

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

Jacqueline de Fátima Teixeira

**UMA PROPOSTA DE SOFTWARE EDUCACIONAL
SIMULADOR PARA ENSINO DE SISTEMAS
OPERACIONAIS**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação.

Prof. José Mazzucco Junior, Dr.

Florianópolis, setembro de 2001.

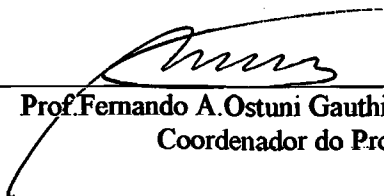
UMA PROPOSTA DE SOFTWARE EDUCACIONAL SIMULADOR PARA ENSINO DE SISTEMAS, OPERACIONAIS

Jacqueline de Fátima Teixeira

Essa Dissertação foi julgada adequada para obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.



Prof. José Mazzucco Junior, Dr.
Orientador

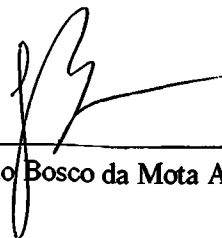


Prof. Fernando A. Ostuni Gauthier, Dr.
Coordenador do Programa

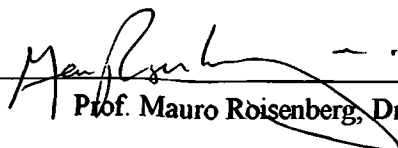
Banca Examinadora:



Prof. José Mazzucco Junior, Dr.



Prof. João Bosco da Mota Alves, Dr.



Prof. Mauro Roisenberg, Dr.

SUMÁRIO

LISTA DE FIGURAS	vii
LISTA DE TABELAS	x
LISTA DE GRÁFICOS	xii
LISTA DE ABREVIATURAS	xiii
RESUMO	xiv
ABSTRACT	xv
CAPÍTULO 1 - INTRODUÇÃO.....	1
1.1. MOTIVAÇÃO.....	1
1.2. OBJETIVOS.....	4
1.2.1. Objetivo Geral.....	4
1.2.2. Objetivos Específicos.....	4
1.3. ESTRUTURA DO TRABALHO.....	4
CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA – PARTE 1: SOFTWARE EDUCACIONAL.....	6
2.1. A RELAÇÃO ENTRE A SOCIEDADE, A TECNOLOGIA DE INFORMÁTICA E A EDUCAÇÃO.....	6
2.1.1. Tecnologias de Informática aplicadas à educação.....	8
2.2. O PROCESSO ENSINO-APRENDIZAGEM: ÊNFASE NO APRENDER OU NO ENSINAR?.....	9
2.3. CLASSIFICAÇÕES DO SOFTWARE EDUCACIONAL.....	12
2.4. SOFTWARE EDUCACIONAL MODALIDADE SIMULAÇÃO.....	15
2.4.1. A simulação e o conceito de micro-mundo.....	17
2.5. SOFTWARE UTILIZADOS PARA ESTUDO E ENSINO DE SISTEMAS OPERACIONAIS.....	18
2.5.1. Sistemas Operacionais Reais.....	18
2.5.2. Sistemas Operacionais Didáticos.....	19
2.5.3. Simuladores de SO.....	20

CAPÍTULO 3 - FUNDAMENTAÇÃO TEÓRICA - PARTE 2: SISTEMA OPERACIONAL.....	22
3.1. CONTEXTO DO SISTEMA OPERACIONAL NA MÁQUINA MULTINÍVEL	22
3.2. OBJETIVOS DO SISTEMA OPERACIONAL.....	25
3.2.1. Histórico Evolutivo dos Sistemas Operacionais	27
3.3. MÓDULOS GERENCIADORES DO SISTEMA OPERACIONAL	30
3.3.1. Módulo Gerenciador de Memória.....	31
3.3.2. Módulo Gerenciador de Arquivos.....	32
3.3.3. Módulo Gerenciador de Sistemas de E/S	33
3.3.4. Módulo Gerenciador de Processos.....	34
3.4. RECURSO.....	34
3.4.1. Recurso como termo de uso geral no estudo de SO.....	35
3.4.2. Recurso como termo usado no estudo de <i>deadlock</i>	35
3.5. MODELO DE PROCESSO.....	36
3.5.1. Definição de Processo	37
3.5.2. Questões sobre Paralelismo	38
3.5.3. Implementação de Processo.....	42
3.5.4. Estados do processo e Transições válidas entre estados	43
3.5.5. Listas de Controle de Processos.....	45
3.6. ESCALONAMENTO	46
3.6.1. Níveis de Escalonamento.....	46
3.6.2. Objetivos do Escalonador.....	47
3.6.3. Algoritmos de Escalonamento	50
3.7. <i>DEADLOCK</i>	63
3.7.1. Conceito de <i>Deadlock</i>	63
3.7.2. Modelo de <i>Deadlock</i> : representação com grafos	66
3.7.3. Estratégias de Tratamento de Situações de <i>Deadlock</i>	69
3.7.4. Detecção de Situações de <i>Deadlock</i>	70

CAPÍTULO 4 - AVALIAÇÃO DE DESEMPENHO DOS ALGORITMOS DE ESCALONAMENTO	79
4.1. ASPECTOS DO MÉTODO DE AVALIAÇÃO	79
4.1.1. Modelos de Avaliação	80
4.1.2. Dados de Entrada	82
4.1.3. Critérios de Avaliação	83
4.2. MÉTODO UTILIZADO PARA AVALIAR O DESEMPENHO DOS ALGORITMOS DE ESCALONAMENTO	84
4.2.1. Modelo de Avaliação.....	84
4.2.2. Dados de Entrada	84
4.2.3. O Ambiente.....	85
4.2.4. Critérios de Avaliação	86
4.3. EXEMPLOS DE ESCALONAMENTO	88
4.3.1. Exemplo de Escalonamento PCPS.....	90
4.3.2. Exemplo de Escalonamento MP	91
4.3.3. Exemplo de Escalonamento Circular	92
4.3.4. Exemplo de Escalonamento MFR.....	96
4.4. ANÁLISE COMPARATIVA DOS RESULTADOS	97
4.4.1. Critérios de avaliação de desempenho de sistema	98
4.4.2. Critérios de avaliação de desempenho de processo	102
 CAPÍTULO 5 – O PROTÓTIPO DO SISO	 105
5.1. APRESENTAÇÃO E ABRANGÊNCIA DO PROTÓTIPO	105
5.2. CARACTERIZANDO O SISO COMO SOFTWARE EDUCACIONAL	107
5.3. MÉTODO DE ANÁLISE DE REQUISITOS DE SOFTWARE	109
5.3.1. Diagrama de fluxo de dados	111
5.3.2. Especificação de procedimentos	113
5.4. MODELAGEM DO PROTÓTIPO DO SISO	118
5.5. APRESENTANDO O PROTÓTIPO DO SISO	118

CAPÍTULO 6 - CONSIDERAÇÕES FINAIS	127
6.1. CONCLUSÕES	127
6.2. PROPOSTAS DE TRABALHOS FUTUROS RELACIONADOS A ESTA PESQUISA	130
REFERÊNCIAS BIBLIOGRÁFICAS	132

LISTA DE FIGURAS

Figura 1.1 – Níveis do sistema computacional.	1
Figura 2.1 – O computador no ensino: um amplificador de capacidades.	8
Figura 3.1 – Relação entre as máquinas virtuais num sistema computacional.	23
Figura 3.2 – Máquinas multiníveis.	24
Figura 3.3 (a) - Dificuldades de comunicação entre <i>hardware</i> e usuários	25
Figura 3.3 (b) - Implementação de camadas sobre o <i>hardware</i> .	25
Figura 3.4 – Histórico Evolutivo dos SO.	28
Figura 3.5 – Organização do Sistema Operacional.	31
Figura 3.6 – Representação de um processo.	37
Figura 3.7 (a) - Paralelismo físico ou multiprocessamento	39
Figura 3.7 (b) - Paralelismo lógico implementando multiprogramação	39
Figura 3.7 (c) - Paralelismo lógico implementando monoprogramação.	39
Figura 3.8 – Processo executando em monoprogramação.	39
Figura 3.9 – Processos executando em multiprogramação.	41
Figura 3.10 – Bloco de Controle de Processo.	42
Figura 3.11 – Diagrama de Estados de Processo.	43
Figura 3.12 – Lista de Controle de Processos.	45
Figura 3.13 – Algoritmos de escalonamento não-preemptivo e preemptivo.	50
Figura 3.14 (a) - Rotina de conclusão de operação de E/S.	51
Figura 3.14 (b) - Rotina de conclusão de processo.	51
Figura 3.14 (c) - Rotina de bloqueio de processo.	51
Figura 3.14 (d) - Rotina de preempção de processo.	51
Figura 3.15 – Esquema do escalonamento não-preemptivo PCPS.	52
Figura 3.16 - Algoritmo de Escalonamento PCPS abordagem não-preemptiva.	53
Figura 3.17-Algoritmo de Escalonamento MP abordagem não-preemptiva.	54
Figura 3.18 – Esquema do Escalonamento Circular.	57
Figura 3.19 - Algoritmo de Escalonamento Circular.	58
Figura 3.20 – Esquema de Escalonamento de MF.	60
Figura 3.21 – Esquema do Escalonador de MFR.	61
Figura 3.22 - Algoritmo de Escalonamento MFR.	62

Figura 3.23 - Exemplo clássico de dois processos em situação de <i>deadlock</i> .	64
Figura 3.24 - Notação de grafos de alocação de recursos.	67
Figura 3.25 - Requisições de recursos de processos concorrentes.	67
Figura 3.26 - Requisição e alocação de recursos representada por grafos dirigidos.	67
Figura 3.27 - Diagrama de Holt.	68
Figura 3.28 - Grafo de alocação de recursos com um recurso de cada tipo.	72
Figura 3.29 - Algoritmo de Detecção com um único recurso de cada tipo.	72
Figura 3.30 - Matrizes para detecção de <i>deadlock</i> com vários recursos de cada tipo.	73
Figura 3.31 - Algoritmo de Detecção com vários recursos de cada tipo.	74
Figura 3.32 - Exemplo de detecção de <i>deadlock</i> com vários recursos de cada tipo.	75
Figura 3.33 - Exemplo de redução de grafos não apresentando situação de <i>deadlock</i> .	76
Figura 3.34 - Exemplo de redução de grafos apresentando situação de <i>deadlock</i> .	76
Figura 3.35 - Alternativas para recuperação de situação de <i>deadlock</i> .	77
Figura 4.1 - Aspectos considerados nas simulações comparativas (realizadas de forma manual) do desempenho dos algoritmos de escalonamento.	84
Figura 4.2 - Esquema de atendimento dos processos no escalonamento PCPS.	90
Figura 4.3 - Esquema de atendimento dos processos no escalonamento MP.	91
Figura 4.4 - Esquema de atendimento dos processos no escalonamento Circular com seleção baseada em prioridade.	93
Figura 4.5 - Esquema de atendimento dos processos no escalonamento Circular com seleção PCPS.	94
Figura 4.6 - Esquema de atendimento dos processos no escalonamento Circular com seleção MP.	95
Figura 4.7 - Esquema de atendimento dos processos no escalonamento MFR.	97
Figura 5.1 - Funções projetadas no SISO.	110
Figura 5.2 - DFD nível 0 do SISO.	111
Figura 5.3 - DFD nível 1 (parcial) - Entrada de dados.	112
Figura 5.4 - DFD nível 1 (parcial) - Simulações e resultados gerados.	112
Figura 5.5 - Relação entre Tabelas com os campos armazenados para cadastramento de processos.	118

Figura 5.6 – Tabelas com os campos armazenados para manter as simulações de escalonamento executadas.	118
Figura 5.7 – Rotinas criadas no protótipo do SISO.	119
Figura 5.8 – Tela Principal do protótipo do SISO.	119
Figura 5.9 – Tela de configurações do SISO.	120
Figura 5.10 – Tela de entrada e manutenção dos processos de usuário a serem simulados.	120
Figura 5.11 – Tela para manter os dispositivos de E/S a serem simulados	121
Figura 5.12 – Tela onde são exibidos os registros das simulações de escalonamento gerados.	122
Figura 5.13 – Primeira parte do relatório de simulação de escalonamento de processos.	122
Figura 5.14 – Segunda parte do relatório de simulação de escalonamento de processos.	123
Figura 5.15 – Tela de entrada de dados para simular detecção de <i>deadlock</i> com vários recursos de cada tipo.	124
Figura 5.16 – Resultado do acionamento do botão Preenche Matriz de Alocação Corrente.	125
Figura 5.17 – Resultado do acionamento do botão Preenche Matriz de Requisições.	125
Figura 5.18 – Tela das informações sobre o desenvolvimento do protótipo.	126

LISTA DE TABELAS

Tabela 2.1 – Relação entre os paradigmas educacionais, suas características e algumas modalidades de SE.	13
Tabela 3.1 – Classificação dos Recursos no contexto do estudo de <i>deadlock</i> .	36
Tabela 4.1 – Duração das fases de uso de CPU e das fases de uso de dispositivos de E/S dos processos de usuário usados nos exemplos.	88
Tabela 4.2 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento PCPS.	90
Tabela 4.3 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento PCPS.	91
Tabela 4.4 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento PCPS.	91
Tabela 4.5 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento MP.	91
Tabela 4.6 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento MP.	92
Tabela 4.7 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento MP.	92
Tabela 4.8 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.	93
Tabela 4.9 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.	93
Tabela 4.10 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.	93
Tabela 4.11 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção PCPS.	94
Tabela 4.12 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção PCPS.	94
Tabela 4.13 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção PCPS.	94

Tabela 4.14 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção MP.	95
Tabela 4.15 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção MP.	95
Tabela 4.16 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção MP.	95
Tabela 4.17 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento MFR.	97
Tabela 4.18 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento MFR.	97
Tabela 4.19 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento MFR.	97
Tabela 4.20 – Resumo dos valores dos critérios de avaliação de desempenho de sistema obtidos nos exemplos de escalonamento.	98
Tabela 4.21 – Valores do critério Produtividade da CPU.	101
Tabela 4.22 – Valores do critério Grau de Multiprogramação.	101
Tabela 4.23 – Valores do critério Tempo de Processamento.	102
Tabela 4.24 – Valores do critério Tempo de Resposta.	103
Tabela 4.25 – Valores do critério Tempo de Espera	104
Tabela 5.1 – Estados válidos para os processos de usuário no protótipo SISO.	114

LISTA DE GRÁFICOS

Gráfico 3.1 – Exemplo de estimativas baseadas em envelhecimento.	55
Gráfico 4.1 – Comparação entre os valores de Tempo Total de Execução obtidos nas abordagens de escalonamento simuladas manualmente.	99
Gráfico 4.2 – Resultados obtidos nos critérios Tempo total de utilização da CPU (por processo de usuário) e Tempo total de utilização da CPU pelo SO.	99
Gráfico 4.3 – Comparação entre os valores de Tempo Total de utilização e Ociosidade da CPU.	100
Gráfico 4.4 – Resultados obtidos no critério Tempo de Processamento.	102
Gráfico 4.5 – Resultados obtidos no critério Tempo de Resposta.	103
Gráfico 4.6 – Resultados obtidos no critério Tempo de Espera.	104

LISTA DE ABREVIATURAS

BCP	Bloco de Controle de Processo
CPU	Central de Processamento de Dados
DFD	Diagrama de Fluxo de Dados
E/S	Entrada e/ou Saída
LCP	Lista de Controle de Processo
LCPP	Lista de Controle de Processos Prontos
MF	Múltiplas Filas
MFR	Múltiplas Filas com Realimentação
MP	Menor Primeiro
PCPS	Primeiro a chegar, Primeiro a ser servido
SE	Software Educacional
SISO	Simulador de Sistema Operacional
SO	Sistema(s) Operacional(is)
UM	Unidade de Memória
UT	Unidade de Tempo

RESUMO

Este trabalho foi desenvolvido objetivando modelar e implementar o protótipo de um *software* educacional que auxilie alunos dos cursos de graduação em Ciência da Computação na compreensão de assuntos específicos da disciplina Sistemas Operacionais. A temática escolhida foi o módulo de Gerência de Processos, especificamente nos assuntos: modelo de processo, estratégias de escalonamento de processador e detecção de situações de *deadlock*. Foram feitas considerações sobre aspectos do processo ensino-aprendizagem, adotando-se no desenvolvimento do protótipo a modalidade de *software* educacional de simulação. O trabalho apresenta ainda o modelo utilizado para avaliação de desempenho dos algoritmos de escalonamento e as fases de desenvolvimento do protótipo.

ABSTRACT

This paper was written with the objective of modeling and implementing an educational *software* prototype that aids students in Computer Science courses in understanding specific subjects in the discipline Operational Systems. The subject chosen was the Processes Management module, specifically in the following areas: process model, processor staggering strategies and detection of *deadlock* situations. Certain aspects of the teaching-learning process were taken into consideration and adopting the modality of educational simulation *software* in developing the prototype. The performance assessment model used on the staggering algorithms is presented as well as the development phases of the prototype itself.

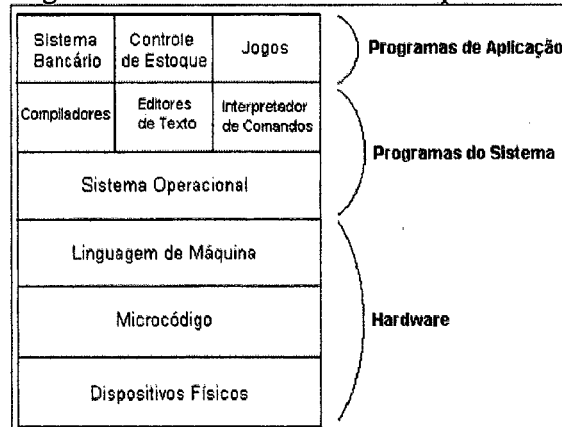
CAPÍTULO 1 - INTRODUÇÃO

As dificuldades no entendimento de conceitos abstratos tratados na disciplina Sistemas Operacionais em cursos de graduação da área de Ciências Exatas e Tecnologia apresentam-se como principal motivador do desenvolvimento deste trabalho, que trata basicamente do desenvolvimento do protótipo de um software educacional denominado SISO (Simulador de Sistemas Operacionais). Neste capítulo são destacadas a motivação e justificativa que levaram ao desenvolvimento deste trabalho, os objetivos geral e específicos e ainda o resumo do conteúdo de cada um dos seis capítulos que o compõe.

1.1. MOTIVAÇÃO

Um sistema computacional pode ser entendido como um grupo de níveis sobrepostos. A Fig. 1.1 apresenta como esses níveis do sistema computacional estão posicionados. Os vários tipos de *software* que foram desenvolvidos ao longo dos últimos 60 anos foram sobrepostos em camadas, denominadas de programas de aplicação e programas de sistema, e eles têm por principal objetivo afastar o usuário da complexidade do *hardware*.

Figural. 1 – Níveis do sistema computacional.



Fonte: TANENBAUM & WOODHULL (2000)

Os **programas de aplicação** são desenvolvidos objetivando atender necessidades específicas de diversas áreas de conhecimento, enquanto que os **programas de sistema** servem de intermediários na comunicação com o *hardware*,

pois independente do grau de facilidade de interação que seja implementada nos programas de aplicação, é no *hardware* que são executadas as instruções dos programas aplicativos.

Os programas do sistema pertencem a uma categoria de *software* denominada **Software Básico** e, segundo TANENBAUM (1995) o **Sistema Operacional** é o mais importante dos programas do sistema. É ele que controla todos os recursos do computador e fornece a base sobre a qual os programas aplicativos são desenvolvidos e executados. Nele são implementadas diversas estratégias para utilização compartilhada dos elementos disponibilizados num sistema computacional como processador, memória, dispositivos de entrada/saída e meios de armazenamento de informações, objetivando o atendimento dos programas de aplicação.

Para o aluno de cursos de graduação da área de Computação é importante conhecer fundamentos de projeto de Sistemas Operacionais para compreender como são implementadas nos Sistemas Operacionais, as estratégias para utilização e compartilhamento de elementos do ambiente computacional para atender aos programas aplicativos de forma satisfatória. Como tais estratégias não são facilmente observáveis pelo nível em que ocorrem nos sistemas computacionais, é necessário que o aluno desenvolva um senso de abstração para compreendê-las.

Como professora da disciplina Sistemas Operacionais nos cursos de graduação de Bacharelado em Ciência da Computação e Tecnologia em Processamento de Dados do Centro de Ensino Superior do Pará (CESUPA), uma Instituição de Ensino Superior da cidade de Belém – Pará, percebe-se que os alunos em geral apresentam certa dificuldade em compreender alguns assuntos do estudo de projeto de Sistema Operacional. Entende-se que esta dificuldade está relacionada ao fato de que muitos dos assuntos tratados no estudo de projeto de Sistema Operacional serem vinculadas diretamente ao *hardware* e serem de difícil demonstração prática.

Assim, o problema identificado como motivador deste trabalho é: como auxiliar o desenvolvimento da disciplina de Sistemas Operacionais a fim de facilitar a compreensão pelos alunos de conteúdos onde predominam assuntos de difícil demonstração, por estarem relacionados a conceitos considerados abstratos.

A fim de resolver este problema foi formulada a hipótese de que o desenvolvimento de um Software Educacional poderia auxiliar os alunos na

compreensão desses conteúdos. A questão então era: entre os vários tipos de *software* educacional, qual seria mais apropriado para explorar os conteúdos técnicos da disciplina?

Segundo TAJRA (2000), *software* educacional pode ser classificado em cinco grupos quanto a sua aplicabilidade: tutoriais, de exercitação, de investigação, simuladores e jogos. Com um *software* educacional do grupo dos simuladores é possível explorar situações fictícias de experimentos que são muito complicados ou que levariam muito tempo para serem processados, ou ainda situações de difícil visualização. No *software* desse grupo, a pedagogia utilizada é a exploração autodirigida, ao invés da instrução explícita e direta própria de *software* educacional tutorial tradicional e de outras modalidades. A simulação envolve a criação de modelos dinâmicos e simplificados do mundo real (dentro do contexto abordado), oferecendo ainda a possibilidade do aluno desenvolver hipóteses, testá-las, analisar resultados e refinar conceitos (VALENTE, 1993). Assim, explorando as características e potencialidades do *software* educacional de modalidade simulação, observou-se que os princípios dessa modalidade poderiam ser interessantes quando aplicados ao desenvolvimento de um *software* educacional onde fossem explorados conteúdos específicos abordados na disciplina Sistemas Operacionais.

Desta forma, a questão de pesquisa norteadora deste trabalho é refletir sobre a questão:

- Que características deve possuir um *software* educacional simulador para auxiliar alunos na compreensão de conteúdos específicos da disciplina Sistemas Operacionais de cursos de graduação relacionados a Ciência da Computação?

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Desenvolver o protótipo de um *software* educacional que permita explorar algumas das estratégias implementadas em um Sistema Operacional (genérico).

1.2.2. Objetivos Específicos

Para atender ao objetivo geral são definidos os seguintes objetivos específicos:

1. Identificar os aspectos pedagógicos associados a um *software* educacional modalidade simulação onde os alunos possam fazer descobertas.

2. Organizar o conteúdo sobre:

- Conceitos gerais sobre Sistema Operacional (SO);
- Modelo de processo;
- Caracterização de situação de *deadlock*, enfatizando a detecção de *deadlock*;
- Estratégias de escalonamento de processador.

3. Definir um modelo de avaliação de desempenho de algoritmos de escalonamento.

4. Modelar e desenvolver o protótipo de um *software* educacional modalidade simulação abrangendo os conteúdos do item 2, utilizando uma metodologia de análise e projeto de sistemas, que vise tratar a qualidade do *software* em todas as fases que compõe o seu desenvolvimento.

1.3. ESTRUTURA DO TRABALHO

Para atender os objetivos propostos, este trabalho está estruturado conforme descrito a seguir.

O Capítulo 1 apresenta as motivações e a justificativa para o desenvolvimento do trabalho, seus objetivos e estrutura.

O Capítulo 2 apresenta questões sobre a fundamentação teórica de *software* educacional, enfatizando características da modalidade simulação. Também são citados *software* educacionais para ensino e estudo de projeto de Sistemas Operacionais.

O Capítulo 3 trata da fundamentação teórica de Sistemas Operacionais apresentando conceitos fundamentais no estudo desse assunto, o contexto em que está inserida a chamada Gerência de Processos, e também são aprofundados os fundamentos dos assuntos específicos enfocados no protótipo (modelo de processo, algoritmos de escalonamento, *deadlock*).

No Capítulo 4 é apresentado um modelo de avaliação de desempenho de algoritmos de escalonamento.

No Capítulo 5 o protótipo do SISO (Simulador de Sistemas Operacionais) é caracterizado como um *software* educacional simulador e são apresentados alguns aspectos da modelagem do protótipo.

No Capítulo 6 são apresentadas as considerações finais deste trabalho e sugestões para trabalhos futuros.

CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA – PARTE 1: SOFTWARE EDUCACIONAL

Neste capítulo são apresentadas cinco seções. A primeira trata da relação da sociedade com a educação, enfatizando o papel das tecnologias de informática e particularmente de *software* educacional como meio para preparar o aluno para o contexto atual, em que é necessário desenvolver o pensar crítico. Na segunda seção discute-se possíveis modelos de aprendizagem, tratando da ênfase no ensinar ou no aprender de cada um deles. Na terceira seção são apresentadas algumas das possíveis classificações de *software* educacional e na quarta seção são enfatizadas as características do *software* educacional modalidade simulação. Na quinta e última seção é feito um apanhado de alguns *software* utilizados para ensino e estudo de Sistemas Operacionais.

2.1. A RELAÇÃO ENTRE A SOCIEDADE, A TECNOLOGIA DE INFORMÁTICA E A EDUCAÇÃO

Vivemos num contexto muito diferente do que existia no início do século em diversos aspectos. Um dos aspectos mais interessantes em que pode-se notar essa incrível diferença é em relação a quantidade de conhecimentos e informações produzidas nas diversas áreas de conhecimento.

Para ter-se idéia de grandeza da quantidade de informações existentes e produzidas, uma analogia muito interessante foi apresentada pelo Prof. Francisco César de Sá Barreto¹, comparando a produção do conhecimento e o volume de informações disponíveis no século XVIII e no século XX com a capacidade de absorção de uma pessoa (considerando que esta é uma pessoa preparada para tal estudo):

¹ O Prof.Dr.Reitor da Universidade Federal de Minas Gerais, Francisco César de Sá Barreto, fez tal comparação na palestra “A universidade brasileira e a universidade do amanhã” proferida em 17/03/2001 na Instituição de Ensino Superior Centro de Ensino Superior do Pará (CESUPA) em Belém-PA.

A quantidade de livros disponíveis no início do século XVIII na Biblioteca de Oxford sobre Filosofia Experimental era da ordem de 200 exemplares. Isso significa que uma pessoa lendo-os de segunda a sexta-feira, 8 horas/dia, teria ao final de um ano lido todos os 200 volumes. No século XX, a produção é da ordem de 20 milhões de trabalhos por ano em todas as áreas de conhecimento. Considerando apenas a área de Bioquímica, onde são publicados 60 mil trabalhos por ano, se uma pessoa passasse um ano, lendo um artigo por hora, durante 10 horas por dia, durante todos os 365 dias do ano, teria lido apenas 6% do que foi publicado somente naquele ano.

Assim, num contexto em que a quantidade de informações já existentes, adicionada à produzida diariamente, excede em muito a que pode ser absorvida por uma pessoa durante toda sua vida é extremamente necessário buscar formas de potencializar o aprendizado das pessoas através de ferramentas apropriadas. É incoerente com a realidade do mundo, que os alunos sejam levados a simplesmente absorver conteúdos de informações apresentados no decorrer de sua vida como estudante pois, antes de tudo, as atividades propostas pelos professores deveriam estimulá-los a desenvolver habilidades para utilizar, relacionar, analisar e avaliar tais informações, levando-as para fora da sala de aula, e relacionando-as com elementos comuns a ele. São essas habilidades que SEABRA (1993) chama de pensar crítico: habilidades compostas por elementos que constituem o pensamento crítico. Se a sala de aula se propõe a desenvolver esse pensar crítico, ele será levado então para fora da sala de aula, sendo extremamente útil na vida profissional do aluno, como fator facilitador frente a tão necessária atualização em qualquer área de conhecimento em que o profissional atue (quer essa atualização seja formal, ou seja, numa instituição de ensino regular ou informal).

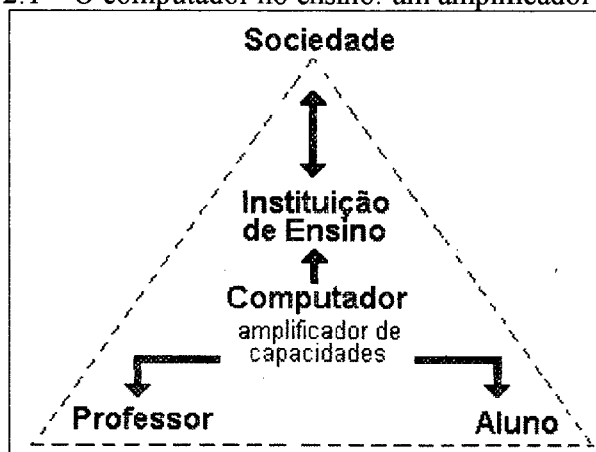
É baseado nesse raciocínio que ALMEIDA (1999,p.7) afirma: *O mercado de trabalho atual está constantemente exigindo que seus profissionais não possuam somente a habilidade de memorizar fatos, mas também que tenham habilidade para aprender novos métodos e novas aptidões.* Isso é o que é chamado na literatura de **aprender a aprender**: mais do que uma tendência futura isso é uma necessidade atual.

2.1.1. Tecnologias de Informática aplicadas à educação

Várias são as ferramentas que podem auxiliar os alunos no processo de aprender a aprender, e o uso do computador nesse processo é uma delas. O computador serve então como um grande aliado nessa proposta tão necessária, servindo como um amplificador de capacidades, e segundo BARRETO (1999, p.216), *ajudando a desenvolver a capacidade de aprender a aprender e personalizando a transmissão de conhecimentos no processo de aprendizado contínuo.*

Na Fig. 2.1 são mostrados os elementos existentes em uma determinada sociedade. As instituições de ensino devem estar atentas às necessidades da sociedade frente às constantes mudanças de seus interesses, e como seus membros também fazem parte dessa sociedade, estão intimamente comprometidos na formação e aperfeiçoamento daqueles colocados sob sua responsabilidade. O computador, que aqui representa as diversas tecnologias de informática existentes e usadas na educação, pode ser usado como amplificador de potencialidades na capacitação dos alunos, professores e da instituição de ensino.

Figura 2.1 – O computador no ensino: um amplificador de capacidades.



Fonte: BARRETO (1999, p.216).

Percebe-se que há motivações diferentes para o uso do computador nas diferentes áreas. Em algumas áreas, o uso do computador é o único meio viável de utilizar determinadas aplicações, como é o caso das reservas de passagens, telecomunicações, serviços bancários, etc. Mas diferente de aplicações dessas áreas estão as aplicações do computador na área de educação (ALMEIDA, 1999). Há vários meios e ferramentas para se trabalhar em educação, sendo o uso do computador apenas

uma delas, e certamente não é o mais apropriado em todas as situações. Tudo depende do uso que se faz dele e dos tipos de conhecimentos.

Concorda-se com RAMOS (1996), quando ela questiona se uma aplicação computacional qualquer, e o *software* utilizado não é intrinsecamente também um ambiente de aprendizado, apesar de alguns já serem desenvolvidos visando o uso no processo educativo, enquanto outros são desenvolvidos para outros fins (como pacotes gráficos, planilhas eletrônicas, linguagens de programação, sistemas operacionais, drivers, etc.). GIRAFFA (1999, p.1) diz que: *Todo programa pode ser considerado um programa educacional desde que utilize uma metodologia que o contextualize no processo ensino-aprendizagem.* Assim, qualquer *software* pode ser utilizado no processo educacional, dependendo do objetivo de seus usuários.

2.2. O PROCESSO ENSINO-APRENDIZAGEM: ÊNFASE NO APRENDER OU NO ENSINAR?

É comum que professores de terceiro grau não tenham formação pedagógica específica e suas concepções sobre aprender sejam as mais diversas possíveis, muitas vezes evoluindo ao longo de sua vida profissional como professor. Segundo MASETTO (1998) essa situação tem conotação histórica pois desde que os cursos superiores e as faculdades começaram a ser criadas e instaladas no país (a partir de 1808), elas voltaram-se diretamente para a formação de profissionais que exerceriam uma determinada profissão, isto é, profissionais competentes em uma determinada área ou especialidade.

Com esse objetivo na formação dos alunos, os cursos superiores e faculdades procuravam compor seu quadro docente com profissionais renomados, com sucesso em suas atividades profissionais, para fazer com que seus alunos se tornassem tão bons profissionais quanto eles. Essa situação se fundamenta numa crença comum a quem convida o profissional e a ele próprio (de quem aceita o convite): quem sabe (fazer), automaticamente sabe ensinar. Atualmente sabe-se que o exercício de qualquer profissão exige capacitação própria e específica e não é diferente com a docência de ensino superior.

A docência de ensino superior exige não apenas domínio de conhecimentos a serem transmitidos por um professor como também um profissionalismo semelhante aquele exigido para o exercício de qualquer profissão (MASETTO, 1998, p.13). Esse profissionalismo exigido do professor implica em atualizações em pelo menos dois aspectos: em sua área de conhecimento e na área de educação, nas diversas facetas do chamado processo ensino-aprendizagem. Ambas são inquestionavelmente imprescindíveis, pois quando mais o professor conhece sobre estudos, pesquisas e teorias relacionados com os processos de ensino e de aprendizagem humana, ele pode aprimorar-se, e é a crença no processo de ensino-aprendizagem que o professor tenha que direcionará seus atos na sua vida docente.

Enquanto **ensino** enfatiza o professor (a sua pessoa, as suas qualidades e as suas habilidades), a **aprendizagem** centra-se no aluno (em suas capacidades, suas possibilidades, suas oportunidades e preocupada em oferecer condições para que o aluno aprenda). Apesar de esses dois aspectos não poderem ser desassociados, entende-se que, dependendo da ênfase que se dê a um ou a outro, o direcionamento das propostas das instituições de ensino e dos professores diversificam-se (ABREU & MASETTO, 1990). Assim, entende-se que o processo ensino-aprendizagem deve privilegiar a aprendizagem dos alunos sobre o ensino de seus professores, pois a instituição de ensino, qualquer que seja seu nível, existe em função do aluno.

Mas o que é **aprender**? Pessoas que atuam como professores, talvez mais do que em qualquer outra profissão, estão envolvidas com a resposta a essa questão, pois dependendo da concepção sobre aprender que um professor tenha, ele definirá sua postura frente a uma turma de alunos, suas estratégias para cumprir um determinado conteúdo a desenvolver com a turma, julgar a evolução do aprendizado dos alunos, enfim, direcionar os seus esforços para cumprir um determinado objetivo.

Assim, a definição de aprender de cada professor (e no caso de quem concebe um *software* educacional), depende de um modelo que caracteriza suas concepções sobre aprender. PORLÁN & RIVERO² apud MORAES (1999), afirmam que as concepções sobre aprender podem ser agrupadas em cinco grandes modelos:

- receptiva,
- por assimilação,
- por descoberta,
- por substituição de conhecimentos e
- por construção.

Será feito um breve apanhado de cada um desses modelos.

No modelo de aprendizagem receptiva o processo de aprendizado é caracterizado pela memorização de conteúdos, pela acumulação de conhecimentos. Assim, a aprendizagem acontece pela retenção a partir da fala do professor, onde os conhecimentos anteriores dos alunos são desconsiderados ou considerados errôneos.

O modelo de aprendizagem por assimilação valoriza o estabelecimento de relações com conhecimentos escolares anteriores, valorizando a aprendizagem significativa (aquilo que pode ser conectado com aprendizagens escolares anteriores). Assim, aprender por assimilação é apropriar-se de um conhecimento novo, e o fato de aplicar aquele novo conhecimento, evidencia o aprendizado.

A aprendizagem por descoberta enfatiza a descoberta de princípios e conceitos por meio do esforço do próprio aluno, valorizando a experimentação e o trabalho prático como fatores facilitadores do aprendizado.

Na aprendizagem por substituição de conhecimentos já existentes, considera-se que cada aluno constrói conhecimento ao longo de sua vida. Entretanto, esses conhecimentos são considerados como errôneos ou informais por não atenderem aos critérios de validade de conhecimentos científicos. Assim, aprender neste modelo, consiste em substituir as concepções erradas ou espontâneas pelos conhecimentos científicos.

Já no modelo de aprendizagem por construção (aprendizagem por evolução de conhecimentos) valorizam-se os conhecimentos prévios dos alunos, tomando-os como ponto de partida para a construção de novos conhecimentos. Assim, aprender é tornar

² PORLÁN, R. RIVERO, ^a El Conocimiento de los profesores. Sevilla. ES. Díada, 1998.

mais complexo o conhecimento cotidiano, fazendo com que a aprendizagem ocorra por mudanças conceituais, no sentido de evolução de conceitos já existentes.

Nos modelos de aprendizagem receptiva, por assimilação e por descoberta, entende-se que o conhecimento se origina no meio (ou ambiente), e o aluno o capta, assimila, absorve ou o descobre. Mesmo no modelo por descoberta, onde o aluno tem uma atitude mais participativa, ele só pode descobrir o que o meio lhe oferece, algo que já está pronto na realidade, e por ele é descoberto. Os modelos por substituição de conhecimentos e por construção defendem que o conhecimento prévio influi em novas aprendizagens, e a aprendizagem é entendida como uma construção do indivíduo, sendo essa construção resultante da interação do aluno com o objeto de conhecimento.

Assim, MORAES (1999) afirma que os três modelos iniciais, onde o conhecimento se origina do meio e o indivíduo o descobre ou assimila, se caracteriza por ter como epistemologia básica o chamado empirismo; e os dois últimos, onde o indivíduo é o construtor ativo do conhecimento, através de sua interação com o meio, têm como base a epistemologia chamada de interacionista. Segundo RAMOS (1996), entender o processo de aprendizagem baseado em uma dessas bases epistemológicas está impregnado de um direcionamento numa determinada visão da natureza humana, e que reflete os ditos paradigmas educacionais a saber: comportamentalista ou construtivista.

