UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

Andreu Carminati

SINCRONIZAÇÃO DE PROCESSOS EM SISTEMAS DE TEMPO REAL NO CONTEXTO DE MULTIPROCESSADORES

Florianópolis

2012

Andreu Carminati

SINCRONIZAÇÃO DE PROCESSOS EM SISTEMAS DE TEMPO REAL NO CONTEXTO DE MULTIPROCESSADORES

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Rômulo Silva de Oliveira, Dr.

Florianópolis

2012

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Carminati, Andreu
Sincronização de processos em sistemas de tempo real no contexto de multiprocessadores [dissertação] / Andreu Carminati ; orientador, Rômulo Silva de Oliveira - Florianópolis, SC, 2012.
166 p. ; 21cm

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Tempo Real. 3. Sincronização. 4. Multiprocessadores. I. Oliveira, Rômulo Silva de. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Andreu Carminati

SINCRONIZAÇÃO DE PROCESSOS EM SISTEMAS DE TEMPO REAL NO CONTEXTO DE MULTIPROCESSADORES

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Rômulo Silva de Oliveira, Dr.

Florianópolis

2012

Andreu Carminati

SINCRONIZAÇÃO DE PROCESSOS EM SISTEMAS DE TEMPO REAL NO CONTEXTO DE MULTIPROCESSADORES

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Engenharia de Automação e Sistemas", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 20 de novembro 2012.

Jomi Fred Hübner, Dr.
Coordenador

Banca Examinadora:

Rômulo Silva de Oliveira, Dr.
Orientador

Raimundo Barreto, Dr.

Patricia Della Méa Plentz, Dra.

Luís Fernando Friedrich, Dr.

RESUMO

Sistemas computacionais de tempo real são identificados como aqueles sistemas computacionais submetidos a requisitos de natureza temporal. Nestes sistemas, os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. Em tais sistemas, quando existe o compartilhamento de recursos, a sincronização de tarefas no acesso a estes é fundamental para garantir tanto a exclusão mútua, quanto a limitação no tempo de espera (evitando inversões de prioridade descontroladas), visto que isto pode induzir as tarefas a gerarem seus resultados em momentos incorretos (perda de *deadlines* ou não atendimento de algum requisito temporal). O não atendimento de um requisito temporal pode resultar em consequências catastróficas tanto no sentido econômico quanto em vidas humanas, dependendo do tipo de sistema. Para sincronização em multiprocessadores, alguns protocolos para escalonamento particionado foram propostos, como o *Multiprocessor Priority Ceiling Protocol* (MPCP), *Flexible Multiprocessor Locking Protocol* (FMLP) e o *Multiprocessor Stack Resource Policy* (MSRP). Neste contexto, esta dissertação de mestrado propõe duas variações para o protocolo MPCP, com as devidas análises de escalonabilidade e fatores de bloqueios associados. No entanto, ambas as variações podem ser encaradas também como variações do FMLP, dependendo do ponto de vista, pois abrangem características comuns a ambos os protocolos. Tais variações são o MPCP não preemptivo e MPCP com enfileiramento FIFO. Esta dissertação também apresenta comparações empíricas entre as propostas apresentadas e os protocolos existentes. Como resultados gerais, as propostas apresentadas se mostraram competitivas tanto em escalonabilidade quanto em *overhead* de implementação. O MPCP com enfileiramento FIFO se posicionou muito bem em sua versão baseada em suspensão. O MPCP não preemptivo, em sua versão baseada em *spin* obteve resultados muito próximos ao FMLP *short*. Do ponto de vista prático, as variações propostas facilitam a utilização em sistemas reais, quando comparadas com as propostas originais.
Palavras-chave: tempo real; sincronização; multiprocessadores

ABSTRACT

Real-time computational systems are identified as those systems subjected requirements of temporal nature. In these systems, the results should be correct not only from logical point of view, but also must be generated at the right time. In such systems, when there is resource sharing, tasks synchronization on access to these resources is essential to ensure both mutual exclusion and limitation of the waiting time (avoiding uncontrolled priority inversion), as this may induce the tasks to generate their results at wrong times (miss of deadlines or not meet a time requirement). A failure to meet a time requirement can result in catastrophic consequences both in the economic sense and in human lives, depending on the type of the system. For synchronization in multiprocessors, some protocols have been proposed for partitioned scheduling, as the Multiprocessor Priority Ceiling Protocol (MPCP), Flexible Multiprocessor Locking Protocol (FMLP) and Multiprocessor Stack Resource Policy (MSRP). In this context, this dissertation proposes two changes to the MPCP protocol, with appropriate schedulability analysis and blocking factors associated. However, both variants can also be viewed as variations of FMLP, depending on the point of view, because they cover features common to both protocols. Such variations are the nonpreemptive MPCP and the MPCP with FIFO queuing. This dissertation also presents empirical comparisons between the previously mentioned well-known protocols and the proposed variations. As general results, the proposals have showed to be competitive in both schedulability and implementation overhead. The MPCP with FIFO queuing was positioned very well in the suspension-based version. The nonpreemptive MPCP, in his spin-based version obtained results very close to the FMLP short. From a practical standpoint, the proposed changes facilitate the implementation in real systems, when compared to the original proposals.
Keywords: real-time; synchronization; multiprocessors

LISTA DE FIGURAS

Figura 1 Inversão de prioridade descontrolada 50
Figura 2 Exemplo de bloqueio local 55
Figura 3 Exemplo de bloqueio remoto 56
Figura 4 Exemplo de suspensão de tarefa ocasionada por bloqueio 57
Figura 5 Espera por *spin* 58
Figura 6 Enfileiramento FIFO 59
Figura 7 Enfileiramento por prioridade 60
Figura 8 Exemplo de tarefas compartilhando recurso pelo MPCP 77
Figura 9 Exemplo de tarefas compartilhando recurso pelo MSRP 83
Figura 10 Exemplo de tarefas compartilhando recurso pelo FMLP *long* . 90
Figura 11 Exemplo de tarefas compartilhando recurso pelo MPCPNP . . . 105
Figura 12 Variação do tamanho das seções críticas 126
Figura 13 Variação do tamanho das seções críticas (desvio padrão relativo) 127
Figura 14 Resultados individuais para cada um dos 30 conjuntos de tarefas do Experimento 1 (seções críticas de 5μs) 128
Figura 15 Resultados individuais para cada um dos 30 conjuntos de tarefas do Experimento 1 (seções críticas de 1280μs) 129
Figura 16 Variação do número de tarefas por processador 130
Figura 17 Variação do número de tarefas usuárias por recurso 132
Figura 18 Variação da utilização dos conjuntos 133
Figura 19 Exemplos de escala de acesso a um determinado recurso 139

LISTA DE TABELAS

Tabela 1 Piores casos no tempo de resposta (exemplo com MPCP) 79
Tabela 2 Piores casos no tempo de resposta (exemplo com FMLP) 91
Tabela 3 Comparação entre os protocolos 99
Tabela 4 Comparação entre os protocolos incluindo as propostas 102
Tabela 5 Piores casos no tempo de resposta (exemplo com MPCPNP) . . 112
Tabela 6 Piores casos no tempo de resposta (exemplo com MPCPF) . . . 117
Tabela 7 Análise subjetiva dos resultados 134
Tabela 8 Configuração das tarefas utilizadas no experimento 140
Tabela 9 Dados estatísticos básicos do experimento (media +- desvio padrão) com 10000 amostras para cada configuração e resultados em ciclos de *clock* 141
Tabela 10 Comparação das amostras obtidas em cada configuração do experimento através do teste estatístico Mann-Whitney-Wilcoxon (protocolos baseados em suspensão) 143
Tabela 11 Comparação das amostras obtidas em cada configuração do experimento através do teste estatístico Mann-Whitney-Wilcoxon (protocolos baseados em *spin*) 143
Tabela 12 Experimento 1: utilizando protocolos baseados em suspensão. A configuração representa o tamanho das seções críticas em μs 161
Tabela 13 Experimento 1: utilizando protocolos baseados em *spin*. A configuração representa o tamanho das seções críticas em μs 161
Tabela 14 Experimento 2: utilizando protocolos baseados em suspensão. A configuração representa o número de tarefas nos conjuntos utilizados . . 162
Tabela 15 Experimento 2: utilizando protocolos baseados em *spin*. A configuração representa o número de tarefas nos conjuntos utilizados 162
Tabela 16 Experimento 3: utilizando protocolos baseados em *suspensão*. A configuração representa o número de tarefas usuárias de cada recurso . 163
Tabela 17 Experimento 3: utilizando protocolos baseados em *spin*. A configuração representa o número de tarefas usuárias de cada recurso 163
Tabela 18 Experimento 4: utilizando protocolos baseados em suspensão. A configuração representa a utilização total dos conjuntos utilizados 163
Tabela 19 Experimento 4: utilizando protocolos baseados em *spin*. A configuração representa a utilização total dos conjuntos utilizados 164

LISTA DE SÍMBOLOS

τ_i	Representa a tarefa i
C_i	Representa o tempo de computação da tarefa τ_i
D_i	Representa o *deadline* da tarefa τ_i
T_i	Representa o período da tarefa τ_i
U_i	Representa a utilização da tarefa τ_i
NC_i	Representa o número total de seções críticas da tarefa τ_i
$NC_{i,h}$	Representa o número de seções críticas que a tarefa τ_i compartilha com a tarefa τ_h
$\beta_{i,h}$	Mais longa entre as $NC_{i,h}$ seções críticas
$v_{i,j}$	Função de atração de uma tarefa τ_i em relação a tarefa τ_j
$s(i)$	Número de segmentos normais de execução de τ_i
$C_{i,j}$	WCET do j-ésimo segmento de execução normal
$C'_{i,j}$	WCET do j-ésimo segmento de seção crítica
$\tau_{i,j}$	j-ésimo segmento de execução normal de uma tarefa τ_i
$\tau'_{i,j}$	j-ésimo segmento de seção crítica de uma tarefa τ_i
$P(\tau_i)$	É o processador que executa a tarefa τ_i
W_i	*Worst-case execution time* total da tarefa τ_i
$W_{i,j}$	Representa o WCRT do j-ésimo segmento de execução normal
$W'_{i,j}$	Representa o WCRT do j-ésimo segmento de seção crítica
B_i^r	Bloqueio remoto total sofrido pela tarefa τ_i
I_i^n	Interferência de escalonamento causada à tarefa τ_i
B_{low}	Bloqueio causado por tarefas de mais baixa prioridade
P_G	Define o *ceiling* base do sistema
PS_{G_i}	Definida para cada recurso, em cada processador P_i, como sendo a mais alta das prioridades entre as tarefas que acessam o recurso e estão alocadas a processadores diferentes de P_i
R_i	Representa o recurso i
S_i	Representa o semáforo relativo ao recurso R_i
λ_i	*Preemption threshold* da tarefa τ_i
π_i	Prioridade nominal da tarefa τ_i

LISTA DE ABREVIATURAS E SIGLAS

BF	*Best-fit*
BFD	*Best-fit decreasing utilization*
BPA	*Blocking-Aware Partitioning Algorithm*
DM	*Deadline Monotonic*
DU	*Decreasing Utilization*
EDF	*Earliest Deadline First*
EDZL	*Earliest Deadline Firs Until Zero Laxity*
EKG	*EDF with task splitting and k processors in a group*
E/S	Entrada e Saída
FF	*First-fit*
FIFO	*First In, First Out*
FMLP	*Flexible Multiprocessor Locking Protocol*
FMLP+	*Flexible Multiprocessor Locking Protocol Improved* ou *FIFO Multiprocessor Locking Protocol*
IPCP	*Immediate Priority Ceiling Protocol*
LITMUS^{RT}	*Linux Testbed for Multiprocessor Scheduling in Real-Time systems*
LLF	*Least Laxity First*
MPCP	*Multiprocessor Priority Ceiling Protocol*
MPCPF	*Multiprocessor Priority Ceiling Protocol* com enfileiramento FIFO
MPCPNP	*Multiprocessor Priority Ceiling Protocol* não preemptivo
MSRP	*Multiprocessor Stack Resource Policy*
NF	*Next-fit*
NUMA	*Non-Uniform Memory Access*
P-SP	*Partitioned Static-Priority*
PCP	*Priority Ceiling Protocol*
Pfair	*Proportionate Fair*
PI	*Priority Inheritance*
RM	*Rate Monotonic*
RUN	*Reduction to Uniprocessor*
SPA	*Synchronization-Aware Partitioning Algorithm*

SRP	*Stack Resource Policy*
WCRT	*Wors-case Response Time*

SUMÁRIO

1 INTRODUÇÃO 25
1.1 CONCEITOS BÁSICOS E MOTIVAÇÃO 26
1.2 OBJETIVO DO TRABALHO 28
1.3 ORGANIZAÇÃO DO TEXTO 29
2 FUNDAMENTAÇÃO SOBRE SISTEMAS DE TEMPO REAL . . 31
2.1 ESCALONAMENTO DE TAREFAS EM SISTEMAS DE TEMPO REAL 31
2.2 ABORDAGENS PARA VERIFICAÇÃO DE ESCALONABILIDADE 32
2.3 ESCALONAMENTO EM SISTEMAS MONOPROCESSADOS . . 33
2.3.1 Modelos de Sistema 33
2.3.2 Com Executivo Cíclico 35
2.3.3 Com Prioridade Fixa 35
2.3.4 Com Prioridade Dinâmica 38
2.4 ESCALONAMENTO EM SISTEMAS MULTIPROCESSADOS . . 38
2.4.1 Modelos de Sistema 40
2.4.2 Escalonamento Particionado 40
2.4.3 Escalonamento Global 42
2.4.4 Escalonamento Híbrido 45
2.5 GERAÇÃO DE TAREFAS PARA AVALIAÇÃO EMPÍRICA DE TESTES DE ESCALONABILIDADE 46
2.5.1 Geração de Utilizações para Sistemas Monoprocessados 47
2.5.2 Geração de Utilizações para Sistemas Multiprocessados 47
2.6 CONCLUSÕES 47
3 SINCRONIZAÇÃO EM SISTEMAS DE TEMPO REAL COM PRIORIDADE FIXA 49
3.1 SINCRONIZAÇÃO EM MONOPROCESSADORES 49
3.1.1 Inversões de Prioridades 50
3.1.2 Protocolos para Sistemas Monoprocessados 51
3.1.2.1 Prevenção de Inversões de Prioridade 51
3.1.2.2 *Priority Inheritance* 51
3.1.2.3 *Priority Ceiling Protocol* 51
3.1.2.4 *Immediate Priority Ceiling Protocol* 52
3.1.2.5 *Stack Resource Policy* 53
3.2 SINCRONIZAÇÃO EM MULTIPROCESSADORES 54
3.2.1 Modelo de Sistema e Convenções 54
3.2.2 Tipos de Recursos em Sistemas Multiprocessados 54

3.2.3 Tipos de bloqueio em Sistemas Multiprocessados 55
3.2.4 Políticas de Controle de Execução 56
3.2.5 Políticas de Escolha/Enfileiramento de Tarefas 58
3.2.6 Algoritmos de Particionamento Cientes de Recursos 60
3.2.6.1 Synchronization-Aware Partitioning Algorithm 61
3.2.6.2 Blocking-Aware Partitioning Algorithm 63
3.2.7 Cálculo do Tempo de Resposta Considerando Bloqueios 66
3.2.8 Protocolos para Sistemas Multiprocessados 71
3.2.8.1 *Multiprocessor Priority Ceiling* Distribuído 71
3.2.8.2 *Multiprocessor Priority Ceiling* para Memória Compartilhada . 72
3.2.8.3 *Multiprocessor Stack Resource Policy* para Prioridade Fixa 79
3.2.8.4 *Flexible Multiprocessor Locking Protocol* - FMLP 83
3.2.8.5 *Flexible Multiprocessor Locking Protocol Improved* 90
3.2.9 Comparação Entre os Protocolos 91
3.2.9.1 Comparação do MSRP com o MPCP 92
3.2.9.2 Comparação do FMLP com o MPCP 94
3.2.9.3 Comparação de Diferentes Políticas de Controle de Execução no MPCP 95
3.3 CONCLUSÕES 96
4 PROPOSTAS DE VARIAÇÕES PARA O PROTOCOLO *MULTIPROCESSOR PRIORITY CEILING* 101
4.1 VARIAÇÃO PROPOSTA 1: MPCP NÃO PREEMPTIVO 101
4.1.1 Regras de Aquisição de Recursos 103
4.1.2 Bloqueios do Protocolo 106
4.1.3 Análise de Escalonabilidade 107
4.1.4 Exemplo 110
4.2 VARIAÇÃO PROPOSTA 2: MPCP COM ENFILEIRAMENTO FIFO 112
4.2.1 Cálculo dos *Ceilings* dos Recursos 113
4.2.2 Regras de Aquisição de Recursos 114
4.2.3 Bloqueios do Protocolo 114
4.2.4 Análise de Escalonabilidade 115
4.2.5 Exemplo 117
4.3 CONCLUSÕES 118
5 AVALIAÇÃO EMPÍRICA DA PROPOSTA 119
5.1 COMPARAÇÃO EMPÍRICA DA ESCALONABILIDADE 119
5.1.1 Condições dos Experimentos 121
5.1.2 Algoritmo de Alocação 121
5.1.3 Geração dos Conjuntos de Tarefas 123
5.1.4 Realização dos Experimentos 123
5.1.4.1 Experimento 1: Variação do Tamanho das Seções Críticas 125

5.1.4.2 Experimento 2: Variação do Número de Tarefas nos Conjuntos . 129
5.1.4.3 Experimento 3: Variação do Número de Tarefas Usuárias por Recurso 131
5.1.4.4 Experimento 4: Variação da Utilização dos Conjuntos 132
5.1.4.5 Resultados 134
5.2 IMPLEMENTAÇÃO DOS PROTOCOLOS NO LINUX PREEMPT-RT 135
5.3 AVALIAÇÃO EMPÍRICA DOS *OVERHEADS* 138
5.3.1 Condições dos Experimentos 140
5.3.2 Análise dos Dados 141
5.4 CONCLUSÕES 144
6 CONSIDERAÇÕES FINAIS 147
REFERÊNCIAS 153
APÊNDICE A – Resultados Numéricos dos Experimentos 161

1 INTRODUÇÃO

Sistemas computacionais de tempo real são identificados como aqueles sistemas computacionais submetidos a requisitos de natureza temporal (FARINES et al., 2000). Nestes sistemas, os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. As falhas de natureza temporal nestes sistemas são, em alguns casos, consideradas críticas no que diz respeito às suas consequências. Na literatura, os sistemas de tempo real são classificados conforme a criticalidade dos seus requisitos temporais (FARINES et al., 2000). Nos sistemas tempo real críticos (*hard* real-time) o não atendimento de um requisito temporal pode resultar em consequências catastróficas tanto no sentido econômico quanto em vidas humanas. Quando os requisitos temporais não são críticos (*soft* real-time) eles apenas descrevem o comportamento desejado. O não atendimento de tais requisitos reduz a utilidade da aplicação mas não a elimina completamente nem resulta em consequências catastróficas.