2.3. CLASSIFICAÇÕES DO SOFTWARE EDUCACIONAL

Quando um *software* educacional (SE) é desenvolvido para ser utilizado como apoio ao processo de aprendizado de um determinado conteúdo, entende-se que uma das etapas no seu desenvolvimento é definir a concepção pedagógica daqueles que estão envolvidos na sua modelagem e/ou implementação. Esse raciocínio é confirmado por RAMOS (1996, p.3) quando ela afirma que essa etapa de desenvolvimento de SE trata-se da *primeira e principal etapa, pois o tipo de uso a que se destina, reflete a concepção pedagógica do software*. Nessa mesma obra, RAMOS apresenta algumas características de *software* de acordo com o paradigma educacional utilizado na concepção e desenvolvimento de SE de diferentes modalidades. Quando um *software* é utilizado para fins educacionais, invariavelmente o mesmo (ou o uso que se faz dele) reflete um desses

dois paradigmas educacionais. A Tabela 2.1, produzida a partir de suas observações, apresenta algumas dessas classificações em relação a modalidades de *software*.

Tabela 2.1 – Relação entre os paradigmas educacionais, suas características e algumas modalidades de SE.

Paradigma Educacional	Visão da natureza humana	Quanto a atividade do aprendiz	Quanto ao direcionamento na utilização do <i>software</i>	Modalidades de SE
Comportamentalista	Empirista e racionalista	Algorítmico	Dura	Tutoriais, Exercitação e prática
Construtivista	Interacionalista	Heurístico	Branda	Simulação, Jogos

Quanto a atividade do aprendiz, um *software* pode ser algorítmico ou heurístico. Num *software* algorítmico é predominante a ênfase na transmissão de conhecimentos do sujeito que sabe para o sujeito que deseja aprender, sendo função do desenvolvedor do *software* projetar uma seqüência bem planejada de atividades que conduzam o aluno ao conhecimento desejado. Já num *software* heurístico, predomina a aprendizagem experimental ou por descobrimento, devido a criação de um ambiente rico em situações que o aluno deve explorar.

Quanto ao direcionamento na utilização do *software*, podem ser consideradas duas abordagens: dura e branda. Na abordagem dura os planos são previamente traçados para uso do computador e as atividades dos alunos resumem-se a responder a perguntas apresentadas, registrando-se e contabilizando-se erros e acertos. Na abordagem branda a atividade e interação com o computador não parecem ter um objetivo definido, fazendo com que o aluno esteja no comando, realizando uma série de atividades consideradas interessantes por ele e onde há desafio. Os erros são fontes de reflexão e desenvolvimento de novos projetos.

TAJRA (2000) classifica *software* quanto a sua natureza como:

- educacionais,
- aplicativos com finalidade tecnológica e
- aplicativos com finalidade educativa.

Os educacionais são aqueles que melhor se adaptam a proposta de ensino do professor/escola para o assunto específico a ser abordado, entre os disponíveis no mercado. Em *software* dessa natureza não há preocupação em repasses de conteúdo tecnológico. Nos aplicativos com finalidade tecnológica, como por exemplo os editores de texto e compiladores, a importância é dada aos conceitos relacionados à informática

e são utilizados principalmente em cursos de formação profissionalizante. Os aplicativos com finalidade educativa são os mesmos da classificação anterior, porém aos usá-los, os alunos são estimulados a perceberem a sua utilização na elaboração de produções integradas ao contexto educacional.

Outra importante classificação dos SE tratada por um grande número de autores é em relação a função que os mesmos desempenham. Assim, os SE podem ser dos seguintes tipos:

- tutoriais,
- de exercitação e prática,
- simuladores,
- jogos educacionais

Apesar de VALENTE (1993) afirmar que essa descrição dos tipos de SE apresentada nas literaturas é bastante didática, sendo impossível encontrar um *software* puramente desenvolvido e utilizado com as características de uma única classificação, será feito um breve resumo de cada um deles.

Nos SE tutoriais o aluno é guiado por distintas fases de aprendizagem, onde seguem-se as fases do processo de aprendizagem descritas por Gagné³. Nos SE deste tipo é adotado o mesmo padrão de ensino da sala de aula tradicional, sendo o conteúdo organizado numa estrutura predefinida, e cabe ao aluno selecionar entre as opções existentes, o conteúdo que deseja estudar.

Geralmente os tutoriais são ricos em significativas inovações tecnológicas como hipertexto, interfaces bem elaboradas (com sons, imagens, animações, etc.), versões para WEB, etc. que tornam a apresentação do conteúdo atraente, retendo a atenção do aluno. Alguns ainda verificam o aprendizado do aluno, através de perguntas, permitindo interação. As respostas são então verificadas e em alguns há uma explicação e/ou sugestão de estudo para as respostas consideradas incorretas. Mais recentemente foram

³ MOREIRA (1981, p.23) afirma que, segundo Gagné, a série típica de eventos que acompanham um ato de aprendizagem pode ser analisada através de oito fases identificadas a seguir, com a indicação do processo envolvido na mesma: motivação (expectativa), apreensão (atenção, percepção seletiva), aquisição (codificação, entrada de armazenamento), retenção (armazenamento na memória), rememoração (recuperação), generalização (transferência), desempenho (resposta) e retroalimentação (reforço). Assim, segundo a concepção de Gagné, a aprendizagem é uma mudança de estado interior que se manifesta através da mudança de comportamento e na persistência dessa mudança. Desta forma, um observador externo pode reconhecer que houve aprendizado quando observa a ocorrência da mudança comportamental e também a permanência dessa mudança, pois a aprendizagem é uma mudança comportamental persistente.

desenvolvidos os chamados Tutores Inteligentes⁴ (ITS – Intelligent Tutoring Systems) que utilizam-se de técnicas de Inteligência Artificial e teorias pedagógicas para conduzir o aluno dentro de um determinado domínio de conhecimento.

Os programas de reforço/exercício ou de exercitação e prática são aplicações que centram-se basicamente em duas das fases de aprendizagem propostas por Gagné: a aplicação e a retroalimentação, sendo sua proposta mais modesta que a dos tutoriais. São utilizados para revisão e memorização de algum assunto já estudado pelo aluno. GIRAFFA (1999) afirma que as versões mais recentes destes programas utilizam recursos hipermídia mas ainda mantendo estas mesmas características.

Software classificado como simuladores apóia-se na construção de situações que se assemelham com a realidade e enfatiza a exploração autodirigida, ao invés da instrução explícita e direta própria de SE tutoriais e de outras modalidades. A simulação envolve a criação de modelos dinâmicos e simplificados do mundo real (micro-mundo), dentro do contexto abordado, oferecendo ainda a possibilidade do aluno desenvolver hipóteses, testá-las, analisar resultados e refinar conceitos. Por tratar-se da modalidade de SE proposta neste trabalho, a seção seguinte explorará mais esse assunto.

Os jogos educacionais⁵ assim como os simuladores apóiam-se na construção de situações que se assemelham com a realidade, sendo que os jogos apresentam ainda um componente lúdico e de entretenimento.

2.4. SOFTWARE EDUCACIONAL MODALIDADE SIMULAÇÃO

A criação de modelos é o início do pensamento puramente teórico sobre o funcionamento das coisas (GRAVINA & SANTAROSA, 1998, p.76). Assim, a característica dominante da modelagem é a explicação, manipulação e compreensão das

⁴ ITS (Intelligent Tutoring Systems) é um *software* capaz de tutorar um aluno num determinado domínio de conhecimento, sabendo o que ensinar, como ensinar e aprende informações relevantes sobre o aluno que esta sendo tutorado, sendo assim capaz de fazer julgamentos sobre o que o aluno sabe, e como ele esta progredindo. Isso é possível graças a definição de um modelo de aprendiz, e se o desempenho do aluno não condiz com o que foi previsto, o sistema busca identificar a deficiência e toma uma decisão instrucional, isto é, determinar uma estratégia de tutoramento (AZEVEDO, 1997).

⁵ As diferenças conceituais entre jogos e simulação podem ser caracterizadas pelo fato de que o jogo é um processo intrinsecamente competitivo (em que co-existem a vitória e a derrota) e uma simulação é a simples execução dinâmica de um modelo previamente definido (GIRAFFA, 1999).

relações entre as variáveis que controlam o ambiente proposto, sendo o feedback oferecido pela máquina um recurso fundamental para o ajuste de idéias.

A simulação envolve a criação de modelos dinâmicos e simplificados do mundo real (VALENTE, 1993, p.7). Com esses modelos é possível explorar situações que são fictícias, de risco, complicadas, inviáveis financeiramente, etc. Assim, SE desenvolvidos na modalidade simulação oferecem a possibilidade de o aluno desenvolver hipóteses sobre um determinado contexto, analisar resultados obtidos com a experimentação e refinar conceitos.

Essas atividades envolvem também aspectos de tomada de decisão, pois se um determinado objetivo é almejado, é necessário definir uma estratégia para alcançá-lo, o que envolve considerar várias possibilidades e decidir por um determinado roteiro de ações. Se o resultado não for o esperado, é possível rever a estratégia e alterá-la, no que se julgar necessário e sem grandes implicações, pois se trata de apenas um modelo que representa um determinado contexto.

Nota-se que bons programas de simulações não são fáceis de desenvolver, pois precisam ser atrativos, e isso implica, além de recursos de *hardware* e *software*, em boa dose de criatividade. Diferente dos jogos, em que a competição é uma grande motivação para seu uso, os simuladores precisam apresentar itens de qualidade de SE que façam com que o aluno se interesse e deseje manipulá-lo.

É necessário ainda complementar o uso de simuladores com outras atividades como aulas expositivas e discussões em sala de aula, pois assim será possível que o aluno relacione os conceitos intrínsecos da simulação com aqueles conteúdos que se deseja explorar.

GIRAFFA (1999) afirma que há dois tipos de simulação: a determinística e a estocástica. Na simulação determinística o resultado é sempre o mesmo quando determinados parâmetros são definidos, e a simulação estocástica apresenta uma aleatoriedade nos parâmetros que faz com que a simulação apresente resultados diferenciados a cada nova execução.

Assim, a simulação implica necessariamente em um modelo computacional que represente eventos que ocorrem num determinado contexto. Tais eventos acontecem de forma contínua em relação ao tempo e de forma discreta em relação as ações, já que são continuamente interrompidos e retomados durante uma seção de simulação.

Segundo GARCIA et all (1997) há duas possíveis abordagens para o uso de sistemas educacionais quanto ao uso de recursos de animações gráficas que podem estar presentes nos SE simuladores: a passiva e a ativa. Na abordagem passiva, de forma similar a quando se assiste a um filme ou a execução de uma seqüência pré-programada de eventos, o usuário se comporta como mero espectador, sendo sua utilização voltada à complementação de livros-texto e aulas. Uma limitação clara dessa abordagem no entanto, está no fato de o usuário não poder testar novos conjuntos de dados e ter de ficar limitado aos conjuntos a ele fornecidos. Já na abordagem ativa são introduzidos no sistema algumas facilidades para que os usuários possam fazer as suas próprias experiências, direcionando o andamento dos algoritmos.

Assim, independente de suas possíveis variações, SEABRA (1993) afirma que atividades que envolvam simulação estimulem os alunos a desenvolverem o chamado **pensar crítico**.

2.4.1. A simulação e o conceito de micro-mundo

A simulação está também relacionada a um conceito denominado **micro-mundo**. Micro-mundo é um cenário relevante para o aprendizado, onde ocorrem fatos dependendo do que o aluno realiza. Nesse ambiente nem toda a complexidade do mundo que é objeto de conhecimento é refletida, mas sim as variáveis consideradas relevantes (GALVIS, 1997). Assim, o comportamento dos componentes do micro-mundo é gerado pelas variáveis que estão sob controle do usuário e pode ser afetado baseado nas ferramentas tecnológicas a seu dispor.

Dependendo da relação com o eixo de aprendizado que se pretende propiciar no SE, os micro-mundos podem ser intrínsecos ou extrínsecos. Os micro-mundos extrínsecos são aqueles que se limitam a emoldurar o cenário compreendido no SE, não tendo grande relação com o cenário inspirador. Os micro-mundos podem ter características intrínsecas, isto é, são aqueles desenvolvidos por pessoas que além de conhecerem bem o assunto foco, conhecem também seus alunos e o contexto em que eles atuam, estando baseados no argumento de que, além de definirem cenários que exploram o assunto que se pretende tratar no micro-mundo, fazem isso de forma

criativa, para que os alunos se interessem e trabalhem ativamente na busca ou na consolidação do conhecimento. É o caso dos SE de simulação e/ou jogos educacionais.

Nos micro-mundos é o aluno quem conduz a ação, é ele quem possui um certo controle dos eventos que deflagram comportamentos desse ambiente. Alguns micro-mundos são sintônicos, isto é, o aluno nem sequer precisa aprender a usá-los, quer seja pela facilidade de interação, quer seja pela familiaridade com os argumentos apresentados. Outros micro-mundos, não são considerados sintônicos pois precisam de uma certa dose de disposição do aluno para compreender o argumento, o tema e a maneira de usar as ferramentas que estão disponíveis para manipulá-lo.

2.5. SOFTWARE UTILIZADOS PARA ESTUDO E ENSINO DE SISTEMAS OPERACIONAIS

São apresentados a seguir alguns produtos existentes usados para ensino e estudo de SO. Alguns são SO reais de código aberto, enquanto outros são SO didáticos, isto é, desenvolvidos com finalidade educacional. Apresenta-se ainda alguns sistemas aplicativos que simulam o funcionamento de um SO.

2.5.1. Sistemas Operacionais Reais

No início do desenvolvimento do SO UNIX na década de 70, quando a AT&T (sua proprietária) permitia o uso de seu código em atividades didáticas, muitas universidades nos EUA o utilizavam em cursos de SO. Outros SO reais existentes na época, eram proprietários e certamente pouco didáticos, por apresentarem pouca documentação, o que dificultaria seu entendimento e estudo (ANIDO, 2000). Quando a versão 7 do UNIX foi lançada pela AT&T, a empresa proibiu a distribuição do seu código e seu uso didático não foi mais possível.

Atualmente, com o aparecimento de SO de código aberto, como o LINUX e FreeBSD, há outras opções para o uso de SO reais nas atividades didáticas.

A principal vantagem no uso de SO reais é a satisfação dos alunos ao manipularem um *software* complexo e real. Porém um ponto tende a dificultar

atividades que envolvam o estudo de SO reais, que é o fato de o código nem sempre ser bem documentado, além de possuírem estruturas de dados e mecanismos muito sofisticados, o que exige dos alunos mais dedicação para compreendê-lo (ANIDO, 2000).

2.5.2. Sistemas Operacionais Didáticos

Três SO didáticos foram desenvolvidos nas décadas de 80 e 90: o MINIX, o XINU e o NACHOS.

O MINIX⁶ (mini-UNIX) é um SO desenvolvido por Andrew Tanenbaum com o objetivo de ser utilizado para estudo de SO, isto é, desenvolvido para ser uma ferramenta educacional. É compatível com UNIX e com processadores Pentium e está escrito em linguagem C. Apesar do MINIX implementar tudo que um SO real possui, é tão pequeno e simplificado que pode ser usado em aulas de laboratório, para que os alunos examinem como um SO real funcionaria por dentro. Mais informações sobre o MINIX podem ser buscadas em [<http://www.cs.vu.nl/~ast/minix.html>].

O sistema XINU foi desenvolvido por Douglas Comer e também trata-se de uma ferramenta com código aberto que pode ser usado para ensino e estudo de SO. Mais informações sobre o XINU podem ser buscadas em [<http://willow.canberra.edu.au/~chrisc/xinu.html>].

O NACHOS é também um SO didático. Foi desenvolvido no início dos anos 90, na University of California, Berkeley, por W.Christopher, Sam Protecter e Tomas Anderson. Esse *software* implementa várias atribuições de um SO moderno como multiprogramação, memória virtual, sistema de arquivos, protocolo de rede, chamada remota a procedures, sistemas distribuídos, etc. É desenvolvido em C++, e roda em diversas plataformas como: DEC MIPS, rodando Ultrix, SUN SPARCstations (testado em SunOS) e HP PA-RISC, rodando HP-UX. Documentação e download podem ser encontrados em [<http://hegel.ittc.ukans.edu/projects/nachos/>].

Segundo ANIDO (2000) estes sistemas didáticos têm a vantagem de serem pequenos, elegantes e relativamente bem documentados. A desvantagem é que eles

⁶ TANENBAUM & WOODHULL (2000) apresentam, além de princípios teóricos do estudo de SO, o código completo e comentado do MINIX.

foram desenvolvidos para uso acadêmico, apresentando por isso, algumas limitações e simplificações quando comparados a SO reais.

2.5.3. Simuladores de SO

O uso de simuladores simplifica o desenvolvimento de tarefas por parte dos alunos, por sua própria proposta que é desenvolver um modelo simplificado de um assunto específico tratado em um SO real, mas com as simplificações próprias de um SO didático. Assim, normalmente os simuladores não implementam toda a funcionalidade de um SO, mas apenas uma pequena parte, relativa ao assunto que se pretende desenvolver com os alunos. A seguir são mencionados alguns exemplos de simuladores.

O projeto denominado de SO proposto por Fernando Albuquerque da Universidade de Brasília, tem por objetivo implementar parte de um SO multiprogramado. Trata-se da descrição de várias rotinas escritas em linguagem C compatível com processadores da linha PC, para descrição das políticas adotadas para gerenciamento de processos e recursos presentes no sistema⁷. Além do projeto SO, o autor também apresenta o código de um programa denominado SIMULADOR, através do qual podem ser testadas e observadas, as operações executadas pelas diversas rotinas que compõem o sistema.

William Shay da University of Wisconsin (Green Bay), apresenta um projeto de um simulador de SO que trata principalmente de escalonamento, gerenciamento de memória e sincronização de processos. Nele, *os processos são gerados aleatoriamente, fazendo solicitações aleatórias de serviços, e requerem um número aleatório de páginas de memória, além de gerarem chamadas de espera e sinal aleatoriamente* (SHAY, 1996, p.686). O projeto tem os programas escritos em linguagem Pascal⁸, e segundo o autor, é independente de *hardware*.

Outra experiência de desenvolvimento de simulador de SO é do Prof. Luiz Paulo Maia, autor do livro “Arquiteturas de Sistemas Operacionais” amplamente adotado nos cursos de graduação. O Prof. Luiz Paulo Maia desenvolveu o SOSim, um simulador de SO que abrange estratégias de gerenciamento de memória virtual com paginação

⁷ Código do programa disponível em ALBUQUERQUE (1990).

⁸ Código do programa disponível em SHAY (1996).

(mapeamento, working set, políticas de alocação e substituição de páginas, swapping e thrashing), gerência do processador (escalonamento não-preemptivo, escalonamento preemptivo por tempo e prioridade) e permite visualizar as estruturas internas do processo (contexto de *hardware*, contexto de *software* e espaço de endereçamento virtual). Desenvolvido em Delphi, esse projeto faz parte de sua dissertação de Mestrado em Informática pela UFRJ/NCE defendida em março/2001. Contatos com o autor podem ser realizados através do endereço eletrônico [lpmaia@ism.com.br].

O VisualOS é um simulador educacional visual de sistema operacional cuja finalidade é ajudar a compreender como trabalham os sistemas operacionais mostrando representações visuais dos aspectos referentes a cada um dos subsistemas simulados: CPU, Memória e dispositivos de E/S. Foi desenvolvido em linguagem C e roda como um aplicativo sobre o SO POSIX. O VisualOS foi desenvolvido por Manuel Estrada Sainz como um projeto vinculado à Universidade de Salamanca (Espanha). Atualmente é utilizado em centros educacionais como Institut National Polytechnique de Grenoble e Institut National des Sciences Appliquées de Rouen, ambos na França. Está disponível no endereço [<http://visualos.sourceforge.net/>].

EduOS é um *software* educacional simulador para estudo de SO. Trata-se de um SO simplificado para rodar nos processadores 80386+, compilador e executado exclusivamente sobre o SO LINUX ou BeOS. Foi desenvolvido por Cornelis Frank e alguns colaboradores. Mais informações estão disponíveis em [<http://inf4serv.rug.ac.be/~fcorneli/eduos/#what>].

O RCOS.java é uma ferramenta projetada para ajudar a compreender o funcionamento interno de um SO. É uma ferramenta animada que simula um SO multitarefa, escrito por Andrew Newman e David Jones. Informações podem ser obtidas em [http://cq-pan.cqu.edu.au/david-jones/Publications/Papers_and_Books/rcos.java/] e também em [<http://www4.cs.uni-dortmund.de/MA/misch/applets/rcos.java/whatis.html>].

CAPÍTULO 3 - FUNDAMENTAÇÃO TEÓRICA -

PARTE 2: SISTEMA OPERACIONAL

Este capítulo é composto de sete seções. Na primeira seção é mostrado o contexto em que o SO atua, posicionando-o entre os diversos níveis de *software* existentes num sistema computacional. Na segunda seção são apresentados seus objetivos que por sua vez justificam sua existência, além de mostrar um breve histórico evolutivo, utilizando para isso, marcos históricos da evolução dos sistemas computacionais. A seção seguinte mostra a divisão do SO em módulos, apontando os objetivos de cada um deles. Na quarta seção é explicado o entendimento do termo **recurso**, já que o mesmo tem um significado geral e amplo, e no contexto do estudo de SO é considerado termo chave. Nas três últimas seções busca-se expressar de forma bastante direcionada a fundamentação teórica dos três principais assuntos abordados nesse trabalho: modelo de processo, estratégias de escalonamento de CPU e estudo de *deadlock*.

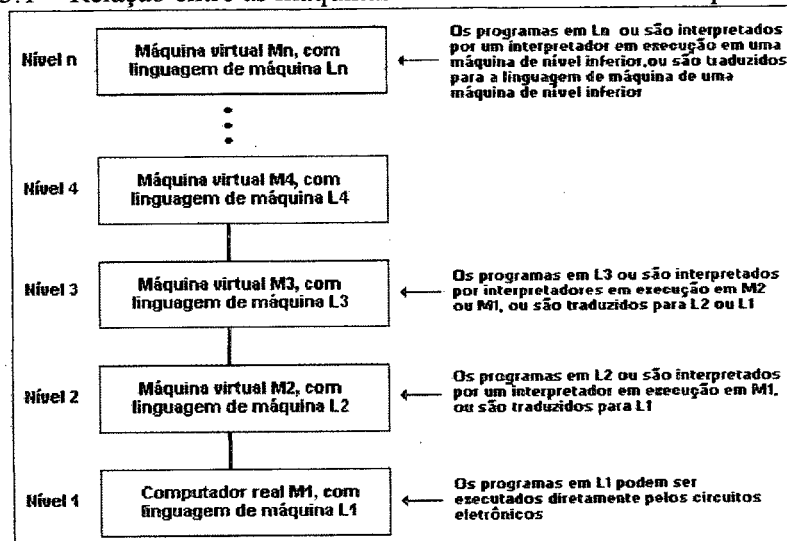
Para produção deste capítulo foi realizada pesquisa bibliográfica utilizando várias referências que tratam de fundamentação teórica sobre SO. Para isso foi utilizado material de fontes diretas e indiretas, de autores nacionais e internacionais, expresso em livros, teses, periódicos e também material disponível na Internet.

3.1. CONTEXTO DO SISTEMA OPERACIONAL NA MÁQUINA MULTINÍVEL

Os computadores atuais podem ser entendidos como constituídos por uma série de níveis sobrepostos, onde cada nível superior recebe funcionalidades dos níveis inferiores. Cada nível representa uma abstração distinta, com a presença de diferentes objetos e operações, onde a camada superior define as funções e a inferior contém os detalhes de mais baixo nível para executá-las. As camadas intermediárias correspondem a estágios de refinamento. Para realizar a comunicação entre os níveis é utilizado algum mecanismo tradutor, como por exemplo a interpretação ou a compilação.

Pode-se então imaginar cada um desses níveis como um computador hipotético ou máquina virtual, com uma linguagem própria para ela (Fig. 3.1). *Um computador com n níveis pode ser considerado como tendo n diferentes máquinas virtuais, cada uma delas com uma diferente linguagem de máquina* (TANENBAUM, 1999, p.3).

Figura 3.1 – Relação entre as máquinas virtuais num sistema computacional.



Fonte: TANENBAUM (1999, p.4).

A idéia de definir níveis ou camadas é a de proporcionar transparência, isto é, distanciar cada vez mais a linguagem utilizada nos diferentes níveis dos detalhes dos níveis inferiores mais próximos ao *hardware*. Por exemplo, os usuários de um nível n nem precisam estar cientes da existência dessas máquinas virtuais em camadas. Para eles, é como se existisse uma máquina (virtual) que executa diretamente o conjunto de comandos da linguagem de programação escritos por eles.

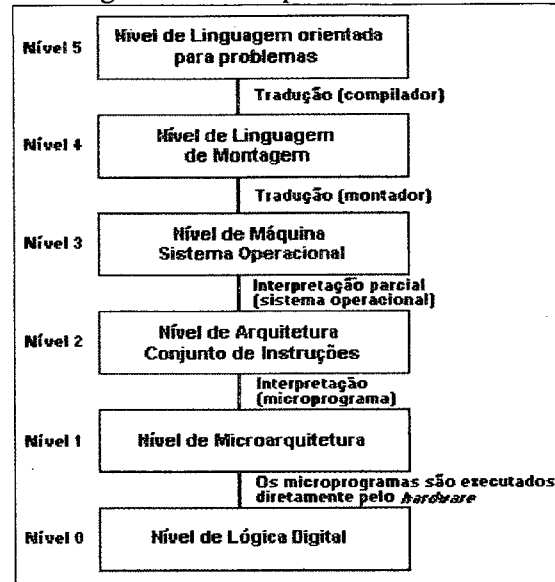
A Figura 3.2 mostra os níveis de um sistema computacional, apresentando ainda os métodos de tradução utilizados entre os níveis, e o *software* que os implementam. Será comentado brevemente, cada um dos níveis para contextualizar o nível 3 onde está localizado o nível de Sistema Operacional.

O nível 0 denominado de nível de lógica digital é o *hardware* propriamente dito, onde há circuitos eletrônicos (portas lógicas) executando as instruções em linguagem de máquina (oriundas do nível 1).

Nos níveis 1 e 2, onde estão os níveis de microprograma e de máquina convencional, são definidos os microprogramas ou equivalentes. Esse conjunto de

instruções⁹, que geralmente não chega a uma centena, é implementado e um determinado processador¹⁰ (ou famílias de processadores) é projetado para reconhecê-lo, interpretá-lo e executá-lo. A comunicação entre os níveis 0, 1, 2 e 3 se dá através de interpretação¹¹ ou execução direta.

Figura 3.2 – Máquinas multiníveis.



Fonte: TANENBAUM (1999, p.5).

O nível 3 (o nível de SO) contém um conjunto de novas instruções tratadas por um interpretador em execução no nível inferior ou diretamente pelo nível de microprograma. Alguns autores como SILBERSCHATZ & GALVIN (2000), SHAY (1996) e NUTT (1997) apresentam esquemas similares ao apresentado anteriormente na Fig. 1.1, onde não há distinção entre os níveis 0, 1 e 2, considerando-os como um único nível de *hardware*, tratando assim o nível de SO como o limite entre os programas aplicativos e o *hardware* do computador.

No nível 4 se encontra o nível da linguagem de montagem¹². Nesse nível há uma relação 1:1 entre um comando e uma das instruções do conjunto de instruções do nível

⁹ Conjunto de instruções - Todas as possíveis instruções que podem ser interpretadas e executadas em um processador.

¹⁰ Nesse trabalho os termos Processador, Processador Central e CPU (Central Processing Unit) serão usados como sinônimos.

¹¹ Interpretação - processo pelo qual um programa escrito em linguagem de alto nível (o programa fonte) tem cada um de seus comandos lido, interpretado, convertido em código executável e imediatamente executado, antes que o comando seguinte seja tratado.

¹² Linguagem de montagem - também chamada de linguagem Assembly, trata-se de uma linguagem simbólica semelhante a linguagem de máquina.

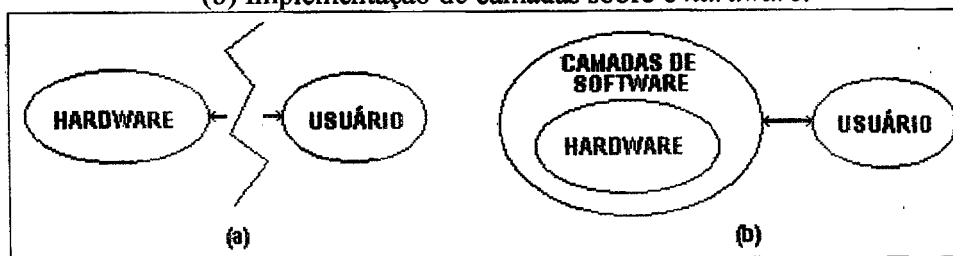
de microprogramação, sendo a comunicação entre o nível imediatamente inferior a este, realizada por um montador¹³. Já no nível 5 estão as linguagens projetadas para serem utilizadas pelos programadores de aplicação com o uso de um compilador¹⁴ ou interpretador. Com esse tipo de *software* de tradução, cada comando disponível na linguagem de programação, ao ser traduzido, será convertido em centenas ou milhares de instruções ao nível de microprogramação.

A seção seguinte trata especificamente do nível 3, o nível do Sistema Operacional.

3.2. OBJETIVOS DO SISTEMA OPERACIONAL

O objetivo fundamental dos sistemas computacionais é executar os programas dos usuários e tornar mais fácil a solução de seus problemas. O *hardware* de um computador é construído com esse objetivo. Mas, como usar somente o *hardware* não é uma tarefa fácil, ao longo do tempo foram desenvolvidas camadas de *software* que envolvem o *hardware* com o intuito de distanciar os usuários¹⁵ dos detalhes físicos de implementação do *hardware*. A Fig. 3.3(a) ilustra essa idéia. Desta forma, em um sistema computacional é através das várias camadas de *software* que os recursos disponíveis no *hardware* podem ser utilizados pelos usuários para processar e acessar informações (Fig. 3.3(b)).

Figura 3.3 (a) Dificuldades de comunicação entre *hardware* e usuários.
(b) Implementação de camadas sobre o *hardware*.



¹³ Montador – programa de conversão (ou tradução) de um programa em linguagem de montagem para seu equivalente executável em linguagem binária.

¹⁴ Compilação – trata-se do processo de análise de um programa escrito em linguagem de alto nível (o programa fonte), para gerar um programa equivalente em linguagem binária (o programa objeto).

¹⁵ Usuários – nesse trabalho entendemos o termo como quaisquer elementos que requisitem algum tipo de serviço do sistema computacional, sejam eles pessoas, equipamentos ou outros computadores.

Os programas pertencentes à chamada **camada de *software*** podem ser de vários tipos (ou categorias), podendo os mesmos serem divididos em dois grandes grupos:

- programas de sistema e
- programas aplicativos.

Os **programas de sistema** gerenciam a operação do computador em si, enquanto que os **programas aplicativos** executam aquilo que o usuário realmente deseja. Desta forma, os programas de sistema servem de apoio aos programas aplicativos (Fig. 1.1).

Um exemplo de programa de sistema é o Sistema Operacional. PRESSMAN (1995, p.20) define que os SO e outros programas de sistema pertencem a uma área de desenvolvimento de *software* denominada Software Básico, isto é, programas escritos para dar apoio a outros programas. Alguns tipos de programas de sistema processam informação complexa mas determinada como por exemplo, os compiladores, editores de texto e utilitários de gerenciamento de arquivos. Outros tipos de programas de sistema processam dados amplamente indeterminados, como é o caso dos SO, dos drivers, dos processadores de telecomunicações, etc. Nos dois casos, a área de Software Básico é caracterizada pela forte interação com o *hardware* de computador, uso intenso de múltiplos usuários, operações concorrentes que exigem escalonamento, compartilhamento de recursos, sofisticada administração do processo, estruturas de dados complexas e múltiplas interfaces externas.

Assim, a realização de operações comuns a vários tipos de aplicações como operações de controle e alocação de recursos, que se caracterizam por serem complexas e repetitivas (como as que controlam dispositivos de Entrada e Saída, por exemplo), são agrupadas num sistema bastante complexo chamado SO.

TANENBAUM & WOODHULL (2000, p.19) definem que o SO pode ser visto sobre dois enfoques. O primeiro enfoque é *top-down*, e diz que o SO é o programa que esconde do usuário a verdade sobre o *hardware*, fazendo com que a abstração oferecida seja mais simples e mais fácil de utilizar que o *hardware* subjacente. Desta forma, o SO apresenta ao usuário o equivalente a uma máquina virtual. O segundo enfoque, o enfoque *botton-up*, diz que o SO gerencia todos os recursos¹⁶ de um sistema computacional comumente complexo. Sua tarefa primária seria monitorar a utilização

¹⁶ Recurso – por tratar-se de termo amplamente utilizado no estudo de SO, seus possíveis enfoques serão tratados na seção 3.4.

de recursos computacionais quanto à requisição e alocação dos mesmos, além de medir a utilização de recursos e mediar as requisições conflitantes de diferentes programas e usuários. Realmente, o SO pode ser visto dessas duas formas, pois as mesmas complementam-se.

Geralmente, o estudo de projeto de SO não se concentra em nenhum processador ou SO específicos, mas concentra-se na discussão de conceitos fundamentais aplicáveis a vários sistemas, na busca de estratégias que visem melhor atender às solicitações realizadas pelos usuários.

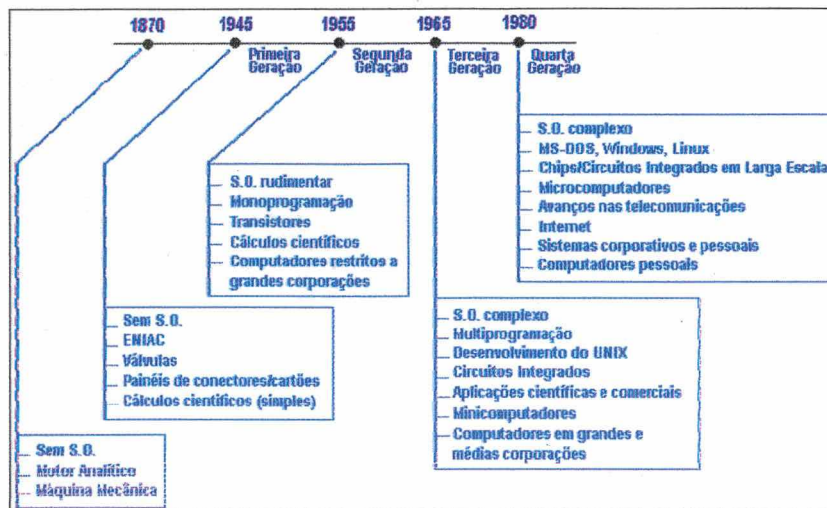
3.2.1. Histórico Evolutivo dos Sistemas Operacionais

É indiscutível o importante papel que os sistemas computacionais têm no nosso cotidiano. De sistemas utilizados somente por técnicos especialistas e com capacidade de processamento e armazenamento mínima, os sistemas evoluíram tornando-se cada vez mais acessíveis quanto à apresentação de interfaces amigáveis, relativamente mais baratos, confiáveis, compactos e com capacidade de armazenamento cada vez mais robustas.

A atual realidade é resultado de fases evolutivas (ou gerações) nas quais os sistemas computacionais foram aprimorados. Entendendo o sistema computacional como um conjunto de esforços das áreas de *hardware* e *software*, podemos afirmar que esta evolução pode ser creditada a inovações tecnológicas em ambas as áreas. Analisando a evolução da área de *hardware* podemos também verificar a evolução do *software*. Esse breve histórico representado na Fig. 3.4, contextualiza, em particular, a evolução dos SO¹⁷.

¹⁷ O histórico da evolução da computação e dos SO está baseada nos textos dos seguintes autores: MACHADO & MAIA (1992), SHAY (1996), TANENBAUM (1995), TANENBAUM & WOODHULL (2000).

Figura 3.4 – Histórico Evolutivo dos SO.



Uma das primeiras iniciativas registradas na literatura foi do visionário Charles Babbage que resultou na produção do primeiro computador considerado digital (aproximadamente 1870). Apesar de sua máquina não possuir SO, Babbage sabia que necessitaria de algum tipo de *software* para gerenciar seu “motor analítico”.

Somente durante a II Guerra Mundial (em 1945) foram construídos os primeiros computadores eletrônicos. Eles ocupavam salas inteiras e utilizavam milhares de válvulas. Com essa tecnologia, a performance das “máquinas de cálculo” era inferior a presente na mais simples das calculadoras atuais. Programava-se através de painéis de conectores, em linguagem de máquina pura. Ainda não existiam SO. O processamento envolvia somente cálculos simples. Esses fatos aconteceram na chamada primeira geração (1945-1955). Ainda nessa geração foram introduzidos os cartões perfurados, mas a rotina de trabalho ainda não distinguia as funções das pessoas no ambiente computacional, isto é, o mesmo grupo de pessoas projetava, construía, programava, operava e mantinha o computador.

A segunda geração (1955-1965) é marcada pela introdução do transistor e pela utilização do primeiro programa similar aos atuais SO. Apesar de acessíveis apenas a grandes corporações (pelo alto preço), a tecnologia do transistor já tornava os computadores confiáveis o suficiente para serem comercializados. Eram utilizados principalmente para cálculos científicos. Linguagens como FORTRAN¹⁸ eram

¹⁸ FORTRAN – FORMula TRANslation - Linguagem de programação de alto nível da década de 50 utilizada principalmente para aplicações com cálculos numéricos.

utilizadas para programar, transformando os comandos em um conjunto de cartões perfurados que eram submetidos à máquina individualmente ou em lote. Esses cartões eram então processados sobre o controle de um programa especial, um SO rudimentar e com funções bastante limitadas que gerenciava o atendimento seqüencial dos *jobs*¹⁹. Nesse tipo de ambiente o processamento era exclusivamente de monoprogramação. Também o papel de cada uma das pessoas envolvidas no ambiente computacional já se tornavam mais definido: havia projetistas, construtores, operadores de computador, programadores e mantenedores.

A terceira geração de evolução dos computadores (1965-1980) é marcada pelo desenvolvimento dos circuitos integrados e o desenvolvimento de SO complexos que possibilitavam executar programas em multiprogramação. Essa técnica iria tornar-se imprescindível para o sucesso dos sistemas computacionais. Nessa época, além das aplicações científicas, surgiram as primeiras aplicações computacionais comerciais. Um marco no histórico dos SO aconteceu nessa geração, quando Ken Thompson desenvolveu o cerne do que mais tarde se tornaria o atual SO UNIX. Também nessa geração foram desenvolvidos os primeiros minicomputadores a um preço mais acessível (no contexto da época) para as corporações de porte médio.

A partir de 1980 entrou-se na quarta geração de computadores, caracterizada pelo desenvolvimento de circuitos LSI (integração em larga escala) que proporcionaram a produção de chips contendo milhares de transistores numa pequena placa de silício, que por sua vez desencadeou a microinformática com computadores pessoais acessíveis ao grande público.

As facilidades de conexão em rede, disponibilidade de recursos gráficos atrativos e *software* com interface amigável para usuários finais foram pontos cruciais para o advento da microinformática, que iniciou nessa geração e continua até hoje. A evolução dos SO teve grande parcela de participação nesse contexto: inicialmente era o MS-DOS (compatível com processadores Intel, e mais tarde com processadores Pentium), depois a grande trajetória do Windows desde suas primeiras versões, que nada mais eram que verdadeiros *shells*²⁰ rodando sobre o MS-DOS, até as mais atuais, totalmente reescritas.

¹⁹ Job – tarefa a ser realizada no sistema computacional. É dividido em passos (ou subjobs), que podem ser comparados aos processos-filhos.

²⁰ Shells – interface que distancia níveis de processamento.

Em 1991, outro marco histórico da evolução dos SO aconteceu, quando o finlandês Linus Torvalds iniciou o desenvolvimento do LINUX, um SO baseado em UNIX para microcomputadores. A grande inovação do LINUX é sem dúvida a proposta embutida no sistema criado: o *software* livre.

Software livre se refere à liberdade dos usuários executarem, copiarem, distribuírem, estudarem, modificarem e aperfeiçoarem o software (O QUE É LINUX?, 2000). Essa mesma fonte afirma que a lista de colaboradores do kernel²¹ LINUX inclui mais de 250 pessoas do mundo inteiro e estima-se que o número de usuários esteja acima de 10 milhões.

Esse histórico evolutivo da tecnologia de *hardware* e *software* não ocorreu (e nem ocorre) ao mesmo tempo em todas as partes do mundo por diferentes contextos sociais, culturais, tecnológicos e econômicos. Essa revisão da evolução da informática mostra que, aliada à fantástica evolução da arquitetura do *hardware* dos últimos 45 anos, está a evolução do *software* de forma geral, e podemos afirmar que os avanços do SO tiveram significativa participação na atual situação.

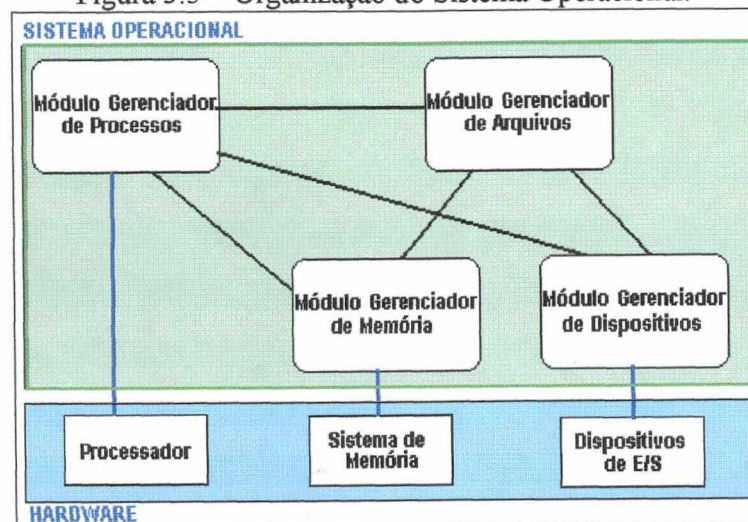
3.3. MÓDULOS GERENCIADORES DO SISTEMA OPERACIONAL

Pode-se afirmar que o SO objetiva proporcionar um ambiente favorável a execução de programas. Mas um sistema grande e complexo como um SO só poderia ser desenvolvido se subdividido em partes menores e mais simples. Assim, a divisão do SO acontece em módulos que focam cada um dos principais recursos (ou conjunto de recursos com características afins) de um sistema computacional, a saber: processador, sistema de memória, sistema de E/S e sistema de arquivos. Para compartilhar cada um desses recursos são implementadas estratégias no SO e em outros tipos de *software* de apoio para acesso, alocação, e liberação dos mesmos por parte dos usuários. Desta forma, os SO são organizados em gerências ou módulos gerenciadores (Fig. 3.5).

Neste trabalho os objetivos e principais características de cada um desses módulos são resumidos e serão detalhados apenas os aspectos relacionados a Gerência de Processos, por esse ser o foco do desenvolvimento do protótipo proposto no trabalho.

²¹ Kernel – núcleo do sistema operacional, onde estão as rotinas básicas.

Figura 3.5 – Organização do Sistema Operacional.



Fonte: Adaptado de NUTT (1997, p.56).

3.3.1. Módulo Gerenciador de Memória

A memória é um recurso importante que deve ser gerenciado de forma cuidadosa, pois a execução das instruções dos processos pelo processador, depende do prévio armazenamento de tais instruções na memória principal. Mas, a memória principal é apenas uma das memórias que compõe a chamada **hierarquia de memória**, isto é, um conjunto de memórias com diferentes capacidade de armazenamento, tecnologia, custo de armazenamento, tempos de acesso aos dados lá armazenados, etc.

Basicamente, há quatro tipos diferentes de memória²²:

- memória secundária,
- memória principal,
- memória cachê,
- registradores.

O módulo Gerenciador de Memória do SO enfatiza o gerenciamento da memória principal, controlando que partes da memória estão em uso e que partes não estão, alocando memória para os processos quando eles necessitarem e desalocando quando eles terminarem, e também gerenciando a troca entre a memória principal e o disco quando o SO implementa técnica de memória virtual e *swapping*.

²² Informações detalhadas sobre hierarquia de memória e as características de cada um dos tipos de memória podem ser obtidas em MONTEIRO (1995).

Se para o usuário o sistema de memória existe para que seus programas ou seus dados tenham lugar a serem armazenados, para o projetista de um SO, o sistema de memória existe para proporcionar a muitos processos tenham um lugar a serem alocados (SHAY,1996). Como se trata de um recurso importante, um dos papéis do SO é gerenciá-lo.

Segundo SILBERSCHATZ & GALVIN (2000) e ALBUQUERQUE (1990), são implementadas nos SO as seguintes atividades relativas ao gerenciamento de memória:

- Manter informações sobre quais partes da memória estão sendo usadas no instante atual e por qual processo²³.
- Decidir quais processos (ou parte deles) devem ser carregados na memória quando algum espaço de memória se torna disponível.
- Transferir processos (ou parte deles) entre a memória principal e memória secundária de modo a liberar espaços para alocar outros processos ativos no sistema.

3.3.2. Módulo Gerenciador de Arquivos

Todas as aplicações computacionais, quaisquer que sejam elas, precisam armazenar e recuperar informação. Tais informações são mantidas em dispositivos de armazenamento externos com características e organização física próprias (como disco magnéticos, fitas magnéticas, unidades de CD, etc.), em unidades denominadas arquivos. Os processos podem então ler informações de arquivos existentes e/ou criar novos, escrevendo informações em tais arquivos. É no projeto do SO que serão definidas as formas de estruturá-los, identificá-los, acessá-los, utilizá-los, protegê-los e implementá-los.

Também chamado de Gerência de Armazenamento Auxiliar, Gerência de Informações ou ainda Sistema de Arquivos, este módulo do SO é o mais visível de todos, pois armazena os arquivos permitindo sua recuperação, seja qual for o meio físico em questão, através de comunicação com os drivers de dispositivos que traduzem as solicitações de entrada e saída em comandos para dispositivos específicos de *hardware*.

²³ Processo – nesse contexto, o termo trata de uma abstração do conceito de um programa em execução. O termo será tratado na seção 3.5.

Desta maneira, poderíamos definir como premissas ao projeto do SO, sobre a manipulação de arquivos:

- Criar e remover arquivos;
- Definir princípios de organização desses arquivos (como por exemplo, diretórios);
- Possibilitar armazenamento e recuperação de grande quantidade de informação;
- Considerar que a informação pode ser compartilhada entre vários processos, observando princípios de segurança.
- Considerar que as informações dos arquivos devem ser persistentes (não-voláteis), isto é, sua permanência não pode estar necessariamente condicionada a criação de novos processos ou término deles.

3.3.3. Módulo Gerenciador de Sistemas de E/S

Implementar meios através dos quais os processos realizem suas operações de E/S é uma das tarefas mais complexas e difíceis atribuídas ao SO. MONTEIRO (1995, p.279) faz referência a algumas considerações que justificam essa complexidade. As diferentes características de cada dispositivo de E/S torna extremamente complicada a comunicação direta entre a CPU e um desses dispositivos, diferenças essas que vão desde questões de forma de transmissão de dados (bit a bit, byte a byte, etc.) até diferenças relativas à parte elétrica de geração e interpretação dos sinais de transmissão. Também questões de diferenças de velocidade de transmissão de sinais estão envolvidas nessa tarefa devido a disparidade entre a velocidade da CPU e dos dispositivos de E/S.

Apesar de todas essas implicações, a comunicação entre esses dois sistemas (CPU-Memória e dispositivos de E/S) acontece o tempo todo durante a execução de processos. Assim, segundo SHAY (1996), são definidos níveis de processamento de E/S com funções específicas que possibilitem que as solicitações dos processos para leitura ou gravação em um dispositivo de E/S possam ser realizadas intermediadas pelo SO e apoiadas por *software* (como o driver) e também por *hardware* específico (como as controladoras de dispositivo) destinados a isso.

É papel do SO tratar essas questões e possibilitar a execução dos processos, tanto para atendimento de instruções diretamente executadas na CPU, quanto para o atendimento de solicitações de operação de E/S.

3.3.4. Módulo Gerenciador de Processos

No módulo Gerenciador de Processos o processador é o recurso a ser gerenciado no atendimento desses processos. SILBERCHATZ & GALVIN (2000) definem como responsabilidades desse módulo:

- Definir o modelo de implementação e controle de processos.
- Criar e remover processos.
- Definir e implementar estratégias para selecionar, executar e suspender processos (critérios de utilização do processador).
- Implementar estratégias de sincronização e comunicação entre processos.
- Implementar (ou não) tratamento de possíveis situações de *deadlock*²⁴.

Nesse trabalho serão detalhadas questões relativas a definição de modelos de processo, estruturas que permitem implementá-los, possibilidade de criação e remoção de processos no sistema, ativação, suspensão e reativação de processos e estratégias para tal, e também estratégias de detecção de *deadlock*.

3.4. RECURSO

O termo recurso tem um significado relativamente amplo, e pode ser utilizado com conotações diferentes no estudo de SO. Particularmente neste trabalho, entende-se recurso com dois significados, dependendo do contexto tratado:

- De uma forma geral, nomeando os elementos a serem compartilhados pelos processos de forma genérica,
- Especificamente no estudo de *deadlock*, como elementos que possam ocasionar o bloqueio de processos.

²⁴ *Deadlock* – situação eventual e indesejável em que processos ficam impedidos de prosseguir sua execução. Suas causas e implicações serão tratadas na seção 3.7.

3.4.1. Recurso como termo de uso geral no estudo de SO

De uma forma mais ampla, o termo recurso é utilizado para designar elementos necessários para o atendimento aos processos, entendido como qualquer componente físico ou lógico, a ser compartilhado no ambiente computacional. Assim, é papel do SO assegurar que esses recursos sejam usados de forma eficiente, provendo serviços adequados aos usuários. Com esse enfoque, CPU, área de memória, dispositivos de E/S, variáveis, direito de entrar em região crítica, mensagens, sinais de comunicação, entre outros, são entendidos como recursos computacionais.

Processos solicitam a utilização de recursos computacionais durante todo seu ciclo de vida. Mas, para que um recurso seja utilizado por um processo, há uma seqüência a ser realizada (SILBERSCHATZ & GALVIN, 2000, p.232):

- Requisição – o recurso pode ou não estar disponível para uso do processo. O recurso fica marcado como requisitado.
- Alocação e Uso – o recurso é alocado ao processo, para ser utilizado por ele.
- Liberação – o processo libera o recurso, fazendo com que o mesmo volte à situação de liberado.

Assim, uma tabela de recursos é mantida pelo SO, registrando a situação de cada um deles. Na solicitação de um recurso que estivesse na situação de alocado, seriam formadas filas de solicitação de serviços.

3.4.2. Recurso como termo usado no estudo de *deadlock*

Já no contexto do estudo de *deadlock*, o termo recurso tem um significado mais restrito. São os elementos ou entidades que ocasionam as situações de bloqueio dos processos. Como define TOSCANI (1987, p.84)²⁵:

No contexto de deadlock, recurso é qualquer entidade que puder ocasionar o bloqueio de processos (um bloqueio acontecerá quando um recurso não estiver disponível e for requisitado por um processo). Os recursos, portanto, são as entidades que os processos estão à espera enquanto estão bloqueados.

²⁵ TOSCANI, Simão Sirineo. *Introdução aos Sistemas Operacionais*. UFRS, Curso de Pós-Graduação em Ciências da Computação. Porto Alegre, 1987. [material apostilado, não publicado].

Assim, CPU não é considerada recurso pois não satisfaz a definição acima, já que um processo que está à espera de CPU está no estado de pronto e não no estado de bloqueado²⁶.

Recursos podem ser definidos sobre dois aspectos (Tabela 3.1):

- Quanto à posse por parte do processo e,
- Quanto a sua finitude no sistema.

Tabela 3.1 – Classificação dos Recursos no contexto do estudo de *Deadlock*.

Quanto a posse por parte do processo.	Preemptivo - aquele que pode ser retirado do processo que está de sua posse, sem maior dano ao processo.
	Não-preemptivo - quando o mesmo não pode ser retirado do processo, e caso seja retirado causará dano ao mesmo.
Quanto a finitude no sistema.	Reutilizáveis serialmente - são aqueles disponíveis no sistema em quantidade fixa, sendo que cada um deles pode ser alocado por apenas um processo por vez.
	Consumíveis - podem ser criados ou eliminados (consumidos), e, depois que o processo os utiliza, eles não podem ser reutilizados (foram consumidos).

São exemplos de recursos preemptivos: área de disco para swapping e tempo de processador; e é exemplo de recurso não-preemptivo o uso da impressora. Também são exemplos de recursos reutilizáveis serialmente: espaço em memória, área de armazenamento em disco, arquivos utilizados para operação de gravação, utilização de dispositivos de E/S, etc. São exemplos de recursos consumíveis: dados, variáveis, passagem de parâmetros, etc.

Apesar da situação de *deadlock* poder ocorrer quando processos acessam quaisquer recursos de forma compartilhada, este estudo está restrito apenas a questões de *deadlock* que envolvem recursos não-preemptivos e reutilizáveis serialmente.

3.5. MODELO DE PROCESSO

No estudo de SO, o conceito-chave a ser compreendido é o de processo. Nesta seção serão tratados os conceitos e terminologias que definem um modelo de processo no contexto de SO, partindo da definição do termo e tratando de questões envolvidas em sua implementação no ambiente computacional.

²⁶ Os estados dos processos serão tratados na seção 3.5.4.

3.5.1. Definição de Processo

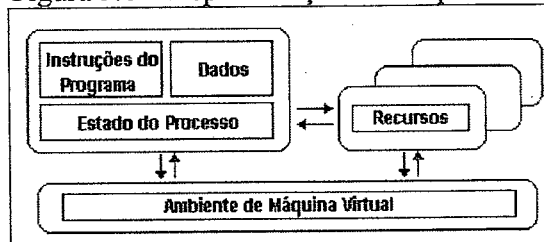
O conceito de processo é fundamental no estudo de SO. Basicamente, um processo é um elemento abstrato composto de vários componentes a ser executado num sistema computacional. Tal sistema consiste de vários processos concorrentes em execução, alguns dos quais são processos do SO (o código desses processos faz parte do código do SO) e os demais são processos de usuários (para os quais o código faz parte de um programa de usuário).

TANENBAUM (1995, p.10) conceitua processo como:

Um programa em execução, sendo constituído de código executável, dos dados referentes ao código, da pilha em execução, do valor do contador de programa, do valor do apontador da pilha, dos valores dos demais registradores do hardware, além de um conjunto de outras informações necessárias à execução do programa.

Assim, processo é um conceito abstrato de uma entidade, que denota principalmente dois aspectos: um código a ser executado e a situação dos elementos de *hardware* de conteúdo variável, necessários à execução desse código. O código mencionado nada mais é que uma determinada quantidade de instruções de máquina, isto é, no formato apropriado para serem executadas pelo processador. Também é importante perceber que um processo é algo dinâmico. Para que um processo seja assim considerado, seu código (ou parte dele) e os dados que esse código manipula precisam estar alocados em memória principal (Fig. 3.6).

Figura 3.6 – Representação de um processo.



Fonte: NUTT (1997, p.22)

Comumente o conceito de processo é confundido com o de programa. Um programa não é um processo. Programa nada mais é que o código, isto é, um conjunto de instruções armazenadas. Essas instruções podem até estar em linguagem de máquina, mas são somente instruções, isto é, um arquivo codificado numa determinada linguagem, mantido num dispositivo de armazenamento (disco, por exemplo). O

conceito de programa não envolve o contexto necessário para sugerir execução. Desta forma, um programa é uma entidade passiva, enquanto um processo é uma entidade ativa. Aí reside a diferença entre programa e processo. A execução de um processo progride de forma seqüencial, isto é, as instruções que compõem o programa, são executadas uma após a outra. Como SHAY (1996) afirma, para o SO não há usuários ou programas, existem somente processos que devem ser executados e esses mesmos processos competem por recursos físicos ou lógicos, que por sua vez só existem em função da existência dos processos²⁷.

A cada momento diferentes processos estão concorrendo pelo uso dos recursos do sistema computacional e cabe ao SO gerenciá-los, dependendo da forma como o SO implementa o atendimento aos processos. Essas questões serão tratadas na seção seguinte.

3.5.2. Questões sobre Paralelismo

Partindo-se da premissa que um processador executa uma e somente uma instrução de cada vez, pode-se observar que essa afirmativa está baseada nas limitações físicas da máquina. Apesar de haver estratégias que permitam que mais de uma instrução (ou parte dela) esteja sendo atendida pelo processador num mesmo espaço de tempo²⁸, a estrutura básica do ciclo de instrução²⁹ limita que num determinado momento, apenas uma única instrução possa ser executada (considerando um único processador). Como uma determinada seqüência de instruções de máquina em execução faz parte de um processo, pode-se concluir que o processador atende a um processo por vez.

Porém, pode-se afirmar que há meios de implementar paralelismo, seja físico ou lógico, na execução de processos, através de recursos de *software* e/ou de *hardware*. (TANENBAUM, 1999).

²⁷ Nesse estudo o termo processo será utilizado de forma genérica como sendo o elemento a ser selecionado pelas estratégias de escalonamento de CPU implementadas pelo SO.

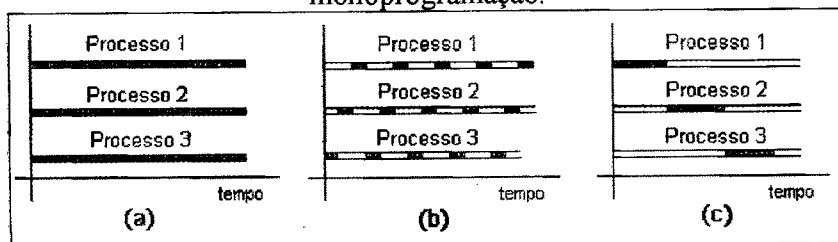
²⁸ Técnicas tratadas nas chamadas categorias de Flynn: SIMD (Fluxo simples de instruções e fluxo múltiplo de dados) e MIMD (Fluxo de instruções múltiplo e de dados múltiplo). São exemplos: Multiprocessadores, Pipeling, Múltiplas UAL Especializadas, Máquinas vetoriais, etc. (TANENBAUM, 1999) e (MACHADO & MAIA, 1992).

²⁹ MONTEIRO (1995, p.386) define ciclo de instrução como o conjunto de etapas previamente determinadas que permite o acesso à memória principal para busca de uma instrução, sua interpretação pela CPU e a conseqüente execução.

Considera-se que os processos são executados fisicamente em paralelo quando cada processo é executado por um processador diferente. Desta forma, o paralelismo físico subentende a existência de multiprocessamento, isto é, mais de um processador atendendo processos simultaneamente. Por exemplo, na Fig. 3.7(a) há três processos sendo simultaneamente executados ao longo do tempo por três processadores distintos.

Já numa execução logicamente em paralelo, os processos são executados em um único processador, e o paralelismo é obtido através de estratégias de compartilhamento de tempo. Esse compartilhamento pode acontecer em duas modalidades: a multiprogramação³⁰, mostrada na Fig. 3.7(b) e a monoprogramação como na Fig. 3.7(c).

Figura 3.7 - (a) Paralelismo físico ou multiprocessamento. (b) Paralelismo lógico implementando multiprogramação. (c) Paralelismo lógico implementando monoprogramação.



Fonte: Adaptada de MACHADO & MAIA (1992, p.40).

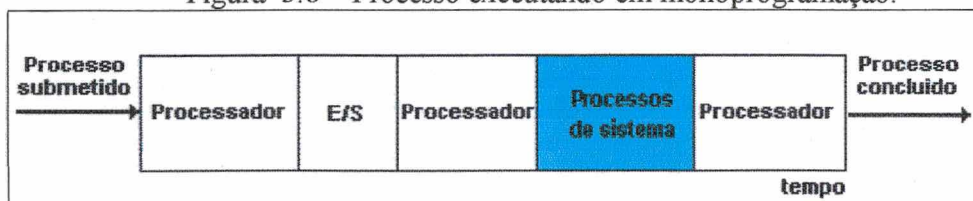
Na monoprogramação (ou monotarefa) todas as instruções de um processo são executadas uma após a outra, sem interrupções de outros processos de usuário. Assim, um processo de usuário monopoliza o uso do processador, não sendo permitido que um segundo processo utilize-o antes que o primeiro seja concluído. Desta forma, nem mesmo nos momentos em que o processo corrente estiver utilizando algum dispositivo de E/S e o processador fica ocioso, outro processo não poderá aproveitar o tempo e utilizar o processador.

Na Fig. 3.8, considera-se uma linha de tempo, na qual um processo é submetido, recebe processador, e a seguir passa a se atendido por um dispositivo de E/S. No espaço de tempo, denominado como E/S, o processador ficará ocioso até que a requisição de E/S do processo seja totalmente concluída. O processo então recebe novamente o processador, até que é forçosamente interrompido para que um dos processos de sistema (gerados pelo SO) seja executado. Esse espaço de tempo é denominado de processos de

³⁰ Multiprogramação, multitarefa e multiprocessamento serão entendidos como sinônimos nesse estudo; da mesma forma os termos monoprogramação, monotarefa e monoprocessamento.

sistema e caracteriza espaços de tempo de uso do processador destinados ao próprio SO, onde o mesmo executa suas próprias rotinas.

Figura 3.8 – Processo executando em monoprogramação.



Vale ressaltar que o conceito de monoprogramação é válido apenas se considerarmos a concorrência entre processos de usuário e, mesmo assim, ainda é bastante questionável. Deve ser notado que mesmo num ambiente em que seja submetido a um único processo, nada garante que esse processo não será desmembrado em vários processos-filhos, que concorrerão entre si (sendo consideradas questões sobre comunicação e sincronização entre processos), o que caracteriza o ambiente de multiprogramação.

Outro ponto de questão diz respeito aos processos de sistema gerados pelo próprio SO. Mesmo com um único processo de usuário, o SO deve compartilhar o tempo de processador (e todos os outros recursos) com o(s) processo(s) de SO (mesmo considerando que os processos gerados pelo SO têm atendimento priorizado frente aos processos de usuário).

Nos SO que implementam multiprogramação (ou multitarefa) as limitações do processador continuam sendo as mesmas (apenas uma instrução é executada de cada vez), mas o paralelismo lógico é conseguido através do compartilhamento do uso do único processador (considerando um ambiente de monoprocesso) entre vários processos, através de implementações do SO.

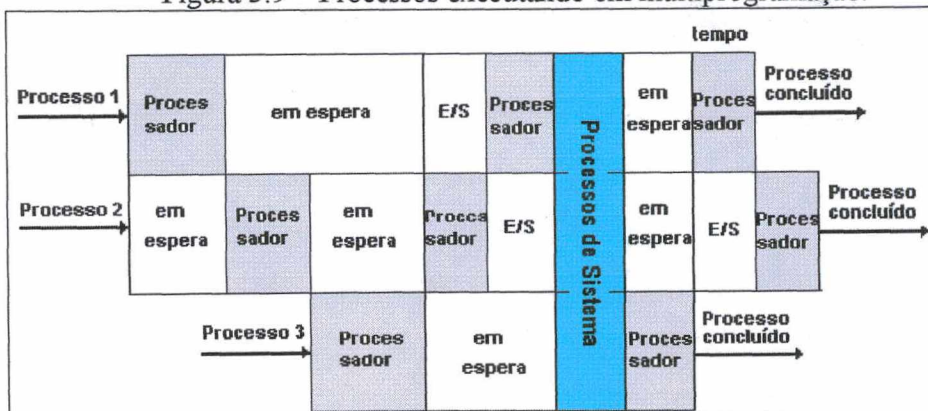
Nessa modalidade de paralelismo lógico, a cada espaço de tempo, um dos processos ativos utiliza o processador e, mesmo antes do primeiro ter sido concluído, o processador passa a atender a um segundo processo, e assim sucessivamente. O processo interrompido deve reiniciar exatamente do ponto onde parou. Para isso o SO mantém estruturas de dados³¹ que suportam as chamadas mudanças de contexto.

³¹ Estruturas de dados como as Tabelas de recursos, os Blocos de Controle de Processos e as Listas de Controle de Processos.

Vale ressaltar que tais estruturas também devem estar presentes nos SO chamados de monoprogamáveis, devido à mudança de contexto entre processos de usuário e de SO.

Na Fig. 3.9 está representada uma linha de tempo, onde três processos de usuário serão atendidos, identificados como 1, 2 e 3. Dois deles são submetidos simultaneamente, enquanto o terceiro só é submetido algum tempo depois. Quando os processos 1 e 2 são submetidos, o processador é destinado ao processo 1, de acordo com algum critério que o privilegiou, o que faz com que o processo 2 fique aguardando sua vez de executar. Os quadros de cor cinza representam os momentos em que o processador executa as instruções dos processos de usuário, enquanto que o quadro azul representa o momento em que o processador está executando as rotinas do próprio SO (processos do sistema). Nota-se portanto que nesse exemplo, o processador está 100% utilizado. Os quadros denominados em espera caracterizam os momentos em que, enquanto um dos processos está executando, os outros aguardam serem selecionados para receber o processador.

Figura 3.9 – Processos executando em multiprogramação.



Através do uso de multiprogramação é possível reduzir os períodos de inatividade do processador, aumentando a eficiência do sistema como um todo. Vale ressaltar que a complexidade de um SO que suporte multiprogramação é muito maior que nos SO monoprogamáveis.

3.5.3. Implementação de Processo

Quer seja num ambiente de mono ou de multiprogramação, TANENBAUM & WOODHULL (2000) afirmam que para implementar o modelo de processo, o SO mantém uma tabela de processos onde haverá uma entrada para cada um dos processos ativos no sistema. Assim, cada uma das entradas na tabela de processos, também chamada de Bloco de Controle de Processo³² (BCP), descreve um processo quer seja de usuário ou de SO (Fig. 3.10).

Figura 3.10 – Bloco de Controle de Processo.

IDENTIFICADOR DO PROCESSO	
TIPO DO PROCESSO	ESTADO DO PROCESSO
REGISTRADORES	
LIMITES DE MEMÓRIA	
LISTA DE ARQUIVOS ABERTOS	
⋮	
PONTEIRO PARA O PRÓXIMO BCP NA LCP	

Fonte: Adaptado de ALBUQUERQUE (1990) e MACHADO & MAIA (1992).

As informações contidas no BCP variam de acordo com a implementação do SO. São basicamente informações sobre estado do processo, contador de programa (indicação da instrução corrente), ponteiro da pilha, conteúdo e identificação dos registradores do processo, alocação de memória, situação de seus arquivos abertos, informações sobre escalonamento e tudo mais que seja necessário para que o processo possa ser interrompido em sua execução e retornar exatamente do ponto onde parou.

O BCP é uma das estruturas de dados mais importantes para o SO, mantendo as informações que possibilitam o gerenciamento dos processos, pois são as informações mantidas no BCP que possibilitam que haja mudança de contexto.

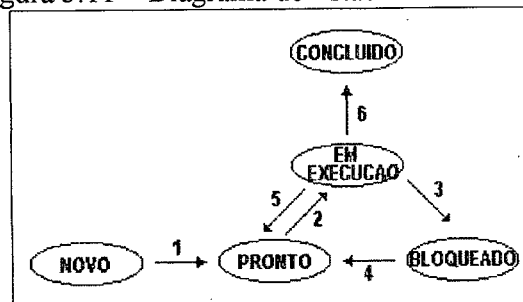
³² TOSCANI (1987) enumera sinônimos utilizados para o BCP: registro descritor, bloco descritor, bloco de contexto, registro de estado ou vetor de estado.

3.5.4. Estados do processo e Transições válidas entre estados

Sabe-se que durante o ciclo de vida dos processos, suas necessidades de atendimento alternam-se entre espaços de tempo de uso de CPU e de uso de dispositivos de E/S. Essas necessidades dos processos são chamadas de fases de uso. Assim, durante o atendimento a suas fases, os processos passam por diferentes estados (ou situações) previstas na implementação do SO, sendo elas: esperando pelo uso da CPU, utilizando CPU, aguardando por alguma operação de E/S solicitada, etc. Esses estados são representados graficamente no Diagrama de Estados de Processo descrito na Fig. 3.11. Nesse diagrama as formas circulares representam os estados previstos e implementados nos SO e as setas direcionadas, as transições ou mudanças válidas entre estados.

Autores como NUTT (1997), SHAY(1996), ALBUQUERQUE (1990), TANENBAUM & WOODHULL (2000) entre outros, apresentam o diagrama de estados com pequenas variações de nomenclatura e detalhamento. Alguns SO têm estados e transições mais refinadas e com mais detalhamento, podendo variar inclusive sua denominação de um SO para outro, mas os mencionados são encontrados em todos os SO.

Figura 3.11 – Diagrama de Estados de Processo.



A seguir, uma breve explicação de cada um dos estados de processo e das transições válidas:

- **Novo** – o processo está sendo criado (submetido no sistema) e recursos como alocação de área de memória principal estão sendo providenciados pelo SO. Nesse ponto também é criado o registro do seu BCP na tabela de processos.
- **Pronto** – todos os recursos necessários para que o processo possa fazer uso do processador já foram alocados ao processo e ele está pronto para ser executado, mas aguarda que a CPU seja a ele alocada.
- **Em Execução** – o processo é selecionado e está sendo executado pela CPU.

- **Bloqueado** – o processo está sendo atendido por um dispositivo de E/S e aguardando que esse evento termine.
- **Concluído** – o processo foi concluído e os recursos alocados a ele são novamente disponibilizados ao sistema. Seu BCP é eliminado da tabela de processos ativos.

As transições válidas entre estados são:

- **Transição 1 - de Novo para Pronto** – o processo é admitido no sistema e passa a competir por CPU.
- **Transição 2 – de Pronto para Em Execução** – o processo é selecionado entre os processos no estado de Pronto e a ele é alocada a CPU.
- **Transição 3 – de Em Execução para Bloqueado** – o processo necessita que seja realizado um evento que depende de um recurso de E/S.
- **Transição 4 – de Bloqueado para Pronto** – é concluído o atendimento em que o processo dependia de um dispositivo de E/S e o processo volta ao estado de Pronto, para novamente competir por CPU.
- **Transição 5 – de Em Execução para Pronto** – A CPU é forçosamente desalocada do processo corrente para que outro processo seja selecionado para usá-la. Essa transição ocorre somente quando a estratégia de seleção de processos (escalonamento) é a chamada preemptiva³³.
- **Transição 6 – de Em Execução para Concluído** – o processo tem sua última fase de uso de CPU atendida e termina sua execução.

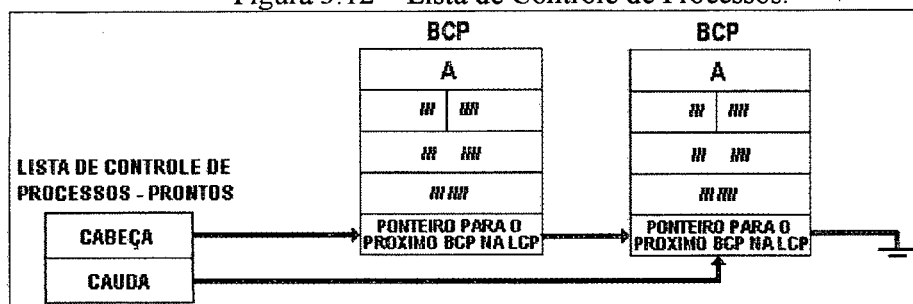
As transições 2, 3 e 5 estão envolvidas no que é chamado de mudança de contexto, isto é, o conjunto de rotinas do SO a serem executadas para que a CPU deixe de atender a um processo e passe a executar outro, sendo um dos exemplos das rotinas de SO (tempos de processo de sistema). O tempo necessário para execução dessas rotinas (e outras de responsabilidade do SO) é relativo e depende de características de *hardware* e estratégias de *software*. Mas, quanto mais tempo o SO utilizar para executar suas próprias rotinas, menos tempo é destinado aos processos de usuário; portanto a duração e frequência desses tempos de processos de sistema devem ser minimizados ao máximo.

³³ Os termos preemptivo e não-preemptivo serão tratados na seção 3.6.3.

3.5.5. Listas de Controle de Processos

A principal estrutura de dados mantida pelo SO que possibilita ocorrer a mudança de contexto é o BCP. Os BCP são organizados em tabelas de processos chamadas de Listas de Controle de Processos (LCP). Segundo ALBUQUERQUE (1990) as LCP são listas ligadas que refletem a situação de cada processo quanto ao seu estado corrente, entre os estados de processo válidos implementados no SO, considerando o dinamismo das mudanças de contexto. Desta forma, há LCP prontos, LCP novos e LCP concluídos. (Fig. 3.12).

Figura 3.12 – Lista de Controle de Processos.



Cada vez que ocorram eventos que resultem na alteração de estados dos processos, o SO analisa os BCP e os atualiza nas LCP, realizando remanejamento(s) para que a situação das LCP reflita a situação corrente. Quando um processo é concluído, seu BCP é eliminado da última LCP a que pertenceu e os recursos que haviam sido a ele alocados, também são liberados.

Mas essas LCP não são os únicos tipos de listas do sistema. Há também as listas de processos em estado de bloqueado, que aguardam por determinados dispositivos de E/S que não estavam disponíveis no momento da solicitação. Considerando que vários processos podem necessitar de um determinado dispositivo de E/S, é razoável imaginar que enquanto um processo é atendido, outros formarão filas de espera. Esses processos também terão seus BCP organizados em LCP para uso de disco, terminal, etc. (SILBERSCHATZ & GALVIN, 2000).

A seção seguinte trata dos níveis de escalonamento e da lógica de funcionamento dos algoritmos de escalonamento.

3.6. ESCALONAMENTO

O objetivo da multiprogramação é maximizar o uso da CPU e, portanto, ter sempre um dos processos ativos em estado de execução. Desta forma, a CPU é freqüentemente transferida entre os processos. O BCP possibilita que os processos alternem seus estados e sempre retornem a executar exatamente do ponto em que foram interrompidos da última vez que estiveram no estado de execução. A organização dos BCP em LCP possibilita controlar os processos por estados.

Considerando que haja mais de um processo ativo, a questão é: entre os processos prontos, isto é, logicamente executáveis, qual deles vai ser selecionado a utilizar a CPU primeiro? A parte do SO composta por rotinas responsáveis por selecionar o processo a ser executado é denominada escalonador. Diferentes níveis de escalonamento são implementados no SO e diferentes algoritmos de escalonamento podem ser usados, favorecendo a diferentes classes de processos.

3.6.1. Níveis de Escalonamento

A alocação de CPU a um processo deve ser precedida de uma série de providências a serem tomadas pelo SO previstas em seus algoritmos, como por exemplo, alocar o processo na memória. Desta forma, em geral, são considerados três níveis de escalonamento destinados a tratar das questões que antecedem e incluem a alocação de CPU propriamente dita: escalonamento de longo, de médio e de curto prazo, chamados por SILBERSCHATZ & GALVIN (2000) como escalonador de processos, intermediário e de CPU, respectivamente.

Como os processos novos não podem ser todos carregados ao mesmo tempo na memória principal por limitações físicas, eles são temporariamente mantidos numa área de memória secundária, e é o escalonador de longo prazo que decide a qual desses processos será alocada área de memória principal, selecionando desta forma, a ordem em que os processos serão aceitos no sistema (a utilização da técnica de memória

virtual³⁴ está envolvida nesse contexto). Já o escalonador de curto prazo determina, entre os processos no estado de pronto, qual será efetivamente selecionado a executar.

Assim, o escalonador de processos pode ser entendido como um macroescalonador, enquanto o escalonador de CPU, como um microescalonador, já que o escalonador de CPU é usado com bastante frequência, considerando as constantes mudanças de contexto em processos com características interativas, por exemplo, enquanto que o escalonador de processos é executado com menos frequência, apenas na seleção e alocação dos processos que transitam entre o estado de novo para pronto. Essa é segundo SILBERSCHATZ & GALVIN (2000), a principal distinção entre esses dois níveis de escalonamento.

Um escalonador intermediário, que realiza a técnica de *swapping*³⁵, também pode ser usado para controlar o nível de multiprogramação do sistema, isto é, a quantidade de processos ativos, decidindo que processos serão mantidos ativos e quais serão momentaneamente removidos da memória, e mais tarde realocados. São as características do SO que determinam a necessidade de dois ou mais níveis de escalonamento.

3.6.2. Objetivos do Escalonador

A implementação da multiprogramação permitiu que o processador fosse compartilhado por diversos processos, e as estratégias de alocação de CPU são a base dos SO multiprogramados. Para isso, é necessário definir critérios para determinar a ordem de escolha dos processos para execução, dentre os que concorrem pela utilização do processador. Esse procedimento de seleção é realizado pelo SO num conjunto de rotinas implementadas em níveis diversos, que a partir de agora será chamado genericamente de Escalonador (o que em TANENBAUM & WOODHULL (2000) é denominado Agendador).

³⁴ Memória Virtual – mecanismo que permite que vários processos estejam ativos sem estarem necessariamente inteiramente carregados na memória principal.

³⁵ *Swapping* – Técnica de gerenciamento de memória onde um processo pode periodicamente ser desalocado de seu espaço de memória principal. Algum tempo depois, o processo pode ser recarregado na memória e sua execução continuar a partir do ponto em que havia sido interrompido. Esse esquema é também chamado de troca de processo, e objetiva reduzir o grau de multiprogramação, isto é, reduzir a quantidade de processos ativos que competem por recursos num determinado momento.