Em tais sistemas, quando a premissa de independência de tarefas não puder ser atendida, a sincronização destas no acesso a recursos é fundamental para garantir tanto a exclusão mútua, quanto a limitação no tempo de espera (evitando inversões de prioridade descontroladas), visto que isto pode induzir a perdas de *deadline*. Em sistemas multiprocessados, sincronização é igualmente importante, sendo ainda mais difícil de ser abordada. A utilização de protocolos de sincronização têm por objetivo suprir dificuldades de se obter sistemas escalonáveis (sistemas onde todas as tarefas possuem seus *deadlines* garantidos) na presença de recursos com acesso exclusivo. Foi provado por Mok (MOK, 1983) que é impossível obter uma escala *online* de execução ótima (no sentido de prover garantias) para um conjunto de tarefas na presença de semáforos/*mutexes* tradicionais. Mok também estendeu o teorema para sistemas multiprocessados. Protocolos de sincronização atacam exatamente a problemática levantada por Mok quanto a impossibilidade de escalonamento sob a presença de exclusão mútua levando em consideração algum modelo específico de sistema, como escalonamento particionado/global e prioridade

fixa/dinâmica. O problema da exclusão mútua foi exaustivamente estudado para sistemas monoprocessados, entretanto, para sistemas multiprocessados, este assunto ainda continua com vertentes em aberto, tendo algumas opções ainda por serem exploradas.

Estudar a problemática da sincronização de processos/tarefas em sistemas multiprocessados no contexto dos sistemas operacionais de tempo real possui muitas abordagens possíveis, algumas com fila única de tarefas (escalonamento global), outras com separação completa das tarefas entre os processadores (escalonamento particionado) (DAVIS; BURNS, 2009). Também é possível usar suspensão ou *spin* (*busy-wait*) (BRANDENBURG et al., 2008). É preciso notar que a memória *cache* do computador tem um grande impacto nos tempos de execução das tarefas, e a política de alocação de recursos adotada pode maximizar ou minimizar os ganhos obtidos pelo uso de memória *cache*.

Com o desenvolvimento deste trabalho, pretende-se contribuir com o tema sincronização, através da proposta de novos protocolos de sincronização baseados em propostas já existentes. Também faz parte da contribuição, estudos empíricos para a comparação de tais propostas com as já existentes. A contribuição se focará em um modelo específico de sistema, que é escalonamento particionado e prioridade fixa, pois boa parte dos estudos envolvendo o tema sincronização se baseia neste tipo de sistema. Outro motivo é que muitos sistemas operacionais de tempo real modernos são dirigidos por prioridade fixa, e alguns deles com escalonamento particionado, como por exemplo o Linux/PREEMPT-RT.

1.1 CONCEITOS BÁSICOS E MOTIVAÇÃO

Em aplicações de controle ou monitoração industrial, automobilística ou aviônica, cada função executada pelo sistema é associada a uma ou a um subconjunto de tarefas (LIU; LAYLAND, 1973). Algumas destas tarefas executam em resposta a eventos externos, enquanto outras a eventos gerados por outras tarefas do sistema ou por temporizadores. Tarefas que são ativadas e executadas em resposta a eventos externos são classificadas como aperiódi-

cas ou esporádicas. Tarefas aperiódicas são associadas a eventos os quais não podem ser previstos temporalmente (quanto o evento pode ou não ocorrer). Quando se sabe que o evento tem um período mínimo entre ocorrências, a este está associado uma tarefa esporádica. Quanto uma tarefa é ativada, ela precisa executar uma série de instruções ou computação (associada a função exercida pela tarefa), cujo tempo de execução no pior caso (*Worst-case execution time*) é denotado por C. Quando existe um limite ou restrição temporal para a tarefa executar sua computação, este é denotado por D, que é o *deadline* da tarefa.

Em geral, tarefas ativadas por temporizadores são chamadas de periódicas. O intervalo de ativação de uma tarefa periódica é chamado de período e é denotado por T. Quando o *deadline* de uma tarefa é igual ao período diz-se que este é implícito, quando é maior é chamado de arbitrário e quando é menor é chamado de limitado. O modelo mais comum de tarefas utilizado tanto na literatura quanto em sistemas reais é o de tarefas periódicas com *deadlines* implícitos.

Em geral, boa parte da teoria de escalonamento adota a premissa de independência entre as tarefas, ou seja, sem relações de precedência (dependência de mensagens trocadas por *rendezvous*, por exemplo) ou bloqueios no acesso a recursos mutuamente exclusivos. Na maioria dos sistemas reais, entretanto, a premissa de independência das tarefas não é válida, dado que podem existir recursos compartilhados cujo acesso deve ser serializado por questões de integridade. Recursos compartilhados podem ser estruturas de dados residentes em memória ou dispositivos de *hardware*, como os de armazenamento de dados ou portas/linhas de E/S. Nestes sistemas, tarefas de alta prioridade podem ser postergadas indefinidamente por conta de tarefas de mais baixa prioridade que estejam de posse de algum recurso necessário, cujo acesso deve ser exclusivo. Este efeito é conhecido como inversão de prioridade, e é considerado descontrolado se não é possível calcular o seu tempo máximo de duração. O uso de protocolos de sincronização permite que recursos sejam compartilhados em sistemas de tempo real de forma que a escalonabilidade do sistema possa ser garantida perante alguma estratégia de escalonamento. Cada protocolo existente apresenta suas particularidades no que diz respeito a inserção nos testes de escalonabilidade, ou mesmo na im-

plementação em sistemas reais. Atualmente existem diversos protocolos para sistemas multiprocessados, como o *Multiprocessor Priority Ceiling Protocol (MPCP)* (RAJKUMAR et al., 1988) (MPCP). O MPCP é um protocolo de sincronização para sistemas multiprocessados com escalonamento particionado e prioridade fixa, embora existam adaptações para outros modelos de sistema (CHEN; TRIPATHI, 1994). O escalonamento particionado é empregado em sistemas como o Linux, onde as tarefas são estaticamente alocadas a processadores (podendo eventualmente serem migradas para outro processador através do balanceamento de carga, comportamento que pode ser inibido por máscaras de afinidades das tarefas/*cpu affinity* (LOVE, 2009)). Outros protocolos muito conhecidos são o *Multiprocessor Stack Resource Policy* (MSRP) (GAI et al., 2001a) e *Flexible Multiprocessor Locking Protocol* (FMLP) (BLOCK et al., 2007).

A motivação por trás deste tema está no fato deste não estar esgotado. Existem diversos protocolos propostos, mas nenhum domina os outros em todas as situações. Ainda existem diversas questões em aberto, como combinações de características de diferentes protocolos.

1.2 OBJETIVO DO TRABALHO

O objetivo geral deste trabalho é propor variações para o protocolo *Multiprocessor Priority Ceiling Protocol* (MPCP) (RAJKUMAR et al., 1988), com o objetivo de melhorar o escalonamento e facilitar possíveis implementações. Tais variações são baseadas em combinações de características ainda não exploradas pela literatura sobre sincronização. Também faz parte do trabalho a validação experimental das propostas.

Para realização dos objetivos anteriores, será utilizada a seguinte metodologia:

- O estudo dos protocolos existentes para sincronização em sistemas multiprocessados com escalonamento particionado e prioridade fixa. Esta dissertação inclui um amplo levantamento bibliográfico das abordagens existentes na literatura para este problema, considerando o modelo de

sistema adotado. Este estudo inclui protocolos de sincronização, algoritmos de particionamento e análise de escalonabilidade na presença de recursos;

- A apresentação de propostas de modificações sobre o protocolo *Multiprocessor Priority Ceiling Protocol*, baseadas nas características fundamentais dos protocolos de sincronização, como alterações nas políticas de enfileiramento em caso de bloqueio, preemptividade de seções críticas e mecanismos de controle de execução em caso de bloqueio de tarefas devido a recursos indisponíveis.

- A apresentação de estudos e experimentos comparativos entre as propostas apresentadas e os protocolos existentes para o mesmo modelo de sistema. Os estudos comparativos envolvem a avaliação de cada protocolo (escalonabilidade) perante cenários de conjuntos de tarefas gerados de forma sintética. Outro estudo comparativo apresentado neste trabalho envolve a medição do *overhead* de implementação de uma das propostas (MPCP não preemptivo) em comparação com o protocolo FMLP, que é um dos protocolos do estado da arte do tema sincronização. Para o último estudo, foi efetuada a implementação dos protocolos, que também serviu como prova de conceito para a proposta. Tanto os experimentos quanto a implementação servem também para justificar e validar as propostas apresentadas.

1.3 ORGANIZAÇÃO DO TEXTO

Esta dissertação está organizada da seguinte maneira: o capítulo 2 apresenta uma introdução a sistemas de tempo real. O capítulo 3 apresenta um levantamento bibliográfico sobre sincronização, bem como o estado da arte relacionado ao assunto. O capítulo 4 apresenta as propostas de modificações para o *Multiprocessor Priority Ceiling Protocol* com as devidas análises de escalonabilidade. O capítulo 5 apresenta comparações empíricas entre as variações propostas e as já existentes, bem como uma implementação realizada como prova de conceito. Finalmente o capítulo 6 apresenta as conclusões e

considerações finais desta dissertação.

2 FUNDAMENTAÇÃO SOBRE SISTEMAS DE TEMPO REAL

Este capítulo apresenta uma fundamentação sobre sistemas de tempo real em sistemas monoprocessados e multiprocessados. Sistemas multiprocessados (*multicore*) surgiram para suprir a demanda de performance em processamento, visto que a estratégia de miniaturização para elevação da frequência de *clock* apresentou diversos problemas, como alto consumo energético, *pipelines* extremamente longos e temperatura elevada. Em sistemas de tempo real, esta tendência de utilização de sistemas multiprocessados em aplicações reais também têm aumentado, embora o suporte teórico tenha sido iniciado a muito tempo atrás, em trabalhos como (LIU; LAYLAND, 1973). Escalonamento em sistemas multiprocessados difere de escalonamento em sistemas monoprocessados por apresentar anomalias e mais situações de não determinismo.

2.1 ESCALONAMENTO DE TAREFAS EM SISTEMAS DE TEMPO REAL

O escalonamento de tempo real é o processo de definir ordens de execução de tarefas (representadas por conjuntos) em um ou mais processadores, com o objetivo de que todas cumpram com suas restrições temporais. Em geral, as tarefas em sistemas de tempo real são recorrentes, ou seja, executam periódica ou ciclicamente. A cada execução de uma determinada tarefa dá-se o nome de *job*, então, uma tarefa pode ser definida como uma sequência possivelmente infinita de *jobs*.

Um conjunto de tarefas é dito *feasible*, com respeito a um dado tipo de sistema, se existir algum algoritmo que escalone todos os *jobs* de todas as tarefas (em todas as possíveis sequências), com todas as restrições temporais sendo garantidas. Um algoritmo de escalonamento é *ótimo* com relação a um tipo de sistema se é capaz de escalonar qualquer conjunto de tarefas que seja escalonado por qualquer outro algoritmo, ou de outra maneira, qualquer conjunto que seja *feasible*.

2.2 ABORDAGENS PARA VERIFICAÇÃO DE ESCALONABILIDADE

De um modo geral, independente do tipo de sistema e da abordagem de escalonamento, existem diferentes tipos de testes que podem prover garantias de que um dado sistema será escalonado com sucesso. Existem basicamente dois tipos de testes:

Suficiente: Um teste é dito suficiente se todo conjunto de tarefas que passar no teste for escalonável, mas nada pode ser dito se o conjunto não passar no teste (pode reprovar conjuntos escalonáveis).

Necessário: Um teste é dito necessário se for simples e não tão restritivo. Um teste necessário garante que se o conjunto de tarefas for reprovado, certamente ele não será escalonável, mas se for aprovado, nada se pode afirmar. Este tipo de teste é útil para descartar de forma prática conjuntos não escalonáveis. Um teste necessário para qualquer modelo de sistema é o de utilização:

$$U = \sum_{i=1}^{n} U_i \leq m \tag{2.1}$$

Onde, na Equação 2.1, U é a utilização total do sistema, U_i é a utilização da tarefa i (também denotada por τ_i) e m é o número de processadores. A utilização de uma tarefa i é dada pela Equação 2.2

$$U_i = \frac{C_i}{T_i} \tag{2.2}$$

Quando um teste é simultaneamente suficiente e necessário ele é dito exato. Exato pois, ele aprova todos os conjuntos que são efetivamente escalonáveis e reprova todos os não escalonáveis.

2.3 ESCALONAMENTO EM SISTEMAS MONOPROCESSADOS

Esta seção apresenta uma visão geral sobre tempo real em sistemas monoprocessados, com ênfase nas respectivas técnicas de escalonamento e modelos de sistema.

2.3.1 Modelos de Sistema

A principal característica de sistemas de tempo real é que eles possuem restrições que aparecem na forma de *deadlines*. Sistemas *hard* real-time são aqueles que possuem *deadlines* que jamais devem ser perdidos. A falha no cumprimento de algum *deadline* deve ser considerada um erro da aplicação. Existem também sistemas *hard* real-time do tipo *safety-critical* (BURNS, 1991) onde a perda de um *deadline* pode ser catastrófica, e se torna ainda mais severa conforme o tempo passa após a perda. Ainda existem aplicações onde a perda de *deadlines* não compromete a integridade do sistema, sendo estes considerados *soft deadlines*. Em sistemas de tempo real, é necessário que as restrições temporais sejam mapeadas em termos de *deadlines*. O *deadline* relativo (ao tempo de início) de uma tarefa τ_i é definido por D_i.

Em sistemas de tempo real existem também 3 formas distintas de estruturas de processos, mais conhecidos como tarefas, sendo elas:

Periódicas: que são tarefas executados de forma regular e periódica. São caracterizados pelo seu período, *deadline* (que geralmente é igual ao período) e o pior caso no tempo de execução ou caso médio (por período). O período de uma tarefa geralmente é denotado por T.

Aperiódicas: São tarefas ativadas aleatoriamente, seguindo por exemplo a distribuição de *Poisson*. Processos aperiódicos também podem ter restrições temporais em forma de *deadlines*. Como a frequência de ativação deste tipo de tarefa não é conhecida, não se pode efetuar uma análise de pior caso na presença destes, logo, tais tarefas não podem ter *deadlines* do tipo *hard*.

Esporádicas: Quando é necessário o cálculo de pior caso na presença de ta-

refas aperiódicas, pode-se assumir um período mínimo entre eventos aperiódicos, dando origem a tarefas esporádicas. Ou seja, tarefas esporádicas representam um subconjunto das tarefas aperiódicas, que apresentam, ou se pode assumir intervalo mínimo entre chegadas.

Também existem restrições utilizadas para descrever o comportamento das tarefas:

Tempo de Computação: é o tempo necessário para execução completa da tarefa. Em geral este parâmetro é denotado por C.

Tempo de Início: é o instante de início da execução de uma tarefa em uma determinada ativação.

Tempo de Término: é o instante de termino da execução de uma tarefa em uma determinada ativação.

Tempo de Chegada: representa o instante em que uma tarefa se torna apta à execução. Em tarefas periódicas este tempo é sempre o início de um determinado período. Em tarefas aperiódicas este tempo corresponde ao instante da solicitação do processamento aperiódico.

Tempo de liberação: tempo de inclusão de uma tarefa na fila de prontos. Este tempo pode coincidir com o tempo de chegada, caso não haja *jitter* de liberação. Em geral este parâmetro é denotado por J.

Desta forma, uma tarefa periódica τ_i tem seu comportamento definido pela quádrupla (J_i, C_i, T_i, D_i). Uma tarefa esporádica τ_i também tem seu comportamento definido por uma quádrupla, que neste caso é $(J_i, C_i, T_i^{min}, D_i)$, onde T_i^{min} é o tempo mínimo entre chegadas.

Em sistemas de tempo real, o escalonamento de tarefas para garantia de *deadlines* pode ser efetuado com algoritmos de prioridade fixa, dinâmica e ainda por estratégias de construção de escala de execução em tempo de projeto. As próximas subseções apresentam as abordagens.

2.3.2 Com Executivo Cíclico

A abordagem de escalonamento mais simples existente chama-se escalonamento por executivo cíclico (FARINES et al., 2000). Nesta abordagem, tanto a verificação da escalonabilidade quando a escala de execução são feitas em tempo de projeto. A escala de execução resume-se a uma tabela que indica qual tarefa deve ocupar o processador em determinado instante de tempo. Esta escala deve ter tamanho finito e deve ser cíclica, onde o fim da escala deve coincidir com início. A verificação da escalonabilidade do sistema ocorre na própria montagem da escala. Esta abordagem é simples de ser implementada, mas pode desperdiçar recursos (a escala reflete o pior caso) e até mesmo ser inviável para conjuntos com muitas tarefas.

2.3.3 Com Prioridade Fixa

Em escalonamento com prioridade fixa, as prioridades são atribuídas em tempo de projeto. *Jobs* de uma mesma tarefa sempre assumirão uma mesma prioridade, podendo ser ela única durante sua execução ou assumindo valores diferentes, mas previamente fixados, como no caso do *dual priority scheduling* (DAVIS; WELLINGS, 1995).