Os principais objetivos do Escalonador são contraditórios:

- Manter o processador ocupado a maior parte do tempo, isto é, combater a ociosidade do processador;
- Balancear o uso do processador entre os vários processos (princípios de justiça e imparcialidade no uso do recurso processador);
- Maximizar o *throughput*³⁶ do sistema;
- Oferecer tempos de resposta razoáveis aos processos de usuários interativos;
- Minimizar o *turnaround*³⁷.

Esses objetivos direcionam as implementações de um escalonador que trate a todos os processos de forma justa, evitando assim que um processo fique indefinidamente esperando pela utilização do processador (*starvation*³⁸), de acordo com as expectativas de atendimento daquele processo (por exemplo tempo de resposta esperado). Desenvolver uma estratégia de escalonamento que abranja todas essas considerações não é tarefa trivial.

Acredita-se que a palavra chave na definição de escalonadores é **flexibilidade**. A implementação de uma determinada estratégia de escalonamento deve ser flexível a ponto de conseguir se adaptar às situações diversas que podem se configurar quando processos das mais diferentes características são submetidos (quantidade e duração de fases imprevisíveis) e devem juntos compartilhar o uso do processador.

Assim, quando um processo é selecionado a executar, o escalonador não pode precisar exatamente quando, por exemplo, ocorrerá uma solicitação de E/S que poderá causar um bloqueio, ou qualquer outro motivo que gere mudança de contexto.

³⁶ *Throughput* – quantidade de processos de usuário ativos num intervalo de tempo. Taxa relacionada a produtividade do ambiente pois determina o chamado grau de multiprogramação.

³⁷ *Turnaround* – uma média de tempo entre o instante em que um processo entra pela primeira vez no estado de pronto e o momento em que o processo é concluído (ou seja, conclui sua última fase de utilização de CPU).

³⁸ *Starvation* – fenômeno relacionado as estratégias de alocação de recursos em que um conjunto de processos é eternamente ignorado devido suas prioridades não serem tão altas quanto dos outros processos. Pode ocorrer no escalonamento de CPU, acesso a recursos de E/S, ou com qualquer outro cenário de alocação de recurso (NUTT, 1997, p.618).

Os agrupamentos de processos com características similares quanto ao uso dos recursos do sistema computacional são propostos por MACHADO & MAIA (1992) quando relacionam algumas características que poderiam ser utilizadas para realizar tal classificação:

- utilização de CPU,
- quantidade de operações de E/S,
- tempo de execução,
- espaço de memória utilizado,
- utilização de disco,
- utilização de terminal,
- utilização de impressora, etc.

Desta forma, poderiam ser definidas classes de processos, e assim, tentar atender a classes diferentes de formas diferentes. Por exemplo, processos da classe 1 seriam aqueles que tivessem as seguintes características: necessidade superior a 100 ms de tempo de processador, não utilização de disco, necessidade de utilização de memória entre 50 e 150 Kbytes, necessidade de tempo de resposta inferior a 2 ms, etc. É baseado nessas características que muitos processos são classificados como batch, tempo compartilhado e tempo real, por exemplo. Outra possível classificação é: *CPU-bound* (onde predominam fases longas de uso de CPU), *IO-bound* (onde predominam fases longas de uso de E/S) ou do tipo balanceado.

Apesar de KLEINROCK³⁹ apud TANENBAUM & WOODHULL (2000, p.70) ser bastante consistente quando afirma que *qualquer algoritmo que se defina no escalonador tende a beneficiar determinada(s) classe(s) de processos em detrimento a outra(s)*, a busca por uma estratégia flexível minimiza os graves problemas de *starvation* fazendo com que o balanceamento na utilização da CPU possa ser relativamente alcançado.

³⁹ KLEINROCK, L. *Queuing Systems*. vol.1. New York: John Wiley, 1975.

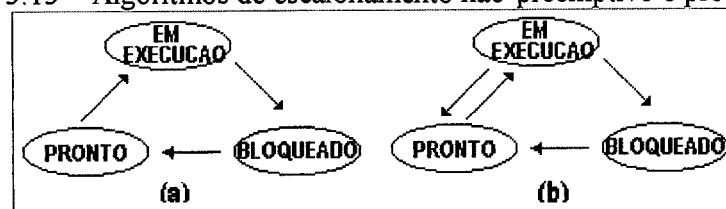
3.6.3. Algoritmos de Escalonamento

A escolha do algoritmo de escalonamento a ser implementado no SO para compartilhamento da CPU entre os processos ativos depende fundamentalmente do tipo de sistema desejado, pois nele serão definidas quais classes de processos se pretende privilegiar. A escolha do algoritmo de escalonamento implica na definição do mecanismo a ser utilizado, e é baseado nesse mecanismo que será definida a política a ser adotada que melhor reflita o que se espera do sistema em termos de alocação de processador.

Os mecanismos determinam como fazer e as políticas definem o que vai ser feito (SILBERSCHATZ & GALVIN, 2000, p.104). Questões de flexibilidade estão envolvidas na definição e aplicação desses dois termos. As políticas tendem a mudar com certa frequência. Porém, quando um mecanismo mais geral é implementado, mudanças na política implicam apenas na alteração de alguns parâmetros.

São definidos dois grupos de algoritmos de escalonamento multiprogramáveis: os não-preemptivos e os preemptivos. Nos algoritmos não-preemptivos, quando um processo ganha o direito de utilizar o processador, nenhum outro processo pode lhe tirar esse recurso até que o mesmo conclua sua execução, ou então seja bloqueado devido a espera pela execução de uma operação de E/S (Fig. 3.13(a)). Já um algoritmo de escalonamento preemptivo (Fig. 3.13(b)) é assim classificado quando o sistema poderia interromper um processo em execução, para que outro processo utilize o processador. É o que NUTT (1997) chama de **troca involuntária**.

Figura 3.13 – Algoritmos de escalonamento não-preemptivo e preemptivo.



Escalonadores não-preemptivos foram muito usados nos primeiros sistemas multiprogramados quando os processos tinham características batch. Já os escalonadores preemptivos são interessantes quando os processos precisam ter acessos curtos ao processador e em intervalos relativamente regulares, como nos processos com característica de tempo compartilhado. Estes estão baseados na definição de um

temporizador, que é inicializado a cada vez que um processo recebe a CPU, registrando o tempo de utilização da mesma. O valor do tempo máximo de utilização do processador por parte dos processos pode ser definido como único ou não.

Alguns algoritmos de escalonamento são apresentados a partir da seção seguinte, numa abordagem didática, isto é, apresentando a definição individual desses algoritmos, embora se saiba que num escalonador real são implementadas características mescladas de mais de um deles. Todos os algoritmos de escalonamento apresentados nas seções seguintes utilizam-se das rotinas apresentadas na Fig. 3.14(a), (b) (c) e (d).

Cada uma dessas rotinas está relacionada com uma das transições entre estados, a saber: a rotina de conclusão de operação de E/S da Fig. 3.14(a) é executada na transição entre os estados de BLOQUEADO e PRONTO; a rotina de conclusão de processo da Fig. 3.14(b) ocorre na transição entre os estados de EM EXECUÇÃO e CONCLUÍDO; a rotina de bloqueio de processo da Fig. 3.14(c) é executada na transição entre os estados de EM EXECUÇÃO e BLOQUEADO e a rotina de preempção de processo (exclusiva para escalonadores preemptivos) da Fig. 3.14(d), é executada na transição entre os estados de EM EXECUÇÃO e PRONTO.

Figura 3.14(a) - Rotina de conclusão de operação de E/S.

- | |
|--|
| <p>Se a operação de E/S de um processo bloqueado for concluída,</p> <ol style="list-style-type: none"> a. Incluir o BCP do processo na cauda da LCP prontos; b. Excluir o BCP da LCP do dispositivo envolvido no bloqueio do processo. |
|--|

Figura 3.14 (b) - Rotina de conclusão de processo.

- | |
|---|
| <p>Se um processo concluir sua última fase de uso de CPU,</p> <ol style="list-style-type: none"> a. Incluir o BCP na LCP concluídos. b. Executar rotina para liberar os recursos (como área de memória) alocados ao processo. |
|---|

Figura 3.14 (c) - Rotina de bloqueio de processo.

- | |
|--|
| <p>Se um processo solicita operação de E/S (iniciar fase de uso de E/S para utilização de determinado dispositivo de E/S,</p> <ol style="list-style-type: none"> a. Atualizar o seu BCP; b. Incluir o BCP na LCP bloqueados do dispositivo de E/S envolvido no bloqueio do processo. |
|--|

Figura 3.14 (d) - Rotina de preempção de processo.

Se um processo tiver sua execução interrompida por preempção,

- a. Atualizar o BCP;
- b. Incluir o BCP do processo na cauda da LCP prontos.

A seguir é apresentada a lógica de funcionamento de seis escalonadores (de multiprogramação):

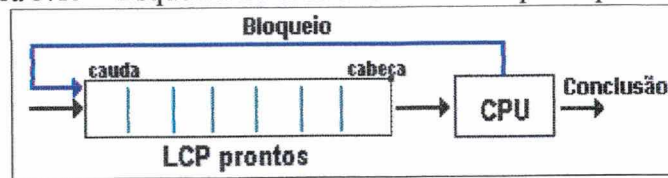
- Primeiro a chegar, primeiro a ser servido.
- Menor Primeiro.
- Por prioridade.
- Circular.
- Múltiplas filas.
- Múltiplas filas com realimentação.

Vale ressaltar que os três primeiros podem também ser utilizados como critério de seleção de processos prontos. Por exemplo, um escalonador Circular pode utilizar como critério de seleção da LCP prontos, a lógica do algoritmo Menor primeiro. A seguir os algoritmos serão explicados.

3.6.3.1. Primeiro a Chegar, Primeiro a ser Servido (PCPS)

Também chamado de FIFO (First In, First Out) nesse escalonamento o critério de atendimento é a ordem de chegada, e assim o processo que chegar primeiro é o primeiro a ser selecionado para execução. Seu algoritmo de implementação é bastante simples, sendo necessária apenas uma única LCP prontos, onde os processos que passam para o estado de pronto entram na sua cauda (final) e são escalonados quando chegarem a cabeça da lista (início). Como se trata de uma abordagem não-preemptiva, quando um processo ganha o processador, ele o utilizará sem ser interrompido, até ser concluído ou ser bloqueado (Fig. 3.15). Caso o processo em execução seja bloqueado, ele será transferido para a fila que represente o seu estado (LCP Bloqueados aguardando disco, por exemplo) sendo que lá é mantido até que um outro evento resulte na modificação do seu estado atual. Quando isto ocorrer, o processo será transferido para o final da fila de processos prontos, na qual permanecerá até que os processos na sua frente venham a ser executados.

Figura 3.15 – Esquema do escalonamento não-preemptivo PCPS.



Na Fig. 3.16 é apresentada a lógica de implementação do escalonador PCPS sendo enfatizado a atuação do BCP, das transições entre os estados e entre as LCP.

Figura 3.16 - Algoritmo de Escalonamento PCPS, abordagem não preemptiva.

1. Selecionar o BCP do processo da cabeça da LCP prontos;
2. Excluir o BCP da LCP prontos;
3. Alocar o recurso processador ao processo selecionado;
4. Executar o processo até que seja bloqueado ou concluído;
5. Se processo for bloqueado,
Executar Rotina de Bloqueio de Processo;
6. Se processo for concluído,
Executar Rotina de Conclusão de Processo;
7. Voltar ao passo 1.

Esse escalonador beneficia o processo que entrar primeiro na LCP, independente de suas características. Também o tempo de resposta médio⁴⁰ tende a ser muito alto, além da impossibilidade de se prever quando um processo terá sua execução iniciada, já que isso varia em função do tempo de execução dos processos que entraram antes na LCP prontos. Vale ressaltar que essa abordagem não é viável, em sua forma original, em ambientes de processos interativos.

3.6.3.2. Menor Primeiro (MP)

O escalonamento Menor Primeiro, que também aparece na literatura com o nome de *Shortest-Job-First*, associa cada processo ao tempo de sua próxima fase de execução. Dessa forma, quando o processador estiver livre, o processo da LCP em estado de pronto cuja próxima fase de uso de CPU for a de menor tempo de duração é selecionado para execução.

⁴⁰ Tempo de resposta médio – Tempo necessário para que o processo saia pela primeira vez da LCP prontos para que tenha sua primeira fase de uso de CPU executada. Esse e outros critérios de avaliação de desempenho de algoritmos de escalonamento serão tratados no capítulo 4.

Figura 3.17- Algoritmo de Escalonamento MP, abordagem não-preemptiva.

1. Organizar os BCP por ordem crescente de próxima fase estimada de uso de CPU;
2. Selecionar o BCP do processo da cabeça da LCP prontos;
3. Excluir o BCP da LCP prontos;
4. Alocar o recurso processador ao processo selecionado;
5. Executar o processo até que seja bloqueado ou concluído:
6. Se processo for bloqueado,
 Executar Rotina de Bloqueio de Processo;
7. Se processo for concluído,
 Executar Rotina de Conclusão de Processo;
8. Voltar ao passo 1.

Esse escalonamento, cujo algoritmo é mostrado na Fig. 3.17, pode ser tanto preemptivo quanto não-preemptivo, dependendo de **quando** é verificada a LCP prontos. Se a LCP só for analisada para escalonar um novo processo quando o atual estiver bloqueado ou for concluído, seguramente será uma implementação de escalonamento não-preemptiva. Porém, se a cada alteração da LCP prontos (por inclusão de processos novos, isto é, transição NOVO-PRONTO ou por inclusão de processos com bloqueio concluído, isto é, transição BLOQUEADO-PRONTO), uma rotina do SO analisar a LCP prontos para verificar se o processo que acabou de ser incluído nela tem a próxima fase de uso de CPU menor que a do processo corrente, o SO poderá interromper o processo corrente para selecionar aquele que acabou de ser incluído na LCP prontos. Desta forma, o escalonador MP assume uma abordagem preemptiva e é chamado de *Shortest Job First Next*.

Apesar de o escalonamento MP favorecer aos processos menores e reduzir o tempo médio de espera em relação ao escalonamento PCPS, apresenta um grave problema: como determinar a duração da próxima fase de uso de CPU para cada processo? Uma maneira de tratar essa questão é buscar uma aproximação, estimando esse valor com base na(s) última(s) fase(s) de uso de CPU daquele processo, pela crença de que, *é esperado para um dado processo, que o valor da próxima fase de uso de CPU seja similar aos valores de fases anteriores* (SILBERSCHATZ & GALVIN, 2000, p.155).

A estimativa com base no comportamento passado é chamada de **envelhecimento** (TANENBAUM & WOODHULL, 2000) e calcula a previsão para a duração da próxima fase de uso da CPU como a média exponencial de durações de fases medidas anteriormente.

A seguir é apresentado um exemplo de como o cálculo de previsão para a duração da próxima fase de uso da CPU baseado em envelhecimento é realizado. Seja t_n a duração da n ésima fase de uso da CPU, e seja t_{n+1} o valor previsto para a duração da $n+1$ ésima fase, adota-se α como um valor de parâmetro (entre 0 e 1) que controla o peso relativo da história passada e da informação recente na próxima previsão. Assim, $t_{n+1} = \alpha t_n + (1 - \alpha) t_n$.

O valor atribuído ao peso α , é normalmente de $\frac{1}{2}$ fazendo com que a história passada e a informação recente tenham pesos iguais. Note que com α igual a 1, $t_{n+1} = t_n$, a próxima fase prevista é igual a última fase registrada (ocorrida), ou seja, apenas a duração da fase de uso de CPU mais recente interessa, pressupondo que a história acumulada é irrelevante. Já com α igual a 0 então $t_{n+1} = t_n$ com a previsão da próxima fase é igual a previsão atual, supondo-se assim que as condições atuais são transitórias e portanto podem ser ignoradas.

Por exemplo, considerando que as quatro primeiras fases de uso de CPU de um determinado processo foram de 5, 4, 6 e 5 ms, qual seria a estimativa da próxima fase de uso de CPU baseada em envelhecimento ?

Nesse exemplo o valor de α será considerado $\frac{1}{2}$. Assim, tem-se $t_0=5$, $t_1=4$, $t_2=6$, $t_3=5$, e deseja-se calcular t_4 . Os cálculos apresentados abaixo, originaram o Gráfico 3.1.

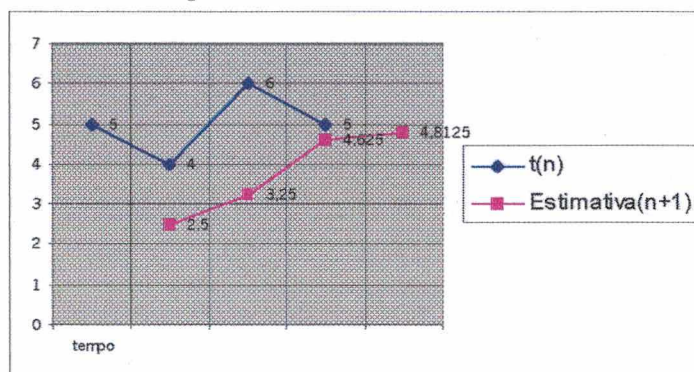
$$t_1 = \frac{1}{2} t_0 = \frac{1}{2} \cdot 5 = 2,5$$

$$t_2 = \alpha t_1 + (1 - \alpha) t_1 = 3,25$$

$$t_3 = \alpha t_2 + (1 - \alpha) t_2 = 4,625$$

$$t_4 = \alpha t_3 + (1 - \alpha) t_3 = 4,8125 \approx 5$$

Gráfico 3.1 – Exemplo de estimativas baseadas em envelhecimento.



Nota-se que o valor estimado tende a se aproximar do valor de duração das últimas fases de uso de CPU, e caso o comportamento do processo continue estável (fases de uso com duração similar as anteriores), esse algoritmo terá sucesso em suas previsões.

3.6.3.3. Baseado em Prioridades

A idéia do escalonamento por prioridade é a de atribuir (interna ou externamente) a cada processo um valor de prioridade de atendimento para uso da CPU. Desta forma, o processo da LCP prontos (implementada como única) com a prioridade mais privilegiada será executado primeiro.

Prioridades são valores numéricos em um determinado intervalo. Não há um consenso sobre se 0 deve ser a prioridade mais alta ou a mais baixa ou se são admitidos valores de prioridade negativos ou não⁴¹. Nesse estudo entendemos que quanto maior for o valor numérico atribuído à prioridade de um processo, mais privilegiado ele será no algoritmo de escalonamento.

ALBUQUERQUE (1990) afirma que a prioridade pode ser definida por critérios internos (utilização de memória, tempo de execução do processo, etc.) ou externos (importância econômica e hierárquica do processo). TANENBAUM & WOODHULL (2000, p.71) enumeram várias situações que justificam determinar prioridades para que os processos não sejam tratados de forma igualitária como no escalonamento Circular, ou ocasional como no escalonamento PCPS.

A atribuição de prioridades aos processos pode ser estática ou dinâmica, isto é, pode variar ou não ao longo da execução do processo. Tratar as prioridades como dinâmicas evita que processos entrem em situação de *starvation*, por sempre existirem outros mais prioritários. Uma alternativa seria priorizar um processo à medida em que aumente o seu tempo de vida no sistema e/ou diminua a prioridade do processo em execução, a cada período de tempo pré-determinado.

Estratégias de escalonamento baseadas em prioridade podem ser implementadas como preemptivas ou como não preemptivas. Numa abordagem preemptiva baseada em

⁴¹ Por exemplo, no LINUX as prioridades variam de -20 (a mais alta) até 19 (a mais baixa), sendo a prioridade padrão que todos os processos de usuário recebem a 0 (zero). Aos processos de usuário pode ser definida prioridade entre 0 e 19, e os valores de prioridade negativos (de -20 a -1) são atribuídos exclusivamente aos processos de SO (<http://www.LINUXdoc.org/LDP/lki/LINUX-Kernel-Internals-2.html>).

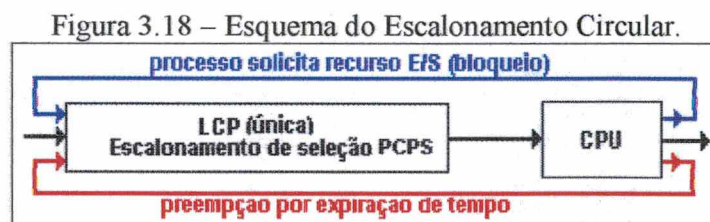
prioridade, em determinados intervalos de tempo o SO executaria uma rotina que reavaliasse as prioridades dos processos da LCP prontos para possivelmente escalonar outro processo. Desta forma, quando um processo entrar na LCP prontos e tiver prioridade maior que a do processo em execução, o sistema deverá interromper o processo corrente, colocá-lo no estado de pronto e selecionar o processo mais prioritário para ser executado.

Já uma abordagem não-preemptiva baseada em prioridade também pode ser implementada, se não for considerado nenhum mecanismo de preempção, e for simplesmente realizada a seleção do processo a ser escalonado, baseado na prioridade atribuída a cada um dos processos.

3.6.3.4. Circular

Esse algoritmo, também chamado de Round Robin ou Revezamento, é bastante semelhante ao PCPS, porém com abordagem preemptiva. Há um tempo limite único para que cada processo utilize a CPU controlado por um temporizador, que é inicializado a cada nova execução; passado esse tempo limite (também chamado de *time slice* ou *quantum*), o processo volta à cauda da LCP prontos, ficando o restante do tempo da fase de uso de CPU para ser executada no próximo escalonamento do processo.

Caso o processo em execução necessite de tempo de processador menor que o quantum, ele deliberadamente liberará o processador (por bloqueio ou conclusão) mas caso contrário, o SO forçosamente fará com que aquele processo volte a LCP prontos. Esse evento é mostrado na Fig. 3.18 como preempção por expiração de tempo.



Nessa abordagem os processos são todos tratados igualmente, apesar disso nem sempre ser desejável. Nota-se que se existem n processos na LCP prontos e o quantum de tempo é definido com q ; então cada processo terá $1/n$ do tempo da CPU em

intervalos de no máximo q unidades de tempo. Assim, cada processo não deve esperar mais do que $q(n-1)$ unidades de tempo até obter a CPU (SILBERSCHATZ & GALVIN, 2000).

O dimensionamento do quantum é uma questão importante. Se o quantum for relativamente grande, isto é, maior ou igual a maior das fases de uso de CPU dos processos, o algoritmo terá o mesmo comportamento do algoritmo PCPS, adotando uma abordagem não-preemptiva. Desta forma, nunca haverá preempção por expiração de tempo. No outro extremo, se o valor do quantum for dimensionado como muito pequeno, fragmentando em demasia a execução das fases de uso de CPU dos processos, acontece o chamado compartilhamento de processador, onde cada um dos processos terá virtualmente um processador com $1/n$ de sua velocidade real.

Outra implicação negativa na definição do quantum pequeno é em relação à necessidade do SO realizar as rotinas para mudança de contexto, que assim serão muito frequentes, fazendo com que o SO utilize o tempo do processador em demasia para ele próprio (para executar suas próprias rotinas), causando o que é chamado de *overhead*⁴².

A lógica de implementação do escalonador Circular é apresentado na Fig. 3.19, com três possíveis critérios de seleção da LCP prontos (PCPS, SJF e baseado em prioridade).

Figura 3.19 - Algoritmo de Escalonamento Circular.

- | |
|---|
| <ol style="list-style-type: none"> 1. Definir o quantum a ser considerado no sistema (único e comum a todos os processos); 2. Ler a LCP prontos; 3. Se a seleção da LCP prontos for Menor Primeiro: <ol style="list-style-type: none"> a. Organizar os BCP por ordem crescente de próxima fase de uso de CPU; 4. Se a seleção da LCP prontos for por Prioridade: <ol style="list-style-type: none"> a. Organizar os BCP por ordem decrescente de prioridade (e PCPS como critério de ordenação secundário); 5. Selecionar o BCP do processo da cabeça da LCP prontos; 6. Excluir o BCP da LCP prontos; 7. Alocar o recurso processador ao processo selecionado; 8. Inicializar o temporizador; 9. Executar o processo até que seja bloqueado, concluído ou interrompido por preempção de tempo, incrementando o temporizador a cada ciclo de relógio; 10. Se processo for bloqueado, <ol style="list-style-type: none"> Executar Rotina de Bloqueio de Processo; 11. Se processo for concluído, <ol style="list-style-type: none"> Executar Rotina de Conclusão de Processo; 12. Se o temporizador \geq quantum <ol style="list-style-type: none"> Executar Rotina de Preempção de Processo; 13. Voltar ao passo 2. |
|---|

⁴² Overhead – nesse contexto pode ser entendido como a situação que deve ser minimizada ao máximo de utilização dos recursos do sistema para o próprio sistema.

3.6.3.5. Múltiplas Filas (MF)

Como os diversos processos ativos geralmente possuem características distintas quanto à duração e frequência das fases de uso de CPU e de uso de dispositivos de E/S, a implementação de um único algoritmo de escalonamento naturalmente beneficiará a uma classe de processos em detrimento a outras. Tendo isso em vista, o escalonamento de MF implementa como o nome sugere, várias LCP prontos onde cada uma dessas listas é submetida a um algoritmo de escalonamento.

Os processos são então classificados de acordo com alguma característica parametrizada internamente pelo SO, e destinados exclusivamente a uma das LCP prontos, onde permanecem até serem selecionados para execução. É apropriado apenas quando os processos são facilmente classificados de acordo com seu tipo de processamento e necessitam de tempos de resposta diferentes.

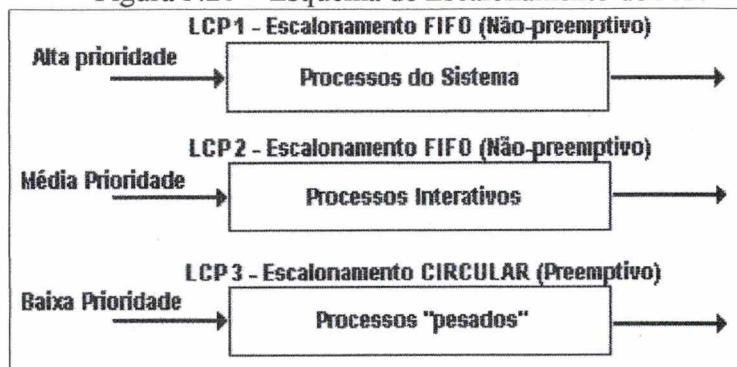
A cada uma das listas é associada uma prioridade (ou intervalo de prioridades), e o sistema somente escalona processos de uma lista menos prioritária quando todas as outras (mais prioritárias) estiverem vazias. A implementação de MF trata os processos de forma estática, isto é, os processos não são transferidos de uma LCP prontos para outra.

A Fig. 3.20 apresenta uma situação em que os processos ativos num determinado sistema são divididos em três grandes grupos:

- os processos do SO,
- os processos interativos (muita E/S e pouco uso de CPU que necessitam de tempo de resposta curto) e
- os processos considerados pesados (pouca E/S e muito processamento para os quais o tempo de resposta não é importante).

Assim, foram definidas três LCP prontos (identificadas como LCP1, LCP2 e LCP3), e de acordo com a prioridade atribuída a cada processo, eles são direcionados à lista que melhor se ajuste as suas características. Nota-se que, nesse exemplo, as listas 1 e 2 têm escalonador não-preemptivo, entendendo-se que o uso da CPU por esses processos é relativamente curto. Percebe-se também, que os processos de uma lista só receberão CPU quando não houver nenhum processo nas listas de alta e média prioridade, e ainda assim com abordagem preemptiva.

Figura 3.20 – Esquema de Escalonamento de MF.

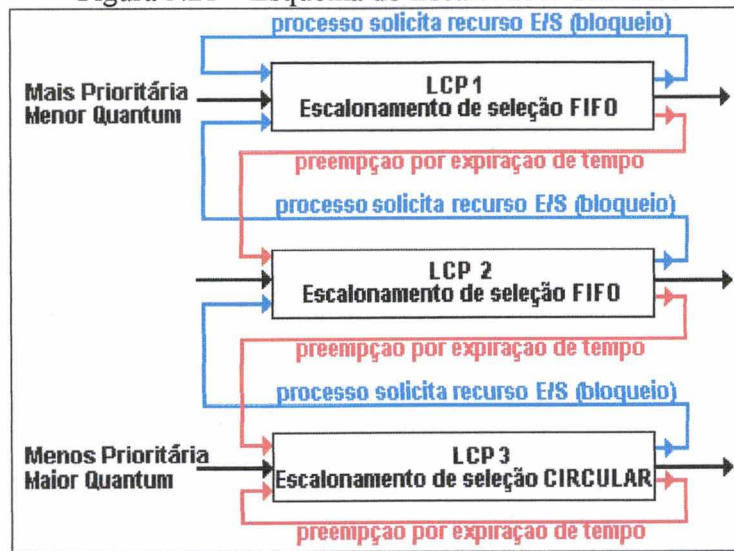


3.6.3.6. Múltiplas Filas com Realimentação (MFR)

Muitos processos são de difícil classificação, pois seu comportamento varia ao longo de sua execução (comportamento dinâmico). O algoritmo de MFR implementa diversas LCP prontos, onde cada uma implementa um algoritmo de escalonamento diferente (ou com parâmetros diferentes). O algoritmo tenta identificar dinamicamente o comportamento de cada processo, fazendo com que o mesmo transite entre LCP prontos diferentes, ajustando assim suas prioridades de execução e também as estratégias de escalonamento a que ele estará sujeito. Esse esquema permite que os processos se movimentem entre LCP prontos, fazendo com que o SO implemente um mecanismo de ajuste dinâmico denominado mecanismo adaptativo, que tem como objetivo ajustar os processos em função do seu comportamento. Assim, apesar de os processos serem inicialmente associados a uma das LCP prontos, em função do seu comportamento são redirecionados entre diversas LCP prontos do sistema ao longo de seu ciclo de vida.

Como mostrado na Fig. 3.21, a cada uma das LCP prontos é associada uma prioridade diferente (ou intervalo de prioridades) e um valor de quantum. Quanto maior é a prioridade da LCP prontos, menor é seu quantum. O escalonamento de um processo em uma determinada LCP prontos só acontece quando não houver nenhum processo pronto nas outras mais prioritárias.

Figura 3.21 – Esquema do Escalonador de MFR.



Quando o processo que está sendo executado deixa o processador por expiração de tempo, ele é redirecionado para a LCP prontos imediatamente inferior a que ele estava, de forma a penalizar processos com longas fases de uso de CPU. Da mesma forma, um processo que pare de executar por bloqueio (aguardando recurso de E/S), quando voltar ao estado de pronto, será redirecionado para uma LCP prontos imediatamente superior a que estava alocado. Assim, processos que utilizem pouca CPU tendem a ser privilegiados nesse algoritmo.

Essa política de escalonamento atende às necessidades de diversos tipos de processos. No caso de processos que necessitem de muita E/S, ela oferece um bom tempo de resposta, já que esses processos são mais prioritários, por permanecerem a maior parte do tempo nas filas de mais alta ordem, mas não monopolizam o uso do processador, pois não o utilizam extensivamente.

Quando processos que necessitem de muito processamento entrarem na fila de maior prioridade ganharão o processador, gastarão seu quantum de tempo e serão normalmente direcionados para uma fila de menor prioridade. Desta forma, quanto mais tempo um processo se utiliza do processador, mais ele vai *caindo* para filas de menor prioridade.

O escalonador de MFR é considerado o mecanismo de escalonamento mais generalista, podendo ser implementado em qualquer tipo de sistema, independentemente da carga de trabalho e característica dela. Se vários processos de mesma característica

são submetidos (vários processos que utilizam muita CPU, por exemplo), todos eles, ao longo de determinado tempo, serão direcionados à LCP menos prioritária entre as existentes, por terem esgotado seu quantum repetidamente. Desta forma, o escalonamento tratará a todos de forma justa.

Se, em meio a esses processos, surgir um com características diferentes (por exemplo, um que necessite de pouco processador e muita E/S), esse será privilegiado, sem no entanto monopolizar a CPU. SHAY (1996) afirma que MFR se assemelha a uma abordagem Circular quando há muita atividade de E/S, e se assemelha a PCPS quando há pouca ou nenhuma atividade de E/S.

O problema que esse escalonador apresenta é que, por sua complexidade, é gerado um certo *overhead* ao sistema, o que, mesmo assim, pode compensar sua implementação.

Na Fig. 3.22 é apresentado o algoritmo do escalonador, atentando para as algumas considerações:

- Fator – valor base para determinar o quantum de cada LCP prontos.
- As prioridades dos processos variam de 9 a 0 (sendo 9 a mais alta prioridade).
- Há 3 LCP prontos identificadas como LCPP, especificadas como:
 - LCPP(1) – prioridade 9 a 7, quantum = fator*1.
 - LCPP(2) – prioridade 6 a 4, quantum = fator *2.
 - LCPP(3) – prioridade 3 a 0, quantum = fator *3.

Figura 3.22 - Algoritmo de Escalonamento MFR.

<p>Para $i=1$ a 3 faça:</p> <ol style="list-style-type: none"> a. Ler a LCPP(i); b. Se LCPP(i) estiver vazia Então, Escapar do laço (voltar ao "Para i" com i incrementado); c. Se a seleção da LCPP(i) for Menor Primeiro: Então, Organizar os BCP da LCPP(i) por ordem crescente de próxima fase de uso de CPU (e PCPS como critério de ordenação secundário); d. Se a seleção da LCPP(i) for por Prioridade: Então, Organizar os BCP da LCPP(i) por ordem decrescente de prioridade (e PCPS como critério de ordenação secundário); e. Selecionar o BCP do processo da cabeça da LCPP(i); f. Excluir o BCP da LCPP(i); g. Alocar o recurso processador ao processo selecionado; h. Inicializar o temporizador; i. Executar o processo até que seja bloqueado, concluído ou interrompido por preempção e incrementar o temporizador a cada ciclo de máquina; j. Se processo for bloqueado, Então, Executar rotina de bloqueio de processo; k. Se processo for concluído, Então, Executar rotina de conclusão de processo; l. Se o temporizador \geq quantum da LCPP considerada Então, Executar rotina de preempção de processo; <p>Fim Para;</p>

3.7. DEADLOCK

Essa seção é dividida em quatro partes. Na primeira apresenta-se o conceito de *deadlock* e o enunciado das condições para sua ocorrência. A segunda parte mostra a modelagem de *deadlock* através do uso de grafos com um e vários recursos de cada tipo, e a terceira parte mostra as estratégias de tratamento de situações de *deadlock*, incluindo a análise da negação das condições de *deadlock* e suas conseqüências e a definição de estados seguros e inseguros para evitar o *deadlock*. Na quarta seção são tratadas as estratégias de detecção e recuperação de situações de *deadlock*.

3.7.1. Conceito de *Deadlock*

Em um SO multiprogramado encontram-se diversos processos que competem durante a sua execução por um número limitado de recursos compartilháveis. O uso destes recursos compartilhados⁴³ pode ocasionar uma situação indesejável denominada *deadlock*⁴⁴, na qual um número de processos se encontram bloqueados, esperando por condições que nunca serão satisfeitas. Assim, uma situação de *deadlock* seria aquela em que processos ficariam bloqueados para sempre, sem qualquer possibilidade de continuarem sendo executados sem algum tipo de intervenção externa.

Segundo TANENBAUM & WOODHULL (2000, p.122):

Um conjunto de processos está em uma situação de deadlock, se cada processo do conjunto estiver esperando por um evento que somente outro processo pertencente ao conjunto poderá fazer acontecer.

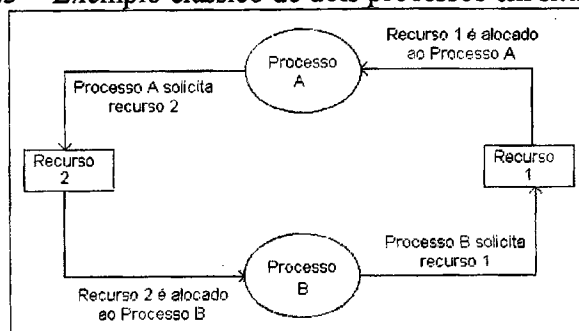
O exemplo clássico da situação de *deadlock* é apresentado na Fig. 3.23 na qual dois processos identificados como A e B encontram-se fazendo uso de dois diferentes recursos denominados de 1 e 2. Vamos supor que o processo A requisite e adquira o recurso 1. Há mudança de contexto e o processo B requisite e adquira o recurso 2. Novamente haveria mudança de contexto, resultando na execução do processo A, que tenta solicitar o recurso 2. Uma vez que o recurso 2 já foi adquirido pelo processo B, o processo A é bloqueado (uma vez que ele necessita alocar a si um recurso que está

⁴³ Recurso nesse contexto é entendido como um elemento que o processo necessita e que, se não disponível fará com que o processo fique bloqueado. Mais detalhes foram apresentados na seção 3.4.

⁴⁴ *Deadlock* é denominado de impasse por SILBERSCHATZ & GALVIN (2000) e TANENBAUM & WOODHULL (2000), e também paralisação e abraço fatal por SHAY (1996).

atualmente ocupado) e aguarda que o recurso 2 seja liberado. O processo B passa novamente a ser executado e tenta adquirir o recurso 1. Uma vez que tal recurso já foi adquirido pelo processo A, o processo B tem sua execução também bloqueada. Esse é o exemplo clássico de uma situação de *deadlock*, com os dois processos bloqueados dependendo de condições que só podem ser satisfeitas pelo outro processo que está na mesma situação.

Figura 3.23 - Exemplo clássico de dois processos em situação de *deadlock*.



Fonte: MACHADO & MAIA (1997, p.101).

Pode ser observado que tanto a quantidade de processos quanto a quantidade de recursos envolvidos no problema é irrelevante, pois pelo fato de todos os processos estarem esperando, nenhum deles poderá fazer acontecer qualquer um dos eventos que possam vir a acordar (fazer executar) um dos demais membros do conjunto. Assim sendo, todos os processos do conjunto vão ficar eternamente bloqueados. Na maioria das vezes, como citado no exemplo, o evento que cada processo está esperando é a liberação de algum dos recursos que está no momento alocado a um outro processo do conjunto. Como nenhum destes processos pode rodar, nenhum recurso será liberado, e nenhum dos processos será acordado.

Autores afirmam que os *deadlocks* podem também ocorrer em situações que não envolvem recursos⁴⁵. A explicação é que num sistema dinâmico, as requisições de recursos ocorrem durante o tempo todo e são então implementadas políticas para decidir a ordem de atendimento dos processos e a ordem de atendimento das requisições de recursos. Algumas dessas políticas podem fazer com que alguns processos nunca sejam atendidos. Por exemplo, se numa fila de impressão for usado como critério de atendimento o tamanho do arquivo a ser impresso, isso fará com que os grandes trabalhos sejam desprestigiados. Apesar de não se tratar de um *deadlock* verdadeiro e

⁴⁵ TOSCANI (1987) chama essa situação de *deadlock* artificial ou postergação indefinida e TANENBAUM (1995) chama de preterição indefinida.

sim de uma falha de implementação do SO, alguns autores o mencionam essa situação como sendo um tipo de *deadlock*.

3.7.1.1. Condições para ocorrência de *deadlock*

Situações de *deadlock* prejudicam os processos fazendo-os ficar impedidos de prosseguir, e portanto essas situações são indesejáveis. Para que ocorram situações de *deadlock* em um sistema são necessárias quatro condições enunciadas por COFFMAN⁴⁶ apud TANENBAUM (1995, p.164). Todas as quatro condições devem ser satisfeitas para que possa ocorrer *deadlock*. Se uma delas estiver ausente, não há possibilidade do *deadlock* ocorrer. São elas:

- **Condição de Exclusão Mútua** - Cada recurso só pode estar alocado a um único processo em um determinado instante. Deve existir pelo menos um recurso que, a cada instante, pode ser usado por apenas um processo. Se outro processo requisitar o recurso, esse processo deve ser bloqueado até que o recurso seja liberado.
- **Condição de Posse e Espera** - Um processo, além dos recursos já alocados a ele, pode solicitar e estar esperando por outro(s) recurso(s). Deve existir, em um dado instante, um processo que esteja usando pelo menos um recurso e esperando que outros recursos, que estão nesse instante sendo usados por outros processos, sejam alocados para seu uso.
- **Condição de Não-Preempção** - Um recurso não pode ser liberado de um processo só porque outro(s) processo(s) deseja(m) o mesmo recurso. Não há possibilidade de preempção, ou seja, um recurso só pode ser liberado voluntariamente pelo processo ao qual está alocado depois que o mesmo terminar de usá-lo.

⁴⁶ COFFMAN E.G., System Deadlocks. *Computing Surveys*, vol.3, p.67-78, jun.1971.

- **Condição de Espera Circular** - Deve haver uma cadeia circular de dois ou mais processos, cada um deles esperando por um recurso alocado a outro processo da cadeia. Como afirmam SILBERSCHATZ & GALVIN (2000, p.233):

Deve existir um conjunto $\{P_0, P_1, P_2, \dots, P_n\}$ de processos em espera, tal que P_0 esteja esperando por um recurso alocado a P_1 , P_1 esteja esperando por um recurso alocado a P_2 , ..., P_{n-1} esteja esperando por um recurso alocado a P_n , e P_n esteja esperando por um recurso alocado a P_0 .

Percebe-se que as quatro condições não são independentes; por exemplo, a condição de espera circular implica na condição de posse e espera.

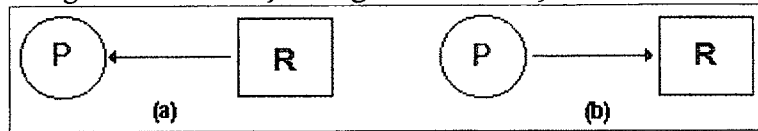
3.7.2. Modelo de Deadlock: representação com grafos

O uso de grafos de alocação de recursos constitui uma ferramenta que permite visualizar se uma dada seqüência de solicitações/liberações leva ou não a situações de *deadlock*. Também chamado de grafo dirigido ou dígrafo, pode ser apresentado como uma seqüência de pares (R, P), onde há dois tipos de nós: os processos (P) representados por círculos e os recursos (R) indicados por quadrados (TANENBAUM, 1995).

No contexto da representação de *deadlock*, um arco saindo de um nó de recurso para um nó de processo significa que o recurso foi solicitado pelo processo e entregue a ele, estando o recurso alocado atualmente ao processo. Um arco indo de um processo para um recurso significa que o processo está bloqueado aguardando tal recurso ser a ele alocado. Assim, $P \rightarrow R$ é um arco de requisição, e $R \rightarrow P$ é um arco de alocação. Nota-se que os grafos são bipartidos, isto é, não existem arestas ligando nós do mesmo tipo (apenas arestas ligando processos a recursos ou vice-versa).

Na Fig. 3.24(a), o processo P está de posse do recurso R, isto é, o recurso no momento está alocado a P. Já na Fig. 3.24(b), o processo P requisita o recurso R, sendo que essa requisição pode ou não resultar numa alocação, dependendo se o recurso está disponível ou não. Após a liberação do recurso, o arco que relacionava processo e recursos desaparecem da representação com grafos.

Figura 3.24 - Notação de grafos de alocação de recursos.



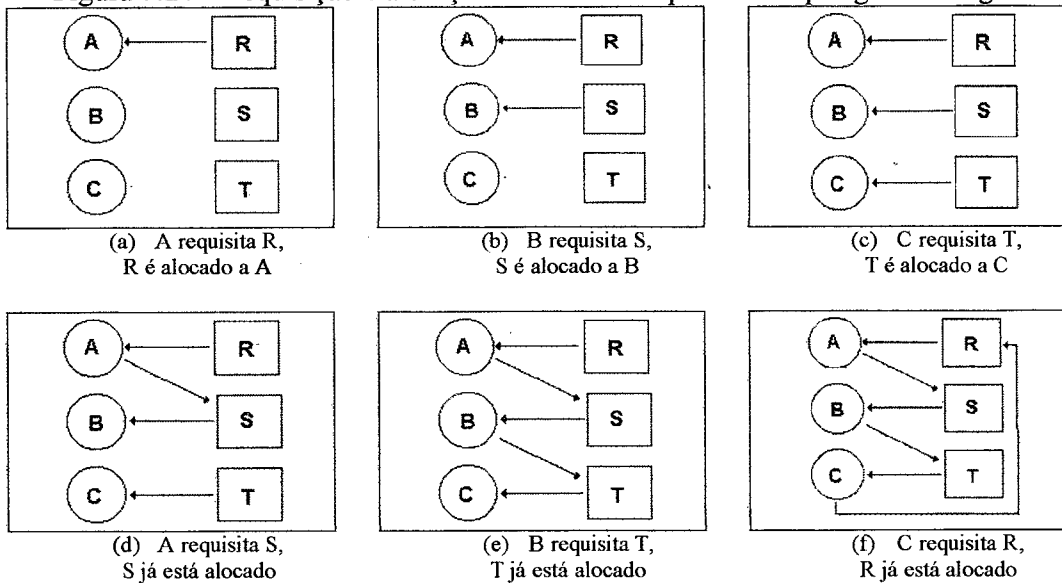
O exemplo apresentado nas Fig. 3.25 e 3.26, adaptado de TANENBAUM & WOODHULL (2000, p.125) mostra a facilidade de visualizar uma situação que pode ou não caracterizar *deadlock*, através do uso dos grafos dirigidos. Supõe-se a seguinte seqüência de requisições e liberações de três processos concorrentes e que compartilhem recursos, onde R, S e T são recursos e A, B e C são processos. A execução dos processos aconteceu de forma que os eventos ocorreram como se segue (Fig. 3.25).

Figura 3.25 – Requisições de recursos de processos concorrentes.

Processo A	Processo B	Processo C
Requisita R	Requisita S	Requisita T
Requisita S	Requisita T	Requisita R

A evolução das requisições e alocações é mostrando graficamente passo a passo na Fig. 3.26, através de grafos de alocação de recursos. De forma visual, a Fig. 3.26(f) apresenta um ciclo (A → S → B → T → C → R → A) e então pode-se afirmar a ocorrência de *deadlock*. Considerando-se ambientes com um recurso de cada tipo, a presença de um ciclo é condição necessária e suficiente para a existência de situação de *deadlock*.

Figura 3.26 - Requisição e alocação de recursos representada por grafos dirigidos.



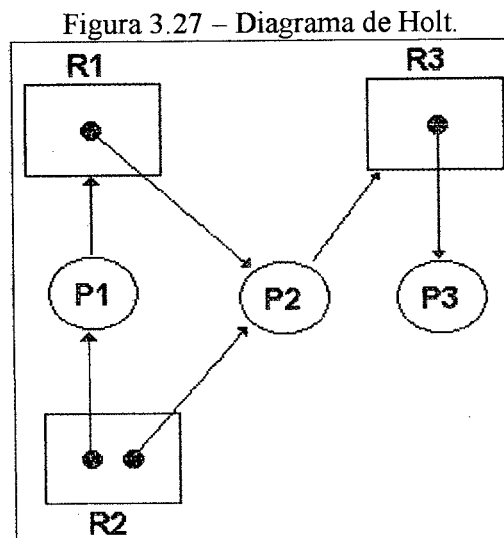
Fonte: Adaptado de TANENBAUM & WOODHULL (2000, p.125).

3.7.2.1. O modelo de HOLT: grafos representando vários recursos de cada tipo

Pode-se dizer que a representação da seção anterior é um caso particular do diagrama de HOLT, pois há dois tipos de nós: quadrados que representam os recursos e os círculos representando os processos. A diferença é que esse modelo permite representar recursos com várias unidades existentes (vários recursos de cada tipo). Os nós recursos têm internamente pequenos círculos que representam as unidades existentes.

Um processo estará bloqueado quando ele estiver requisitando recursos que não estão disponíveis, e não estará bloqueado quando todos os recursos que ele estiver requisitando estiverem disponíveis.

A Fig. 3.27 apresenta um exemplo, onde há um recurso do tipo R1, dois do tipo R2, um do tipo R3 e há três processos ativos (P1, P2, P3).



O processo P1 tem um recurso R2 e está esperando por um recurso do tipo R1. O processo P2 está de posse de um recurso do tipo R1 e solicita um recurso R2 e um recurso R3. O processo P3 está de posse de um recurso do tipo R3 e de um recurso do tipo R2. Já que há uma seqüência de execuções que possibilita que todos os processos possam ser executados, pode-se afirmar que não há *deadlock*, já que se P3 receber processador e for concluído, os recursos R3 e R2 (um deles) seriam liberados e poderiam assim ser alocados a P2, o que possibilitaria também sua conclusão. Em seguida poderia ocorrer a conclusão de P1. A possível ocorrência de *deadlock* está vinculada, nesse exemplo, a seqüência considerada para execução dos processos.

Nota-se que nesse modelo (onde podem ser representados vários recursos de cada tipo) um ciclo não implica necessariamente na ocorrência de *deadlock* ($R1 \rightarrow P2 \rightarrow R2 \rightarrow P1 \rightarrow R1$). A existência de um ciclo é condição necessária, mas não suficiente para ocorrência de *deadlock*.

3.7.3. Estratégias de Tratamento de Situações de Deadlock

Segundo SILBERSCHATZ & GALVIN (2000) há basicamente três formas de abordar a questão do *deadlock*:

- Implementar estratégias para garantir que o mesmo nunca ocorra.
- Permitir que o *deadlock* ocorra e depois resolver o problema.
- Ignorar as possíveis ocorrências de *deadlock*.

Para garantir a ausência de *deadlock*, o sistema pode implementar dois métodos diferentes. No primeiro é feito com que pelo menos uma das quatro condições necessárias para a ocorrência de *deadlock* não seja satisfeita. Esse método evita *deadlock* por meio de restrições sobre como as requisições de recursos podem ser feitas. No segundo método, informações adicionais sobre quais recursos cada processo ativo necessita durante sua execução são fornecidas antecipadamente e mantidas pelo SO, servindo de base para decidir, a cada requisição, se o processo que fez a requisição deve esperar ou não. Desta forma, o *deadlock* é impedido baseado na análise da seqüência de execuções dos processos.

Caso o SO não implemente nenhum dos dois métodos anteriores, poderão ocorrer situações de *deadlock*, e podem ser implementadas estratégias que examinem o estado do sistema para verificar se há situação de *deadlock* (estratégias de detecção de *deadlock*), e em caso positivo, tratá-la (estratégias de recuperação de *deadlock*).

Se não for implementada nenhuma estratégia para impedir, detectar e/ou recuperar possíveis situações de *deadlock*, o sistema não poderá reconhecer uma possível ocorrência de *deadlock*. Apesar de essa atitude não parecer razoável, ela é usada em alguns SO, por seus projetistas entenderem que a maioria dos usuários preferiria a ocorrência de *deadlocks* ocasionais, em vez da implementação no SO de rotinas *pesadas* que comprometam a performance do sistema ou então, implementações

que, por exemplo, forcem todos os processos a somente poderem alocar um recurso de cada vez, para negar uma das quatro condições de ocorrência do *deadlock*.

Assim, o não tratamento de *deadlock* é justificado pela observação da probabilidade de ocorrência dele, que muitos projetistas consideram baixa, e pelo *overhead* causado no sistema para seu tratamento. Autores como TANENBAUM (1995) chamam essa atitude como o *Algoritmo do Avestruz*, fazendo analogia à suposta reação desse animal que, quando se depara com uma situação de problema, enterra a cabeça na areia se omitindo a enfrentá-la.

Por ser um dos focos de estudo nesse trabalho, as subseções seguintes limitam-se a tratar especificamente de estratégias de detecção de *deadlock*.

3.7.4. Detecção de Situações de Deadlock

Se não for implementada nenhuma das estratégias que impeçam a ocorrência de *deadlock*, podem ser implementados meios que verifiquem se há existência de *deadlock* entre os processos ativos e, caso o *deadlock* seja detectado, identificar os processos e recursos envolvidos, para que possíveis estratégias de recuperação de situações de *deadlock* sejam executadas.

Uma questão importantíssima na detecção de *deadlock* é quando eles devem ser procurados no sistema. É interessante perceber que, quanto mais mecanismos de detecção de *deadlock* forem implementados no SO, maior será o tempo alocado às rotinas do próprio SO (*overhead*), o que poderá comprometer o tempo de atendimento aos processos de usuário. Uma possibilidade é verificar a ocorrência de *deadlock* a cada novo recurso solicitado pelos processos ativos. Nota-se que seria necessário um considerável tempo de processamento para executar as rotinas de detecção para uma periodicidade tão curta. Uma outra solução seria executar rotinas de detecção de *deadlock* sempre que a utilização do processador atingisse um nível de performance considerado baixo (já que esse nível poderia ser indicativo de uma possível situação de *deadlock*).

Para elaborar um algoritmo que permita verificar se há processos e recursos envolvidos em *deadlock*, é necessário considerar duas situações (TANENBAUM,

1995): uma situação hipotética considerando um único recurso de cada tipo e uma situação mais real quando há mais de um recurso de cada tipo.

3.7.4.1. Detecção de Deadlock com um único recurso de cada tipo

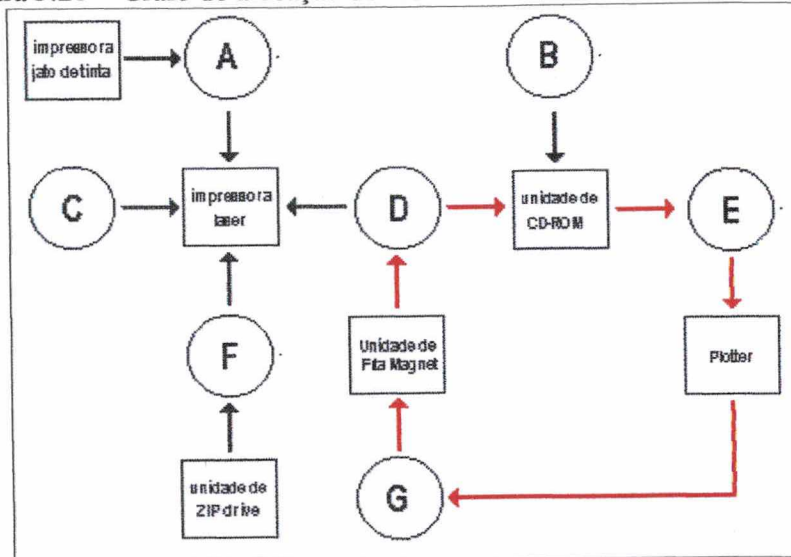
Considera-se um ambiente onde só exista um recurso de cada tipo, por exemplo, somente uma impressora, uma unidade de disco, uma unidade de fita, entre outros; de tal forma que os processos identifiquem o recurso que solicitam de forma específica. Podemos então construir um grafo de recursos e processos, e se tal grafo contiver um ou mais ciclos, está caracterizada a ocorrência de *deadlock*. Caso não existam ciclos, o sistema não estará em *deadlock*.

Tomando por base o exemplo de TANENBAUM (1995, p.167), onde há sete processos (de A a G) e seis recursos, observa-se que em determinado instante a situação é a seguinte:

- O processo A está de posse da impressora jato de tinta e solicita a impressora laser.
- O processo B não está de posse de nenhum recurso e solicita a unidade de CD-ROM.
- O processo C não está de posse de nenhum recurso e solicita impressora laser.
- O processo D está de posse da Unidade de Fita Magnética e solicita impressora laser e a unidade de CD-ROM.
- O processo E está de posse da unidade de CD-ROM e solicita plotter.
- O processo F está de posse do recurso unidade de Zip-Drive e solicita impressora laser.
- O processo G está de posse do recurso plotter e solicita unidade de fita magnética.

Através da análise visual da Fig. 3.28, pode-se perceber a existência de um ciclo (mostrado em vermelho) envolvendo os processos D, G e E, o que é condição necessária e suficiente para afirmar-se que os processos estão em *deadlock*.

Figura 3.28 – Grafo de alocação de recursos com um recurso de cada tipo.



Fonte: Adaptado de TANENBAUM (1995, p.168).

O algoritmo da Fig. 3.29 usa o princípio tratado na análise visual: ele inspeciona um grafo e a análise termina é encontrado um ciclo (e portanto, *deadlock*), ou quando um ciclo não é encontrado (não existência de *deadlock*). É utilizada uma estrutura de dados – uma lista de nós L – e durante a execução do algoritmo, os arcos são marcados para indicar que já foram inspecionados.

Figura 3.29 – Algoritmo de Detecção com um único recurso de cada tipo.

Para cada nó N do grafo, realizar os passos a seguir, com N sendo o nó inicial:

1. Inicializar L como uma lista vazia, e designar todos os arcos como desmarcados;
2. Colocar o nó corrente no fim da lista L, e verificar se ele aparece mais de uma vez em L. Caso sim, o grafo contém um ciclo cujos nós estão identificados em L, e ir para FIM.
3. A partir deste nó, verificar se há algum arco desmarcado partindo do nó. Se houver ir para o passo 4, senão ir para o passo 5.
4. Escolher aleatoriamente um arco não assinalado partindo deste nó e marcá-lo; em seguida pegar o próximo nó, e voltar ao passo 2.
5. Voltar ao nó anterior (o tratado antes do atual), considerar novamente como corrente e voltar ao passo 2. Se este nó for o inicial, o grafo não contém ciclos e então FIM.

FIM.

Fonte: TANENBAUM (1995, p.168).

3.7.4.2. Detecção de *deadlock* com vários recursos de cada tipo

Num ambiente podem existir várias cópias dos mesmos tipos de recursos (agrupados em classes) e os processos não identificam especificamente os recursos que solicitam, isto é, um processo pode, por exemplo, solicitar serviços de impressão em qualquer uma das impressoras jato de tinta existentes, e não necessariamente de uma específica. Assim, é necessária a adoção de uma abordagem diferente da apresentada anteriormente para detectar possíveis situações de *deadlock*. Para isso, são mantidas estruturas de dados atualizadas dinamicamente pelo SO, capazes de armazenar informações sobre os recursos existentes, os recursos já alocados para cada processo ativo, e os processos que estão em espera por determinados recursos.

Essas estruturas de dados são matrizes unidimensionais (para guardar a quantidade de recursos existentes e disponíveis de cada tipo) e bidimensionais (onde o número de linhas depende do número de processos ativos e o número de colunas depende da quantidade de classes de recursos existentes).

Como ilustrado na Fig. 3.30, as estruturas de dados mantidas são as seguintes (onde m representa os recursos e n representa os processos):

- **E** – matriz unidimensional – indica o número total de unidades de cada um dos m recursos existentes.
- **D** – matriz unidimensional – indica o número de recursos atualmente disponíveis de cada tipo.
- **A** – matriz bidimensional – indica a alocação corrente, isto é, informa quantos recursos de cada tipo estão atualmente alocados a cada processo. Nessa matriz há uma linha para cada um dos n processos ativos.
- **R** – matriz bidimensional – indica as requisições, isto é, indica o número de recursos de cada tipo cada um dos processos ainda necessita. Nessa matriz há uma linha para cada um dos n processos ativos.

Figura 3.30 – Matrizes para detecção de *deadlock* com vários recursos de cada tipo.

$$\begin{array}{l}
 E = [e_1 \ e_2 \ \dots \ e_m] \quad D = [d_1 \ d_2 \ \dots \ d_m] \\
 A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & r_{nm} \end{bmatrix}
 \end{array}$$

Será considerado que: n representa o número de processos ativos; m indica quantos são os tipos de recursos existentes num determinado ambiente; j indica o processo corrente (variando de 1 a m) e i indica o recurso corrente (variando de 1 a n). Assim por exemplo, e_j indica a quantidade de recursos existentes do tipo j , a_{ij} indica a quantidade de recursos do tipo j que estão alocados ao processo i , e r_{ij} indica qual a quantidade de recursos do tipo j o processo i necessita além das unidades já alocadas a ele. Desta forma, $d_j = e_j - \sum_{i=1}^n (a_{ij})$.

O algoritmo de detecção de *deadlock* com vários recursos de cada tipo está baseado na comparação das matrizes unidimensionais R e D . Com essas estruturas é possível definir um algoritmo (Fig. 3.31) onde cada processo é considerado inicialmente como desmarcado e os processos serão marcados, indicando quais deles estão habilitados a terminar seu processamento por não estarem envolvidos em situações identificadas como de *deadlock*. Quando o algoritmo termina, todos os processos desmarcados (se houver algum) estarão em *deadlock*. O que esse algoritmo faz é simular uma seqüência de execução para os processos ativos não bloqueados, isto é, aqueles cujas requisições possam ser atendidas.

A justificativa do algoritmo é que, se existe *deadlock* num determinado momento num sistema é porque existem processos bloqueados que continuarão assim, mesmo que os outros processos liberem todos os recursos alocados a eles. Por outro lado, se a partir de um determinado estado, existir alguma maneira de se alocar recursos de modo que processos possam evoluir, então é porque não há situação de *deadlock* entre eles (TOSCANI, 1987).

Assim, se ao final do algoritmo todos os processos forem desbloqueados, é porque não houve *deadlock*. O desempenho do algoritmo pode ser melhorado se inicialmente marcar-se todas as linhas i para as quais $A_i=0$, já que esses processos não têm nenhum recurso alocado, e portanto não podem estar em situação de *deadlock*.

Figura 3.31 - Algoritmo de Detecção com vários recursos de cada tipo.

1. Fazer W receber a matriz D (considera-se W uma matriz unidimensional auxiliar de m elementos, inicialmente zerada).
2. Enquanto existir linha i (não marcada) E $(R_i \leq W_i)$ fazer:
 - 2.1. W recebe $W + A_i$
 - 2.2. Marcar a linha i .
3. Fim.

Fonte: TOSCANI (1987).

A Fig. 3.32 mostra um exemplo de uma situação onde será aplicado o algoritmo de detecção de *deadlock* com vários recursos de cada tipo. Nota-se que são consideradas inicialmente como desmarcadas todas as linhas de A, e W é igual a [0 1 0]. A lógica do algoritmo é buscar-se linhas em $R \leq D$. Inicialmente o único processo que iria satisfazer essa condição é o processo P2, e sua linha seria marcada. O vetor W então ficaria [1 2 1]. Sendo o processo P3 marcado, W ficaria [1 3 1], e assim P1 seria marcado e W ficaria [2 3 1]. Como ao final não haveria nenhum processo desmarcado, pode-se concluir que não haveria situação de *deadlock*.

Figura 3.32 – Exemplo de detecção de *deadlock* com vários recursos de cada tipo.

Matriz E (recursos existentes)				Matriz D (recursos disponíveis)			
R1 (Unid.Fita)	R2 (Plotter)	R3 (Impressora)		R1 (Unid.Fita)	R2 (Plotter)	R3 (Impressora)	
2	3	1		0	1	0	

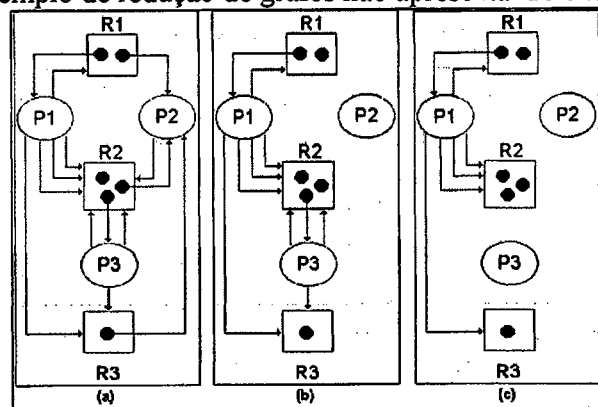
Matriz A (recursos alocados)				Matriz R (recursos requisitados)			
Processos	R1 (Unid.Fita)	R2 (Plotter)	R3 (Impressora)	Processos	R1 (Unid.Fita)	R2 (Plotter)	R3 (Impressora)
P1	1	0	0	P1	1	3	1
P2	1	1	1	P2	0	1	0
P3	0	1	0	P3	0	2	1

A versão visual desse algoritmo é a chamada operação de **redução de grafos** (considerando o modelo de Holt). A idéia é percorrer o grafo e eliminar, sucessivamente as arestas correspondentes aos processos não bloqueados. Se todas as arestas são eliminadas diz-se que o grafo é completamente redutível. A eliminação das arestas pode ocasionar desbloqueio de novos processos, os quais também podem ter então suas arestas eliminadas, e assim sucessivamente.

Existe situação de *deadlock* quando o grafo correspondente a esse estado não for completamente redutível. Assim, os processos cujas arestas não podem ser eliminadas são os que se encontram em *deadlock*.

O mesmo exemplo da Fig. 3.32 é mostrado graficamente na Fig. 3.33, onde inicialmente todas as requisições e alocações dos processos são mostradas na Fig.3.33(a), e na seqüência, os processos P2 e P3 têm suas arestas reduzidas (b) e (c).

Figura 3.33 – Exemplo de redução de grafos não apresentando situação de *deadlock*.

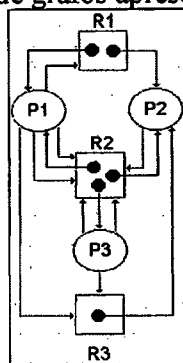


Fonte: TOSCANI (1987).

Como o grafo da Fig.3.33(a) representa um ambiente com vários recursos de cada tipo, apesar de haver um ciclo no mesmo ($P2 \rightarrow R2 \rightarrow P3 \rightarrow R3 \rightarrow P2$) isso não necessariamente caracteriza uma situação de *deadlock*.

Nota-se que com uma pequena alteração no exemplo anterior, pode-se verificar que ocorreria *deadlock*. Se P1 estivesse de posse de um recurso R2, o gráfico ficaria como na Fig. 3.34, e W seria [0 0 0]. Nenhum processo poderia portanto ter suas requisições satisfeitas e seriam os três marcados como estando em *deadlock*.

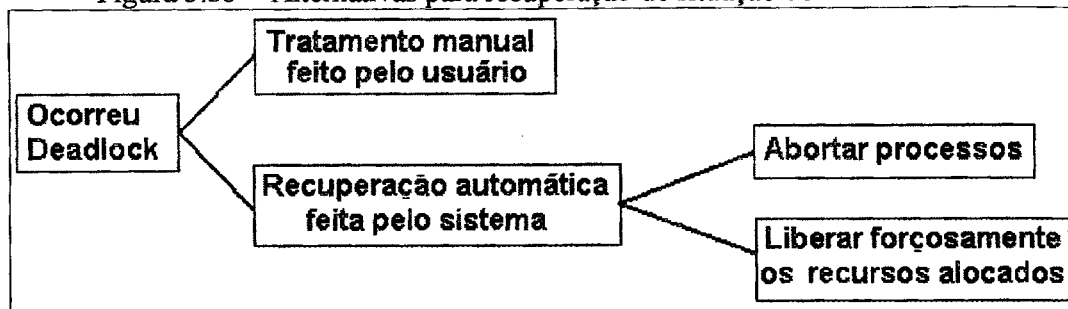
Figura 3.34 – Exemplo de redução de grafos apresentando situação de *deadlock*.



3.7.4.3. Recuperação de Situações de Deadlock

Após detectar uma situação caracterizada como *deadlock* é possível implementar algum tipo de estratégia objetivando retornar à condição normal buscando corrigir o problema. Para isso há basicamente duas alternativas (Fig.3.35). A primeira é informar ao usuário que processos estão em situação de *deadlock*, e orientá-lo a resolver a situação manualmente. A segunda alternativa é ter implementado no SO rotinas que objetivem recuperar automaticamente a situação.

Figura 3.35 – Alternativas para recuperação de situação de *deadlock*.



A recuperação automática de situações de *deadlock* está baseada na quebra de duas condições de ocorrência de *deadlock*: espera circular e não preempção de recursos. Na quebra da espera circular há a eliminação de um ou mais processos ativos (e conseqüentemente na liberação dos recursos a eles alocados) e na eliminação da preempção de recursos, eles (os recursos) são forçosamente liberados sucessivamente e alocados a outros processos (NUTT, 1997).

A recuperação, seja qual for a estratégia de quebra implementada, tem implicações muito sérias para o sistema, que vão desde a definição de critérios para a escolha do(s) processo(s) a ser(em) eliminado(s), até dificuldades em recuperar (mecanismo de *rollback*) o que já foi realizado pelo(s) processo(s) escolhido(s) para ser(em) abortado(s).

As danosas conseqüências de qualquer uma das abordagens propostas para recuperação de *deadlock* são verdadeiros efeitos colaterais, por isso afirma-se que qualquer uma das duas alternativas (tratamento manual ou tratamento automático) é questionável. A primeira porque expõe ao usuário uma falha de alocação de recursos, que gerou, como conseqüência, a impossibilidade dos processos prosseguirem. Mesmo que falhas desse tipo ocorram ocasionalmente e propositalmente por questões

estratégicas (diminuir o overhead do sistema e conseqüentemente melhorar a performance), nenhum usuário aceita calmamente que falhas ocorram, gerando questionamentos sobre a confiabilidade do sistema. A segunda conseqüência é o fato de ser extremamente difícil implementar algum algoritmo de recuperação de *deadlock* que prejudique ao mínimo a execução de processos principalmente se eles são cooperantes⁴⁷.

⁴⁷ Processos cooperantes são aqueles cujas saídas servem de entradas para outros processos, e/ou aqueles cujas entradas foram geradas como saídas de outros processos.

CAPÍTULO 4 - AVALIAÇÃO DE DESEMPENHO DOS ALGORITMOS DE ESCALONAMENTO

Este capítulo contém quatro seções onde será apresentada uma possível forma de avaliar manualmente (para fins didáticos) o desempenho dos algoritmos de escalonamento tratados no capítulo anterior. Na primeira seção são apresentados aspectos teóricos sobre pontos importantes na definição de um método de avaliação de desempenho de escalonamento como carga de trabalho, possíveis modelos de avaliação e questões sobre os critérios de avaliação. Na segunda seção são apresentados detalhes sobre o método de avaliação utilizado, justificando tais escolhas. Na terceira seção são apresentados exemplos do comportamento dos algoritmos de escalonamento utilizando o método definido, e na quarta e última seção os valores obtidos em cada um dos critérios de avaliação são comparados e apresentados graficamente.

4.1. ASPECTOS DO MÉTODO DE AVALIAÇÃO

Foram tratadas as características de seis abordagens de escalonamento: Primeiro a Chegar Primeiro a ser Servido, Menor Primeiro, Circular, Baseado em Prioridade, Múltiplas Filas e Múltiplas Filas com Realimentação, apresentando quase que de forma evolutiva, possíveis formas de compartilhamento do processador. Cada um dos algoritmos possui complexidades diferentes e trata de situações específicas. Assim a adequação de cada abordagem a um determinado tipo de ambiente deve ser analisada. Então, qual dos algoritmos de escalonamento é mais adequado para determinada situação? Qual apresenta melhor desempenho?

Para comparar o desempenho de cada um dos algoritmos de escalonamento, inicialmente serão definidos três pontos envolvidos, por entender-se que são essenciais para realizar a avaliação e escolha de qual algoritmo utilizar:

- Modelo de avaliação,
- Dados de entrada e
- Critérios de avaliação (ou medidas de eficiência).

CAPÍTULO 4 - AVALIAÇÃO DE DESEMPENHO

DOS ALGORITMOS DE ESCALONAMENTO

Neste capítulo corremos quatro seções onde são apresentadas algumas formas de avaliar manualmente (para fins didáticos) o desempenho dos algoritmos de escalonamento tratados no capítulo anterior. Na primeira seção são apresentados aspectos técnicos sobre pontos importantes na definição de um método de avaliação de desempenho de escalonamento como carga de trabalho, possíveis modelos de avaliação e questões sobre os critérios de avaliação. Na segunda seção são apresentados detalhes sobre o método de avaliação utilizado, justificando tais escolhas. Na terceira seção são apresentados exemplos de comportamento dos algoritmos de escalonamento utilizando o método definido, e na quarta e última seção os valores obtidos em cada um dos critérios de avaliação são comparados e apresentados graficamente.

4.1. ASPECTOS DO MÉTODO DE AVALIAÇÃO

Foram tratadas as características de seis abordagens de escalonamento: Primeiro a Chegar Primeiro a ser Servido, Menor Primeiro, Circular, Baseado em Prioridade, Múltiplas Filas e Múltiplas Filas com Realimentação, apresentando duas de formas evolutivas, possíveis formas de compartilhamento do processador. Cada um dos algoritmos possui complexidades diferentes e trata de situações específicas. Assim a avaliação de cada abordagem a um determinado tipo de ambiente deve ser analisada. Então, qual dos algoritmos de escalonamento é mais adequado para determinada situação? Qual apresenta melhor desempenho?

Para comparar o desempenho de cada um dos algoritmos de escalonamento inicialmente serão definidos três pontos em/olvidos, por entender-se que são essenciais para realizar a avaliação e escolha de qual algoritmo utilizar:

- Modelo de avaliação
- Dados de entrada
- Critérios de avaliação (ou medidas de eficiência)

4.1.1. Modelos de Avaliação

ALBUQUERQUE (1990, p.51) afirma que comparar diversos algoritmos de escalonamento é complexo devido a *característica de aleatoriedade associada ao comportamento dos processos*. Os métodos destinados a realizar a comparação entre os diversos algoritmos de escalonamento são classificados como:

- Modelo Analítico: modelagem determinística,
- Modelo de Filas: teoria de filas,
- Modelo de Simulação e,
- Modelo de Implementação.

Nos métodos de avaliação analítica, uma determinada carga de trabalho é submetida a cada um dos algoritmos *para produzir uma fórmula ou número que expresse o desempenho do algoritmo para aquela carga de trabalho* (SILBERSCHATZ & GALVIN, 2000, p.169). Um dos tipos de avaliação analítica é a chamada modelagem determinística, onde uma carga de trabalho predeterminada é considerada e calcula-se o desempenho de cada algoritmo para aquela carga de trabalho. Essa modelagem é simples e rápida, fornecendo números exatos, o que facilita a comparação entre os algoritmos analisados. Porém, como são requeridos números exatos dados como entrada, sua aplicação se restringe a esse caso. É interessante notar que nos casos em que os mesmos processos são executados repetidamente e os requisitos necessários para o processamento dos mesmos podem ser medidos de forma exata. Assim, sua aplicação a um conjunto de exemplos pode indicar tendências a serem analisadas separadamente para comprovação.

Um ramo da estatística chamado de teoria das filas trata do estudo da espera em filas. A aleatoriedade característica dos ambientes computacionais do ponto de vista do sistema, pode ser tratada com ferramentas matemáticas apropriadas para tratar eventos aleatórios como estes. No modelo de filas o sistema computacional é descrito como um conjunto de clientes (aqueles que esperam por determinado serviço) e servidores (aqueles que fornecem um determinado serviço); assim, a CPU e os dispositivos de E/S são servidores, as LCP em cada um dos estados são filas de espera pelos serviços oferecidos pelos servidores e os processos são os clientes.

Através da teoria das filas é possível definir o ambiente matematicamente, após estudá-lo exaustivamente, determinando a distribuição do uso da CPU e dos

dispositivos de E/S. Desta forma, essas distribuições podem ser medidas e então usadas em aproximações ou estatísticas. Segundo SHAY (1996), as fórmulas permitem dispor de respostas prováveis baseadas em informações observadas do sistema como: distribuição dos tempos nos quais os processos ficam prontos para serem executados (distribuição dos tempos de chegada), probabilidade da CPU ser usada durante um dado intervalo de tempo (taxa de utilização), tamanho médio das filas, taxa média de entrada de novos processos, etc.

As limitações dessa abordagem são principalmente porque as distribuições dos tempos de chegada e uso de dispositivos de E/S são sempre matematicamente tratáveis, mas não necessariamente realistas. Assim, com esse modelo pode-se obter uma aproximação do mundo real, mas a precisão obtida nos resultados não é inteiramente confiável.

Nos métodos baseados em simulação são implementados modelos que representem os elementos do sistema a ser simulado. Um simulador é idealizado para ter um comportamento mais próximo possível de um sistema real (SHAY, 1996). São então gerados dados de entrada, submetidos ao modelo simulador e são coletados valores de saída para determinar o desempenho do algoritmo que as manipulou. Segundo SILBERSCHATZ & GALVIN (2000), a geração dos dados de entrada pode ser realizada de duas maneiras:

- Através de gerador de números aleatórios,
- Através da produção de fitas de rastreamento.

O método mais comum na geração dos dados de entrada para uso de simuladores é através de um gerador (manual ou automatizado) de números aleatórios de acordo com as distribuições de probabilidade, para gerar a duração de fases de uso de CPU, tempos de chegada, término de atendimento de E/S, etc. Essas distribuições podem ser definidas matematicamente⁴⁸ ou empiricamente. Numa distribuição definida empiricamente são feitas medidas relativas ao sistema, onde os resultados são utilizados como entrada na simulação. Já as fitas de rastreamento são criadas através da gravação e armazenamento do monitoramento do sistema real, de acordo com a ocorrência dos eventos reais. A precisão é uma característica dessa forma de gerar a carga de trabalho.

48 SHAY (1996, p.519-539) apresenta detalhadamente os processos de Poisson, distribuição exponencial de tempos de chegada, derivação da distribuição exponencial, taxas de nascimento e morte da teoria das filas para definir-se matematicamente entradas para simulação de escalonadores.