Para o modelo de sistema com *deadlines* implícitos, a política ótima de atribuição de prioridades é o *Rate Monotonic* (LIU; LAYLAND, 1973). Nesta política, as prioridades mais altas são atribuidas às tarefas cujas frequências de ativação sejam maiores (menor período).

Um teste suficiente para o *Rate Monotonic*, baseado em utilização foi proposto por (LIU; LAYLAND, 1973), e é baseado nas seguintes premissas:

- Todas as tarefas são periódicas;
- Tarefas são liberadas no início do período;
- Tarefas são independentes (sem compartilhamento de recursos ou precedência);
- Sem auto-suspensão de tarefas;

- Tarefas totalmente preemptivas;
- Sem *overheads* de escalonamento;
- Somente um processador.

O teste foi desenvolvido com base na Teorema do Instante Crítico (LIU; LAYLAND, 1973), que diz que o pior caso do tempo de resposta das tarefas de um sistema ocorre quando todas são liberadas simultaneamente em um mesmo instante (o instante de tempo zero). O teste proposto define que um conjunto é escalonável se satisfizer a seguinte condição:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{2.3}$$

Na Equação 2.3, conforme n cresce, a utilização converge para 0,69 (69%), ou seja:

$$\lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = 0,69 \tag{2.4}$$

Como o teste anterior é suficiente, ele pode descartar conjuntos escalonáveis cuja utilização seja superior a 0,69. Entretanto, esta condição pode ser relaxada quando as tarefas possuem períodos harmônicos (múltiplos do período da tarefa de mais alta prioridade). Quando esta condição for satisfeita, então o conjunto passa ser garantidamente escalonável se o somatório das utilizações não exceder 1.

Porém, o teste anterior apresenta certas limitações. Ele é um teste pessimista (suficiente mas não necessário), impõe restrições quanto aos *deadlines* (precisam ser iguais aos períodos) e supõe que as prioridade são atribuídas segundo *Rate Monotonic*.

Um teste mais preciso foi proposto por Lehoczky em (LEHOCZKY et al., 1989). Este teste é baseado na função Demanda de Processador $W(t)$ e diz que um conjunto é escalonável se as tarefas iniciarem a execução simultaneamente no instante zero e a seguinte condição for respeitada para cada tarefa τ_i, com $0 < t \leq T_i$:

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t \tag{2.5}$$

Neste teste, para cada instante t, a demanda total de processador não deve exceder a demanda disponível. Com relação as prioridade das tarefas, sejam duas tarefas τ_i e τ_j, se $j < i$, então a prioridade de τ_j é mais alta que a prioridade de τ_i.

Quando a política de atribuição de prioridades for qualquer (inclui-se aqui o *Rate Monotonic*) o teste anterior também pode ser aplicado, conforme provado por (NASSOR; BRES, 1991).

Outro teste que pode ser utilizado para verificação da escalonabilidade de um sistema é o teste de tempo de resposta proposto por Audsley quase simultaneamente ao teste de demanda de tempo em (AUDSLEY et al., 1991). Este teste é baseado em uma equação de ponto fixo que calcula o tempo de resposta de uma tarefa i, ativada no instante crítico:

$$R_i^{n+1} = C_i + \sum_{h \in hp(i)} \left\lceil \frac{R_i^n}{T_h} \right\rceil C_h \tag{2.6}$$

Se na Equação 2.6, para cada tarefa, o valor de R_i convergir para uma valor menor ou igual a T_i, então o conjunto de tarefas é escalonável, onde $hp(i)$ representa o conjunto das tarefas com prioridades mais altas que a da tarefa τ_i.

Os testes anteriores foram desenvolvidos primariamente sob a condição de que as tarefas possuem prioridade atribuídas segundo a política *Rate Monotonic*, que por sua vez pressupõe *deadline* igual ao período. Entretanto, se o *deadline* for menor que o período, o *Rate Monotonic* deixa de ser ótimo. Para estes casos, deve ser utilizado o *Deadline Monotonic* (DM) proposto por Leung (LEUNG; WHITEHEAD, 1982), que é ótimo para *deadlines* menores que o período. O *Deadline Monotonic* atribui as prioridades mais altas para tarefas com menores *deadlines* relativos. Os testes de escalonabilidade para o DM são os mesmos do RM, exceto o de utilização, que não se aplica.

2.3.4 Com Prioridade Dinâmica

Em escalonamento com prioridade dinâmica, as prioridade são atribuídas a *jobs* individuais. Estes algoritmos por vezes são chamados de algoritmos dirigidos por *deadlines*. O algoritmo de escalonamento por prioridade dinâmica mais conhecido é o *Earliest Deadline First* (EDF). Neste algoritmo, as prioridades são atribuídas de forma inversamente proporcional ao *deadline* absoluto dos *jobs*. A primeira análise para o EDF também foi proposta proposta por Liu e Layland (LIU; LAYLAND, 1973), sob as mesmas premissas utilizadas para o RM. O teste considera como escalonável conjuntos de tarefas que atenderem a seguinte restrição:

$$U = \sum_{i=1}^{n} U_i \leq 1 \tag{2.7}$$

Ou seja, o algoritmo EDF escalona qualquer conjunto que possua utilização menor ou igual a 1, então, ele é ótimo em relação ao escalonamento preemptivo em sistemas monoprocessados, como foi mostrado por Dertouzos em (DERTOUZOS, 1974). O teste de utilização é exado para o EDF (para o RM ele é somente suficiente). Outro algoritmo de escalonamento com prioridade dinâmica, que também é ótimo é o *Least Laxity First* (LLF), proposto por (MOK, 1983). O LLF escalona as tarefas de acordo com a folga das mesmas. Na prática, o LLF é pouco utilizado devido ao seu alto *overhead*.

Também existem testes para algoritmos de prioridade dinâmica baseados em demanda de processador, como os propostos em (BARUAH et al., 1990b) e que foram ampliados para comportar tarefas esporádicas (BARUAH et al., 1990a).

2.4 ESCALONAMENTO EM SISTEMAS MULTIPROCESSADOS

Escalonamento em sistemas de tempo real tem por objetivo básico resolver dois problemas: o da alocação, que diz respeito a onde executar uma tarefa (em que processador) e com qual prioridade (ordem em relação as outras

tarefas).

Desta forma, os algoritmos de escalonamento para estes sistemas são classificados de acordo com suas políticas de mudança/atribuição de prioridade e como efetuam a alocação das tarefas em processadores. Em relação às prioridades, os algoritmos podem ser:

De prioridade fixa por tarefa: todos os *job* de uma mesma tarefa recebem uma mesma prioridade, como RM e DM por exemplo;

De prioridade fixa por *job*: *jobs* de uma mesma tarefa podem ter prioridades diferentes, mas fixas durante a ativação de um *job* particular, como no EDF por exemplo;

De prioridade dinâmica: um *job* de uma tarefa pode ter diferentes prioridades em diferentes instantes de tempos durante uma ativação, como no LLF, por exemplo.

Em relação à alocação:

Sem migração: nesta configuração, as tarefas são estaticamente alocadas aos processadores, ou seja, todos os *jobs* de uma determinada tarefa executam em um único processador. Estes algoritmos também são conhecidos como particionados;

Migração em nível de tarefa: *jobs* de uma tarefa podem executar em diferentes processadores, entretanto, cada *job* executa em somente um processador;

Migração em nível de *job*: um *job* de uma tarefa pode executar em diferentes processadores, ou seja, no meio de uma ativação pode haver uma migração. Entretanto, a execução de um *job* em mais de um processador simultaneamente não é permitida. Estes algoritmos também são conhecidos como algoritmos globais;

Com relação à migração, ainda existem as abordagens híbridas, que permitem a migração seletiva de tarefas. Existe ainda a classificação dos algoritmos em:

Preemptivos: uma tarefa pode ser preemptada por outra de mais alta prioridade a qualquer momento;

Não preemptivos: uma tarefa não poderá ser preemptada após iniciada sua execução;

Cooperativos: tarefas somente podem ser preemptadas em pontos específicos de sua execução.

2.4.1 Modelos de Sistema

Além dos modelos de sistemas/tarefas, que são idênticos aos modelos para sistemas monoprocessados, existem os modelos dos processadores, que podem ser:

Heterogêneos: Neste modelo os processadores são diferentes, onde a taxa de execução depende tanto da tarefa quando do processador. Neste modelo, processadores podem ter aplicação específica, então, nem todas as tarefas podem estar aptas à execução em todos os processadores;

Homogêneos: Neste modelo os processadores são idênticos, ou seja, a taxa de execução das tarefas não varia de um processador para outro;

Uniforme: Modelo onde a taxa de execução de uma tarefa depende somente da velocidade do processador.

A seguir serão apresentados os algoritmos classificados de acordo com as políticas de migração (particionados, globais e híbridos).

2.4.2 Escalonamento Particionado

Em escalonamento particionado, o problema recai em escalonamento para sistemas monoprocessados. O real problema deste tipo de escalonamento é a alocação de tarefas entre os vários processadores que compõem o sistema. Este problema de alocação é análogo ao problema do *bin packing* (COFFMAN et al., 1978), que é NP-Hard.

Em geral, o problema da alocação é resolvido por heurísticas de *bin packing* sub-ótimas, como por exemplo *"First-Fit"* (FF), *"Next-Fit"* (NF), *"Best-Fit"* (BF) e *"Worst-Fit"* (WF), combinados com estratégias de ordenação, como utilização decrescente (DU). Mais do que particionar um conjunto de tarefas, é preciso garantir a escalonabilidade perante alguma política de atribuição de prioridades. Neste cenário, por exemplo, existem algoritmos como RM-FFDU, que particiona o conjunto de tarefas segundo a estratégia *"First-fit"*, partindo do conjunto ordenado por utilização decrescente (DU) e escalonando os processadores individualmente por RM. Os testes de escalonabilidade para monoprocessadores devem ser utilizados ao longo do processo de particionamento, com o intuito de verificar a validade das escolhas do algoritmo (verificar se a inserção de uma tarefa em um determinado processador mantém o sistema/processador escalonável).

A maneira mais comum de avaliar algoritmos de alocação de tarefas é utilizando um conceito chamado de taxa de aproximação ($\Re$). Taxa de aproximação serve para comparar a performance de um determinado algoritmo em relação a um algoritmo ótimo. Seja um conjunto de tarefas $\prod$ e sabendo que um algoritmo ótimo escalona o sistema com $M_O(\prod)$ processadores e um algoritmo A escalona com $M_A(\prod)$ processadores, então a taxa de aproximação do algoritmo $\Re_A$ será:

$$\Re_A = \lim_{M_O(\prod)\to\infty} \left(\underset{\forall \tau}{max} \left(\frac{M_A(\prod)}{M_O(\prod)} \right) \right) \tag{2.8}$$

Se um determinado algoritmo A possui taxa de aproximação $\Re_A = 1$, então este algoritmo também é ótimo. O valor de $\Re$ sempre será maior ou igual a 1.

Também existe o conceito de limites de utilização para avaliação de performance de algoritmos de alocação em conjunto de tarefas com *deadlines* implícitos. O pior caso de limite de utilização para um algoritmo A (U_A) é definido como a mínima utilização de qualquer conjunto de tarefas que pode ser escalonado com o algoritmo A. Em outras palavras, não existe conjunto com utilização total menor ou igual a U_A que não seja escalonável pelo algo-

ritmo A. Segundo (ANDERSSON et al., 2001), o maior limite de utilização para qualquer algoritmo de particionamento é:

$$U_{OPT} = (m+1)/2 \tag{2.9}$$

Para qualquer algoritmo de particionamento com prioridade fixa por tarefa, foi mostrado por (OH; BAKER, 1998), que existe um limite de utilização, que é limitado por:

$$U_{OPT(FTP)} < (m+1)/(1+2^{1/(m+1)}) \tag{2.10}$$

Existem outros algoritmos de particionamento e outras maneiras de se derivar os limites de utilização, baseados em parâmetros como número de tarefas e utilizações individuais (DAVIS; BURNS, 2009). Para o caso de conjuntos de tarefas com *deadlines* arbitrários e restritos também existem outros algoritmos de particionamento/escalonamento.

Apesar de existir limites de utilização para algoritmos ótimos, na realidade não existe um algoritmo que consiga escalonar qualquer conjunto de tarefas que seja escalonável por qualquer outro algoritmo de particionamento.

Para o caso de tarefas com relações de dependência por exclusão mútua, a inclusão dos fatores de bloqueio em equações de tempo de resposta não é tão direta como no caso de sistemas monoprocessados. Apenas quando não existe tais relações as equações são idênticas. A inclusão de fatores de bloqueio será mostrada no próximo capítulo, onde serão apresentadas noções de sincronização em sistemas multiprocessados.

2.4.3 Escalonamento Global

Em escalonamento global, as tarefas podem migrar de um processador para outro. Isto pode ser visto como o compartilhamento de uma única fila por todos os processadores. Em (DAVIS; BURNS, 2009) são apontadas vantagens e desvantagens da abordagem global, como por exemplo:

- Menor quantidade de chaveamentos de contexto e preempções. Em

escalonamento global, tarefas somente serão preemptadas quando não houver nenhum processador *idle*;

- Aproveitamento de processador por todas as tarefas quando uma tarefa executar por tempo menor que o WCET;

- Facilidade de utilização em sistemas abertos, visto que não precisa de particionamento ou balanceamento quando o conjunto de tarefas sofre alteração.

Como dito anteriormente, escalonamento global é equivalente a migração a nível de *job*, pois a maioria dos trabalhos nesta área pressupõem isto como premissa.

Ao contrário do que ocorre em sistemas monoprocessados, EDF não é um algoritmo de escalonamento ótimo no contexto de sistemas multiprocessados globais. Foi mostrado por (DHALL; LIU, 1978) que o limite de utilização do EDF global é 1 + ε, para qualquer ε arbitrariamente pequeno. Isto ocorre pois, um sistema com m tarefas (e m processadores) com utilizações arbitrariamente pequenas e uma tarefa com utilização muito próxima a 1 não pode ser escalonado por EDF. Esta característica ficou conhecida como efeito "Dhall" e acabou mostrando que outras abordagens eram necessárias.

Atualmente, existem várias classes de escalonadores globais, que podem ser classificados como:

Prioridade Fixa por *Job*: para conjuntos de tarefas com *deadline* implícito, o limite máximo utilização para qualquer algoritmo de prioridade fixa por *job* é:

$$U_{OPT} = (m+1)/2 \tag{2.11}$$

Para esta classe de algoritmos existem diversas variantes do EDF, como EDF-US (Srinivasan, AnandBaruah, 2002) e EDF(k) (BAKER, 2005).

Prioridade Fixa por Tarefa: Da mesma forma como o EDF sofre do efeito "Dhall", algoritmos como RM global e DM global também sofrem. Por

este motivo, outros algoritmos de atribuição de prioridade foram desenvolvidos para conjuntos de tarefas periódicas com *deadlines* implícitos, como o TkC (ANDERSSON; JONSSON, 2000) por exemplo, sendo que o limite de utilização para qualquer algoritmo com prioridade fixa por tarefa em escalonamento global é:

$$U_{OPT} \leq 0.41m \tag{2.12}$$

Prioridade Dinâmica: existem diversos algoritmos de prioridade dinâmica por *job*, sendo alguns deles ótimos em relação a escalonamento de tarefas periódicas com *deadlines* implícitos. Entretanto, não existem algoritmos ótimos para escalamento *online* de tarefas esporádicas. Escalonamento global domina todas as outras classes de algoritmos.

Dentre os algoritmos ótimos, um dos mais conhecidos é o Pfair (*Proportionate fair*) proposto por (BARUAH et al., 1996). O Pfair é baseado na teoria do escalonamento fluído, onde cada tarefa progride proporcionalmente a sua utilização. Por ser ótimo, o Pfair possui o limite de utilização:

$$U_{PFAIR} = m \tag{2.13}$$

Entretanto, devido ao fato do algoritmo apresentar alto *overhead* de execução por tomar decisões de escalonamento a cada instante infinitesimal de tempo, foram propostas versões discretizadas do mesmo. O objetivo destas versões é obter um compromisso entre *overhead* e otimalidade.

Existem outros algoritmos ótimos, como LLREF (CHO et al., 2006) também baseado em escalonamento fluído e RUN (REGNIER et al., 2011) que é baseado em *dual scheduling*.

Outras abordagens não ótimas existem, como o EDZL (*Earliest Deadline First until Zero Laxity*) proposto por (LEE et al., 1994). Este algoritmo funciona exatamente como o EDF, exceto quando alguma tarefa atinge folga zero, então ela deve ser executada imediatamente, senão

perderá seu *deadline*. O EDZL possui um bom limite de utilização para tarefas com *deadline* implícito, que é:

$$U_{EDZL} \leq 0.63m \quad (2.14)$$

2.4.4 Escalonamento Híbrido

Existem abordagens híbridas criadas para suprir algumas deficiências das abordagens anteriores. Por exemplo, abordagens globais podem apresentar *overheads* excessivos de comunicação entre processadores e de despejo de *cache* na migração. Por outro lado, a abordagem particionada tende a fragmentar a capacidade de processamento total do sistema, tendo um limite superior de utilização de no máximo 50%. Abordagens híbridas possuem características de ambas as abordagens anteriores e podem ser classificadas em Semi-particionadas e baseadas em Clusterização.

Abordagens Semi-particionadas: Abordagens Semi-particionadas tem por objetivo o aproveitamento de capacidade disponível entre os diversos processadores em sistemas particionados. Em tal abordagem, algumas tarefas são divididas em partes a serem executadas em diferentes processadores (que possuem capacidade para executar esta parte).

Para conjuntos de tarefas periódicas com *deadlines* implícitos, existe o EKG (ANDERSSON; TOVAR, 2006), por exemplo, que quebra algumas tarefas em componentes que serão executados em diferentes processadores. Para conjuntos de tarefas esporádicas com *deadlines* implícitos, existem outras abordagens que se diferenciam pela forma como as quebras são efetuadas, como por exemplo, a nível de *job* (ANDERSSON et al., 2008).