Assim como na modelagem determinística, o modelo de simulação também resulta números exatos, o que facilita a comparação entre os algoritmos analisados. As limitações do modelo de simulação dizem respeito aos custos relacionados ao processamento da simulação de cada um dos algoritmos que se pretende comparar, a geração e armazenamento dos arquivos de fitas de rastreamento (se for o caso), e principalmente pelos esforços relacionados ao desenvolvimento do sistema simulador de cada um dos algoritmos de escalonamento.

SILBERSCHATZ & GALVIN (2000) consideram a implementação o único método completamente preciso de avaliação dos algoritmos de escalonamento, pois essa abordagem coloca o algoritmo de escalonamento a ser analisado no sistema real para avaliação sob condições reais de operação. De todos os métodos de avaliação de algoritmos, sem dúvida esse é o mais confiável. Porém, as implicações de implementar um algoritmo de escalonamento apenas iniciam com as dificuldades em modificar partes do SO para cada um dos algoritmos de escalonamento que se pretende avaliar, dada a complexidade do *software*. Além dos custos envolvidos nas alterações, há também os aspectos que envolvem a reação negativa de usuários de um sistema em produção frente as mudanças constantes no SO, que afeta o desempenho do sistema pela forma com que os diferentes algoritmos de escalonamento tratam os processos. Fatos como esses influenciam diretamente na inviabilidade dessa abordagem, apesar de seus aspectos técnicos positivos.

4.1.2. Dados de Entrada

Os dados de entrada (ou carga de trabalho) impostos a um sistema para realizar avaliação de desempenho devem, na medida do possível, representar a maioria dos casos que podem ser encontrados num ambiente real. Podem ser gerados por números aleatórios definidos matematicamente ou definidos de forma empírica, e dependem essencialmente do modelo de avaliação adotado.

4.1.3. Critérios de avaliação

Os critérios de avaliação (também chamados de medidas de eficiência) influenciam de maneira significativa na determinação do desempenho dos algoritmos de escalonamento. Esses critérios podem ser relativos ao sistema como um todo ou indicadores de desempenho processo a processo.

SILBERSCHATZ & GALVIN (2000, p.151-2) definem que independente do ambiente que se pretende analisar, geralmente os critérios de avaliação de algoritmos de escalonamento tratam de aspectos que objetivam:

1. Maximizar a utilização de CPU,
2. Maximizar a produtividade (número de processos finalizados por unidade de tempo),
3. Minimizar o tempo de processamento por processo,
4. Minimizar o tempo de espera na(s) LCP prontos e,
5. Minimizar o tempo de resposta por processo.

Para determinar que um algoritmo de escalonamento apresenta melhor desempenho que outro, deve-se primeiramente definir a importância relativa desses critérios. Se no ambiente predominam processos de usuários interativos, os aspectos identificados acima, como 3, 4 e 5, são extremamente importantes. Já quando predominam processos *batch*, o ponto 5 não é crucial.

Num ambiente heterogêneo (que é o mais próximo do real) deve haver um balanceamento na utilização da CPU pelos processos, para que não ocorram situações de *starvation*, e sejam alcançados os objetivos enumerados acima em conjunto. Desta forma, normalmente são definidas metas a serem tratadas como premissas e, a partir delas, definem-se os critérios do que se deseja avaliar. Por exemplo, considerando uma determinada premissa tida como essencial num contexto: todos os processos interativos devem ter tempo de resposta inferior a 5 ms, serão definidos os critérios para que se possa avaliar como cada um dos algoritmos de escalonamento se comporta, e qual deles melhor atende à(s) premissa(s) formulada(s).

4.1.3. Critérios de avaliação

Os critérios de avaliação (também chamados de medidas de eficiência) influenciam de maneira significativa na determinação do desempenho dos algoritmos de escalonamento. Esses critérios podem ser relativos ao sistema como um todo ou indicadores de desempenho processo a processo.

SILBERSCHATZ & GALVIN (2000, p 121-2) definem que independentemente do ambiente das se pretende analisar, geralmente os critérios de avaliação de algoritmos de escalonamento tratam de aspectos que objetivam:

1. Maximizar a utilização de CPU;
2. Maximizar a produtividade (número de processos finalizados por unidade de tempo);
3. Minimizar o tempo de processamento por processo;
4. Minimizar o tempo de espera nas(s) TCP pronto(s);
5. Minimizar o tempo de resposta por processo.

Para determinar que um algoritmo de escalonamento apresenta melhor desempenho que outro, deve-se primeiramente definir a importância relativa desses critérios. Se no ambiente predominantemente processos de usuários interativos, os aspectos identificados acima, como 3, 4 e 5, são extremamente importantes, já quando predominantemente processos batch, o ponto 2 não é crucial.

Num ambiente heterogêneo (que é o mais próximo do real) deve haver um planejamento na utilização de CPU pelos processos, para que não ocorram situações de starvation e sejam alcançados os objetivos enunciados acima em conjunto. Desta forma, normalmente são definidas metas a serem atingidas como premissas e a partir delas, definem-se os critérios do que se deseja atingir. Por exemplo, considerando uma determinada premissa não essencial num contexto, todos os processos interativos devem ter tempo de resposta inferior a 2 ms, estão definidos os critérios para que se possa avaliar como cada um dos algoritmos de escalonamento se comporta e qual deles

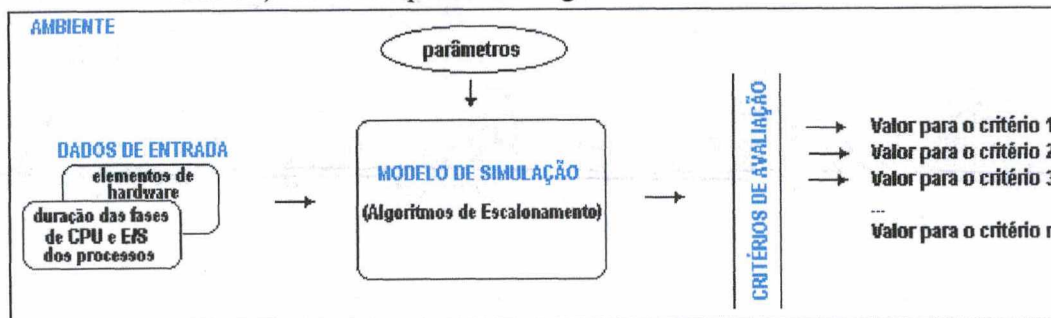
melhor atende a(s) premissa(s) formulada(s).

4.2. MÉTODO UTILIZADO PARA AVALIAR O DESEMPENHO DOS ALGORITMOS DE ESCALONAMENTO

Nesta seção são definidos aspectos do ambiente onde serão apresentados exemplos (executados de forma manual) de simulações comparativas de desempenho dos algoritmos de escalonamento. Para formular esses exemplos considerou-se (Fig. 4.1):

- O modelo de avaliação adotado,
- A definição dos dados de entrada,
- As características do ambiente,
- Os critérios de avaliação de desempenho.

Figura 4.1: Aspectos considerados nas simulações comparativas (realizadas de forma manual) do desempenho dos algoritmos de escalonamento.



4.2.1. Modelo de Avaliação

O modelo de avaliação adotado será o modelo de simulação. É considerado o comportamento dos algoritmos de escalonamento no atendimento aos processos conforme tratado na seção 3.6.3, atentando para os parâmetros informados quando for o caso (por exemplo, valor de quantum).

4.2.2. Dados de entrada

Para a simulação do escalonamento dos processos foram gerados valores numéricos hipotéticos definidos de forma aleatória, que definem a priori:

- A duração das fases de uso de CPU e a duração das fases de uso de dispositivos de E/S dos processos de usuário,
- A duração dos tempos de sistema (do processo de SO) para que sejam realizadas as mudanças de contexto entre os processos de usuário e as rotinas envolvidas na conclusão dos processos de usuário,
- Os tempos de chegada dos processos de usuário.

A duração das fases de uso de CPU e a duração das fases de uso de dispositivos de E/S dos processos foram definidas a fim de representar três tipos de processos quanto ao uso dos recursos processador e dispositivos de E/S: *CPU-bound*, *I/O-bound*⁴⁹ e um terceiro tipo que será denominado balanceado. Assim, não haverá predominância de nenhum tipo de processo. Serão utilizados em todos os diferentes algoritmos de escalonamento analisados os mesmos dados de entrada, a fim de facilitar a comparação dos resultados obtidos.

Da mesma forma, os dados de entrada da simulação de *deadlock* são valores aleatórios que definem:

- Quais os dispositivos que terão suas requisições e alocações simuladas e,
- A quantidade desses dispositivos existentes no ambiente.

4.2.3. O Ambiente

O ambiente criado para simular o atendimento dos processos de acordo com cada um dos métodos de escalonamento será extremamente simples se comparado a um ambiente real, mas considerando os propósitos desse trabalho, o contexto definido é adequado.

A abrangência desse contexto restringe-se a definição de um modelo de processo, ao escalonamento do processador e a detecção de *deadlock*, e desta forma, entende-se que aspectos relacionados à alocação do processo em memória (virtual e

⁴⁹ Os processos do tipo *CPU-bound* passam a maior parte do tempo no estado de execução, ou seja, utilizando o processador. Esse tipo de processo realiza poucas operações de E/S, sendo típico de aplicações matemáticas e/ou científicas. Já os processos do tipo *I/O-bound* passam a maior parte do tempo no estado de bloqueado, pois realiza um elevado número de operações de E/S. Esse tipo de processo é encontrado em aplicações comerciais, que se baseiam em leitura, processamento e gravação, assim como os processos interativos, pela forma de comunicação entre o usuário e o sistema, normalmente lenta, devido a utilização de terminais. (MACHADO & MAIA, 1992, p.59-60).

principal), comunicação com dispositivos de E/S, aspectos relacionados a acesso a arquivos, entre dezenas de outros pontos, não estão abrangidos por esses exemplos.

Considera-se que há elementos de *hardware* simulados como: CPU, dispositivos de E/S (considerados como preemptivos e reutilizáveis serialmente), memória principal e área de memória virtual. Também as estruturas mantidas pelo SO para gerenciamentos são consideradas como simuladas: a(s) LCP de cada um dos estados válidos, uma tabela de processos mantendo um registro, isto é, um BCP para cada processo ativo, etc.

A variável fictícia identificada como Tempo Total para execução dos processos é definida para representar o relógio do sistema. À medida que o valor armazenado nessa variável é incrementado, o simulador modifica o estado do sistema para refletir as atividades dos elementos simulados.

4.2.4. Critérios de avaliação

Os critérios de avaliação são identificados como em dois grupos:

- de sistema, e
- de processo.

Os critérios de sistema são aqueles que tratam de valores que representem de uma forma geral o desempenho do ambiente; já os critérios de avaliação de processo são indicativos individuais de performance processo a processo.

Desta forma, são considerados critérios de avaliação de desempenho de sistema:

- **Utilização da CPU** –Esse critério de avaliação contabiliza o total de tempo em que a CPU ficou utilizada, tanto para atendimento de processos de usuário, quanto para atendimento de processos de sistema. O percentual de Utilização da CPU é calculado pela razão entre os valores de Utilização da CPU e Tempo Total para execução dos processos.
- **Ociosidade da CPU** – Indica quanto tempo a CPU ficou ociosa, isto é, sem executar processos (enquanto há processos ativos). O valor de Ociosidade da CPU é calculado pela diferença entre os valores de Tempo Total para execução dos processos e Tempo de Utilização da CPU. O percentual de Ociosidade da CPU pode ser calculada por $(1 - \text{Percentual de Utilização da CPU})$.

principal) comunicação com dispositivos de E/S, aspectos relacionados a acesso a arquivos, entre outras de outros pontos, não estão abordados por estes exemplos.

Considera-se que há elementos de hardware simulados como CPU, dispositivos de E/S (considerados como dispositivos e recursos estruturais) memória principal e área de memória virtual). Também as estruturas mantidas pelo SO para gerenciamento são consideradas como simuladas. A(s) TCP de cada um dos estados variáveis, uma tabela de processos mantendo um registro, isto é, um BCP para cada processo ativo, etc.

A variável fictícia identificada como Tempo Total para execução dos processos é definida para representar o relógio do sistema. A medida que o valor aumentado nessa variável é incrementado, o simulador modifica o estado do sistema para refletir as atividades dos elementos simulados.

4.2.4. Critérios de avaliação

Os critérios de avaliação são identificados como em dois tipos:

- de sistema;
- de processo.

Os critérios de sistema são aqueles que tratam de valores que representam de uma forma geral o desempenho do ambiente, já os critérios de avaliação de processo são indicadores individuais de performance processo a processo.

Desta forma, são considerados critérios de avaliação de desempenho de sistemas:

- Utilização da CPU - Esse critério de avaliação contabiliza o total de tempo em que a CPU ficou utilizada, isto para arquivamento de processos de usuário, quanto para arquivamento de processos de sistema. O percentual de Utilização da CPU é calculado pela razão entre os valores de Utilização da CPU e Tempo Total para execução dos processos.

- Ociosidade da CPU - indica quanto tempo a CPU ficou ociosa, isto é, sem executar processos (quando há processos ativos). O valor de Ociosidade da CPU é calculado pela diferença entre os valores de Tempo Total para execução dos processos e Tempo de Utilização da CPU. O percentual de Ociosidade da CPU pode ser calculada por (1 - Percentual de Utilização da CPU).

(CPU)

- **Utilização da CPU pelo SO** – somatório do tempo que o processador foi utilizado pelo SO (especificamente para realizar rotinas que envolvam mudança de contexto e conclusão de processos de usuário). É também especificado o número de mudanças de contexto ocorridas, incluindo a primeira intervenção do SO, que representa a primeira seleção de processo de usuário da LCP prontos.
- **Produtividade da CPU** – indica o número de processos de usuário concluídos a cada intervalo de tempo pré-definido. Num ambiente com processos relativamente pequenos essa taxa tende a ser alta.
- **Grau de Multiprogramação** - quantidade de processos de usuário ativos com execução parcial ou total a cada intervalo de tempo pré-definido. Nos ambientes de monoprogramação, o valor varia entre 1 e 0 (considerando apenas processos de usuário).

Os critérios analisados por processo de usuário são explicados a seguir:

- **Tempo de Processamento** (*turnaround time*) - Mede o tempo total de atendimento de cada processo, desde o momento de sua submissão até sua conclusão. Inclui tempo de espera para receber recursos, tempo de espera na LCP prontos, tempo de execução na CPU e tempo na(s) LCP bloqueado.
- **Tempo de Resposta** (*response time*) – Um processo de usuário interativo pode ser atendido parcialmente e produzir uma saída também parcial. Para o usuário, isso significa que ele está sendo atendido (ainda que não completamente). Esse parâmetro indica o tempo necessário para que um processo tenha o primeiro atendimento de CPU, isto é, o tempo que o processo ficou em espera antes de sua primeira execução.
- **Tempo de Bloqueio** - Objetiva mostrar quanto tempo o processo passa na(s) LCP bloqueados, isto é, indica o somatório de tempo que cada processo fica aguardando por algum evento que envolva operação de E/S. Vale ressaltar que o escalonador não tem influência sobre esse critério, servindo apenas para medir o atendimento do processo.
- **Tempo de Espera** (*waiting time*) – Objetiva mostrar quanto tempo o processo passa na LCP prontos, isto é, indica o somatório de tempo que cada

Utilização da CPU pelo SO - somatório do tempo que o processador foi utilizado pelo SO (especificamente para realizar tarefas que envolvam mudanças de contexto e conclusão de processos de usuário). É também especificado o número de mudanças de contexto ocorridas, incluindo a primeira intervenção do SO, das respostas a primeira seleção de processo de usuário da LCP prontas.

Produtividade da CPU - indica o número de processos de usuário concluídos a cada intervalo de tempo pré-definido. Num ambiente com processos relativamente poucos essa taxa tende a ser alta.

Grav de Multiprogramação - quantidade de processos de usuário ativos com exceção parcial ou total a cada intervalo de tempo pré-definido. Nos ambientes de multiprogramação, o valor varia entre 1 e 0 (considerando apenas processos de usuário).

Os critérios analisados por processo de usuário de usuário são explicados a seguir:

Tempo de Processamento (turnaround time) - Mede o tempo total de atendimento de cada processo, desde o momento de sua submissão até sua conclusão. Inclui tempo de espera para receber recursos, tempo de espera na LCP prontas, tempo de execução na CPU e tempo nas (s) LCP bloqueadas.

Tempo de Resposta (response time) - Um processo de usuário interativo pode ser atendido parcialmente e produzir uma saída também parcial. Para o usuário, isso significa que ele está sendo atendido (ainda que não completamente). Esse período mede o tempo necessário para que um processo tenha o primeiro atendimento de CPU. Isto é, o tempo que o processo fica em espera antes de sua primeira execução.

Tempo de Bloqueio - Objeto a mostrar quanto tempo o processo passa nas LCP bloqueadas, isto é, indica o somatório de tempo que cada processo fica aguardando por algum evento que envolva operação de E/S. Vale ressaltar que o escalonador não tem influência sobre esse critério, servindo apenas para medir o atendimento do processo.

Tempo de Espera (waiting time) - Objeto a mostrar quanto tempo o processo passa na LCP prontas, isto é, indica o somatório de tempo que cada

processo fica aguardando por execução na fila de processos prontos. Inclui o tempo de resposta daquele processo, mas não inclui o(s) tempo(s) de bloqueio.

4.3. EXEMPLOS DE ESCALONAMENTO

Definidos os aspectos relativos ao método adotado para comparação de desempenho dos algoritmos de escalonamento, serão definidos exemplos do atendimento aos processos de acordo com algumas das abordagens de escalonamento. Assim, o objetivo desta seção é apresentar esquemas que representem como, ao longo do tempo, processos de usuário seriam atendidos por um SO que implementasse determinado Escalonador de CPU. Desses esquemas produzidos serão calculados os valores dos critérios de avaliação de desempenho citados na seção seguinte. Com isso, é possível comparar o desempenho de cada uma das abordagens tratadas. A fim de facilitar a comparação das medidas de eficiência, em todos os exemplos será considerada a mesma carga de trabalho composta de três processos de usuário. Na Tabela 4.1 são apresentadas as fases de uso de CPU e de dispositivos de E/S para cada um dos três processos de usuário, identificados como A, B e C, submetidos nessa ordem, simultaneamente, a partir do momento 0 (zero).

Tabela 4.1 – Duração das fases de uso de CPU e das fases de uso de dispositivos de E/S dos processos de usuário usados nos exemplos.

	Processo A	Processo B	Processo C
Fases de Tempos de uso de CPU e de Entrada/Saída	10 UT processando 4 UT aguardando Entrada 6 UT processando	5 UT processando	1 UT processando 6 UT aguardando Entrada 3 UT processando 3 UT aguardando Saída 2 UT processando
Ordem de entrada na LCP prontos	1º	2º	3º
Prioridade	9	0	4

A seguir são relacionadas algumas suposições consideradas nos exemplos que ilustram o comportamento dos algoritmos de escalonamento. É suposto que...

- ... o ambiente é de monoprocessamento e o SO implementa multiprogramação.
- ... a unidade de tempo tratada será fictícia e identificada como UT (unidade de tempo).
- ... se conheça a priori as fases de uso de CPU com a duração de cada uma dessas fases para cada um dos processos de usuário (trata-se de uma parte dos dados de entrada).
- ... também são conhecidas as fases em que o processo está no estado de bloqueado e o tempo que o dispositivo de E/S leva para atender a esse processo (em cada uma de suas fases de bloqueio).
- ... haja um único processo de SO, que realiza especificamente as rotinas relativas à mudança de contexto e conclusão de processo, sendo que para cada mudança de contexto é necessário 1 UT e para atender um processo que chegue ao estado de concluído, também é necessário 1 UT.
- ... o processo de SO nunca necessite realizar uma operação de E/S para si próprio.
- ... para as medidas de eficiência que utilizam Intervalos de Tempo, deve-se considerar intervalos de 10 UT cada.
- ... nunca são formadas filas para utilizar os serviços de E/S.
- ... nunca acontece situação de *deadlock*.

Os exemplos de escalonamento apresentados a seguir englobam as seguintes abordagens:

- Abordagens não-preemptivas:
 - PCPS e
 - MP.
- Abordagens preemptivas:
 - Circular com critério de seleção por prioridade,
 - Circular com critério de seleção PCPS,
 - Circular com critério de seleção MP e,
 - MFR com três LCPP.

Tabela 4.6 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento MP.

	Intervalos de Tempo			
	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40
Produtividade da CPU	1	0	1	1
Grau de Multiprogramação	2	2	2	1

Tabela 4.7 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento MP.

	Processos de Usuário		
	A	B	C
Tempo de Processamento	35	08	27
Tempo de Resposta	14	03	01
Tempo de Bloqueio	04	-	09
Tempo de Espera	15	03	12

4.3.3. Exemplo de Escalonamento Circular

O algoritmo de escalonamento utilizado é o Circular (Round Robin) com abordagem preemptiva, onde há revezamento na utilização da CPU pelos processos, havendo um tempo limite único para todos os processos na utilização da CPU de forma contínua, tempo esse denominado de *quantum* (ou *time slice*) com valor de 4 UT. Considera-se ainda que a LCP no estado de pronto é única.

São apresentadas três possíveis abordagens para o exemplo de escalonamento circular onde varia-se o critério de seleção dos processos da LCP prontos:

- Escalonamento Circular com seleção baseada em prioridade,
- Escalonamento Circular com seleção PCPS e,
- Escalonamento Circular com seleção MP.

4.3.3.1. Exemplo de Escalonamento Circular com seleção baseada em prioridade

Sobre o critério de seleção baseado em prioridade, considera-se que:

- A prioridade atribuída aos processos é estática.
- As prioridades válidas estão no intervalo de 0 a 9 (inclusive).
- Quanto maior é o valor numérico atribuído a prioridade de um processo, mais privilegiado ele será no algoritmo de escalonamento.

Na Fig. 4.4 é apresentado um esquema que representa como o atendimento aos processos A, B e C ocorre seguindo os princípios do algoritmo de escalonamento Circular com seleção baseada em prioridade. As Tabelas 4.8, 4.9 e 4.10 mostram os valores numéricos resultantes das observações feitas no esquema.

Figura 4.4 - Esquema de atendimento dos processos no escalonamento Circular com seleção baseada em prioridade.

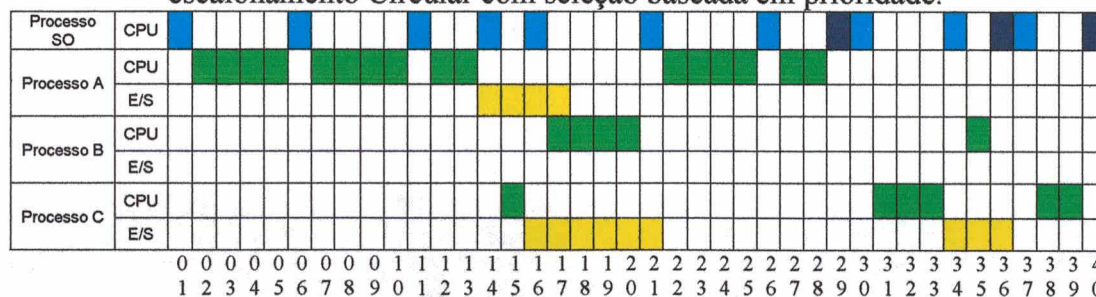


Tabela 4.8 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.

	Quantidade de UT	Percentual
Tempo Total para execução dos processos	40 UT	-
Utilização da CPU	40 UT	100%
Ociosidade da CPU	0 UT	0%
Utilização da CPU pelo SO	13 UT	32,5%

Tabela 4.9 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.

	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40
Produtividade da CPU	0	0	1	2
Grau de Multiprogramação	1	3	3	2

Tabela 4.10 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção baseada em prioridade.

	Processos de Usuário		
	A	B	C
Tempo de Processamento	28	35	39
Tempo de Resposta	01	16	14
Tempo de Bloqueio	04	-	09
Tempo de Espera	08	30	24

4.3.3.2. Exemplo de Escalonamento Circular com seleção PCPS

Na Fig. 4.5 é apresentado um esquema que representa como o atendimento aos processos A, B e C ocorre seguindo os princípios do algoritmo de escalonamento Circular com seleção Primeiro a chegar, Primeiro a ser servido. As Tabelas 4.11, 4.12 e 4.13 mostram os valores numéricos resultantes das observações feitas no esquema.

Figura 4.5 - Esquema de atendimento dos processos no escalonamento Circular com seleção PCPS.

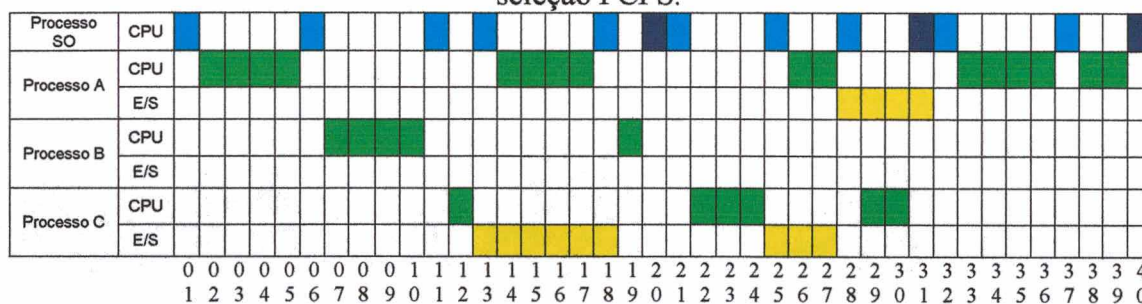


Tabela 4.11 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção PCPS.

	Quantidade de UT	Percentual
Tempo Total para execução dos processos	40 UT	-
Utilização da CPU	40 UT	100%
Ociosidade da CPU	0 UT	0%
Utilização da CPU pelo SO	13 UT	32,50%

Tabela 4.12 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção PCPS.

	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40
Produtividade da CPU	0	1	0	2
Grau de Multiprogramação	2	3	2	1

Tabela 4.13 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção PCPS.

	Processos de Usuário		
	A	B	C
Tempo de Processamento	39	19	30
Tempo de Resposta	01	06	11
Tempo de Bloqueio	04	-	09
Tempo de Espera	19	14	15

4.3.3.3. Exemplo de Escalonamento Circular com seleção MP

Na Fig. 4.6 é apresentado um esquema que representa como o atendimento aos processos A, B e C ocorre seguindo os princípios do algoritmo de escalonamento Circular com seleção Menor Primeiro. As Tabelas 4.14, 4.15 e 4.16 mostram os valores numéricos resultantes das observações feitas no esquema.

Figura 4.6 - Esquema de atendimento dos processos no escalonamento Circular com seleção MP.

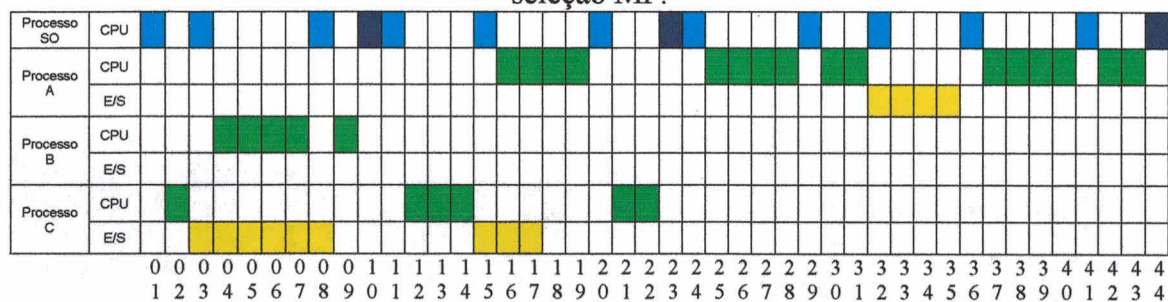


Tabela 4.14 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento Circular com seleção MP.

	Quantidade de UT	Percentual
Tempo Total para execução dos processos	44 UT	-
Utilização da CPU	41 UT	93,18%
Ociosidade da CPU	3 UT	6,82%
Utilização da CPU pelo SO	14 UT	34,14%

Tabela 4.15 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento Circular com seleção MP.

	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40	UT 41 a 50
Produtividade da CPU	1	0	1	0	1
Grau de Multiprogramação	2	2	2	1	1

Tabela 4.16 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento Circular com seleção MP.

	Processos de Usuário		
	A	B	C
Tempo de Processamento	43	09	22
Tempo de Resposta	15	03	01
Tempo de Bloqueio	04	-	09
Tempo de Espera	23	04	07

4.3.4. Exemplo de Escalonamento MFR

O algoritmo de escalonamento utilizado é de Múltiplas Filas com Realimentação (preemptivo), onde:

- São definidas três LCP prontos (LCPP) identificadas como 1, 2 e 3.
- A LCPP 1 recebe os processos com prioridade no intervalo de 7 a 9 (inclusive), a LCPP 2 recebe os processos com prioridade no intervalo de 4 a 6 (inclusive), e a LCPP 3 recebe os processos com prioridade no intervalo de 0 a 3 (inclusive).
- A prioridade 9 é considerada a mais alta.
- O quantum das LCPP 1, 2 e 3 é, respectivamente de 2, 4 e 6 UT.
- O critério de seleção de cada uma das LCPP será PCPS.
- A transição entre as LCPP é dinâmica, baseado no seguinte:
 - um processo com quantum esgotado, desce um degrau na prioridade (se não for a última implementada), e é encaixado na LCPP correspondente, e,
 - um processo que foi bloqueado, quando voltar ao estado de pronto é encaixado na LCPP imediatamente superior a que estava quando houve o bloqueio (se não for a primeira).
- As LCPP receberão os processos de acordo com as prioridades a eles definidas;

Na Fig. 4.7 é apresentado um esquema que representa como o atendimento aos processos A, B e C ocorre seguindo os princípios do algoritmo de escalonamento de Múltiplas Filas com Realimentação. As Tabelas 4.17, 4.18 e 4.19 mostram os valores numéricos resultantes das observações feitas no esquema.

Figura 4.7 - Esquema de atendimento dos processos no escalonamento MFR.

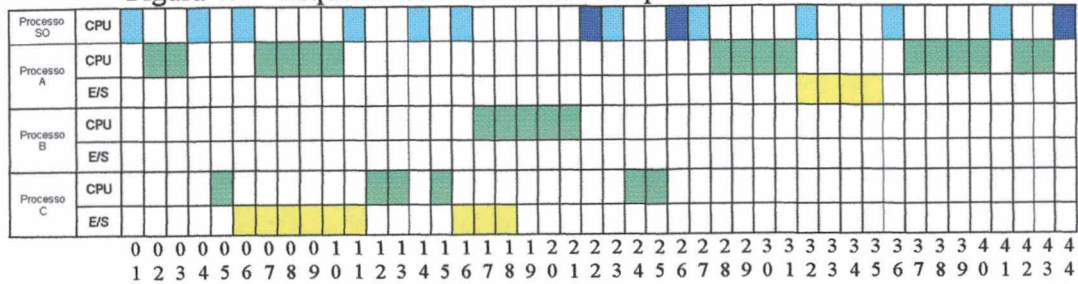


Tabela 4.17 – Valor numérico e percentual dos critérios de sistema resultantes do exemplo de escalonamento MFR.

	Quantidade de UT	Percentual
Tempo Total para execução dos processos	44 UT	-
Utilização da CPU	41 UT	93,18%
Ociosidade da CPU	3 UT	6,82%
Utilização da CPU pelo SO	14 UT	34,14%

Tabela 4.18 – Valor numérico dos critérios de sistema que utilizam intervalo de tempo resultantes do exemplo de escalonamento MFR.

	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40	UT 41 a 50
Produtividade da CPU	0	0	2	0	1
Grau de Multiprogramação	2	3	3	1	1

Tabela 4.19 – Valor numérico dos critérios de processo resultantes do exemplo de escalonamento MFR.

	Processos de Usuário		
	A	B	C
Tempo de Processamento	43	21	25
Tempo de Resposta	01	16	04
Tempo de Bloqueio	04	-	09
Tempo de Espera	23	16	10

4.4. ANÁLISE COMPARATIVA DOS RESULTADOS

Nesta seção serão realizadas comparações dos resultados obtidos nos critérios de desempenho na simulação manual do atendimento aos processos usados como exemplo (apresentados na Tabela 4.1) segundo cada abordagem de escalonamento tratada. Como a carga de trabalho é pequena, essa análise servirá apenas como referência para próximos estudos, não refletindo reais potencialidades e limitações das abordagens de escalonamento apresentadas.

Essa seção está dividida em duas partes, sendo que na primeira são analisados os critérios de desempenho de sistema, a saber:

- Utilização da CPU,
- Ociosidade da CPU,
- Utilização da CPU pelo SO,
- Produtividade da CPU, e
- Grau de Multiprogramação.

Na segunda parte são analisados os critérios por processo de usuário, abrangendo:

- Tempo de Processamento,
- Tempo de Resposta, e
- Tempo de Espera.

4.4.1. Critérios de avaliação de desempenho de sistema

A Tabela 4.20 apresenta o resumo dos valores obtidos nos critérios de avaliação de desempenho de sistema.

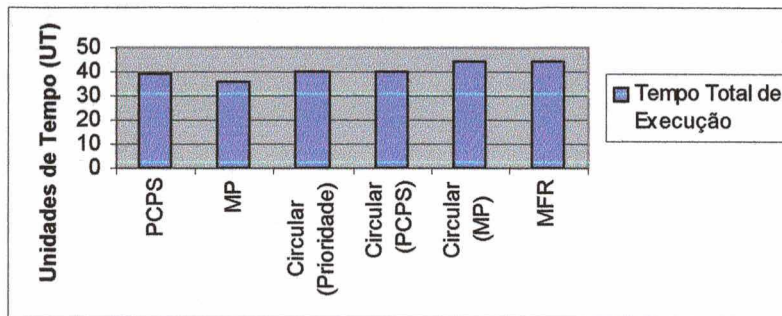
Tabela 4.20 – Resumo dos valores dos critérios de avaliação de desempenho de sistema obtidos nos exemplos de escalonamento.

	Tempo Total para execução dos processos	Utilização da CPU	Ociosidade da CPU	Utilização da CPU pelo SO
Escalonamento Primeiro a chegar, primeiro a ser servido	39	37	02	10
Escalonamento Menor Primeiro	36	36	0	09
Escalonamento Circular com seleção baseada em prioridade	40	40	0	13
Escalonamento Circular com seleção Primeiro a chegar, primeiro a ser servido	40	40	0	13
Escalonamento Circular com seleção Menor Primeiro	44	41	03	14
Escalonamento Múltiplas Filas com Realimentação	44	41	03	14

O Gráfico 4.1 compara o tempo total de atendimento de cada uma das abordagens analisadas. Nota-se que as abordagens Circular com seleção MP e MFR

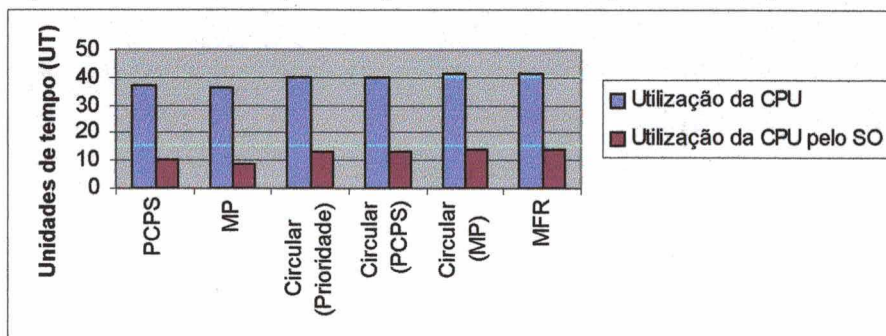
apresentaram os maiores tempos de execução, e portanto a pior performance nesse critério. O melhor desempenho nesse critério foi do escalonador MP (36 UT).

Gráfico 4.1 – Comparação entre os valores de Tempo Total de Execução obtidos nas abordagens de escalonamento simuladas manualmente.



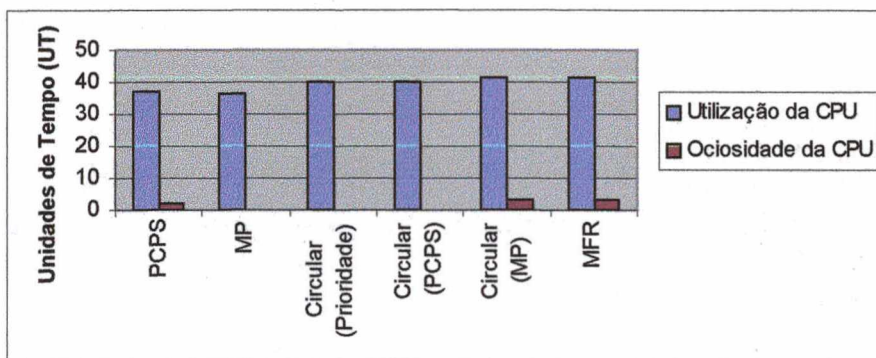
SILBERSCHATZ & GALVIN (2000, p.121) afirmam que os tempos gastos pelo SO na CPU *são trabalhos adicionais requeridos pelo próprio sistema e é altamente dependente do suporte de hardware*. O Gráfico 4.2 apresenta a relação entre dois critérios de desempenho de sistema: Utilização da CPU e Utilização da CPU pelo SO. Sabe-se que o total de tempo de CPU utilizado para atender aos processos de usuário é fixo e corresponde a 27 UT em todas as abordagens (já que a carga de trabalho analisada em todas as abordagens foi a mesma), e assim, a comparação entre esses dois critérios objetiva mostrar quanto do tempo total de CPU utilizado de acordo com cada abordagem foi para simular a execução de rotinas do SO (nos exemplos exclusivamente para executar rotinas de mudanças de contexto e conclusão de processos). Os maiores valores percentuais de tempos de CPU utilizado para realizar rotinas de SO ocorreram nas abordagens Circular com seleção MP e MFR (34,14%).

Gráfico 4.2 – Resultados obtidos nos critérios Tempo total de utilização da CPU (por processo de usuário) e Tempo total de utilização da CPU pelo SO.



O Gráfico 4.3 mostra a relação entre os critérios de Tempo de Utilização da CPU e Tempos de Ociosidade de mesma (inatividade). A utilização da CPU pode variar de 0 (*idle*) a 100%, sendo desejável que ele fique ocupada o máximo possível. SILBERSCHATZ & GALVIN (2000, p.151) afirmam que em um sistema real, esse percentual varia entre 40 e 90%. As abordagens MP, Circular com base em prioridade e Circular com seleção PCPS apresentaram Ociosidade de CPU inexistente, enquanto que as abordagens Circular com seleção MP e MFR apresentaram os maiores valores percentuais (6,82%) e portanto a pior performance.