Clusterização: Clusterização pode ser vista como uma abordagem particionada, cujas partições são alocadas a *clusters* de processadores, e não a processadores individuais. Com esta abordagem a fragmentação é reduzida em relação a sistemas particionados. Outra característica é que o tamanho da fila de tarefas de cada *cluster* pode ser consideravelmente

menor que a fila única em sistemas globais, influenciado assim, na redução da migração. Em sistemas clusterizados, pode-se também aproveitar características de *hardware* na hora da montagem dos *clusters*, como por exemplo, compartilhamento de *cache*.

2.5 GERAÇÃO DE TAREFAS PARA AVALIAÇÃO EMPÍRICA DE TESTES DE ESCALONABILIDADE

Para avaliação empírica de escalonamento, afim de comparar diferentes algoritmos ou testes de escalonabilidade em sistemas monoprocessados e multiprocessados, deve-se utilizar conjuntos de tarefas sintéticos gerados sob algum critério. Segundo (EMBERSON et al., 2010), o problema da geração de conjuntos de tarefas possui os seguintes requisitos:

Eficiência (*Efficiency*): a fim de se obter um número estatisticamente grande de amostras, um grande número de conjuntos de tarefas deve ser gerado para cada alteração de algum parâmetro que é examinado no experimento;

Independência (*Independency*): deve ser possível variar cada parâmetro e/ou propriedade do conjunto de tarefas independentemente. Por exemplo, se um teste é afetado pelo período das tarefas do conjunto, então este será o único parâmetro a ser variado independentemente;

Imparcial (*Unbiased*): a distribuição do conjunto de tarefas deve ser equivalente a seleção aleatória a partir dos conjuntos de todas as tarefas possíveis, descartando as que não se enquadram nos parâmetros desejados.

Em geral, se aplica algum algoritmo de geração de utilizações (descritos abaixo), sendo o tempo de computação calculado a partir da utilização e do período de cada tarefa. O período por sua vez, pode ser gerado de acordo com algum critério específico ou seguindo uma distribuição log-uniforme ou mesmo uniforme.

2.5.1 Geração de Utilizações para Sistemas Monoprocessados

Uma maneira de gerar conjuntos de forma a não introduzir viés em resultados empíricos é utilizar o algoritmo de geração UUniFast (BINI; BUTTAZZO, 2005). Este algoritmo gera conjuntos de tarefas com utilização aproximadamente igual a 1, para uma dada quantidade de tarefas. A distribuição das utilizações geradas pelo UUniFast seguem uma distribuição uniforme.

2.5.2 Geração de Utilizações para Sistemas Multiprocessados

Para sistemas multiprocessados poderia também ser utilizado o UUniFast, bastando configurar a utilização total do sistema para um valor igual ao número de processadores. Entretanto, esta abordagem pode gerar conjuntos de tarefas inválidos, devido a possibilidade da geração de tarefas com utilização maior que 1. Para contornar esta situação foi proposto o algoritmo UUniFast-Discard (DAVIS; BURNS, 2011), que funciona simplesmente descartando conjuntos de tarefas inválidos. A única deficiência do UUniFast-Discard é que a geração pode se tornar inviável (no sentido de ineficiente) para certos valores e para certas quantidades de tarefas e determinados valores de utilização, pois pode haver a necessidade de descarte de muitos conjuntos de tarefas. Para contornar este problema, pode-se utilizar o algoritmo *Randfixedsum* de Stafford, descrito em (EMBERSON et al., 2010). Este algoritmo é eficiente para quaisquer parâmetros do conjunto de tarefas, pois não efetua nenhum descarte.

2.6 CONCLUSÕES

Este capítulo apresentou um introdução a sistemas de tempo real no contexto de prioridade fixa por tarefa, fixa por *job* e dinâmica. Foram apresentados conceitos relativos ao escalonamento em sistema monoprocessados e multiprocessados, com uma breve explanação das várias abordagens possíveis. Para a parte de sistemas monoprocessados foram mostradas as políticas

mais comuns de atribuição de prioridades e os testes clássicos para análise de escalonabilidade, como o Teste de Tempo de Resposta e o Teste de Demanda de Tempo. Para a parte relativa a sistemas multiprocessados, foram expostas as abordagens classificadas quanto a alocação das tarefas aos processadores, que neste contexto, foram as globais, particionadas e híbridas. A última parte do capítulo apresentou noções de geração de tarefas sintéticas para análise de algoritmos de escalonamento/e testes de escalonabilidade em sistemas mono-processados e multiprocessados.

A literatura sobre escalonamento de sistemas de tempo real é imensa e apenas os conceitos relevantes foram apresentados. Diversas sub-áreas não foram mencionadas. Por exemplo, modelos probabilistas não foram considerados neste trabalho e por isto os conceitos ligados a este tipo de modelagem não foram apresentados.

3 SINCRONIZAÇÃO EM SISTEMAS DE TEMPO REAL COM PRIORIDADE FIXA

A utilização de protocolos de sincronização têm por objetivo suprir dificuldades de se obter sistemas escalonáveis na presença de recurso com acesso exclusivo. Foi provado por Mok (MOK, 1983) que é impossível obter uma escala *online* de execução ótima para um conjunto de tarefas na presença de semáforos/*mutexes* tradicionais. Mok também estendeu o teorema para sistemas multiprocessados. Protocolos de sincronização atacam exatamente a problemática levantada por Mok quanto a impossibilidade de escalonamento sob a presença de exclusão mútua. O problema da exclusão mútua foi muito estudado para sistemas monoprocessados, entretanto, para sistemas multiprocessados, este assunto ainda continua em aberto.

Esta parte do trabalho apresenta uma descrição dos protocolos existentes tanto para sistemas monoprocessados quanto multiprocessados, entretanto, com mais ênfase nos que tangem a última categoria. Para a parte relativa a sistemas multiprocessados, serão abordadas questões específicas a estes sistemas, como tipos de bloqueios possíveis e as diferentes políticas de controle de execução, algoritmos de alocação de tarefas (cientes de recursos) e testes de escalonabilidade considerando bloqueios. Para este capítulo, bem como para o restante do trabalho, serão consideradas apenas alternativas baseadas em prioridade fixa por tarefa.

3.1 SINCRONIZAÇÃO EM MONOPROCESSADORES

Antes de se estudar com profundidade a problemática da sincronização em sistemas multiprocessados, é útil uma breve explanação do problema sob a ótica de sistemas monoprocessados. A sincronização em sistemas de tempo real busca limitar a ocorrência de inversões de prioridade descontroladas. O problema da sincronização em monoprocessadores foi extensivamente estudado e inúmeras soluções foram propostas (SHA et al., 1990; RAJKUMAR, 1990; BAKER, 1990), a maioria envolvendo algum tipo de herança de priori-

dade. A próxima subseção introduz o problema da inversão de prioridades no âmbito de sistemas monoprocessados.

3.1.1 Inversões de Prioridades

Inversões de prioridade ocorrem quando uma tarefa de mais alta prioridade é obrigada a esperar por uma tarefa de mais baixa prioridade. Isto facilmente acontece quando uma tarefa de mais alta prioridade disputa recursos com tarefas de prioridade mais baixa. Devido à exclusão mútua, apenas uma das tarefas poderá acessar o recurso em um determinado instante. Se a tarefa de mais baixa prioridade detiver o recurso quando a de mais alta prioridade precisar deste, a última deverá esperar a primeira liberar o recurso, ocorrendo uma inversão de prioridades.

Uma inversão de prioridade é dita descontrolada quando não há um limite para a sua duração, ou seja, a tarefa de mais alta prioridade pode ser postergada indefinidamente. Um ponto importante é que inversões ocorrem somente devido a bloqueios causados por tarefas de mais baixa prioridade. Tarefas com prioridade mais alta não causam bloqueios mas sim interferência, que já é prevista naturalmente nos testes exatos para prioridade fixa. Uma inversão de prioridade descontrolada é exemplificada na Figura 1. Nesta figura, a tarefa de mais alta prioridade τ_0 bloqueia em um recurso que está de posse de uma segunda tarefa, τ_2, de prioridade mais baixa. τ_2 por sua vez é preemptada por τ_1, que transitivamente adiciona mais atraso à tarefa τ_0.

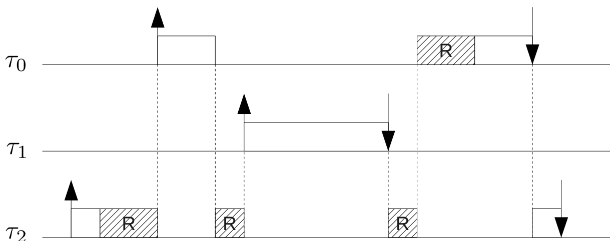


Figura 1: Inversão de prioridade descontrolada

3.1.2 Protocolos para Sistemas Monoprocessados

Dada a problemática das inversões de prioridades, a maneira mais usual de controlá-las em sistemas de tempo real, é com a utilização de protocolos de acesso a recursos (ou protocolos de sincronização). A seguir serão apresentadas as alternativas de controle de inversões de prioridades para sistemas monoprocessados (com prioridade fixa por tarefa).

3.1.2.1 Prevenção de Inversões de Prioridade

Inversões de prioridade podem ser evitadas se nenhuma tarefa puder acessar uma seção crítica se houver a chance de uma tarefa de alta prioridade ser bloqueada. Esta abordagem existe e foi proposta por (KEITH et al., 1990). Entretanto, esta abordagem reduz aproximadamente pela metade a utilização do sistema pela inserção de tempo *idle*.

3.1.2.2 *Priority Inheritance*

Priority Inheritance (PI) é uma maneira de controlar dinamicamente inversões de prioridade. Ela funciona basicamente alterando a prioridade de tarefas que estejam causando inversões. Segundo este protocolo, quando uma tarefa é bloqueada, ela cede sua prioridade para a tarefa que a bloqueia em forma de herança, ou seja, a tarefa que bloqueia herda a prioridade da tarefa bloqueada. Foi mostrado por (SHA et al., 1990) que o protocolo *Priority Inheritance* limita o número de vezes em que uma tarefa pode ser bloqueada a um número igual ao de seções críticas, pois no pior caso, cada seção estará sendo acessada por uma tarefa de mais baixa prioridade.

3.1.2.3 *Priority Ceiling Protocol*

Embora o protocolo *Priority Inheritance* possa limitar o número de vezes que uma tarefa é bloqueada, este tempo pode ser extremamente pes-

simista. Este pessimismo surge com a possibilidade de ocorrência de encadeamento de bloqueios (em caso de seções críticas aninhadas ou múltiplas seções por tarefa). Por exemplo, uma tarefa τ_i é bloqueada por uma tarefa τ_j, que por sua vez é bloqueada por outra tarefa τ_k, formando assim uma cadeia de bloqueios. Outro característica é que o *Priority Inheritance* não impede a ocorrência de *deadlocks*. O protocolo *Priority Ceiling Protocol* (PCP) ou apenas *Ceiling Protocol* (SHA et al., 1990) aborda exatamente estes problemas, limitação aceitável nos tempo de bloqueio e prevenção de *deadlocks*. Geralmente este protocolo é implementado como regras de aquisição de semáforos binários/*mutexes*. Segundo este protocolo:

- Todas as tarefas apresentam prioridades definidas estaticamente (RM por exemplo);
- Todo *mutex* apresenta um valor *ceiling*, que é definido como sendo a prioridade mais alta entre as prioridades das tarefas que o acessam;
- Uma tarefa possui, em tempo de execução, uma prioridade dinâmica, que é a maior entre a sua própria prioridade estática e as prioridades herdadas de tarefas de mais alta prioridades bloqueadas por esta;
- Uma tarefa somente poderá adquirir um *mutex* se sua prioridade dinâmica for maior que todos os *ceilings* de todos os *mutexes* atualmente bloqueados por outras tarefas.

Segundo este protocolo, uma tarefa poderá ser bloqueada somente uma vez em cada ativação, e por qualquer tarefa de mais baixa prioridade.

3.1.2.4 *Immediate Priority Ceiling Protocol*

O *Immediate Priority Ceiling Protocol* (IPCP) ou apenas *Ceiling Priority* (KLEIN; RALYA, 1990) pode ser visto como um simplificação do *Priority Ceiling Protocol*. Segundo este protocolo, a prioridade de uma tarefa deve ser ajustada imediatamente ao *ceiling* assim que esta efetua *lock* em um *mutex*. Os ceiling dos recursos/*mutexes* são atribuídos de maneira idêntica ao *Ceiling Protocol*. O piores casos nos tempos bloqueio deste protocolo também

são iguais aos tempos do *Priority Ceiling Protocol*, sendo que neste, ocorrem menos chaveamentos de contexto.

3.1.2.5 *Stack Resource Policy*

O *Stack Resource Policy* (SRP) é um protocolo de sincronização proposto por (BAKER, 1990), que tem por objetivo, além do controle de inversões de prioridade, o reaproveitamento de pilha. Este protocolo é versátil, pois pode ser utilizado tanto com EDF quanto com RM. Segundo o protocolo, a cada tarefa τ_i é associada uma prioridade ρ_i e um *preemption level* λ_i, tal que, τ_i somente poderá ser preemptada pela tarefa τ_j se $\lambda_i < \lambda_j$. Para cada recurso R é atribuído um ceiling estático definido por:

$$ceil(R) = max\{\lambda_i | \tau_i \quad usa \quad R\} \tag{3.1}$$

Também é definido um *system ceiling* dinâmico, que é definido por

$$system_ceiling = max[\{ceil(R) | R \quad esta \quad sendo \quad acessado\} \cup \{0\}] \tag{3.2}$$

A regra de escalonamento é simples: uma tarefa não poderá executar enquanto sua prioridade não for a maior entre as prioridade de todas as outras tarefas ativas e o seu *preemption level* não for maior que o *system_ceiling*. O protocolo garante que, se uma tarefa começar a executar, ela não será mais bloqueada, somente preemptada por tarefas de mais alta prioridade. Todo bloqueio experimentado pela tarefa ocorrerá antes do início da execução desta, similarmente ao *Immediate Priority Ceiling*. Em (BAKER, 1990) também é proposto um teste de escalonabilidade baseado em utilização. Assim como o MPCP, o SRP também evita *deadlocks*, com o adicional de utilizar somente uma pilha para todas as tarefas.

3.2 SINCRONIZAÇÃO EM MULTIPROCESSADORES

3.2.1 Modelo de Sistema e Convenções

Para este trabalho será considerado apenas escalonamento particionado com prioridade fixa por tarefa (ou (1, 1)-*restricted* na classificação de (CARPENTER et al., 2004)). No escalonamento particionado, as tarefas são alocadas estaticamente aos processadores através de filas dedicadas. O *dual* do escalonamento particionado é o global, onde uma única fila é compartilhada por todos os processadores, o qual não será considerado neste trabalho. O escalonamento particionado com prioridade fixa também é conhecido como P-SP (*partitioned static priority*). Este tipo de escalonamento é abordado com algoritmos para monoprocessador em cada processador, de modo que o alvo do problema é como efetuar o particionamento das tarefas. O particionamento de tarefas entre processadores é efetivamente o problema do *Bin Packing* que tem complexidade combinatória NP-*hard*. Uma estratégia possível é particionar segundo alguma heurística, como BFD (*Best-fit decreasing*) ou agrupar por utilização de recursos, como será visto mais adiante. As próximas seções preocupam classificar os tipos de recursos em sistemas multiprocessados, tipos de bloqueios e outras características relevantes à sincronização.

3.2.2 Tipos de Recursos em Sistemas Multiprocessados

Em sistemas de tempo real com escalonamento particionado, recursos[1] podem ser classificados de acordo com sua visibilidade no sistema. Os recursos podem ser:

Recursos Locais: Quando um recurso é visível em apenas um processador (ou acessível somente para tarefas alocadas a um determinado processador), este é classificado como local.

Recursos Globais: São recursos acessíveis a partir de mais de um processa-

[1] Recursos são quaisquer estruturas, objetos ou dispositivos, os quais não devem ser utilizados simultaneamente por mais de uma tarefa, por questões de segurança ou integridade.

dor. Podem ser, por exemplo, estruturas em memória utilizadas para coordenação e comunicação entre tarefas. Também podem ser dispositivos de hardware (linhas de comunicação, portas de E/S, etc).

Quando se particiona tarefas por utilização de recursos, pode-se transformar recursos globais em locais. Desta forma, pode-se abordar o problema utilizando somente protocolos para sistemas monoprocessados. Na prática, é muito difícil conseguir um particionamento completamente disjunto em relação aos recursos utilizados. O que pode acontecer também é que os conjuntos obtidos excedam a capacidade individual de um processador, sendo assim, não escalonáveis.

3.2.3 Tipos de bloqueio em Sistemas Multiprocessados

Existem basicamente dois tipos de bloqueio que podem ocorrer em sistemas multiprocessados na presença de recursos:

Bloqueio Local: Bloqueios locais ocorrem em consequência do uso de recursos locais. Quando uma tarefa é bloqueada devido a uma tarefa de baixa prioridade que executa no mesmo processador dizemos que este é um bloqueio local. Bloqueios locais ocorrem exclusivamente por conta de tarefas de mais baixa prioridade, visto que tarefas de mais alta prioridade só causam interferência.

Um exemplo de bloqueio local é apresentado na Figura 2. Nesta figura, a tarefa τ_0 sofre bloqueio ao tentar adquirir o recurso que esta detido pela tarefa τ_1 no mesmo processador.

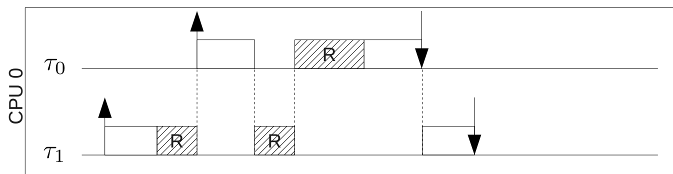


Figura 2: Exemplo de bloqueio local

Bloqueio Remoto: Uma tarefa também pode sofrer bloqueio por conta de tarefas que não estejam executando no mesmo processador que esta (ou seja, em um processador remoto). Este bloqueio pode ser induzido tanto por tarefas de mais baixa prioridade quanto de mais alta prioridade. A presença de bloqueio remoto pode alterar significativamente a complexidade dos cenários de bloqueio.

Um exemplo de bloqueio remoto é apresentado na Figura 3. Neste exemplo, a tarefa τ_0 sofre bloqueio até a tarefa τ_1 (em outro processador) liberar o recurso necessário.

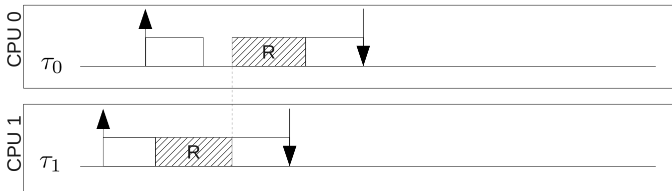


Figura 3: Exemplo de bloqueio remoto

3.2.4 Políticas de Controle de Execução

Em sincronização de tarefas quanto ao acesso a recursos, um detalhe importante é a forma como se lida com a contenção. Contenção ocorre quando uma tarefa é obrigada a esperar por um determinado recurso que encontra-se indisponível no momento (está sob a posse de outra tarefa). As próximas partes desta seção apresentam as duas formas de controle de execução utilizadas em caso de contenção na sincronização em tempo real.

Abordagens Baseadas em Suspensão de Tarefa: Quando uma tarefa precisar de um recurso que está de posse de uma tarefa que executa em processador remoto, ela será suspensa, permitindo que outras tarefas de mais baixa prioridade utilizem o processador. Quando o recurso necessário à tarefa de mais alta prioridade se tornar disponível, esta deverá ser ativada novamente (inserida na fila de aptos). Esta política tange tanto

bloqueios locais quanto remotos.

É apresentado na Figura 4 um exemplo de suspensão, onde a tarefa τ_0, na CPU 0 é suspensa ao bloquear no recurso *R*, que está de posse da tarefa τ_2 na CPU 1. Quando τ_2 libera o recurso, a tarefa τ_0 é desbloqueada e preempta τ_1 (com prioridade mais baixa), que estava em execução na CPU 0.

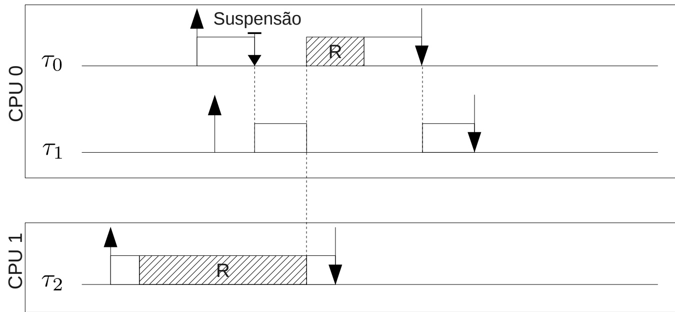


Figura 4: Exemplo de suspensão de tarefa ocasionada por bloqueio

Abordagens baseadas em Espera Ocupada (*Spin*): Outra abordagem em situações de bloqueio é esperar pelo recurso em uma espécie de *polling* (também conhecido como *spinlock*). Este *polling* ou *spin* pode ser executado de duas formas distintas: preemptiva e não preemptiva. Na forma preemptiva, a tarefa simplesmente executa o *spin* para impedir que tarefas de mais baixa prioridade executem, entretanto interferências ainda são possíveis. No *spin* não preemptivo, como o próprio nome diz, nenhuma tarefa poderá executar (seja ela de mais baixa ou mais alta prioridade) senão a própria que realiza o *spin*.

Um exemplo de *spin* (não preemptivo) em bloqueio pode ser visto na Figura 5. Neste exemplo, a tarefa τ_0 na CPU 0, bloqueia em espera ocupada ao tentar acessar o recurso *R* que está de posse da tarefa τ_2, na CPU 1. Como τ_0 não libera a CPU 0 enquanto bloqueada, τ_1 é impedida de executar.

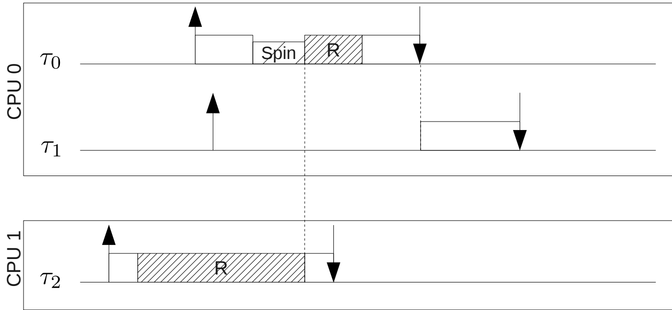


Figura 5: Espera por *spin*

Uma das desvantagens do *spin* é o desperdício de ciclos de CPU, o que não ocorre com suspensão. A técnica de *spin* pode ser interessante em casos onde existem seções críticas muito pequenas, onde chavear contexto em caso de suspensão induziria um *overhead* desnecessário. Brandenburg et al. (BRANDENBURG et al., 2008) apresentam uma comparação de políticas de controle de execução. Existe ainda uma maneira de evitar o desperdício de CPU efetuando o chamado *virtual spin* (LAKSHMANAN et al., 2009). Nesta técnica, tarefas poderão executar quando alguma outra entrar em *spin*, contudo, serão bloqueadas em qualquer tentativa de acesso a recursos. Embora esta técnica alivie o desperdício de CPU, ela só tem impacto no caso médio dos tempo de resposta, não influenciando no pior caso.

3.2.5 Políticas de Escolha/Enfileiramento de Tarefas

Um dos aspectos importantes em relação a sincronização é a forma como as tarefas são ordenadas para acessar recursos. Basicamente pode-se utilizar duas maneiras de ordenação (utilizadas para tempo real), embora existam outras maneiras específicas para recursos específicos, como discos por exemplo.

Enfileiramento FIFO: A maneira mais intuitiva e simples de ordenação é

por FIFO, ou seja, o primeiro a solicitar é o primeiro a receber o recurso. O artigo (LORTZ; SHIN, 1995) apresenta resultados e argumenta a favor de enfileiramento FIFO (levantam a possibilidade de FIFO apresentar melhor escalonabilidade utilizando *rate monotonic*). É apresentado na Figura 6 um exemplo em que a atribuição de recursos às tarefas é efetuada por esta política.

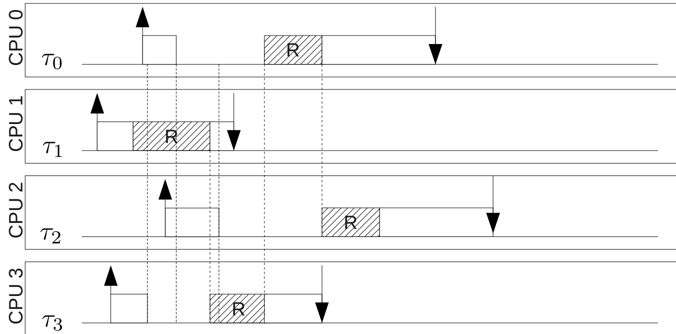


Figura 6: Enfileiramento FIFO

No exemplo anterior a primeira tarefa a acessar o recurso é a τ_1. Enquanto τ_1 está com o recurso, outras 3 tarefas solicitam este, que é atribuído por ordem de chegada quando liberado pela tarefa atual.

Enfileiramento Baseado em Prioridades: A outra maneira de enfileiramento de tarefas é de acordo com suas prioridades. É apresentado na Figura 7 um exemplo de uso desta política. Também é possível atribuir uma prioridade de enfileiramento diferente da nominal, como por exemplo uma prioridade ótima, que leve a uma melhor escalonabilidade. Entretanto esta última possibilidade foi provada (LORTZ; SHIN, 1995) como sendo NP-completa, mas com a possibilidade de implementação de boas heurísticas.

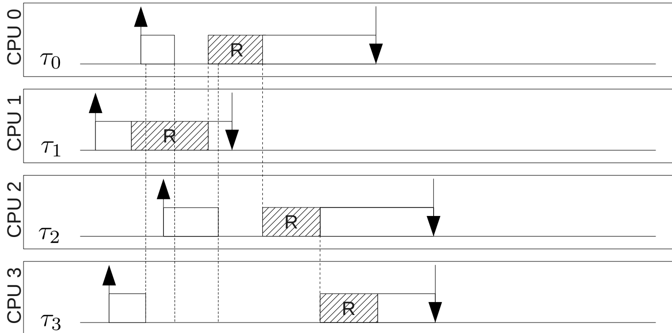


Figura 7: Enfileiramento por prioridade

No exemplo anterior, a tarefa foi enfileirada de acordo com sua prioridade nominal. Existem abordagens em que as tarefas podem ser enfileiradas com outras prioridades (sem ser a prioridade nominal da tarefa). Isto pode ser visto em trabalhos como (LORTZ; SHIN, 1995), onde a prioridade de enfileiramento é independente.

3.2.6 Algoritmos de Particionamento Cientes de Recursos

Em sincronização para sistemas multiprocessados com prioridade fixa, foram propostos alguns algoritmos que consideram a utilização de recursos no momento de efetuar a distribuição das tarefas entre os diferentes processadores. Para sistemas onde não há compartilhamento de recursos, as soluções de particionamento foram apresentadas no capítulo anterior. Basicamente, existem dois algoritmos cientes de recursos para particionamento de tarefas: o *Synchronization-Aware Partitioning Algorithm* e o *Blocking-Aware Partitioning Algorithm*.

3.2.6.1 Synchronization-Aware Partitioning Algorithm

O *Synchronization-Aware Partitioning Algorithm* (SPA) (LAKSHMANAN et al., 2009) é um algoritmo de particionamento de tarefas que leva em consideração aspectos de sincronização. Este algoritmo é uma adaptação do algoritmo BFD (*Best-fit decreasing utilization*). Uma descrição mais detalhada do algoritmo é apresentada em (NEMATI et al., 2010b). O algoritmo é composto das seguintes etapas:

1. Primeiramente tarefas que compartilham recursos direta ou indiretamente são agrupadas em pacotes chamados de *macrotasks*. O algoritmo inicia com um conjunto de processadores cuja utilização total seja suficiente para comportar a utilização total do conjunto de tarefas. Se uma tarefa não compartilha recursos, então ela estará contida em uma *macrotask* com apenas um elemento (ela própria).

2. As *macrotasks* são ordenadas em ordem não crescente de utilização. Em seguida, o algoritmo tenta colocar cada *macrotask* em um processador (sem adicionar mais processadores). Se a *macrotask* couber (escalonável) no processador, então as tarefas desta *macrotask* são alocadas a este processador. Todas as *macrotasks* que não couberem em seus processadores devem ser deixadas de lado. Ao final deste passo, os processadores estarão ordenados em ordem não crescente de utilização.

3. As *macrotasks* que foram deixadas de lado no passo anterior são ordenadas por custo de quebra. Os custos de quebra são definidos em termos do *overhead* causados pelos bloqueios quando se transforma recursos locais em globais. O custo de transformar um recurso local R_k em global é definido como:

$$Cost(R_k) = GlobalOverhead(R_k) - LocalDiscount(R_k) \quad (3.3)$$

Onde o $GlobalOverhead$ pode ser calculado como:

$$GlobalOverhead(R_k) = max(|Cs_k|)/\max_{\forall \tau_i}(\rho_i) \qquad (3.4)$$

Onde $max(|Cs_k|)$ é a seção crítica mais longa entre todas as seções relativas ao recursos R_k e ρ_i é a prioridade da tarefa τ_i (quanto menor o valor, mais alta a prioridade).

O *LocalDiscount* é definido por:

$$LocalDiscount(R_k) = \max_{(\forall \tau_i \quad acessando \quad R_k)} (max(|Cs_{i,k}|)/\rho_i) \qquad (3.5)$$

Sendo $max(|Cs_{i,k}|)$ a mais longa seção crítica da tarefa τ_i relativa ao recurso R_k. O custo total para quebrar uma *macrotask* é calculado como:

$$Cost(macrotask_t) = \sum_{(\forall R_k \quad acessado \quad pela \quad macrotask_t)} Cost(R_k) \qquad (3.6)$$

4. Em seguida, a *macrotask* com o menor custo de quebra é então partida em duas partes tal que a utilização de uma das partes seja tão próxima quanto possível da maior utilização disponível entre todos os processadores. Em uma *macrotask* as tarefas são ordenadas por ordem decrescente de utilização, sendo que as tarefas são adicionadas nesta ordem aos processadores. Então a *macrotask* fica assim dividida em dois grupos: o das tarefas que couberam no processador e as que não couberam neste. Se ao final do processo, a alocação não for viável, então é adicionado um novo processador ao sistema e o algoritmo é executado todo novamente. Para verificar se uma alocação é viável, pode-se, por exemplo verificar se todas as tarefas possuem seus *deadlines* garantidos.

3.2.6.2 Blocking-Aware Partitioning Algorithm

Uma segunda abordagem de particionamento é utilizada pelo *Blocking-Aware Partitioning Algorithm* (BPA) (NEMATI et al., 2010b, 2010a). O objetivo deste algoritmo é reduzir o bloqueio total das tarefas, melhorando assim a escalonabilidade das mesmas. Melhorar a escalonabilidade das tarefas geralmente tem o efeito de reduzir o número de processadores necessários para escalonar o sistema. De forma mais específica, estes objetivos podem ser enunciados como:

- Reduzir o número de seções críticas globais, agrupando tarefas que utilizam os mesmos recursos sempre que possível.
- Reduzir a taxa e o tempo de acesso à recursos, agrupando tarefas que acessam tais recursos mais vezes e por mais tempo, sempre que possível.

O algoritmo é executado em duas rodadas (*rounds*), sendo que o resultado final será a saída da rodada com o melhor resultado de particionamento. Primeiramente, são executados alguns passos comuns as duas rodadas:

1. São atribuídos pesos às tarefas. O peso de uma tarefa depende da utilização e de parâmetros de outras tarefas que possam causar bloqueios remotos. O peso w_i de uma tarefa τ_i é calculado da seguinte forma:

$$w_i = u_i + \left\lceil \left(\sum_{\rho_h < \rho_i} NC_{i,h} \times \beta_{i,h} \times \left\lceil \frac{T_i}{T_h} \right\rceil + NC_i \times \max_{\rho_l \geq \rho_i} \beta_{i,l} \right) / T_i \right\rceil \quad (3.7)$$

   Na Equação 3.7, NC_i representa o número total de seções críticas da tarefa τ_i e $NC_{i,h}$ representa o número de seções críticas que a tarefa τ_i compartilha com a tarefa τ_h, sendo $\beta_{i,h}$ a mais longa. Segundo (NEMATI et al., 2010b), este peso é fortemente baseado no protocolo *Multiprocessor Priority Ceiling*, que será discutido mais adiante.

2. São criadas as *macrotasks* de forma semelhante ao *Synchronization-Aware Partitioning Algorithm*, ou seja, tarefas que direta ou indiretamente compartilham recursos são agrupadas. Podem haver dois tipos de

macrotasks: as quebradas (*broken*) e as não quebradas (*unbroken*). As não quebradas são sempre colocadas em um mesmo processador pelo algoritmo (podem ser escalonadas utilizando protocolos para sistemas monoprocessados). Se uma *macrotask* é quebrada, significa que ela não cabe em um único processador, então as tarefas devem ser distribuídas em diferentes processadores. A cada *macrotask* não quebrada é atribuído um peso que é a soma das utilizações das tarefas que a compõe.

3. *Macrotasks* não quebradas e tarefas que não pertencem a nenhuma *macrotask* não quebrada são colocadas em uma lista ordenada pelo peso das tarefas (equação 3.7) e peso das *macrotasks* baseado na utilização das tarefas. Esta lista recebe o nome de lista mista (*mixed list*).

Em ambas as rodadas, as tarefas são alocadas utilizando o conceito de atração entre tarefas. A função de atração de uma tarefa τ_i em relação a tarefa τ_j é definida pela capacidade da tarefa τ_j em induzir bloqueios remotos na tarefa τ_i se estas tarefas forem alocadas em diferentes processadores. Esta função de atração $v_{i,j}$ é definida por:

$$v_{i,j} = \begin{cases} NC_{i,j} \times \beta_{i,j} \times \left\lceil \frac{T_i}{T_j} \right\rceil & \rho_j < \rho_i \\ NC_{i,j} \times \beta_{i,j} & \rho_j \geq \rho_i \end{cases} \tag{3.8}$$

A equação anterior (3.8) é baseada no *Multiprocessor Priority Ceiling Protocol*, logo, para outros protocolos ela deve ser ajustada para refletir as particularidades dos mesmos. Definida a função de atração, a execução das rodadas ocorre da seguinte maneira:

Rodada 1: Na primeira rodada, o algoritmo é executado enquanto todas as tarefas não forem alocadas. Considera-se que os processadores sempre estejam em ordem não crescente de utilização, então o primeiro objeto da lista mista é retirado. A ação a ser tomada depende do tipo de objeto retirado:

Tarefa que não pertence a nenhuma *macrotask*: se a tarefa não pertencer a nenhuma *macrotask* logo ela não compartilha recursos.

Esta tarefa deverá se alocada ao primeiro processador que a comportar (a inserção da tarefa mantém o sistema escalonável). Se ela não puder ser colocada em nenhum processador, então um novo processador é adicionado ao sistema e a tarefa é então inserida nele.

***Macrotask* não quebrada:** se o objeto for uma *macrotask* não quebrada, então todas as tarefas desta deverão ser inseridas no primeiro processador que comportar todas elas. Similarmente, se a *macrotask* não puder ser colocada em nenhum processador, então um novo processador é adicionado ao sistema e a *macrotask* é então inserida nele.

Tarefa pertencente a uma *macrotask* quebrada: se a tarefa τ_i pertencer a uma *macrotask* quebrada, então o algoritmo ordena o restante das tarefas não alocadas pertencentes a mesma *macrotask* por ordem de atração das mesmas em relação a τ_i e as insere em uma lista que terá como primeiro elemento a própria tarefa τ_i. Em seguida, será selecionado o processador que couber a maior quantidade de tarefas desta lista, e se não houver nenhum, um novo processador é adicionado. Entretanto, se o novo processador não comportar alguma das tarefas (algum outro processador se tornou não escalonável pela inserção de tal tarefa no novo processador, por exemplo), o algoritmo falha, reinicia, e entra na segunda rodada.