Gráfico 4.3 – Comparação entre os valores de Tempo Total de utilização e Ociosidade da CPU.



A Tabela 4.21 apresenta os valores obtidos nos critérios de avaliação de desempenho de sistema denominado Produtividade da CPU, obtidos nas abordagens de escalonamento tratadas. Por esses dados percebe-se que nas abordagens não-preemptivas, os quatro primeiros intervalos foram suficientes para que os três processos de usuário fossem concluídos, enquanto que em duas das abordagens não-preemptivas (com exceção da abordagem Escalonamento Circular com seleção PCPS e MFR) foram necessários 5 intervalos de tempo. Atribui-se essa situação aos tempos de execução das rotinas do SO, que foram mais freqüentes nas abordagens preemptivas.

Tabela 4.21 – Valores do critério Produtividade da CPU.

	Intervalos de Tempos				
	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40	UT 41 a 50
Escalonamento Primeiro a chegar, primeiro a ser servido	0	1	1	1	0
Escalonamento Menor Primeiro	1	0	1	1	0
Escalonamento Circular com seleção baseada em prioridade	0	0	1	2	0
Escalonamento Circular com seleção Primeiro a chegar, primeiro a ser servido	0	1	0	2	0
Escalonamento Circular com seleção Menor Primeiro	1	0	1	0	1
Escalonamento Múltiplas Filas com Realimentação	0	0	2	0	1

A Tabela 4.22 apresenta os valores obtidos nos critérios de avaliação de desempenho de sistema denominado Grau de Multiprogramação. Percebe-se que nas abordagens preemptivas, a distribuição dos processos ativos por intervalo de tempo é praticamente uniforme. Já nas abordagens não-preemptivas, nota-se que devido a particularidades dos algoritmos de escalonamento PCPS e Circular com seleção baseada em prioridade, o primeiro intervalo de tempo foi utilizado exclusivamente pelo processo A, o que influencia a análise desse critério de avaliação.

Tabela 4.22 – Valores do critério Grau de Multiprogramação.

	Intervalos de Tempos				
	UT 01 a 10	UT 11 a 20	UT 21 a 30	UT 31 a 40	UT 41 a 50
Escalonamento Primeiro a chegar, primeiro a ser servido	1	3	2	1	0
Escalonamento Menor Primeiro	2	2	2	1	0
Escalonamento Circular com seleção baseada em prioridade	1	3	3	2	0
Escalonamento Circular com seleção Primeiro a chegar, primeiro a ser servido	2	3	2	1	0
Escalonamento Circular com seleção Menor Primeiro	2	2	2	1	1
Escalonamento Múltiplas Filas com Realimentação	2	3	3	1	1

4.4.2. Critérios de avaliação de desempenho de processo

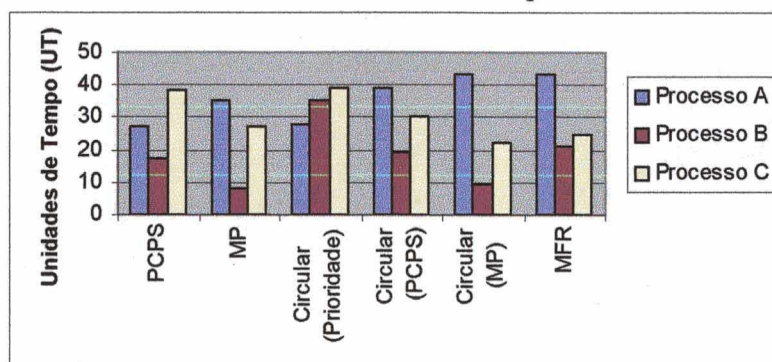
A Tabela 4.23 apresenta os valores do critério de avaliação de desempenho de processo denominado Tempo de Processamento obtidos nas diferentes abordagens de escalonamento. Nesse critério quanto maior o valor, pior o atendimento que o processo recebeu. Observando a média apresentada na Tabela 4.23 pode-se observar que nesse critério, a abordagem MP apresentou um tempo médio de conclusão de processos menor e portanto melhor desempenho (23,3 UT). Em contrapartida, o pior atendimento médio foi no escalonador Circular baseado em prioridade (34 UT).

Tabela 4.23 – Valores do critério Tempo de Processamento.

	Processos de Usuário			
	Processo A	Processo B	Processo C	Tempo de Processamento Médio
Escalonamento Primeiro a chegar, primeiro a ser servido	27	17	38	27,3
Escalonamento Menor Primeiro	35	8	27	23,3
Escalonamento Circular com seleção baseada em prioridade	28	35	39	34,0
Escalonamento Circular com Seleção Primeiro a chegar, primeiro a ser servido	39	19	30	29,3
Escalonamento Circular com seleção Menor Primeiro	43	9	22	24,6
Escalonamento Múltiplas Filas com Realimentação	43	21	25	29,6

Através do Gráfico 4.4 pode-se observar que os processos A, B e C foram melhor atendidos, respectivamente, nas abordagens PCPS, MP e Circular com seleção MP.

Gráfico 4.4 – Resultados obtidos no critério Tempo de Processamento.



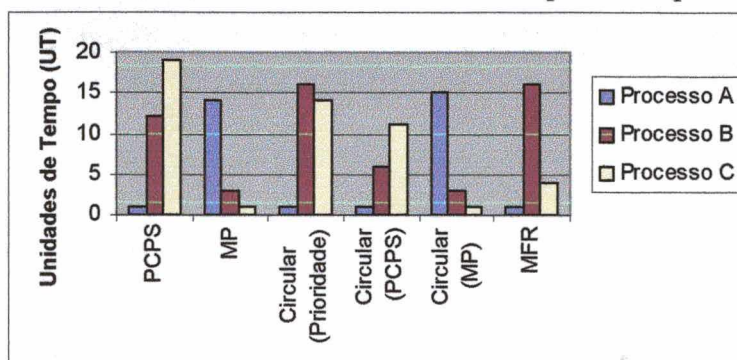
A Tabela 4.24 apresenta os valores do critério de avaliação de desempenho de processo denominado Tempo de Resposta obtidos nas diferentes abordagens de escalonamento. Nesse critério quanto menor o valor obtido, melhor o atendimento dado ao processo. Nota-se que na média, os piores atendimentos nesse critério foram obtidos nas abordagens PCPS e Circular com base em prioridade.

Tabela 4.24 – Valores do critério Tempo de Resposta.

	Processos de Usuário			
	Processo A	Processo B	Processo C	Tempo de Resposta Médio
Escalonamento Primeiro a chegar, primeiro a ser servido	01	12	19	10,6
Escalonamento Menor Primeiro	14	03	01	6,0
Escalonamento Circular com seleção baseada em prioridade	01	16	14	10,3
Escalonamento Circular com seleção Primeiro a chegar, primeiro a ser servido	01	06	11	6,0
Escalonamento Circular com seleção Menor Primeiro	15	03	01	6,3
Escalonamento Múltiplas Filas com Realimentação	01	16	04	7,0

Observando o Gráfico 4.5, o processo A como foi definido como de alta prioridade e o primeiro a chegar na LCP prontos, é privilegiado nas abordagens que utilizam ordem de chegada como critério de seleção (PCPS e Circular com seleção PCPS) e que utilizam prioridade como critério de seleção (MFR e Circular com base em prioridade). Da mesma forma, o processo C é privilegiado nas abordagens MP e Circular com seleção MP. Já o pior atendimento ao processo C aconteceu com a abordagem PCPS, quando o mesmo passa 19 UT para ter o primeiro atendimento realizado. Para o processo B os piores atendimentos nesse critério ocorreram nas abordagens Circular com base em prioridade e MFR que esperou 16 UT para ser selecionado pela primeira vez da LCP prontos.

Gráfico 4.5 – Resultados obtidos no critério Tempo de Resposta.



A Tabela 4.25 apresenta os valores do critério de avaliação de desempenho de processo denominado Tempo de Espera obtidos nas diferentes abordagens de escalonamento. Para o critério Tempo de Espera quanto maior o valor apresentado numa determinada abordagem de escalonamento mais tempo o processo passou pronto (na LCP prontos) enquanto o processador não estava disponível, isto é, estava ocupado com outro processo. Dessa forma, esse critério deve ser o menor possível.

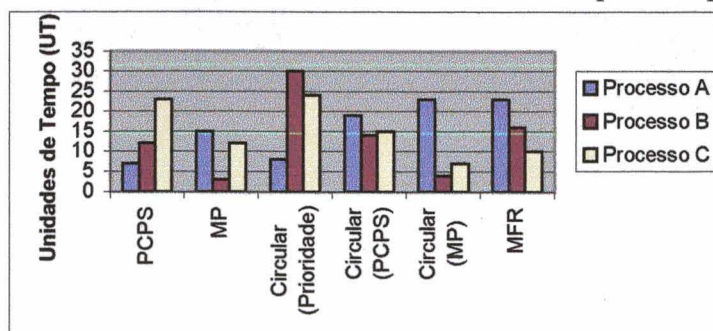
Em termos de tempos de espera médio, a mais alta média, e portanto a pior, foi observada na abordagem Circular com seleção por prioridade, e o melhor desempenho nesse critério foi na média da abordagem MP.

Tabela 4.25 – Valores do critério Tempo de Espera.

	Processos de Usuário			
	Processo A	Processo B	Processo C	Tempo de Espera Média
Escalonamento Primeiro a chegar, primeiro a ser servido	07	12	23	14,0
Escalonamento Menor Primeiro	15	03	12	10,0
Escalonamento Circular com seleção baseada em prioridade	08	30	24	20,6
Escalonamento Circular com seleção Primeiro a chegar, primeiro a ser servido	19	14	15	16,0
Escalonamento Circular com seleção Menor Primeiro	23	04	07	11,3
Escalonamento Múltiplas Filas com Realimentação	23	16	10	16,3

No Gráfico 4.6 observa-se que o tempo de resposta para o processo A nas abordagens preemptivas foi muito alto, variando entre 08 e 23 UT, e foi melhor atendido nesse critério na abordagem PCPS. O processo B teve os melhores tempos de resposta nas abordagens MP e Circular com seleção MP, porém na abordagem Circular baseada prioridade teve o pior atendimento (30 UT na LCP prontos). Já o processo C foi melhor atendido na abordagem Circular seleção MP (apenas 7 UT na LCP prontos).

Gráfico 4.6 – Resultados obtidos no critério Tempo de Espera.



CAPÍTULO 5 – O PROTÓTIPO DO SISO

Este capítulo é composto por cinco seções. Na primeira são apresentados a abrangência, o contexto de sua utilização, o público alvo e algumas características técnicas do protótipo do sistema desenvolvido denominado de SISO. Na segunda seção o SISO é caracterizado como um *software* educacional segundo as classificações apresentadas anteriormente no capítulo 2. Na terceira seção são apresentadas as fases de desenvolvimento e são também tratadas questões relativas à fase de análise de requisitos do protótipo utilizando princípios de Análise Estruturada. Na quarta seção são apresentados alguns aspectos da modelagem do protótipo e na quinta e última seção são apresentados as principais telas do protótipo desenvolvido sendo explicadas, ainda, as suas funcionalidades.

5.1. APRESENTAÇÃO E ABRANGÊNCIA DO PROTÓTIPO

SISO (Simulador de Sistemas Operacionais) é o protótipo⁵¹ de um *software* educacional abrangendo alguns assuntos do módulo de Gerência de Processos, a saber: modelo de processo, algoritmos de escalonamento de CPU e detecção de *deadlock*. O protótipo objetiva ser utilizado para apoio didático nas aulas das disciplinas Sistemas Operacionais e Sistemas Operacionais I e II dos cursos de Bacharelado em Ciência da Computação e Tecnologia em Processamento de Dados da Instituição de Ensino Superior em Belém (Pará), o CESUPA - Centro de Ensino Superior do Pará, na qual a autora deste trabalho atua como professora.

Optou-se por modelar os assuntos citados acima principalmente por dois motivos: primeiro por serem assuntos intimamente relacionados uns com os outros e serem de abrangência bem delimitada. A segunda razão diz respeito às dificuldades observadas nos alunos pelo alto grau de abstração relacionada a alguns conceitos da

⁵¹ Por protótipo entende-se que se trata de um modelo de *software* que capacita o desenvolvedor a visualizar e mensurar dificuldades e potencialidade de algo que será implementado posteriormente (PRESSMAN, 1995, p.35). Assim, esse protótipo implementa um subconjunto da função exigida do *software* desejado: um simulador que trate do gerenciamento de todos os recursos do ambiente computacional (memória, sistema de arquivos, etc.) abrangendo os outros módulos do SO. Esse sistema completo poderá ser desenvolvido posteriormente.

área de *software* básico, que dificultava a compreensão dos mesmos, entendendo-se que, com a utilização do sistema, essas dificuldades seriam amenizadas.

Pelas características dos assuntos que são de difícil demonstração entendeu-se que um SE modalidade simulação seria o mais adequado, e essa decisão influenciou as fases posteriores.

Foi iniciada, então, a fase de especificações do *software*. Foram definidas as principais especificações com as informações necessárias que deveriam ser mantidas pelo sistema e os procedimentos necessários para manipulá-las. Nessa etapa foram definidas as variáveis envolvidas, chegando-se a um modelo de grande complexidade, que foi por sua vez simplificado para servir como um modelo educacional.

Saber o que deve ser abordado e o que deve ficar de fora (ser simplificado) na definição de um modelo é difícil de ser encontrado. Concorda-se com SEABRA (1993, p.49) quando o mesmo afirma que:

Uma simulação deve ser tão simples que permita manipular com clareza algumas variáveis (pois se tratar de todas as variáveis, tem-se a realidade propriamente dita, e não um modelo dela), mas não simples a ponto que se torne inverídica.

A próxima fase foi a de codificação do protótipo. Foi utilizada a linguagem de programação Delphi 5, utilizando o Banco de dados Paradox 7.

O protótipo desenvolvido é capaz de gerar ambientes de descoberta, discussão e experimentação no estudo de SO. Por exemplo, o professor pode definir um problema: supõe-se que dada uma determinada carga de trabalho onde é definida a quantidade e duração das fases de uso de CPU e de E/S de uma certa quantidade de processos de usuário, seja definido um objetivo que deve ser alcançado, como por exemplo, a produtividade da CPU no atendimento aos processos deve ser superior a 90%. O aluno então, utilizando o protótipo do SISO, submeteria tal carga de trabalho e deveria encontrar, entre as estratégias de escalonamento de CPU implementadas no SISO, buscar qual deles se aproximará mais do objetivo determinado. De acordo com as características predominantes dos processos (*CPU-bound*, *I/O-bound* ou balanceado), o aluno poderá deduzir qual das estratégias melhor os atenderia.

Com exercícios desse tipo explora-se dois focos do protótipo: modelo de processo e estratégia de escalonamento. Após executar uma determinada simulação, ele analisará os resultados e poderá simular novamente com as mudanças nos parâmetros de

escalonamento que julgar necessárias, até obter um resultado satisfatório. Assim, o aluno deverá relacionar suas ações, ao manipular o sistema, com os conceitos tratados em sala de aula (e complementados com leitura nas bibliografias disponíveis sobre o assunto), pois só assim poderá definir uma estratégia coerente com os objetivos que pretende alcançar, entre as centenas de combinações possíveis de serem simuladas no sistema.

No estudo de *deadlock*, o aluno poderá criar uma situação hipotética definindo o número de processos ativos e recursos existentes, e alocar os recursos para testar se uma situação apresenta *deadlock* ou não. Assim o aluno pode observar em que situações específicas o sistema indica ocorrência de *deadlock*.

5.2. CARACTERIZANDO O SISO COMO SOFTWARE EDUCACIONAL

Para caracterizar o SISO de acordo com as várias classificações possíveis apresentadas no capítulo 2, inicialmente entende-se que, em relação a função que o *software* desempenha se trata de um *software* da modalidade de simulador, pois, analisando as características dos SE apresentadas por RAMOS (1996) nota-se que o SISO, quanto à atividade do aprendiz, é do tipo heurístico, pois apresenta um ambiente que pode ser explorado, onde predomina a aprendizagem experimental. Quanto ao direcionamento na utilização do *software*, entende-se que trata-se de uma abordagem do tipo branda, pois pela experimentação pode-se testar novas hipóteses e novos projetos, fazendo com que, ao utilizá-lo, o aluno esteja no comando, fazendo uma série de atividades que sugerem desafio.

Entende-se que o modelo do protótipo desenvolvido oscila entre os modelos por descoberta e por construção (segundo a classificação apresentada por MORAES, 1999). **Por descoberta** porque com o uso do protótipo o próprio aluno poderá encontrar relações entre os resultados obtidos nas simulações, experimentando e tirando suas conclusões sobre os resultados. Já quando entende-se o protótipo como do modelo **por construção**, nota-se que o uso do protótipo facilitará a evolução dos conhecimentos dos alunos, aprimorando-os.

Segundo a classificação de GIRAFFA (1999) quanto aos dois possíveis tipos de SE modalidade simulação (determinístico ou aleatório), entende-se que o SISO é um simulador do tipo **determinístico** na simulação de escalonamento, pois dada a mesma carga de trabalho e os mesmos parâmetros de execução, os resultados serão sempre os mesmos (no caso os valores obtidos nos critérios de desempenho), e a simulação é do tipo **aleatório** na simulação de detecção de *deadlock*, pois mesmo informando-se a mesma quantidade de processos e de dispositivos de E/S existentes, a simulação não necessariamente gera os mesmos resultados.

Segundo a classificação⁵² de VALENTE (1993, p.8) o SISO poderia ser classificado como um programa de controle de procedimentos, sendo a vantagem deste tipo de *software* eliminar certos aspectos tediosos de descrição de fenômenos. Por exemplo, quando os alunos desenvolviam manualmente os esquemas de atendimento de processos durante as aulas sobre o assunto de algoritmos de escalonamento de CPU, era muito comum observar que a elaboração do esquema passava a ser mais importante que o uso do mesmo para compreender como o atendimento aos processos ocorreu. Isso pode ser atribuído a nossa natural dificuldade em trabalhar com dados extensos muito detalhados. Ao final, a tarefa era tediosa e o mais importante, sem grandes contribuições para o entendimento da lógica dos escalonadores de processos. A construção do esquema nada mais era do que um meio de extrair dados para organizá-los para, a partir de então, realizar a análise do atendimento dos processos naquele escalonador. Pequenos erros comprometiam os resultados obtidos, fazendo com que muitas vezes a análise fosse prejudicada.

Realizando o mesmo tipo de exercício com o uso do *software* (e não mais construindo o esquema manualmente), poderia-se gerar vários esquemas de atendimento com várias políticas de escalonamento e verificar qual dentre elas apresentou melhor desempenho. O fato de o computador ser usado para simular o atendimento dos processos através de um *software* (de simulação), faz com que o esquema e os dados extraídos desse, possam ser usados para entender como o atendimento foi feito, analisar resultados e investir em melhores alternativas para alcançar resultados mais produtivos.

⁵² VALENTE (1993, p.8) classifica que, quando o computador é utilizado como ferramenta educacional, os programas educacionais podem ser: aplicativos para uso do aluno e do professor, resolução de problemas através do computador, produção de música, programas de controle de processo e computador como comunicador.

A seguir são apresentadas algumas fases genéricas de desenvolvimento de *software*, onde é apresentado e justificado o método utilizado no desenvolvimento do SISO.

5.3. MÉTODO DE ANÁLISE DE REQUISITOS DE SOFTWARE

Um programa (ou conjunto de programas) funcionando é somente uma parte de uma configuração de *software* que inclui elementos como planejamento, especificação de requisitos, projeto, estrutura de dados, codificação e especificação de testes. Essas fases em conjunto resultam no programa funcionando. A documentação resultante desses elementos constitui os alicerces para um desenvolvimento bem-sucedido, fornecendo um guia para a tarefa de manutenção que possivelmente será desenvolvida posteriormente, buscando o aprimoramento do *software*. Assim, independente do paradigma de Engenharia de Software⁵³ escolhido, da área de aplicação, do tamanho ou da complexidade do projeto, são desenvolvidas três fases: definição⁵⁴, desenvolvimento⁵⁵ e manutenção⁵⁶.

Até onde foi pesquisado não foi encontrado, na literatura, referencial de apoio para o desenvolvimento específico de *software* educacional. Assim no desenvolvimento da primeira das fases genéricas do protótipo do SISO foram usados os princípios da

⁵³ Segundo PRESSMAN(1995, p.31) trata-se de uma parte da engenharia de sistemas e da engenharia de *hardware*, e pode ser definida como *o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais*.

⁵⁴ Definição – nessa fase focaliza-se o *o que*, isto é, o desenvolvedor tenta identificar quais informações têm de ser processadas, qual função e desempenho são desejados, quais interfaces devem ser estabelecidas, quais restrições de projeto existem e quais critérios de validação são exigidos para se definir um sistema bem-sucedido. Nessa fase, as exigências fundamentais do sistema e do *software* são identificadas. Inclui três etapas: Análise do sistema (define o papel de cada elemento num sistema baseado em computador, atribuindo em última análise, o papel que o *software* desempenhará definido seu escopo), Planejamento do projeto de *software* (nessa etapa são analisados os riscos, os recursos necessários, alocar tais recursos, custos envolvidos e a programação de trabalho) e Análise de requisitos de *software* (para entender a natureza dos programas a serem desenvolvidos, o analista (ou engenheiro) de *software* deve compreender o domínio da informação para o *software*, além do desempenho e interface exigidos).

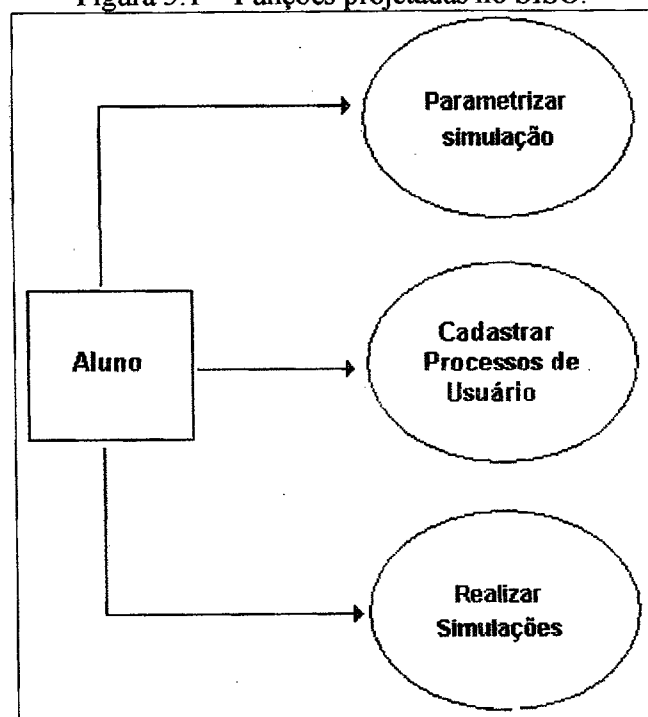
⁵⁵ Desenvolvimento – nessa fase focaliza-se o *como*. É dividida em três etapas: Projeto de *software* (é de fato um processo de múltiplos passos que se concentra em quatro atributos distintos do programa: estrutura de dados, arquitetura de *software*, detalhes procedimentais e caracterização da interface. O projeto traduz as exigências numa representação do *software* que pode ser avaliada quanto à qualidade antes que a codificação comece), Codificação (nessa etapa o projeto é traduzido numa forma legível por máquina) e Realização de testes do *software* (após gerar o código, iniciam-se os testes do programa; essa etapa concentra-se em aspectos lógicos internos, isto é, garantir que todas as instruções tenham sido testadas e funcionais externos, isto é, buscar erros e garantir que a entrada definida produza resultados esperados).

⁵⁶ Manutenção – as mudanças realizadas depois de realizados os testes ocasionados por correção (erros encontrados), adaptação (necessidade de adaptação do *software* para acomodar mudanças em seu ambiente externo), e/ou melhoria funcional (acréscimos funcionais ou de desempenho).

Análise Estruturada, por ser, como todos os métodos de análise de requisitos de *software*, uma atividade de construção de modelos, e desta forma poderia retratar o fluxo e o conteúdo da informação (tanto de dados como de controle), dividindo o sistema em partições funcionais e comportamentais, e, assim, possibilitando que fosse descrito a essência daquilo que deveria ser construído (PRESSMAN, 1995).

Basicamente o SISO foi projetado para que o aluno (seu principal usuário) realize as funções apresentadas na Fig. 5.1: Possibilitar a entrada de parâmetros utilizados na simulação, Permitir o cadastramento dos dados dos processos de usuário e Simular a execução dos processos cadastrados de acordo com o escalonador definido e simular a detecção de situação de *deadlock*.

Figura 5.1 – Funções projetadas no SISO.



Serão apresentados os seguintes modelos para representar o protótipo SISO:

- Diagrama de fluxo de dados⁵⁷ (DFD),
- Especificação de procedimentos⁵⁸.

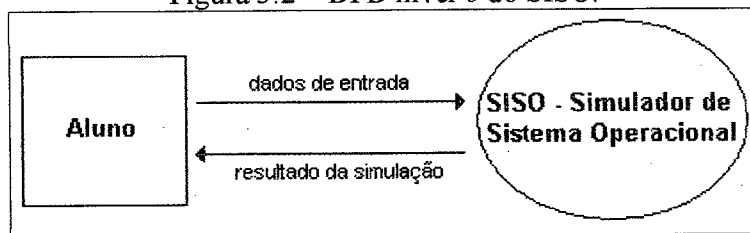
⁵⁷ Diagrama de fluxo de dados - é uma técnica gráfica que descreve o fluxo de informações que são aplicadas à medida que os dados se movimentam da entrada para a saída. PRESSMAN (1995) afirma que independente do tamanho e complexidade de um sistema, é possível criar um modelo de fluxo de informação, já que o principal elemento de um sistema é o que o motivo de sua existência é a manipulação de informações.

⁵⁸ Especificação de procedimentos - descreve detalhes dos procedimentos do nível final de refinamento, servindo como guia para o projeto dos componentes de programa que serão implementados. Deve incluir uma especificação precisa do processamento, com seqüência de eventos, pontos de decisão, operações repetitivas e estrutura/organização de dados.

5.3.1. Diagrama de fluxo de dados

O DFD de nível 0 é esquematizado de forma a exibir as entidades externas que produzem informação para serem usadas pelo sistema e as entidades externas que recebem as informações que são processadas pelos procedimentos⁵⁹. Como apresentado na Fig. 5.2, no caso do SISO tem-se uma única entidade externa⁶⁰ a manipular o mesmo – a entidade externa Aluno, que interage com o sistema, fornecendo informações quanto aos processos de usuário e parâmetros de definição do escalonamento, e o sistema, por sua vez, terá como saída os dados de uma execução simulada em forma de um relatório.

Figura 5.2 – DFD nível 0 do SISO.



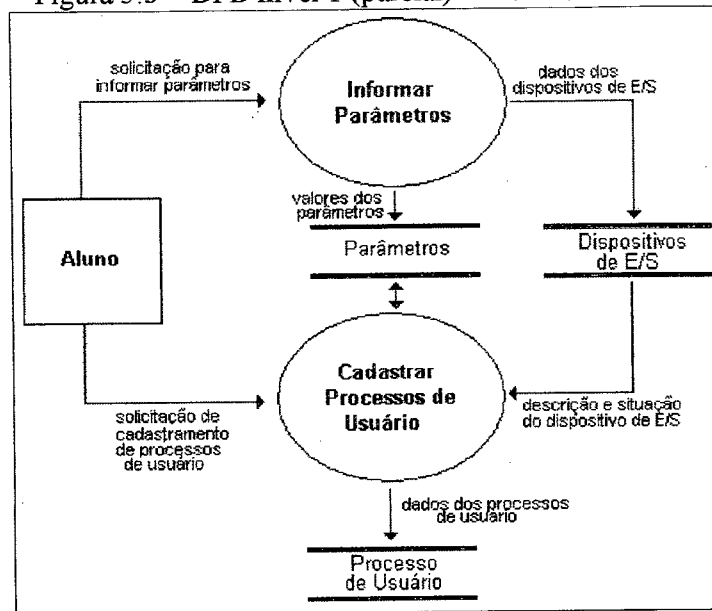
Quando o DFD é refinado em níveis de detalhes maiores, é realizada uma decomposição funcional implícita do sistema (PRESSMAN, 1995). A Fig. 5.3 apresenta uma parte do nível 1 de refinamento do SISO, onde são introduzidos os dados necessários para realizar a simulação. Assim, foram identificados dois procedimentos principais: Informar Parâmetros e Cadastrar processos de usuário. São apresentados também os depósitos de dados⁶¹ visíveis nesse nível.

⁵⁹ Nos capítulos 3 e 4, o termo processo foi usado para definir o conceito abstrato de um programa em execução, relativo ao contexto de SO. No contexto de Engenharia de Software o termo processo é também utilizado para especificar um conjunto de atividades de manipulação de informações. Para evitar tratar significados diferentes utilizando o mesmo termo, no contexto de Engenharia de Software é adotado o termo *procedimento* como sinônimo de processo.

⁶⁰ Entidade Externa - Um produtor e/ou consumidor de informações que esteja fora dos limites de abrangência do sistema modelado.

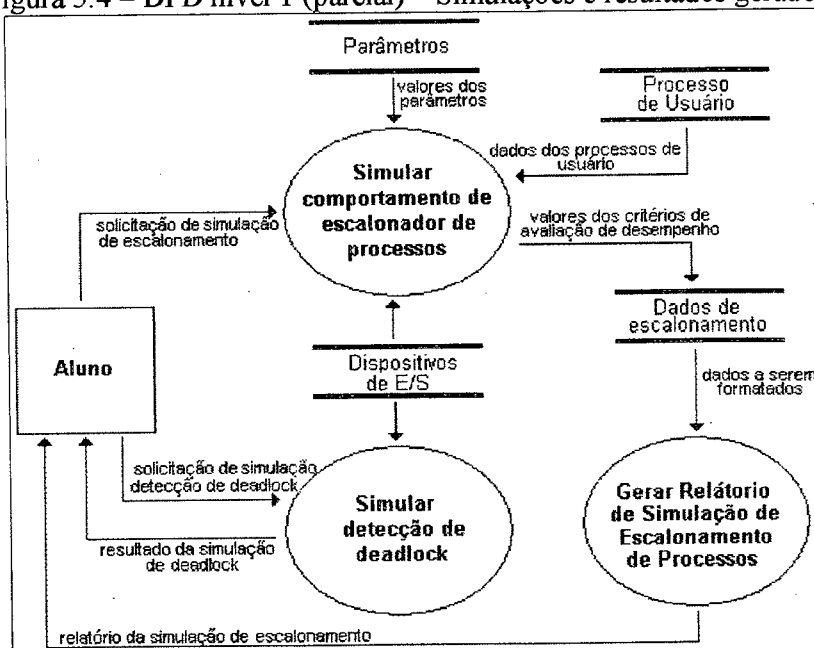
⁶¹ Depósito de dados – informações armazenadas que são usadas pelo sistema. Trata-se de um repositório de dados que são armazenados para serem utilizados por um ou mais procedimentos. No DFD não é especificada a natureza desse elemento. PRESSMAN (1995, p.281) afirma que podem ser tão simples quanto um buffer ou tão sofisticado quanto um banco de dados relacional.

Figura 5.3 – DFD nível 1 (parcial)– Entrada de dados.



A simulação tratada no sistema envolve dois aspectos: simulação de escalonamento de processos (seção 3.6) e simulação de detecção de *deadlock* para vários recursos de cada tipo (seção 3.7.4.2). O DFD exibido na Fig. 5.4 detalha esses dois procedimentos, assim é possível visualizar detalhes das funcionalidades do protótipo.

Figura 5.4 – DFD nível 1 (parcial) – Simulações e resultados gerados.



5.3.2. Especificação de procedimentos.

Será feito um resumo das ações a serem realizadas em cada um dos procedimentos apresentados nos DFD, a saber:

- Informar Parâmetros.
- Cadastrar processos de usuário.
- Simular comportamento do escalonador de processos.
- Gerar relatório de simulação de escalonamento de processos.
- Simular detecção de *deadlock*.

Antes de dar início à especificação dos procedimentos, serão feitas algumas considerações sobre o que o sistema se propõe a fazer e outras observações de suma importância ao entendimento das características, abrangência e limitações do protótipo:

- Considera-se um ambiente de monoprocessamento e de multiprogramação;
- A unidade de tempo considerada é fictícia e será identificada como UT (unidade de tempo);
- Cada UT corresponde a um determinado número de ciclos de máquina (comparação ilustrativa), onde poderiam ser executadas n instruções de máquina;
- Considera-se uma quantidade de unidades fictícias para determinar o tamanho da Memória Virtual a ser utilizada pelos processos, identificadas como UM (unidade de memória). Esse valor é informado como parâmetro e objetiva limitar a duração das fases de uso de CPU e E/S informadas pelos alunos;
- Serão considerados dois tipos de processo: processo de SO e processo de usuário;
- Será considerado um único processo de SO identificado como 0 (zero), que fará uso exclusivamente de tempo de processador, isto é, vamos considerar que o processo de SO nunca necessite realizar uma operação de E/S para si próprio.
- São duas as situações em que será simulada a execução de processos de SO: para realizar a conclusão de processos (1 UT) e para realizar mudança de contexto (1 UT). A quantidade de UT para cada uma dessas atividades é fixa e determinada internamente.

- Os estados válidos para os processos serão inspirados na definição de estados do LINUX⁶² exibidos na Tabela 5.1.

Tabela 5.1 – Estados válidos para os processos de usuário no protótipo SISO.

Código	Significado	Descrição
0	EM EXECUÇÃO (RUNNING)	Em execução, fazendo uso da CPU; instruções do código do processo estão sendo executadas.
1	PRONTO (READY)	Na fila de processos prontos, aguardando por CPU; o processo está pronto para obter o controle do processador.
2	EM ESPERA (WAIT)	Bloqueado, fazendo uso de recurso de E/S; o processo está esperando pelo término de uma operação de E/S.
4	CONCLUÍDO (ZOMBIE)	Finalizado, concluído.
6	NOVO	Processo é submetido e admitido no sistema, recebendo recurso área de memória a ser alocado.

No protótipo são assumidos como válidos todos os estados e transições entre estados de processo mencionadas na Fig. 3.11.

5.3.2.1. Informar parâmetro

- Receber valor de memória virtual, valor do intervalo de UT (para calcular os critérios de avaliação de desempenho de sistema) e valor do quantum (no caso de escalonador preemptivo por tempo) e armazená-los em *Parâmetros*.

5.3.2.2. Cadastrar processos de usuário

Nesse procedimento é realizado o cadastro dos processos de usuário, onde os processos de usuário serão incluídos, alterados e excluídos pelo aluno:

- É exibida tela onde é apresentada a identificação do processo e solicitado o valor de prioridade do processo.
- Cada processo de usuário terá uma identificação numérica, reutilizada, inteira e maior que 0.
- A cada processo de usuário é opcionalmente atribuído um valor de prioridade. As prioridades variam de 0 (mais baixa) até 9 (mais alta). A prioridade *default* que todos os processos submetidos recebem é 0 (zero).

⁶² Com adaptações simplificadas – ver <http://www.LINUXdoc.org/LDP/ki/LINUX-Kernel-Internals-2.html>

- Para cada UT (de necessidade de processador ou de E/S) considerada para cada processo, será entendido que o mesmo necessitaria de uma UM (unidade de memória), isto é, a relação UT:UM será 1:1;
- O limite de processos de usuário ativos é dado pela relação 1:1 entre tamanho da memória virtual (em UM) e somatório das necessidades dos processos (em UT); A tentativa de incluir processos que excedam esse limite é impedida pelo sistema;
- O aluno deverá incluir uma ou mais fases de uso de processador (obrigatoriamente) para cada processo de usuário a serem incluídas no depósito de dados *Processos de Usuário*.
- O aluno poderá incluir, opcionalmente, fase(s) de uso de dispositivos de E/S a serem incluídas no depósito de dados *Processos de Usuário*, sendo que cada uma delas deve estar relacionada a um dos dispositivos de E/S cadastrados no depósito *Dispositivos de E/S*.
- O cadastramento no depósito de dados *Dispositivos de E/S* é também feito através desse formulário. Considerando escalonamento de processos haverá apenas uma instância de cada um dos dispositivos de E/S. Esses dispositivos de E/S são identificados como de três possíveis tipos: de Entrada, de Saída ou de Entrada ou Saída, e também registrados como Ativos ou Inativos.
- Cada fase de uso de dispositivos de E/S deverá estar relacionada a um dos recursos existentes.
- Para a inclusão de uma fase de uso de dispositivo de E/S só serão apresentados os dispositivos com situação Ativo.

5.3.2.3. Simular comportamento do escalonador de processos

- O sistema suporta 4 possíveis abordagens de escalonamento:
 - Primeiro a chegar, primeiro a ser servido (PCPS) - Sem passagem de parâmetros.
 - Menor Primeiro (MP) - Sem passagem de parâmetros.
 - Circular (Round Robin) - Parâmetros obrigatórios: valor do quantum e o critério de seleção da LCP prontos que poderá ser PCPS, MP ou Prioridade.

- MFR – Múltiplas Filas com Realimentação – onde são implementadas quatro LCP prontos. Parâmetros obrigatórios: valor base do quantum (sendo que o valor de quantum de cada uma das LCP será em função deste) e o critério de seleção da LCP poderá ser PCPS ou MP.
- O atendimento aos processos de usuário deve simular o comportamento dos algoritmos de escalonamento apresentados na seção 3.6.3, e realizar o cálculo dos valores dos critérios de avaliação de desempenho tratados na seção 4.2.4, armazenando o resultado em *Dados de Escalonamento*.

5.3.2.4. Gerar relatório de simulação de escalonamento de processos

Apresentar um relatório com formato apresentado na Fig. 5.5, onde é exibido um exemplo, simular ao tratado na seção 4.3.3.2, a saber:

- O cabeçalho refere-se à identificação do sistema com informações de data e hora de emissão do relatório.
- Na parte 1 são apresentados os dados de entrada dos processos de usuário (identificação dos processos, prioridade e quantidade e duração das fases de uso de CPU e E/S) e também o parâmetro de tamanho de memória virtual (UM) e a quantidade percentual de sua utilização.
- A parte 2 indica os parâmetros de escalonamento informados pelo aluno, como tipo de escalonador (PCPS, MP, Circular ou MFR), critério de seleção (PCPS, MP ou Prioridade), valor do quantum a ser considerado e valor do intervalo de tempo a ser considerado.
- A parte 3 apresenta o esquema de atendimento gerado usando a idéia de linha de tempo representada em colunas onde cada estado pelo qual o processo passa seja registrado.
- Na parte 4 são indicados os valores dos critérios de desempenho (de sistema e por processo) obtidos com a simulação.