Rodada 2: Para a segunda rodada, os seguintes passos também são executados enquanto todas as tarefas não forem alocadas. Primeiramente, o primeiro objeto é retirado da lista mista, a ação a ser tomada também depende do tipo do objeto:

Tarefa que não pertence a nenhuma *macrotask*: se o objeto não pertencer a nenhuma *macrotask*, então o tratamento é igual ao da primeira rodada.

***Macrotask* não quebrada:** se a *macrotask* for não quebrada, então o tratamento também é igual ao da primeira rodada.

Tarefa pertencente a uma *macrotask* quebrada: se a tarefa τ_i pertencer a uma *macrotask* quebrada, então os processadores são ordenados em uma lista, em dois passos:

1. Processadores que contém uma ou mais tarefas pertencentes a mesma *macrotask* que τ_i são inseridos em ordem não crescente de atração (somatório da atração de todas as tarefas em relação a τ_i).
2. Processadores que não contém nenhuma tarefa pertencente a mesma *macrotask* que τ_i são adicionados a lista em ordem não crescente de utilização.

Após a execução dos passos anteriores, os processadores que contém ao menos uma tarefa pertencente a mesma *macrotask* estão no topo da lista. A tarefa τ_i será colocada, então, no primeiro processador da lista ordenada de processadores em que ela couber. Nesta fase, um novo processador também poderá ser adicionado ao sistema. Se a tarefa não puder ser adicionada neste novo processador, a rodada falha.

A saída do algoritmo será o melhor particionamento obtido entre as duas rodadas. Se as duas falharem, o algoritmo por inteiro falha. Se apenas uma rodada falhar, a saída será o particionamento obtido na rodada que não falhou.

3.2.7 Cálculo do Tempo de Resposta Considerando Bloqueios

Em (LAKSHMANAN et al., 2009) é apresentada uma extensão do teste baseado em tempo de resposta, que leva em consideração bloqueios. Este teste foi proposto originalmente para o *Multiprocessor Priority Ceiling*, mas pode ser modificado para suportar qualquer protocolo desenvolvido para escalonamento particionado e prioridade fixa. A notação que será utilizada ao longo deste capítulo e no próximo é a mesma utilizada em (LAKSHMANAN et al., 2009). Nesta notação, uma tarefa τ_i é definida por:
$\tau_i : ((C_{i,1}, C'_{i,1}, C_{i,2}, C'_{i,2}, ..., C'_{i,s(i)-1}, C_{i,s(i)}), T_i)$

Onde:

- $s(i)$ é o número de segmentos normais de execução de τ_i.

- $s(i) - 1$ é o número de segmentos de seção crítica de τ_i.

- $C_{i,j}$ é o WCET do j-ésimo segmento de execução normal. Nesta notação, o tempo de execução de uma tarefa é segmentado em seções críticas e seções de execução normal (intercaladas). O pior caso no tempo de execução (WCET) de uma tarefa τ_i pode ser calculado como sendo a soma de todos os segmentos, sendo eles normais e críticos. Este tempo pode ser calculado pela seguinte equação:

$$C_i = \sum_{j=1}^{s(i)} C_{i,j} + \sum_{k=1}^{s(i)-1} C'_{i,k} \tag{3.9}$$

  Pela Equação 3.9, pode-se ver que o número de segmentos de seção crítica deve ser igual ao número de segmentos de execução normal menos um.

- $C'_{i,j}$ é o WCET do j-ésimo segmento de seção crítica.

- T_i é o período de uma tarefa τ_i.

- $\tau_{i,j}$ é a j-ésimo segmento de execução normal de uma tarefa τ_i.

- $\tau'_{i,j}$ é a j-ésimo segmento de seção crítica de uma tarefa τ_i.

- $P(\tau_i)$ é o processador que executa a tarefa τ_i.

Dada duas tarefas τ_i e τ_j, se $i < j$, então a prioridade de τ_i é mais alta que a prioridade de τ_j.

Baseada na notação e nos tempos de bloqueios calculados para um determinado protocolo, é definido o tempo de resposta de uma tarefa (a tarefa τ_i possui tempo de resposta W_i). Este tempo pode ser calculado através da seguinte equação de ponto fixo:

$$W_i^{n+1} = C_i + B_i^r + I_i^n + B_i^{low} \quad (3.10)$$

Na Equação 3.10, C_i é o WCET da tarefa τ_i, B_i^r é o bloqueio remoto total sofrido pela tarefa (que depende do protocolo de sincronização utilizado), I_i^n é a interferência imposta por tarefas de mais alta prioridade. B_i^{low} representa o bloqueio imposto por tarefas de mais baixa prioridade. O valor inicial desta equação de ponto fixo deve ser $W_i^0 = C_i + B_i^r$. B_i^{low} existe pelo simples fato da tarefa τ_i ser impedida de executar como resultado de uma tarefa de mais baixa prioridade estar ocupando o processador de modo não preemptivo. Isto ocorre pois todos os protocolos executam seções críticas de modo não preemptivo (totalmente ou parcialmente). O cálculo dos valores de interferência I_i^n e do bloqueio causado por tarefas de mais baixa prioridade B_i^{low} dependem de como o protocolo lida com bloqueios (suspensão ou *spin*). Estes tempos podem ser calculados da seguinte maneira:

- **Para protocolos baseados em suspensão:** para protocolos que bloqueiam tarefas por suspensão, o cálculo da interferência deve ser efetuado pela seguinte equação:

$$I_i^n = \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n + B_h^r}{T_h} \right\rceil C_h \quad (3.11)$$

  Na Equação 3.11, o tempo de bloqueio de uma tarefa de mais alta prioridade aparece como um aumento no WCRT. Esta equação captura o efeito chamado *back-to-back execution* (RAJKUMAR, 1991), onde um tarefa sofre interferência adicional por parte de tarefas de mais alta prioridade que se auto-suspendem (este tempo não é considerado na análise normal da interferência). Neste efeito, uma tarefa que normalmente sofreria a interferência de apenas um ativação de uma determinada tarefa de alta prioridade, passa a sofrer a interferência de duas ativações sucessivas. Um limite superior para este tipo de interferência é B_h^r, que precisa se adicionado a W_i^n como uma espécie de *jitter*.

Para o cálculo do tempo de bloqueio causado por tarefas de mais baixa prioridade, deve ser utilizada a seguinte equação:

$$B_i^{low} = s(i) \times \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C^{'}_{l,k} \tag{3.12}$$

Na Equação 3.12, cada vez que uma tarefa bloquear em algum recursos (no pior caso, ela irá bloquear em cada tentativa de acesso a recurso) e antes da ativação, tarefas de mais baixa prioridade podem executar. Estas tarefas de mais baixa prioridade podem bloquear em recursos. Quando estas tarefas de mais baixa prioridade receberem seus respectivos recursos, elas irão preemptar a tarefa de mais alta prioridade em questão, pois todos os protocolos de sincronização executam seções críticas com prioridades maiores que as prioridades normais do sistema. Esta equação oferece um limite superior para este bloqueio.

- **Para protocolos baseados em *spin*:** para protocolos baseados em *spin*, o cálculo da interferência deve ser efetuado através da seguinte equação:

$$I_i^n = \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n}{T_h} \right\rceil (C_h + B_h^r) \tag{3.13}$$

O tempo de *spin* de uma tarefa (quando bloqueada em algum recurso) aparece como um aumento no tempo de computação do ponto de vista de tarefas de mais baixa prioridade. Então, este tempo deve ser adicionado ao tempo de computação de cada tarefa de mais alta prioridade, como pode ser visto na Equação 3.13.

O cálculo do tempo de bloqueio causado por tarefas de mais baixa prioridade deve ser feito através da seguintes equações:

  – Para *spin* preemptivo: quando o *spin* é realizado de maneira a impedir apenas a execução de tarefas de mais baixa prioridade, o

tempo de bloqueio deve ser calculado com a seguinte equação:

$$B_i^{low} = \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C'_{l,k} \quad (3.14)$$

Na Equação 3.14, quando uma tarefa bloqueia, ela impede que tarefas de mais baixa prioridade também bloqueiem. Entretanto, antes da ativação da tarefa em questão, no pior caso, ocorre o encadeamento de bloqueios, como no seguinte exemplo: a tarefa menos prioritária (tarefa A) bloqueia em um recurso e inicia o *spin*. Em seguida, a segunda tarefa menos prioritária (tarefa B) preempta a tarefa A para em seguida bloquear também (*spin*) e assim sucessivamente com as tarefas C e D. Assim que a tarefa D inicia o *spin*, a tarefa E (tarefa que sofrerá o bloqueio) inicia a execução. Durante a execução da tarefa E todas as tarefas de mais baixa prioridade são desbloqueadas e acabam por preemptar esta, impondo o bloqueio.

– Para *spin* não preemptivo: quando o *spin* é realizado de modo a impedir a execução de qualquer tarefa, o tempo de bloqueio deve ser calculado pela seguinte equação:

$$B_i^{low} = \max_{l>i \& \tau_l \in P(\tau_i) \& 1 \leq k < s(l)} (C'_{l,k} + B^r_{l,k}) \quad (3.15)$$

Na Equação 3.15, quando uma tarefa bloqueia (em *spin* não preemptivo), a execução de qualquer outra tarefa será impedida. Entretando, como a execução do *spin* é não preemptiva, pode haver uma tarefa de mais baixa prioridade efetuando *spin* no momento que uma tarefa qualquer iniciar sua execução. Esta tarefa deverá esperar a execução do *spin* e da seção crítica da tarefa de mais baixa prioridade, antes de iniciar sua execução propriamente dita. Como mostrado na Equação 3.15, um limite superior para este bloqueio é a maior soma de tempo de *spin* (bloqueio $B^r_{l,k}$ para a k-ésima seção crítica da tarefa τ_l) com seção crítica (a k-ésima seção crítica da tarefa τ_l é definida por $C'_{l,k}$), entre as tarefas de mais baixa prioridade.

Nas equações anteriores, foi utilizado um termo chamado $B^r_{i,j}$. Este tempo define o bloqueio remoto correspondente ao máximo tempo de espera experimentado pela tarefa τ_i quando requisita a j-ésima seção crítica global (correspondente ao segmento $C^{'}_{i,j}$). O tempo de bloqueio remoto total de uma tarefa τ_i é denotado por B^r_i e pode ser calculado utilizando a seguinte equação:

$$B^r_i = \sum_{q=1}^{s(q)-1} B^r_{i,q} \tag{3.16}$$

Como o valor $B^r_{i,q}$ é calculado depende de qual protocolo é utilizado. Muitos fatores podem influenciar este cálculo como política de enfileiramento em caso de bloqueio e preemptabilidade de seções críticas. Conforme os protocolos forem apresentados, será mostrado como calcular o valor de $B^r_{i,q}$ para cada um deles.

3.2.8 Protocolos para Sistemas Multiprocessados

Esta seção apresenta os protocolos existentes para sincronização em sistemas multiprocessados com escalonamento particionado e prioridade fixa. Primeiramente será apresentado brevemente o *Multiprocessor Priority Ceiling* Distribuído. Em seguida serão apresentados o *Multiprocessor Priority Ceiling* para Memória Compartilhada, o *Multiprocessor Stack Resource Policy* e o *Flexible Multiprocessor Locking Protocol*. Ao final será comentado sobre o *Flexible Multiprocessor Locking Protocol Improved* (FMLP+) por ser uma proposta recente.

3.2.8.1 *Multiprocessor Priority Ceiling* Distribuído

Este protocolo foi proposto por Rajkumar (RAJKUMAR et al., 1988) como sendo uma extensão do PCP(SHA et al., 1990) para sistemas distribuídos. Segundo este protocolo, para o caso de tarefas acessando recursos locais, este se reduz ao PCP (por consequência, para o caso de haver só um processador no sistema, o protocolo automaticamente se reduz ao PCP também). Neste protocolo, os recursos são alocados a processadores chamados *synchroniza-*

tion processors. No modelo mais simples, existe somente um *synchronization processor* no sistema, mas em um modelo mais complexo são admitidos vários destes processadores. Quando uma tarefa desejar acessar um recurso, se ela não residir no *synchronization processor*, ela deverá ser migrada para este temporariamente para acessar o recurso. Quando a tarefa liberar o recurso ela deverá voltar para o processador ao qual está alocada. Existem implementações que não fazem uso de migração, mas sim de mecanismos parecidos com RPC (BRANDENBURG; ANDERSON, 2008b) entre as tarefas e o gerente de recursos do *synchronization processor*. O mais relevante deste protocolo, que são as regras de aquisição de recursos, são iguais às do protocolo que será apresentado na próxima seção, que é o *Multiprocessor Priority Ceiling* para Memória Compartilhada.

3.2.8.2 *Multiprocessor Priority Ceiling* para Memória Compartilhada

Este protocolo foi desenvolvido após o MPCP Distribuído por Rajkumar (RAJKUMAR, 1990). A versão anterior levava em consideração sistemas cujos espaços de memória eram locais a cada processador (ou algo semelhante a arquiteturas com tempo de acesso à memória não uniforme - NUMA), e que a comunicação entre processadores era efetuada através de trocas de mensagens. Entretanto, essas premissas fogem da realidade quando entra em cena as atuais arquiteturas multiprocessadas, visto que estas, em geral, possuem uma visão simétrica da memória (acesso uniforme ou multiprocessamento simétrico).

O protocolo possui a definição de *ceiling*, que é um pouco diferente da definição utilizada no PCP. Seja P_H a mais alta das prioridades entre todas as prioridades das tarefas do sistema. Em um sistema monoprocessado, nenhuma seção crítica será executada com prioridade superior a P_H, então, o *ceiling* de qualquer seção crítica local deve ser mais baixa ou igual a P_H. Seções críticas globais, entretanto, devem ser executadas com prioridade superior a P_H, isto garante que os tempos de bloqueio sempre estarão em função dos tempos das seções críticas.

O *ceiling* de um recurso local é definido da mesma forma que no PCP,

ou seja, a mais alta prioridade entre as prioridades das tarefas que acessam o recurso. O *ceiling* de um recurso global é definido como sendo mais alto que P_H. Também existe o fato de, dado que a mais alta prioridade entre as prioridades das tarefas que acessam o semáforo S_i seja denotada por PS_i e $PS_i > PS_j$, então o *ceiling* de S_i é mais alto que o *ceiling* de S_j.

O *ceiling* de recursos globais é definido por uma composição de dois *ceilings*. O primeiro é o *ceiling* base P_G, tal que $P_G > P_H$. A segunda componente é PS_{G_i}, que é definida para cada processador P_i como sendo a mais alta das prioridades entre as tarefas que acessam o recurso e estão alocadas a processadores diferentes de P_i. Então o *ceiling* final será $P_G + PS_{G_i}$, sendo a segunda componente, como dito, variável para cada processador.

Para recursos globais (S_i, por exemplo), a tarefa que acessar tal recurso, irá faze-lo com sua prioridade ajustada a uma prioridade mais alta que todas as prioridades das tarefas do sistema (ou seja, PS_i), ou seja, de modo não preemptivo (não será preemptada por tarefas executando fora de seções críticas). Isto é feito para manter os tempos de bloqueio sempre em função dos tempos das seções críticas.

Quando uma tarefa tentar acessar um recurso global que não está livre, ela será então inserida em uma fila de espera ordenada por prioridades e em seguida bloqueada. Bloquear a tarefa significa que o processador poderá ser utilizado por tarefas de mais baixa prioridade. Quando um recurso global é liberado, será desbloqueada a tarefa que está no topo da fila (tarefa de mais alta prioridade, caso haja).

Este protocolo possui algumas restrições quanto ao aninhamento de seções críticas. Aninhamento de seções locais e globais ou apenas globais pode gerar inversões de prioridades descontroladas, e por isso não é permitido.

Uma extensão possível para o MPCP é, por exemplo, impedir que tarefas de mais baixa prioridade executem quando existirem tarefas de mais alta prioridade bloqueadas em recursos globais. Neste caso, a tarefa de mais alta prioridade, quando desbloqueada, não terá a possibilidade de ser atrasada por um tempo maior devido a tarefas de mais baixa prioridade executando seções críticas, visto que estas estarão com prioridade temporária superior. Uma ex-

tensão recente do protocolo (LAKSHMANAN et al., 2009) é utilizar *spin* para controle de execução. Nesta extensão, quando uma tarefa bloqueia, ele permanece em *spin* preemptivo para evitar que tarefas de prioridade mais baixa executem, bloqueiem, e venham a preemptar esta no futuro causando bloqueio local.

Para calcular o tempo de resposta de um conjunto de tarefas que compartilham recursos através do protocolo MPCP, primeiro deve-se calcular os tempos de bloqueio. Além dos bloqueios enunciados anteriormente, ainda existe a execução *back-to-back* devido a suspensão de tarefas. Este tipo de efeito ocorre quando tarefas de alta prioridade se auto-suspendem (ao bloquearem em recursos) causando *jitter* em tarefas de prioridade mais baixa (ao receberem o recurso). Para efetuar o cálculo do tempo de resposta deve-se calcular os tempos de bloqueio e aplicá-los nas equações de tempo de resposta apresentadas anteriormente.

O pior caso no tempo de resposta de uma determinada seção crítica pode ser obtido pela seguinte equação:

$$W_{i,k}^{'} = C_{i,k}^{'} + \sum_{\tau_u \in P(\tau_i)} \max_{1 \leq v \leq s(u) \& gc(u,v) > gc(i,k)} C_{u,v}^{'} \tag{3.17}$$

Na Equação 3.17, $gc(i,k)$ é o *ceiling* da seção crítica global $C_{i,k}^{'}$. Então, o tempo de execução de uma seção crítica pode sofrer acréscimo com tempos de seções críticas com *ceilings* mais altos. Em outra palavras, seções críticas podem ser preemptadas e sofrerem interferência/atraso de outras seções críticas.