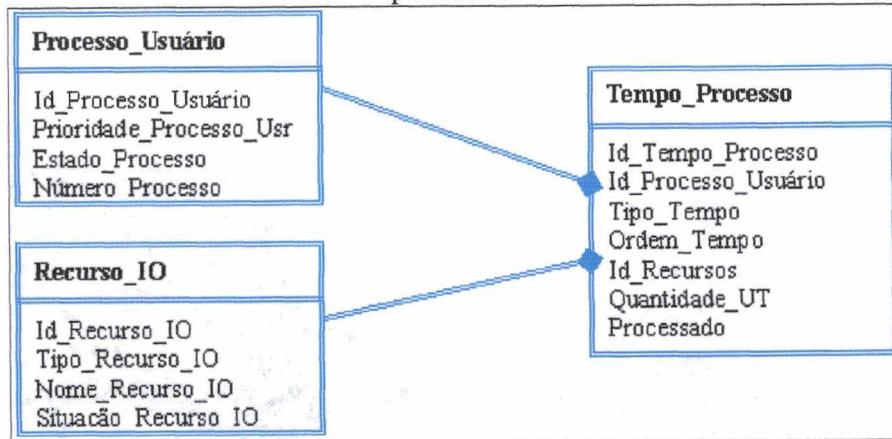
5.3.2.5. Simular detecção de *deadlock*

- O aluno deve informar: Quantidade de processos ativos: [n] e Quantidade de dispositivos de E/S existentes [m], onde n pode receber valor ≥ 2 e m pode receber valor ≥ 0 .
- Criar vetor P com n posições, onde $P(n-3) = P_{n-3}, \dots, P(n) = P_n$. Por exemplo, se $n=3$ então P com 3 posições onde $P(1)=P_1, P(2)=P_2, P(3)=P_3$
- Criar vetor E com 1x5 posições, onde $E(1)=m_1, E(2)=m_2, \dots, E(5)=m_5$
- Criar vetor D com 1x5 posições inicialmente zerado.
- Criar vetor auxiliar W com 1x5 posições inicialmente zerado.
- Criar matriz A com nx5 e inicialmente zerada. Por exemplo, $a(1,1), a(1,2), a(1,3), a(1,4), a(1,5), \dots, a(n,1) \dots a(n,5)$
- Criar matriz R com nx5 e inicialmente zerada. Idem anterior.
- Gerar valores aleatórios para as posições da coluna 1 da matriz A $a(1,1), \dots, a(n,1)$, de tal forma que o somatório desses valores seja $\leq m_1$. (idem para as outras colunas).
- Atualizar a matriz D, onde $D(1)=E(1)-\sum[a(1,1) \dots a(n,1)]$. Idem para as outras colunas.
- Gerar valores aleatórios para as posições $r(1,1), \dots, r(n,1)$, da matriz R. (idem para as outras colunas)
- Fazer W receber os valores de D.
- Para cada uma das linhas i fazer,
 - Se houver alguma linha i de $R \leq$ vetor W (comparar posição a posição), então guardar o valor de i (é necessário saber quais as linhas marcadas)
 - fazer $W = W + A_i$.
- Exibir em tela os vetores E, D, e as matrizes A e R.
- Se alguma linha não foi marcada (houve *deadlock*) então, exibir “HÁ DEADLOCK ENTRE OS PROCESSOS :” linhas não marcadas
senão, exibir “NÃO HÁ SITUAÇÃO DE DEADLOCK”.

5.4. MODELAGEM DO PROTÓTIPO DO SISO

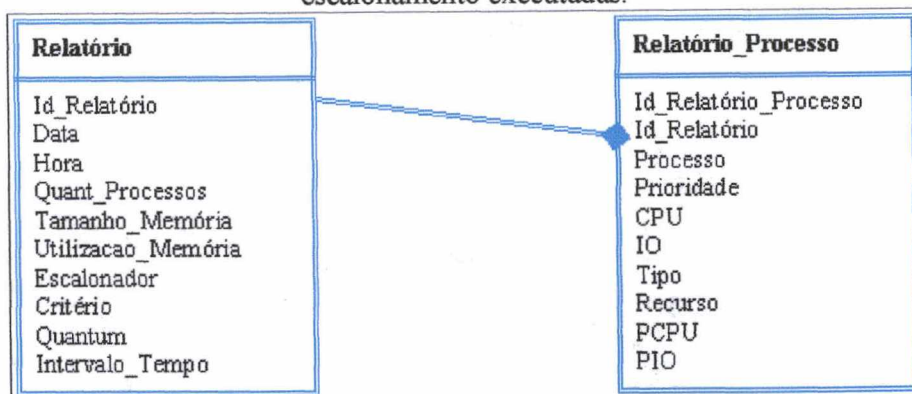
Nesta seção são tratados os aspectos de projeto do protótipo do sistema. A Fig. 5.5 apresenta as principais informações mantidas nos depósitos de dados criados para cadastramento dos processos de usuário com registro das fases de uso de CPU e de dispositivos de E/S.

Figura 5.5 – Relação entre Tabelas com os campos armazenados para cadastramento de processos.








Na Fig.5.6 são apresentadas as duas tabelas mantidas para guardar os registros de simulações feitas no sistema.

Figura 5.6 – Tabelas com os campos armazenados para manter as simulações de escalonamento executadas.



Foram criadas rotinas individuais para cada uma das tarefas objetivadas no sistema. A Fig.5.7 relaciona cada um dos botões usados no sistema com as rotinas e sua descrição resumida.

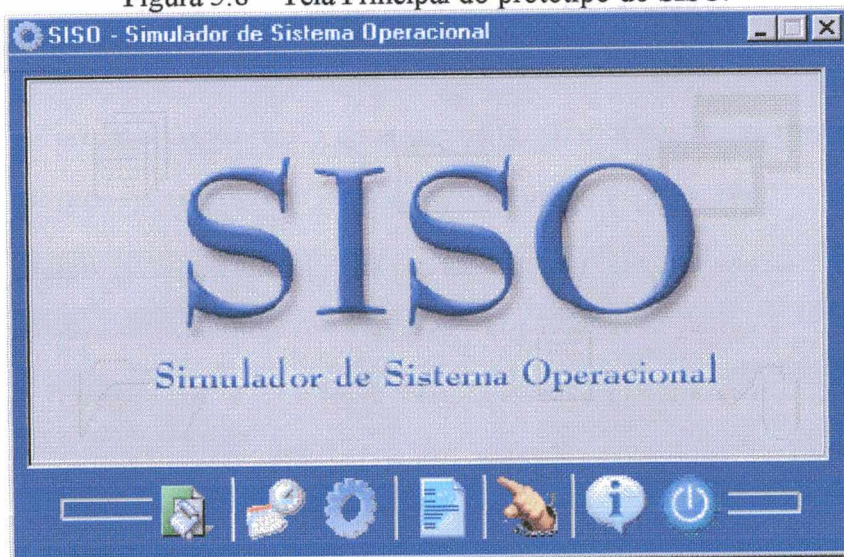
Figura 5.7 – Rotinas criadas no protótipo do SISO.

	Executa a DLLCONFI que armazena as informações sobre a memória virtual, intervalo de tempo e <i>quantum</i> em um arquivo de sistema no diretório /Windows com o nome de SISO.INI.
	Executa a DLLPROCE que cria e manipula os processos, tempos de processo e os recursos de E/S utilizando as tabelas: Processo_Usuário, Tempo_Processo e Recurso_IO.
	Executa a DLLFIFO que insere os dados na tabela Relatório e Relatório_Processo.
	Executa a DLLRELAT que lê os dados na tabela Relatório e Relatório_Processo e gera o relatório HTML.
	Executa a DLLDEADL que lê os dados na tabela Recurso_IO.

5.5. APRESENTANDO O PROTÓTIPO DO SISO

Nesta seção são apresentadas algumas telas e funcionalidades do protótipo desenvolvido. A tela principal do SISO (Fig. 5.8) apresenta sete funcionalidades acionadas pelos ícones do rodapé da tela principal, a saber: Configurações, Processos, Simulação PCPS, Relatório de Escalonamento, Detecção de Deadlock, Sobre e Sair.

Figura 5.8 – Tela Principal do protótipo do SISO.




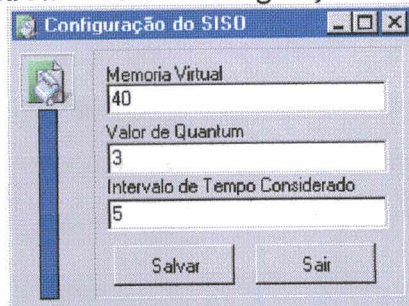
Através do botão Configurações identificado pelo ícone  é exibida tela (Fig.5.9) onde é possível informar o tamanho simulado de memória virtual a ser considerada para limitar a duração do somatório das fases de uso de CPU e de E/S de todos os processos de usuário cadastrados para simulação de escalonamento, e também informa-se qual o intervalo de tempo que deve ser considerado para calcular alguns dos critérios de avaliação de desempenho de sistema e o valor do quantum para simular o atendimento aos processos nos escalonadores preemptivos.

Figura 5.9 – Tela de configurações do SISO.




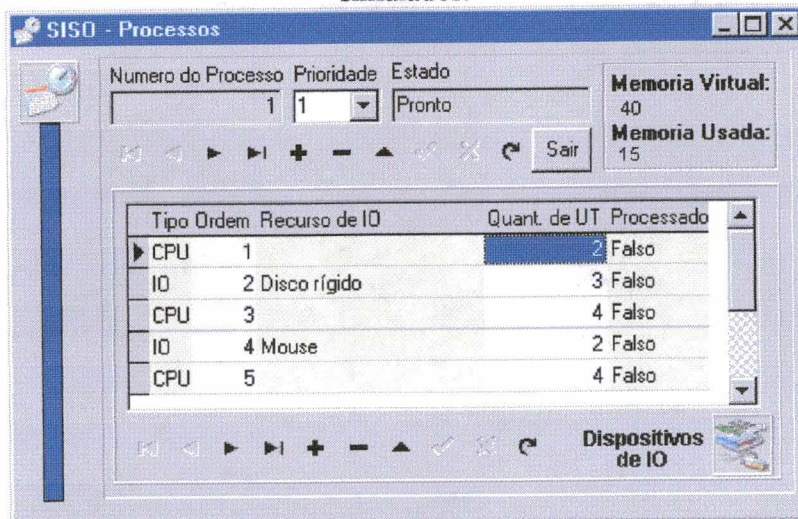
Acionando o ícone  do botão Processos é apresentada tela para cadastrar, alterar, excluir e consultar os processos de usuário que terão sua execução simulada (Fig.5.10). Devem ser informados a prioridade do processo (de 0 a 9, sendo 9 a mais prioritária) e a duração das fases de uso de CPU e fases de uso de dispositivos de E/S. A identificação do processo (Número do Processo) é atribuída internamente pelo sistema.

Figura 5.10 – Tela de entrada e manutenção dos processos de usuário a serem simulados.




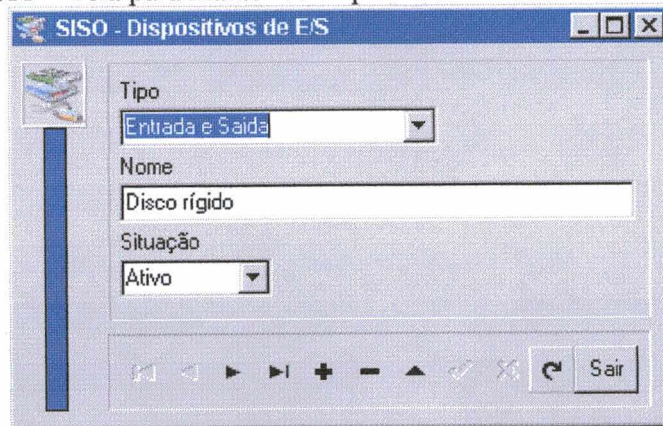

Acionando o ícone  apresentado na tela de cadastro de processos de usuário, é exibida a tela da Fig.5.11 onde se pode cadastrar, alterar, excluir e consultar os dispositivos de E/S a serem simulados no escalonamento de processos.

Figura 5.11 – Tela para manter os dispositivos de E/S a serem simulados.



No botão Escalonamento  pode-se simular a execução dos processos anteriormente cadastrados usando os princípios do escalonador escolhido no desenvolvimento do protótipo, o escalonador não-preemptivo PCPS. É ainda apresentada mensagem confirmando ou não a simulação que foi solicitada. Cada simulação executada tem os dados gerados armazenados identificando-a através da data e hora da geração. Assim é possível gerar várias simulações para depois compará-las.


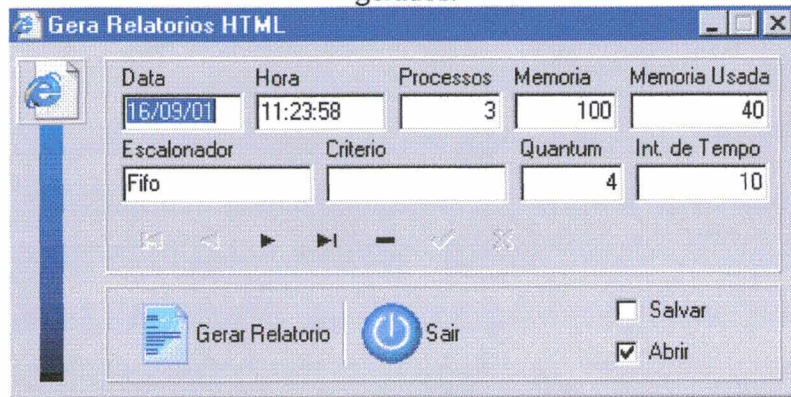
É através do botão Relatório de Escalonamento  que é exibida a tela apresentada na Fig. 5.12 onde pode ser escolhida entre as simulações geradas aquela que terá o relatório de escalonamento com os critérios de avaliação de desempenho criado. Pode-se optar ainda em visualizar o relatório na tela (opção Abrir) ou salvá-lo num arquivo HTML (opção Salvar).

Figura 5.12 – Tela onde são exibidos os registros das simulações de escalonamento gerados.



As Figuras 5.13 e 5.14 mostram um exemplo de relatório gerado em HTML de simulação de escalonamento. Na Fig.5.13 é exibido o cabeçalho informando a data e hora da geração do mesmo, além de apresentar os dados de entrada que serviram de base a simulação, como quantidade de processos, informações relativas a tamanho informado e utilização de memória virtual simulada e também a quantidade e duração das fases de uso de CPU e E/S dos processos de usuário.

Figura 5.13 – Primeira parte do relatório de simulação de escalonamento de processos.

SISO – Simulador de Sistema Operacional

Data: 16/09/01

Relatório de Simulação de Escalonamento de Processos

Hora: 11:23:58

1 - DADOS DE ENTRADA

Quantidade de processos de usuário = 3

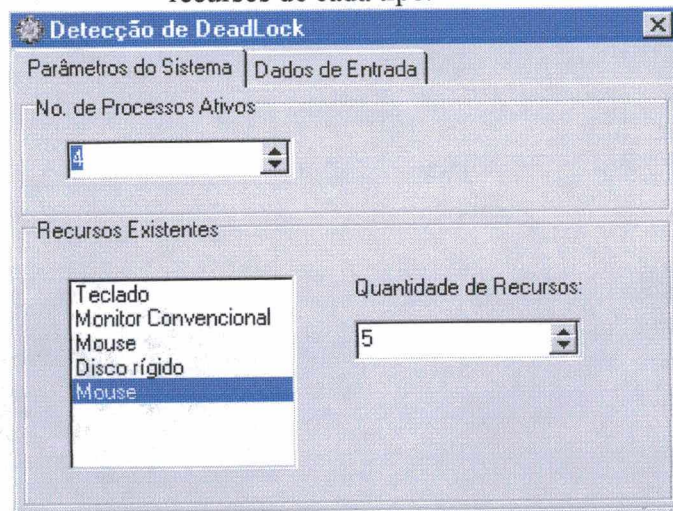
Tamanho da Memória Virtual = 100 UM

Utilização da Memória Virtual = 40 UM - 40 %

Identificação dos processos de usuário	Prioridade	Fases de uso			
		Duração das fases de uso de CPU (em UT)	Fases de uso de dispositivo de E/S		
			Duração (em UT)	Tipo de Operação	Dispositivo requisitado
1	9	10	4	Entrada	mouse
	9	6			
2	5	5			
3	4	1	6	Entrada e Saida	disco rígido
	4	3	3	Entrada	teclado
	4	2			
TOTAL		27	13		
TOTAL GERAL		40			

processos ativos a serem considerados na simulação, e também são exibidos os dispositivos de E/S Ativos previamente cadastrados para que seja informada a quantidade considerada existente de cada um deles (Fig. 5.15).

Figura 5.15 – Tela de entrada de dados para simular detecção de *deadlock* com vários recursos de cada tipo.



Na segunda parte (Dados de Entrada) são apresentadas as cinco matrizes utilizadas para realizar a simulação (Fig.5.16). Acionando o botão **Preenche Matriz de Alocação Corrente** a matriz A (Matriz de Alocação Corrente) é preenchida com valores aleatórios para cada uma das linhas dentro do limite de recursos existentes de cada tipo. A matriz D (Matriz de Recursos Atualmente Disponíveis) é preenchida com a diferença entre os existentes e os alocados. O botão **Preenche Matriz de Requisições** gera valores aleatórios simulando requisições que os processos façam para cada um dos recursos, como mostrado na matriz R (Matriz de Recursos Requisitados).

Figura 5.16 – Resultado do acionamento do botão Preenche Matriz de Alocação Corrente.

Detecção de DeadLock - Algoritmo do Banqueiro

Parâmetros do Sistema | Dados de Entrada

Matriz de Recursos Existentes (E):

1	2	3	4	5
---	---	---	---	---

Matriz de Recursos Atualmente Disponíveis (D):

0	0	0	1	2
---	---	---	---	---

Matriz de Alocação Corrente (A):

0	0	1	0	0
0	1	1	3	3
1	1	0	0	0
0	0	1	0	0

Status:

--

Matriz de Recursos Requisitados (R):

Preenche Matriz de Alocação Corrente

Preenche Matriz de Requisições

Detectar DeadLock

Preenchidas as matrizes E, A, D e R, deve-se acionar o botão **Detectar DeadLock** que irá, seguindo os princípios do Algoritmo do Banqueiro, verificar quais dos processos estariam em *deadlock*. A matriz denominada de Status exibe a situação verificada, identificando a situação de cada um dos processos (Fig.5.17).

Figura 5.17 – Resultado do acionamento do botão Preenche Matriz de Requisições.

Detecção de DeadLock - Algoritmo do Banqueiro

Parâmetros do Sistema | Dados de Entrada

Matriz de Recursos Existentes (E):

1	2	3	4	5
---	---	---	---	---

Matriz de Recursos Atualmente Disponíveis (D):

0	0	0	1	2
---	---	---	---	---

Matriz de Alocação Corrente (A):

0	0	1	0	0
0	1	1	3	3
1	1	0	0	0
0	0	1	0	0

Status:

DeadLock
OK!
DeadLock
OK!

Matriz de Recursos Requisitados (R):

1	1	2	2	3
0	0	0	1	1
1	2	3	4	4
0	1	1	2	2

Preenche Matriz de Alocação Corrente

Preenche Matriz de Requisições

Detectar DeadLock


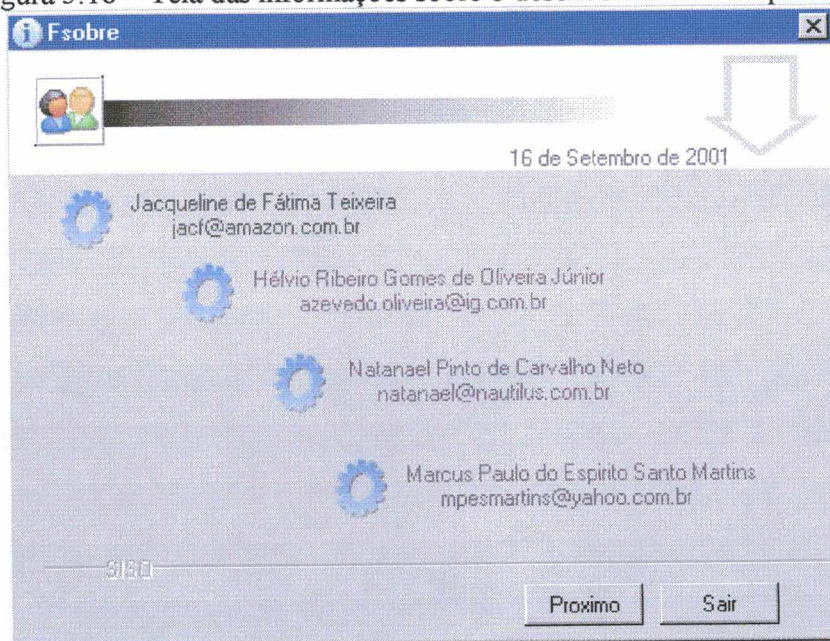
No botão Sobre  são exibidas informações para contato com as pessoas diretamente envolvidas no desenvolvimento dessa versão do protótipo (Fig.5.18).

Figura 5.18 – Tela das informações sobre o desenvolvimento do protótipo.



Além das pessoas relacionadas na tela Sobre, outras também participaram de forma direta ou indireta do desenvolvimento de versões anteriores que serviram de base para a atual versão do protótipo do SISO (apresentada por ocasião da defesa dessa dissertação). Agradecimentos especiais a José Haroldo Sena de Oliveira Filho (jsena@webset.net), Luiz Sérgio Carvalho Mácola (lumacola@bol.com.br), Maurício Santos Teixeira (mteixeira@webset.net) e Raimundo Nonato Monteiro Junior (rj_bill@zipmail.com.br).

CAPÍTULO 6 - CONSIDERAÇÕES FINAIS

Neste capítulo são destacados a relevância do tema abordado no presente trabalho, a colaboração que o mesmo oferece através dos itens propostos e desenvolvidos, e são relacionados os resultados obtidos em relação aos objetivos propostos. Numa segunda seção são sugeridos alguns tópicos que podem ser explorados para aprimoramento deste trabalho e pesquisas futuras.

6.1. CONCLUSÕES

Passaram-se apenas cerca de 60 anos desde que a primeira geração de computadores eletrônicos foi desenvolvida. As limitações quanto ao tamanho dos equipamentos e capacidade de armazenamento e processamento deram lugar em pouquíssimo tempo à capacidade de manipulação e armazenamento de informação a velocidades e em quantidades há poucos anos inimagináveis.

Aliado aos avanços da tecnologia de *hardware* estão os incontestáveis avanços da área de *software*. Com relação às linguagens de programação também houve avanços significativos. Das primeiras linguagens de montagem existentes, atualmente os desenvolvedores dispõem de linguagens de programação que oferecem recursos internos mais facilmente implementáveis e poderosíssimos, que possibilitam o desenvolvimento de sistemas aplicativos com interfaces cada vez mais amigáveis aos usuários.

Entendendo o sistema computacional como uma série de camadas de *software* que circundam o *hardware* em si, sabe-se que os sistemas aplicativos pertencem a camadas que são consideradas como de mais alto nível e portanto mais distantes da camada de *hardware*. Assim, grande parte das facilidades oferecidas pelos sistemas computacionais tanto do ponto de vista do desenvolvedor quanto dos usuários dos mesmos, devem-se a inovações tecnológicas implementadas nas camadas intermediárias, particularmente nos chamados programas de sistema.

Os programas de sistema são pertencentes a um grupo de programas denominado de Software Básico e possuem características bem peculiares, devendo servir de meio de comunicação de *software* aplicativo com o *hardware*, sem contudo

servir de meio de comunicação de *software* aplicativo com o *hardware*, sem contudo transparecer os detalhes do mesmo. Pertencente a essa categoria de *software* está o foco desse estudo: o Sistema Operacional.

Extremamente complexo e projetado para atender objetivos específicos diversos e muitas vezes conflitantes, o Sistema Operacional deve implementar estratégias que, ao mesmo tempo mantenha níveis aceitáveis de produtividade na utilização de elementos de *hardware* diversos e muitas vezes escassos (como tempo de processador, área de memória, etc.) e mantenha níveis de segurança aceitáveis, mas ao mesmo tempo, não deva executar rotinas próprias que monopolizem o uso dos elementos de *hardware* (*overhead*).

Dada sua importância no ambiente computacional, o conhecimento de noções de projeto e implementação de estratégias de SO é crucial na formação dos alunos de cursos da área de Ciência da Computação. Nos últimos dois anos essa importância foi ressaltada pela nova tendência do *software* ser de código aberto. Assim, o conhecimento de detalhes do projeto de muitos SO do mercado não é mais exclusivamente restrito aos seus próprios projetistas, mas sim aberto a comunidade. É a proposta do chamado *software* livre.

O ensino e estudo de SO, devido a suas características próprias (nível de implementação complexo muito aproximado do *hardware*), envolve muitos conceitos abstratos de difícil demonstração, e assim torna-se essencial buscar formas de ilustrar como determinadas estratégias são implementadas no SO e as conseqüências das mesmas para os usuários que têm suas aplicações submetidas à execução num ambiente computacional.

Uma das possíveis formas de minimizar as dificuldades encontradas para o ensino de conceitos e estratégias inerentes ao estudo de SO envolve o desenvolvimento de aplicações específicas, ou seja, desenvolvimento de *software* educacional. Assim, o desenvolvimento do SISO (como ferramenta de auxílio pedagógico), um protótipo de um Software Educacional modalidade simulação, nada mais é do que uma forma de facilitar a compreensão de certos assuntos específicos relacionados com a disciplina SO.

Como uma das modalidades de desenvolvimento de *software* educacional, os simuladores apresentam vários benefícios didáticos, como por exemplo:

- aprender fazendo,

- realização de exercícios combinando teoria com prática
- uso de *software* de apoio educacional que quebra a rotina de sala de aula,
- fazer o aluno pensar no que foi feito e a analisar os resultados e o porquê desses resultados ocorrerem,
- despertar o interesse dos alunos, com a utilização de *software* educacional que tenha uma nova dinâmica de ensino.

Sabe-se que estar envolvido com Educação na Sociedade da Informação significa muito mais do que treinar pessoas para o uso de tecnologias de informação e comunicação: trata-se de formar indivíduos para **aprender a aprender**, de modo a serem capazes de lidar positivamente com a contínua e acelerada transformação da base tecnológica.

A busca de conhecimentos técnicos por parte dos alunos e a busca pela melhor forma de tratar tais conhecimentos por parte dos professores, é apenas uma parte da rotina dessas pessoas numa instituição de ensino. A dinâmica da Sociedade da Informação requer educação continuada ao longo da vida, que permita ao indivíduo não apenas acompanhar as mudanças tecnológicas, mas sobretudo inovar. Assim, entende-se que a Educação em seus vários aspectos é o elemento-chave para a construção de uma Sociedade da Informação.

Concorda-se com MASETTO (1998) quando ele afirma que as Instituições de Ensino Superior podem ser consideradas como um local de encontro e de convivência entre educadores e educandos, que constituem um grupo que se reúne e trabalha para que ocorram situações favoráveis ao desenvolvimento dos alunos nas diferentes áreas do conhecimento, no aspecto afetivo-emocional, nas habilidades, nas atitudes e nos valores. Mas não deve-se esquecer que isso só é possível se a capacitação pedagógica e tecnológica dos educadores for continuamente desenvolvida.

6.2. PROPOSTAS DE TRABALHOS FUTUROS RELACIONADOS A ESTA PESQUISA

O prosseguimento deste trabalho tem por objetivo incorporar novos assuntos nas próximas versões do sistema como por exemplo:

- 1) No estudo de Gerencia de Processador.
 - a) Implementar o escalonador não preemptivo MP e os preemptivos Circular e MFR.
 - b) Gerar gráficos comparativos entre duas ou mais simulações de escalonamento geradas para facilitar a comparação dos valores obtidos nos critérios de avaliação de desempenho.
- 2) No estudo de SO no módulo Gerência de Memória.
 - a) Implementar uma opção onde possam ser simuladas as estratégias de substituição de páginas (ou segmentos),
 - b) Mostrar a relação entre tamanho do endereço (virtual e real), tamanho do frame, a quantidade de registros na Tabela de Páginas e quantidade de páginas.
- 3) No estudo de SO no módulo Gerência de Dispositivos de E/S.
 - a) no estudo do relacionamento entre níveis de E/S mostrando a função específica de drivers, controladoras e a função do SO na solicitação de operação de E/S para os processos e recebimento de retorno dessa solicitação,
- 4) No estudo de outras disciplinas que tratem de assuntos que também têm a característica de serem dificilmente visualizados por parte do aluno, por exemplo: o estudo do ciclo de execução de instruções da disciplina Organização de Computadores⁶³.
- 5) Desenvolver um módulo completo de ajuda (HELP dinâmico) para tratar de dúvidas que o usuário possa ter durante a manipulação do *software* e também que venha a esclarecer as dúvidas mais freqüentes.

⁶³ Assim, numa aplicação usando princípios de simulação poderia ser mostrado como ocorre a busca da instrução (indicada pelo Contador de Programa) de um determinado endereço de memória principal para o Registrador de Dados de Memória. Em seguida poderia ser mostrado o relacionamento entre os elementos da CPU (como decodificador, Unidade aritmética e lógica, registradores de armazenamento, etc.) na interpretação da instrução, a busca dos dados que essa instrução possa vir a manipular (dependendo do modo de endereçamento dos operandos) e efetiva execução da mesma.

ESTRATÉGIAS DE TRABALHO PARA REVISIONISTAS

O desenvolvimento deste trabalho tem por objetivo incorporar novos assuntos nas próximas versões do sistema como por exemplo:

- 1) No estudo de Gerência de Processador
 - a) Implementar o escalonador não preemptivo MFC e os preemptivos Circular e MFC.
 - b) Gerar gráficos comparativos entre duas ou mais simulações de escalonamento geradas para facilitar a comparação dos valores obtidos nos critérios de avaliação de desempenho.

- 2) No estudo de SO no módulo Gerência de Memória
 - a) Implementar uma opção onde possam ser simuladas as estratégias de substituição de páginas (ou segmentos).
 - b) Estudar a relação entre tamanho do endereço (virtual e real), tamanho do frame, quantidade de registros na Tabela de Páginas e quantidade de páginas.

- 3) No estudo de SO no módulo Gerência de Dispositivos de E/S
 - a) No estudo do relacionamento entre níveis de FMS mostrando a função específica de controle controladora e a função do SO na solicitação de operação de E/S para os processos e recebimento de retorno dessa solicitação.

- 4) No estudo de outras disciplinas que tenham de assuntos que também têm as características de serem difíceis de analisar por parte do aluno. Por exemplo, o estudo do ciclo de execução de instruções da disciplina Organização de Computadores.

- 5) Desenvolver um módulo completo de ajuda (HELP, dinâmico) para tirar as dúvidas que o usuário possa ter durante a manipulação do software e também que possa esclarecer as dúvidas mais frequentes.

Assim como qualquer outro trabalho de simulação, o usuário deve estar atento a busca da instrução indicada pelo cursor de programa de um determinado endereço de memória física para o Registro de Endereços de Memória. Em seguida, poderá ser mostrado o relacionamento entre os elementos de CPU (como o Registrador, Unidade de Controle, Registradores de endereço, etc.) no momento da instrução e busca dos dados que esta manipula (dependendo do modo de endereçamento das operações) e a instrução (dependendo do modo de endereçamento das operações).

Com o desenvolvimento do protótipo conclui-se ser viável a implementação de um *software* seguindo a modalidade de simulação e pretende-se concluí-lo com todas as funcionalidades mencionadas anteriormente.

A próxima etapa então será definir-se um conjunto de procedimentos de validação de *software* junto a um grupo de usuários (professores e alunos), e ao utilizá-lo como ferramenta de apoio didático nas atividades desenvolvidas durante as aulas da disciplina Sistemas Operacionais, realizar análise dos sistema quanto a aspectos como operacionalidade, eficiência, manutenibilidade, portabilidade, rentabilidade, integridade, fidedignidade e legibilidade, buscando aprimorá-lo.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, Maria Célia de. MASETTO, Marcos T. **O professor universitário em aula**. 8.ed. São Paulo: MG Ed. Associados. 1990.

ALBUQUERQUE, Fernando. **Projeto de Sistemas Operacionais em Linguagem C**. Rio de Janeiro: EBRAS-Berkeley, 1990.

ALMEIDA, Maria Aparecida Fernandes. Aprender, atividade inteligente: e se esta inteligência for parcialmente artificial? **Dissertação de Mestrado**. UFSC. Florianópolis, 1999. [on line] Disponível em <http://www.inf.ufsc.br/~mafa> [Capturado em 03/09/2000].

ANIDO, Ricardo de Oliveira. Uma proposta de plano pedagógico para a matéria Sistemas Operacionais. **Anais do II Curso de qualidade de cursos de graduação da área de computação e informática**. SBC – Sociedade Brasileira de Computação. Curitiba, 2000.

AZEVEDO, Breno Fabrício Terra. **Tópicos em Construção de Software Educacional**. UFES. Vitória, 1997. [on line] Disponível em <http://www.inf.ufes.br/~tavares/trab3.html> [Capturado em 22/01/2001].

BARRETO, Jorge Muniz. **Inteligência Artificial no limiar do século XXI**. Jorge Muniz Barreto. Florianópolis: J.M.Barreto, 1999.

FANTAUZZI, Elizabeth. A informática e o desenho geométrico: uma proposta construtivista. **3º Encontro de Informática na Educação**. SENAC, São Paulo: 2000.

FIGUEIREDO, Wânia Meira Matos, BRITO, Gláucia da Silva. **Metodologia Para Produção De Softwares Educacionais: Um Estudo de Caso no Curso de Mecânica**. [on line] Disponível em <http://dainf.cefetpr.br/~figuered/isenac.html> [Capturado em 08/03/2001].

FONTES, Carlos. **Navegando na Filosofia: Escolas, Saberes e Tecnologia**. [on line] Disponível em <http://www.terravista.pt/guincho/5198/> [Capturado em 25/03/2001].

FUNDAMENTOS DE SISTEMAS OPERACIONAIS (FSO). Área Técnico-operacional – RENPAC – Básico. Departamento de Treinamento. EMBRATEL – Empresa do Sistema TELEBRÁS. 1990.

GALVIS, Álvaro H. Software Educativo Multimídia: aspectos críticos no seu ciclo de vida. **Revista Brasileira de Informática na Educação**. Florianópolis, SBC, n.1, set.1997.

GARCIA, Islene C. REZENDE, Pedro J de. CALHEIROS, Felipe C. Astral: um ambiente para ensino de estruturas de dados através de animações de algoritmos. **Revista Brasileira de Informática na Educação**. n.1. set./1997. [on line] Disponível em <http://www.inf.ufsc.br/sbc-ie/revista/nr1> [Capturado em 27/03/2001].

GIRAFFA, Lúcia M.M. Uma arquitetura de tutor utilizando estados mentais. **Tese de Doutorado**. Porto Alegre: CPGCC/UFRGS, 1999.

GRAVINA, Maria Alice. SANTAROSA, Lucila Maria Costi. A aprendizagem da matemática em ambientes informatizados. **Informática na Educação: teoria e prática**. Curso de Pós-Graduação em Informática na Educação. vol.1, n.1 out.1998. Porto Alegre: UFRGS. p.73-88.

HOLT, R.C. et all. **Structured Concurrent Programming with Operating Systems Applications**. Addison-Wesley Publishing Company. Toronto, 1978.

LOPES, Josiane. Vygotsky – o teórico social da inteligência. **Revista Nova Escola**. Dez.1996.p.33-38.

MACHADO, Francis Berenger, MAIA, Luiz Paulo. **Introdução à Arquitetura de Sistemas Operacionais**. Rio de Janeiro: LTC – Livros Técnicos e Científicos, 1992.

MASETTO, Marcos Tarciso. Professor universitário, um profissional da educação na atividade docente. **Docência na universidade**. Campinas: Papyrus, 1998. p.9-26.

MONTEIRO, Mário A. **Introdução a Organização de Computadores**. 2.ed. Rio de Janeiro: LTC, 1995.

MORAES, Roque. Concepções de aprender de professores de terceiro grau. **Revista da ADPPUC** (Associação dos professores e pesquisadores da Pontifícia Universidade Católica do Rio Grande do Sul). Porto Alegre. n.1. out.1999.

MOREIRA, Marco Antonio. **Ensino e Aprendizagem** - enfoques teóricos. 3.ed. São Paulo: MORAES Ltda, 1981.

NUTT, Gary. **Operating Systems: a modern perspective**. Miami: Addison-Wesley, 1997.

O QUE É LINUX? [on line] Disponível em <http://www.emporio-LINUX.com.br/sobreLINUX/oqueeLINUX.html> [Capturado em 03/01/2001].

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

RAMOS, Edla Maria Faust RAMOS. Análise ergonômica do sistema HiperNet buscando aprendizado da cooperação e da autonomia. **Tese de Doutorado**. UFSC. Florianópolis, 1996. [on line] Disponível em <http://www.inf.ufsc.br/~edla/> [Capturado em 05/12/2000].

SEABRA, Carlos. O computador na criação de ambientes interativos de aprendizagem. **Em Aberto**. Brasília, ano 12, n.57, jan./mar., 1993. p.45-50.

SHAY, William A. **Sistemas Operacionais**. São Paulo: Makron Books, 1996.

SILBERSCHATZ, Abraham, GALVIN, Peter Baer. **Sistemas Operacionais: conceitos**. São Paulo: Prentice Hall, 2000.

SILVA, Edna Lúcia da, MENEZES, Estera Muszkat. **Metodologia da pesquisa e elaboração de dissertação**. Florianópolis: Laboratório de Ensino a Distância da UFSC, 2000.

SOCIEDADE DA INFORMAÇÃO NO BRASIL: Livro Verde. Brasília: Ministério da Ciência e Tecnologia, 2000.

TAJRA, Sanmya Feitosa. **Informática na Educação: novas ferramentas pedagógicas para o professor da atualidade**. 2.ed. São Paulo: Érica, 2000.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. Rio de Janeiro: Prentice Hall, 1995.

TANENBAUM, Andrew S. **Structure Computer Organization**. 4.ed. New Jersey: Prentice Hall. 1999.

TANENBAUM, Andrew S. WOODHULL, Albert S. **Sistemas Operacionais: projeto e implementação**. 2.ed. Porto Alegre: Bookman, 2000.

TERRA JUNIOR, Osvaldo Gomes. **A Educação e a Informática**. UFES. Vitória, 1997. [on line] Disponível em <http://www.inf.ufes.br/~tavares/trab1.html> [Capturado em 22/01/2001].

VALENTE, José Armando. Diferentes usos do computador na educação. **Em Aberto**. Brasília, ano 12, n.57, jan./mar. 1993. p.3-16.