Conforme foi apresentado na seção 3.2.7, $B_{i,j}^{r}$ representa o pior caso no tempo de bloqueio remoto para uma tarefa τ_i quando da aquisição da seção crítica global $C_{i,j}^{'}$. No MPCP este bloqueio é limitado pela convergência (onde $B_{i,j}^{r,0} = \max_{l > i \& (\tau_{l,u}^{'}) \in R(\tau_{i,j})} (W_{l,u}^{'})$):

$$B_{i,j}^{r,n+1} = \max_{l>i\&(\tau'_{l,u})\in R(\tau_{i,j})} (W'_{l,u})$$
$$+ \sum_{h<i\&(\tau'_{h,v}\in R(\tau_{i,j}))} \left(\left\lceil \frac{B_{i,j}^{r,n}}{T_h} \right\rceil + 1 \right) (W'_{h,v}) \quad (3.18)$$

Na Equação 3.18, no pior caso, a tarefa τ_i pode ser bloqueada por somente uma tarefa de mais baixa prioridade τ_l, que tem a maior seção crítica relativa ao recurso $R(\tau_{i,j})$ (entre todas as tarefas de mais baixa prioridade), e que já estava executando no interior da seção crítica quando a tarefa τ_i efetuou a tentativa de acesso a este recurso. Entretanto, a tarefa τ_i pode ser bloqueada por muitas tarefas de mais alta prioridade (que acessam $R(\tau_{i,j})$), varias vezes cada uma. Isto ocorre devido ao fato das filas de acesso aos recursos serem ordenadas pelas prioridades das tarefas no MPCP.

Para o cálculo do tempo de resposta total das tarefas, existem duas abordagens possíveis:

Para a versão baseada em suspensão: para o cálculo do tempo de resposta utilizando a versão baseada em suspensão, deve-se utilizar a Equação de interferência 3.11 e a de bloqueio local 3.12 utilizando os tempos de bloqueio apresentados anteriormente, ou seja:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i\&\tau_h\in P(\tau_i)} \left\lceil \frac{W_i^n + B_h^r}{T_h} \right\rceil C_h \quad (3.19)$$
$$+ s(i) \times \sum_{l>i\&\tau_l\in P(\tau_i)} \max_{1\leq k<s(l)} C'_{l,k}$$

Para a versão baseada em *spin*: para o cálculo do tempo de resposta utilizando a versão baseada em *spin*, deve-se utilizar a Equação de interferência 3.13 e a de bloqueio local para protocolos baseados em *spin* preemptivo 3.14 utilizando também os tempos de bloqueio apresentados anteriormente, ou seja:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n}{T_h} \right\rceil (C_h + B_h^r) + \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C'_{l,k} \tag{3.20}$$

É apresentado na Figura 8 um exemplo da execução do protocolo em um sistema com 4 tarefas, 2 CPUs e um recurso, que é representado pela área hachurada em xadrez. A escala de tempo é apresentada na parte inferior da figura. Neste exemplo, ocorre a seguinte sequência de eventos:

$t = 0$: a tarefa τ_2 inicia sua execução na CPU 0;

$t = 1$: a tarefa τ_3 inicia a execução na CPU 1;

$t = 2$: a tarefa τ_2 adquire o recurso, que anteriormente estava livre;

$t = 3$: a tarefa τ_0 vai para a fila de prontos na CPU 0, mas não pode executar por conta da tarefa τ_2 estar executando a seção crítica com prioridade elevada. Neste mesmo tempo, a tarefa τ_3 bloqueia (suspende sua execução) ao solicitar o recurso;

$t = 4$: a tarefa τ_1 inicia sua execução na CPU 1;

$t = 5$: a tarefa τ_1 bloqueia ao solicitar o recurso;

$t = 6$: a tarefa τ_2 libera o recurso, que é imediatamente atribuído a tarefa τ_1, que está na cabeça da fila (ordenada por prioridades). A tarefa τ_1 começa a execução da sua seção crítica. Neste mesmo tempo, a tarefa τ_0 inicia sua execução pelo fato da tarefa τ_2 ter se tornado preemptável novamente;

$t = 8$: a tarefa τ_0 bloqueia ao solicitar o recurso (que está com a tarefa τ_1). Neste mesmo momento, a tarefa τ_2 retoma sua execução;

$t = 9$: a tarefa τ_1 libera o recurso, que é atribuído à tarefa τ_0. A tarefa τ_0 inicia a execução da sua seção crítica.

$t = 11$: a tarefa τ_0 libera o recurso, que é atribuído a tarefa τ_3;

$t = 13$**:** finalmente a tarefa τ_3 termina de utilizar o recurso.

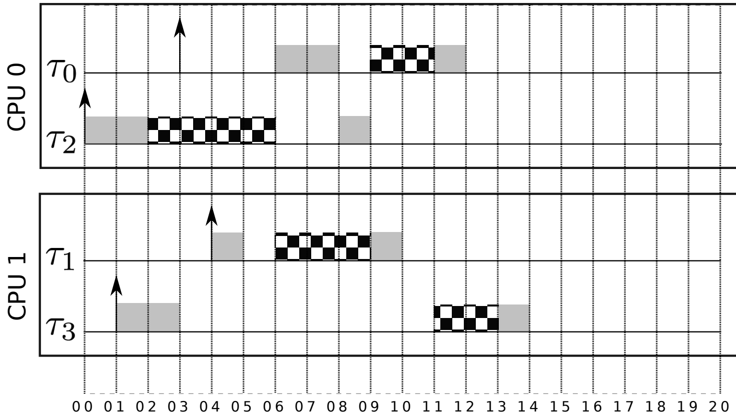


Figura 8: Exemplo de tarefas compartilhando recurso pelo MPCP

Para ilustrar o MPCP, esta subseção apresenta um exemplo numérico. O exemplo consiste de 3 processadores, 9 tarefas, 4 recursos globais e (S_1, S_2, S_3 e S_4). As tarefas apresentam as seguintes propriedades:

$$\tau_0: \; C_{0,1} = 1, C'_{0,1} = 2, C_{0,2} = 1$$
$$R_{0,1} = S_0$$
$$T_0 = 50$$
$$P(0) = P_1$$

$$\tau_1: \; C_{1,1} = 1, C'_{1,1} = 1, C_{1,2} = 2$$
$$R_{1,1} = S_2$$
$$T_1 = 85$$
$$P(1) = P_1$$

$$\tau_2: \; C_{2,1} = 2, C'_{2,1} = 1, C_{2,2} = 2$$
$$R_{2,1} = S_2$$
$$T_2 = 105$$

$$P(2) = P_1$$

$$\tau_3: \; C_{3,1} = 1, C'_{3,1} = 1, C_{3,2} = 1, C'_{3,2} = 1, C_{3,3} = 1$$
$$R_{3,1} = S_0, R_{3,2} = S_1$$
$$T_3 = 45$$
$$P(3) = P_2$$

$$\tau_4: \; C_{4,1} = 1$$
$$T_4 = 70$$
$$P(4) = P_2$$

$$\tau_5: \; C_{5,1} = 1, C'_{5,1} = 2, C_{5,2} = 1, C'_{5,2} = 1, C_{5,3} = 1$$
$$R_{5,1} = S_3, R_{5,2} = S_0$$
$$T_5 = 85$$
$$P(5) = P_2$$

$$\tau_6: \; C_{6,1} = 1, C'_{6,1} = 2, C_{6,2} = 1$$
$$R_{6,1} = S_3$$
$$T_6 = 135$$
$$P(6) = P_2$$

$$\tau_7: \; C_{7,1} = 2, C'_{7,1} = 2, C_{7,2} = 2$$
$$R_{7,1} = S_1$$
$$T_7 = 75$$
$$P(7) = P_3$$

$$\tau_8: \; C_{8,1} = 2, C'_{8,1} = 3, C_{8,2} = 2$$
$$R_{8,1} = S_1$$
$$T_8 = 100$$
$$P(8) = P_3$$

Para o cálculo do pior caso no tempo de resposta do conjunto de tarefas, foi utilizada a Equação 3.19 para a versão baseada em suspensão e a Equação 3.20 para a baseada em *spin*. Os resultados são apresentados na Tabela 1.

Tabela 1: Piores casos no tempo de resposta (exemplo com MPCP)

Tarefa	W_i **com Suspensão**	W_i **com *Spin***
τ_0	22	20
τ_1	10	23
τ_2	13	27
τ_3	26	18
τ_4	6	19
τ_5	26	31
τ_6	16	33
τ_7	22	19
τ_8	23	33

3.2.8.3 *Multiprocessor Stack Resource Policy* para Prioridade Fixa

O protocolo *Multiprocessor Stack Resource Policy* (MSRP) foi proposto por Gai (GAI et al., 2001b), baseado no SRP (BAKER, 1990). Apesar deste protocolo ter sido inicialmente proposto para EDF, ele pode ser facilmente utilizado com prioridade fixa (RM). O MSRP também classifica os recursos como locais e globais. Para recursos locais, este protocolo se comporta como o SRP (monoprocessador). Em geral, quando uma tarefa tentar alocar um recurso que está ocupado, ela terá duas opções: ser suspensa, como no MPCP, ou efetuar *busy-wait*, como em um *spinlock* normal. Para manter uma das características originais do SRP, que é o compartilhamento de pilha, o MSRP adota a segunda opção para recursos globais. Quando uma tarefa executa uma seção crítica, sua prioridade deve ser ajustada para a prioridade máxima do sistema, ou seja seções críticas são não preemptáveis.

A algoritmo utiliza a noção de *preemption threshold*, que é uma forma de priorização estática sobre tarefas com prioridade dinâmica. Quando uma tarefa adquire um recurso, todas as possibilidades de preempção que esta pode sofrer levam em conta os *preemption threshold* das outras tarefas, bem como

o dela própria.

Um dos objetivos do trabalho é integrar recursos com utilização eficiente de pilha. Quando uma tarefa é escalonada de maneira preemptiva (e não cooperativa), o número de *frames* de tarefas na pilha é igual ao número de níveis de prioridade. Por outro lado, escalonamento não preemptivo pode ser feito com apenas 1 *frame* na pilha, ao custo de ser menos responsivo e levar a mais conjuntos de tarefas não escalonáveis. Então, o artigo propõe a desabilitação seletiva de preempção, tendo como compromisso a manutenção da escalonabilidade do sistema. Isso pode ser alcançado com o uso de *preemption thresholds*, como proposto por (SAKSENA, 2000). A ideia básica é que, a cada tarefa do sistema é associada uma prioridade nominal π_i e um *preemption threshold* λ_i tal que $\pi_i \leq \lambda_i$. Quando uma tarefa é ativada, ela é inserida na fila de aptos de acordo com sua prioridade nominal, mas, quando ela começa a executar, a prioridade é aumentada para o *preemption threshold*. Isso faz com que a tarefa não possa ser preemptada por tarefas com prioridade nominal menor ou igual ao *preemption threshold*. Isto leva a definição de tarefas (T_i e T_j) mutuamente não preemptivas, ou seja, tarefas nas quais $(\pi_i \leq \lambda_j) \wedge (\pi_j \leq \lambda_i)$.

O trabalho observa que os *preemption threshold* são similares aos *ceilings* dos recursos do SRP. Quando uma tarefa acessa uma recurso, o *ceiling* de sistema é ajustado para o mais alto *ceiling* entre o do sistema e o do recurso. Esta característica também é uma maneira de limitar preempções, assim como os *preemption threshold*. Para tornar duas tarefas mutuamente não preemptivas, pode-se forçar o compartilhamento de um *pseudo* recurso ρ^k entre elas. O *ceiling* de ρ^k deverá ser o mais alto entre o *preemption level* das duas tarefas. Quando uma destas tarefas começar a executar, ela deverá bloquear ρ^k e reter este recurso até o término da ativação.

Utilizando a noção de *preemption threshold* e *preemption level*, o algoritmo do MSRP funciona através do seguinte conjunto de regras:

Regra 1: Como citado anteriormente, para recursos locais o algoritmo se comporta como o SRP. Em particular, são definidos um nível de preempção para cada tarefa, um *ceiling* para cada recurso e um *ceiling* de sistema para cada recurso.

Regra 2: O aninhamento de seções críticas locais é permitido, bem como o de locais e globais. Já o aninhamento de seções globais não é permitido, pois pode gerar *deadlocks*.

Regra 3: Para cada recurso global, cada processador P_k define um *ceiling* maior ou igual ao nível de preempção máximo entre as tarefas em P_k.

Regra 4: Quando uma tarefa τ_i, alocada ao processador P_k tentar acessar um recurso global R_j, então o *ceiling* de sistema Π_k (do processador P_k) deve ser elevado para $ceil(R_j)$, fazendo com que a tarefa fique não preemptável. Independentemente do recurso estar livre ou não, o *ceiling* Π_k é mantido com o valor de $ceil(R_j)$, sendo alterado para o valor antigo somente quando τ_i liberar o recurso. Em seguida ela deve verificar se o recurso está livre, e caso afirmativo, ela bloqueará o recurso e irá executar a seção crítica correspondente. Caso o recurso não estiver livre, ela será inserida em uma fila FIFO de acesso a este, e aguardará em *busy-wait*.

Regra 5: Quando uma tarefa τ_i, alocada ao processador P_k liberar um recurso R_j, ela deverá verificar a fila de acesso a este. Caso haja uma tarefa aguardando, ela irá conceder o recurso à tarefa. No caso de não haver tarefas aguardando, o recurso será marcado como livre. A última etapa é restaurar o *ceiling* Π_k para o seu antigo valor.

Outro fator é que, quando uma tarefa iniciar, ela não deverá ser bloqueada, apenas preemptada por tarefas de mais alta prioridade.

No caso de uma tarefa tentar acessar um recurso detido por uma tarefa em outro processador, ela será inserida em uma fila FIFO e aguardará em *busy-wait* (como um *spinlock* normal). Esta característica de efetuar bust-wait garante a possibilidade de utilização de pilha única (uma para cada processador), de forma a preservar esta característica herdada do SRP.

O tempo de bloqueio do MSRP, no pior caso, é igual a uma seção crítica (executada por uma tarefa com *preemption level* menor), mais o tempo de *busy-wait* gasto na espera por recursos globais. Desta forma, o tempo de bloqueio B_i para uma tarefa T_i será:

$$B_i = max(B_i^{local}, B_i^{global}, B_i^{pseudo}) \qquad (3.21)$$

Onde têm-se que:

B_i^{local}: Define o bloqueio devido a recursos locais.

B_i^{global}: Define o bloqueio devido a recursos globais.

B_i^{pseudo}: Define o bloqueio devido a pseudo-recursos. Ocorre pelo fato da tarefa T_i poder ser mutuamente não preemptiva com relação a tarefas no mesmo processador.

As equações completas para obter-se cada componente da Equação 3.21 podem ser vistas no trabalho original (GAI et al., 2001b).

É apresentado na Figura 9 um exemplo de utilização do protocolo em um sistema com 4 tarefas, 2 CPUS e 1 recurso. Este exemplo é o mesmo utilizado no MPCP. Abaixo é descrito a sequência de eventos do exemplo.

$t = 0$: a tarefa τ_2 inicia sua execução na CPU 0;

$t = 1$: a tarefa τ_3 inicia a execução na CPU 1;

$t = 2$: a tarefa τ_2 adquire o recurso, que anteriormente estava livre;

$t = 3$: a tarefa τ_0 vai para a fila de prontos na CPU 0, me não pode executar por conta da tarefa τ_2 estar executando a seção crítica com prioridade elevada. Neste mesmo tempo, a tarefa τ_3 bloqueia (inicia o o *busy-wait*) ao solicitar o recurso;

$t = 4$: a tarefa τ_1 é inserida na fila de aptos na CPU 1, mas não pode iniciar sua execução devido ao *busy-wait* da tarefa τ_3;

$t = 6$: a tarefa τ_2 libera o recurso, que é imediatamente atribuído a tarefa τ_3, que está na cabeça da fila (FIFO, na qual ela é o único elemento). A tarefa τ_3 começa a execução da sua seção crítica. Neste mesmo tempo, a tarefa τ_0 inicia sua execução pelo fato da tarefa τ_2 ter se tornado preemptável novamente;

$t = 8$**:** a tarefa τ_3 libera o recurso. Neste mesmo instante, a tarefa τ_0 solicita o recurso, que lhe é atribuído imediatamente, por estar livre. Neste mesmo momento a tarefa τ_1 retoma sua execução;

$t = 9$**:** a tarefa τ_1 bloqueia ao solicitar o recurso;

$t = 10$**:** a tarefa τ_0 libera o recurso, que é atribuído a tarefa τ_1. A tarefa τ_1 sai do *busy-wait* e acessa sua seção crítica;

$t = 13$**:** a tarefa τ_1 termina de utilizar o recurso.

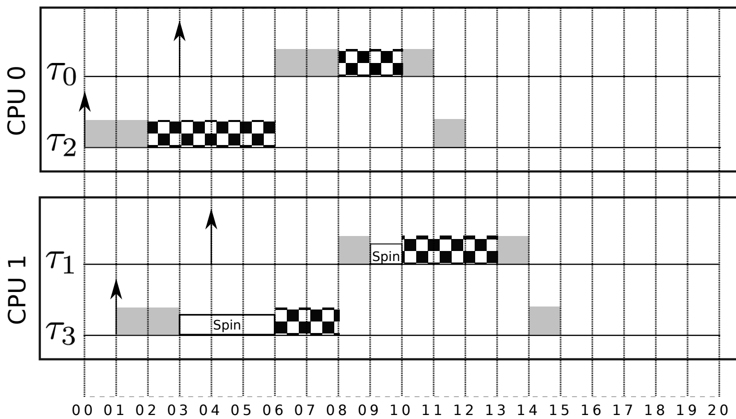


Figura 9: Exemplo de tarefas compartilhando recurso pelo MSRP

3.2.8.4 *Flexible Multiprocessor Locking Protocol* - FMLP

O protocolo *Flexible Multiprocessor Locking Protocol* foi proposto por Block (BLOCK et al., 2007) para escalonamento EDF global e particionado e Pfair PD^2 (ANDERSON; SRINIVASAN, 2001). A extensão do protocolo para prioridade fixa particionada é apresentada em (BRANDENBURG; ANDERSON, 2008b). O protocolo é considerado flexível por duas razões. A primeira é que pode ser utilizado tanto com escalonamento particionado quanto global. A segunda é pelo fato dele ser agnóstico quando ao tipo de controle de execução adotado (suspensão ou *spin*).

Este protocolo se baseia na definição de dois tipos de recursos globais, os *short* e os *long*. A definição se baseia no tempo de acesso a estes (tempo de seção crítica). Recursos *short* são acessados via *locks* baseados em fila, enquanto os *long* são acessados de acordo com um protocolo baseado em semáforo. A classificação (*short* ou *long*) dos recursos fica a cargo do programador.

Na definição original, o FMLP desencoraja preempção de tarefas que estão de posse de recursos. Quando uma tarefa detentora de um recurso é preemptada, outras tarefas que estão esperando este mesmo recurso podem ser substancialmente atrasados. Então, segundo o FMLP, recursos *short* são acessados de modo não preemptivo, enquanto recursos *long* são acessados utilizando herança de prioridade. Como esta dissertação leva em conta somente a versão para prioridade fixa com escalonamento particionado, as futuras descrições serão baseadas em (BRANDENBURG; ANDERSON, 2008b). No MPCP, tarefas detentoras de recursos globais devem ter sua prioridade elevada para não serem preemptadas por tarefas executando sem a posse de recursos. O FMLP utiliza uma abordagem simplificada, que consiste em elevar a prioridade das tarefas desbloqueados de forma igual (prioridade máxima, ou seja, de forma não preemptiva). Tarefas com prioridade elevada (com posse de recursos), são escalonados com ordem FIFO. Em recursos *short* , embora as respectivas seções críticas sejam executadas de forma não preemptiva, o ajuste de prioridade não é necessário, pois para esse tipo de recurso não há suspensão de tarefas. Esta elevação de prioridade não é utilizada nas outras variantes do FMLP (para Pfair, EDF, entre outras).

Dadas as características do protocolo apresentadas anteriormente, são definidas as seguintes regras para aquisição de recursos segundo o protocolo FMLP, supondo o não aninhamento de recursos de tipos diferentes:

Requisição de recursos *short*: uma tarefa τ_i executa uma requisição para um recurso global R_j. Esta tarefa se torna não preemptável. Se o recurso estiver disponível, este é atribuído a tarefa, e se não estiver, a tarefa deve ser bloqueada (*spin* não preemptivo). A ordem de atribuição do recurso às tarefas é em ordem FIFO. A tarefa se torna preemptável quando liberar o recurso.

Requisição de recursos *long*: uma τ_i executa uma requisição para um recurso global R_j. Se o recurso estiver disponível a tarefa se torna não preemptável (o que é chamado de *priority boost* no artigo) e entra na seção crítica, caso contrário ela será bloqueada (suspensa). A ordem de atribuição do recurso às tarefas é em ordem FIFO. A tarefa se torna preemptável (via prioridade original) quando liberar o recurso.

O fato de não precisar de ajuste de prioridade nos recursos *short* é devido ao *busy-wait/spin*. A tarefa que recebe um recurso *short* após um tempo de espera já estava em execução, logo não precisa de uma prioridade elevada para voltar a executar o mais cedo possível, pois já estava efetivamente em execução (em *busy-wait* de modo não preemptivo).

Para recursos locais, o protocolo sugere o uso do SRP, pois a definição trata basicamente de recursos globais sendo o tratamento para os locais de livre escolha (desde que seja uma escolha adequada). Segundo o artigo, a classificação entre recursos *short* e *long* serve apenas para recursos globais.

O tempo máximo de bloqueio pode ser calculado pela soma dos seguintes fatores:

***Boost blocking*:** Ocorre quando τ_i não pode executar devido a uma tarefa de mais baixa prioridade executando uma seção crítica *long*. Se a tarefa τ_i é a de mais baixa prioridade no processador ou as tarefas de prioridade mais baixa não acessem recursos, então este fator é igual a zero.

***Arrival blocking*:** Ocorre quando τ_i se torna apto à executar (no *release* ou retornando de uma suspensão) mas não pode executar devido a uma tarefa executando de forma não preemptiva. Se nenhuma das tarefas de mais baixa prioridade compartilhar recursos *short*, então este fator é igual a zero.

***Short blocking*:** Ocorre quando τ_i solicitar um recurso *short* que atualmente está de posse de outra tarefa (em outro processador). Tarefas locais não causam este tipo de bloqueio. Se τ_i não utiliza recursos *short*, então este fator é igual a zero.

***Local blocking*:** Ocorre quando τ_i solicitar um recurso *long* que atualmente está de posse de outra tarefa (em outro processador). Tarefas locais não causam este tipo de bloqueio (causam interferência) bem como tarefas de mais baixa prioridade (que é contabilizado no *Boost blocking*). Se τ_i não utiliza recursos *short*, então este fator é igual a zero.

***Deferred blocking*:** Ocorre quando uma tarefa de mais alta prioridade adia sua execução (via suspensão) para posteriormente retomar a execução e voltar a competir por processador com τ_i. Se nenhuma das tarefas de mais alta prioridade utilizar recursos *long*, então este fator é igual a zero.

As equações completas para cada componente da fórmula enumerada anteriormente podem ser vistas no trabalho original (BRANDENBURG; ANDERSON, 2008b). Em (BRANDENBURG; ANDERSON, 2008a) foi feita uma comparação do FMLP com outros protocolos utilizando a plataforma LITMUS (CALANDRINO et al., 2006).

Com relação ao aninhamento de recursos, o problema pode ser tratado com o agrupamento de *locks*. Neste caso, cada grupo contem somente recursos *short* ou *long*. Dois recursos R_0 e R_1 pertencem ao mesmo grupo se e somente se uma tarefa faz uma requisição ao recurso R_0 que está contida dentro de uma requisição do recurso R_1 e ambos os recursos são *short* ou *long*. Recursos não aninhados possuem grupos próprios, o que facilita a implementação.

Para propósitos de análise de escalonabilidade, sem perda de generalidade, será assumido que um sistema contém somente recursos *short* ou somente *long*. Não será entrado no mérito da classificação de recursos. As análises apresentadas nas próximas subseções, são todas baseadas na análise do MPCP proposta por (LAKSHMANAN et al., 2009).

Utilizando a abordagem apresentada anteriormente (LAKSHMANAN et al., 2009), para o cálculo do pior caso no tempo de resposta de uma seção crítica $C'_{i,k}$, existem duas abordagens possíveis:

Para versão *long*: para a versão *long*, que é baseada em suspensão, o pior

caso deverá ser calculado pela seguinte equação:

$$W'_{i,k} = C'_{i,k} + \sum_{\tau_u \in P(\tau_i)} \max_{1 \leq v \leq s(u)} C'_{u,v} \quad (3.22)$$

Ou seja, no pior caso, uma tarefa desbloqueado em uma seção crítica deverá esperar pela maior seção crítica de cada tarefa ativa, supondo que todas os tarefas do processador estejam desbloqueadas em suas seções críticas. Isto ocorre devido a não preemptividade das seções críticas, pelo fato de tarefas sob o mesmo nível de prioridade (não preemptividade pode ser encarada como todas as seções sendo executadas com prioridade máxima) serem executadas em ordem FIFO em um processador.

Para versão *short*: Para a versão *short* o cenário é consideravelmente diferente e deve ser calculado pela seguinte equação:

$$W'_{i,k} = C'_{i,k} \quad (3.23)$$

Na equação anterior (3.23), o pior caso de uma seção crítica é o próprio tempo de execução da seção crítica. Isto ocorre pelo fato do FMLP *short* utilizar *spin* não preemptivo como mecanismo de controle de execução. Desta maneira, somente uma tarefa pode estar bloqueada em algum recurso em um determinado instante, logo, quando desbloqueada a tarefa não precisará competir com nenhuma outra na utilização do processador em sua seção crítica. Embora o pior caso da seção seja menor, o bloqueio local tende a ser maior devido ao *spin* não preemptivo.

Finalmente, $B^r_{i,j}$ (que representa o bloqueio remoto para uma tarefa τ_i na aquisição da seção crítica global $C'_{i,j}$) deve ser calculado da seguinte maneira:

Para versão *long*: como o enfileiramento é FIFO, no pior caso, uma tarefa deverá esperar por todas as tarefas que acessam o recurso (soma de todos os tempos de computação de todas as seções críticas) na versão baseada em suspensão:

$$B^r_{i,j} = \sum_{h \neq i \& (\tau'_{h,v} \in R(\tau_{i,j}))} W'_{h,v} \quad (3.24)$$

Esta equação é caracterizada por um somatório pois, a ordem de acesso é FIFO, e no pior caso, a tarefa em questão está posicionada no fim da fila, estando na ativa todas as outras tarefas com intenção de acessar o recurso.

Para versão *short*: para a versão baseada em *spin*, $B^r_{i,j}$ deve ser calculado como a soma da maior seção crítica relativa ao recurso $B^r_{i,j}$ de cada processador, como segue:

$$B^r_{i,j} = \sum_{\forall p | p \neq P(\tau_i)} \max_{\forall h | P(\tau_h) = p \& (\tau'_{h,v} \in R(\tau_{i,j}))} W'_{h,v} \quad (3.25)$$

Esta diferença ocorre pois, na versão baseada em *spin* (não preemptivo) pode haver somente uma tarefa bloqueada em algum recurso em cada processador. Diferentemente da versão *long* (baseada em suspensão), onde podem haver várias tarefas bloqueadas em recursos em cada processador.

Para o cálculo dos tempos de resposta as seguintes abordagens devem ser utilizadas, considerando os tempos de bloqueio e fatores apresentados para este protocolo:

Para a versão *long*: Para a versão *long* do protocolo deve-se utilizar equação de tempo de resposta para protocolos baseados em suspensão, ou seja:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h < i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n + B_h^r}{T_h} \right\rceil C_h \quad (3.26)$$
$$+ s(i) \times \sum_{l > i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C'_{l,k}$$

Para a versão *short*: Para a versão *short*, deve ser utilizada a versão da equação de tempo de resposta para protocolos baseados em *spin* não preemptivo, ou seja:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n}{T_h} \right\rceil (C_h + B_h^r) + \max_{l>i \& \tau_l \in P(\tau_i) \& 1 \leq k < s(l)} (C'_{l,k} + B_{l,k}^r) \quad (3.27)$$

É apresentado na 9 um exemplo de utilização do protocolo em um sistema com 4 tarefas, 2 CPUs e 1 recurso. Este exemplo é o mesmo utilizado no MPCP e MSRP, sendo que o recurso é considerado *long*, pois se fosse *short*, ficaria idêntico ao último protocolo (MSRP). Neste exemplo, ocorre a seguinte sequência de eventos:

$t = 0$: a tarefa τ_2 inicia sua execução na CPU 0;

$t = 1$: a tarefa τ_3 inicia a execução na CPU 1;

$t = 2$: a tarefa τ_2 adquire o recurso, que anteriormente estava livre;

$t = 3$: a tarefa τ_0 vai para a fila de prontos na CPU 0, mas não pode executar por conta da tarefa τ_2 estar executando a seção crítica de forma não preemptiva. Neste mesmo tempo, a tarefa τ_3 bloqueia (suspende sua execução) ao solicitar o recurso;

$t = 4$: a tarefa τ_1 inicia sua execução na CPU 1;

$t = 5$: a tarefa τ_1 bloqueia ao solicitar o recurso;

$t = 6$: a tarefa τ_2 libera o recurso, que é imediatamente atribuído a tarefa τ_3, que está na cabeça da fila (ordem FIFO). A tarefa τ_3 começa a execução da sua seção crítica. Neste mesmo tempo, a tarefa τ_0 inicia sua execução pelo fato da tarefa τ_2 ter se tornado preemptável novamente;

$t = 8$: a tarefa τ_0 bloqueia ao solicitar o recurso (que está com a tarefa τ_1). Neste mesmo momento, a tarefa τ_2 retoma sua execução. Também

ocorre a liberação do recurso pela tarefa τ_3, que é atribuído a tarefa τ_1;

$t = 11$: a tarefa τ_1 libera o recurso, que é atribuído a tarefa τ_0. A tarefa τ_0 inicia a execução da sua seção crítica. Assim termina o sequenciamento do recurso entre as tarefas.

Para ilustração do cálculo do tempo de resposta, o exemplo utilizado no protocolo MPCP foi repetido para o FMLP também. Os tempos de resposta obtidos são mostrados na Tabela 2, sendo que para o cálculo dos tempo de resposta da versão *long* foi utilizada a Equação 3.26 e para a versão *short* a Equação 3.27.

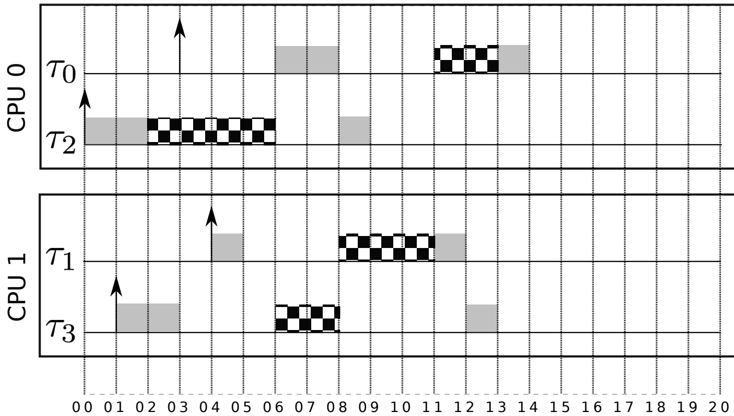


Figura 10: Exemplo de tarefas compartilhando recurso pelo FMLP *long*

3.2.8.5 *Flexible Multiprocessor Locking Protocol Improved*

O *Flexible Multiprocessor Locking Protocol Improved* (FMLP+) (BRANDENBURG, 2011), também conhecido como *FIFO Multiprocessor Locking Protocol*, busca criar uma versão preemptiva do FMLP. Assim como ocorre no MPCP, seções críticas podem ser preemptadas no FMLP+, através de regras de desempate. Empates ocorrem quando duas ou mais tarefas estão aptas a

Tabela 2: Piores casos no tempo de resposta (exemplo com FMLP)

Tarefa	W_i **com a versão *long***	W_i **com a versão *short***
τ_0	17	6
τ_1	10	10
τ_2	13	14
τ_3	31	13
τ_4	6	14
τ_5	22	21
τ_6	16	23
τ_7	17	11
τ_8	18	15

executar suas seções críticas em uma mesma CPU. No FMLP este desempate ocorre em ordem FIFO ao desbloqueio (primeira a ser desbloqueada, primeira a executar). No FMLP+ este desempate ocorre pela ordenação FIFO relacionada com o instante da requisição do recurso (utilizando *timestamps*) e não com o instante de debloqueio. Diferentemente do FMLP, o FMLP+ não possui divisão entre versão *short* e *long* pois na versão baseada *short* (tratada com *spin*) naturalmente não ocorrem empates, pois pode haver somente uma tarefa bloqueada em cada processador (a tarefa bloqueada estará em *spin*, não havendo maneira de outra tarefa executar e bloquear).

3.2.9 Comparação Entre os Protocolos

As seções anteriores se preocuparam somente em apresentar os protocolos. Entretanto, é interessante comparar de alguma forma tais protocolos e existem várias maneiras para efetuar tais comparações. A maneira mais intuitiva é comparar analiticamente as equações que definem o tempo de bloqueio de cada protocolo. Entretanto esta é uma opção pouco viável, visto que tais equações geralmente são extremamente complexas pois envolvem operações sobre conjuntos e muitas variáveis.

Uma segunda opção, que na prática é frequentemente utilizada, é a comparação de percentagem de conjuntos de tarefas escalonáveis por cada protocolo. Para tanto, deve-se gerar conjuntos de tarefas segundo algum cri-

tério e avaliar a escalonabilidade perante cada protocolo (ou é escalonável ou não é). Para sistemas multiprocessados, a definição dos conjuntos de testes deve levar em conta muitos fatores, como o número de tarefas, período, tempo de computação, *deadline*, taxa de compartilhamento de recursos, número de processadores entre outros.

Uma terceira opção é a simulação. Neste caso também são gerados conjuntos de tarefas que são simulados afim de se observar eventos como perdas de *deadlines*. Este tipo de comparação atua como um teste necessário mas não suficiente sobre o sistema na presença de recursos, onde a perda de um *deadline* implica na não escalonabilidade, mas a não observação de perdas não reflete a escalonabilidade do sistema. Para este tipo de comparação deve-se utilizar os mesmos critérios de simulação, de forma que esta seja imparcial em relação aos protocolos. Com simulação pode-se observar a contagem de chaveamentos de contexto ou número de preempções efetuadas, mas sem mensurar o custo real de tais operações. Uma das vantagens da simulação é que ela evita a utilização das equações na comparação, visto que para cada protocolo, as equações correspondentes podem apresentar graus diferentes de pessimismo.

Uma quarta alternativa é a experimentação. Para tal, deve-se implementar os protocolos os quais se deseja comparar em um sistema real e avaliar a execução destes. Em experimentação pode-se observar efeitos como o *overhead* do protocolo/implementação, avaliar a escalabilidade, chaveamentos de contexto entre outros. A experimentação também é útil para levantar parâmetros que serão utilizados em testes de escalonabilidade.

3.2.9.1 Comparação do MSRP com o MPCP

Em (Di Natale et al., 2003) é apresentada uma comparação entre o MPCP e o MSRP. Segundo o trabalho, as equações para o tempo de bloqueio do MPCP, embora pessimistas, podem ser reduzidas com um exame das particularidades do conjunto de tarefas que será analisado. Também é apontado que o MSRP pode gastar muitos ciclos de CPU efetuando *busy-wait*, embora tenha as equações de tempo de bloqueio mais simples, pois não lidam com

suspensão. O trabalho é baseado em uma arquitetura com somente 2 CPUs. Então os resultados não são conclusivos, pois com o aumento no número de CPUs, o protocolo MSRP tende a desperdiçar mais e mais ciclos de máquina e isso não chegou a ser observado. Outro ponto é que no trabalho, o MSRP é utilizado com EDF como política de atribuição de prioridades, o que torna a comparação um pouco injusta, sendo atenuada com a escolha de períodos harmônicos para aumentar a utilização com RM. Embora o trabalho utilize uma plataforma automotiva como referência, esta só serve para fornecer os parâmetros (tempos em geral) para a geração dos conjuntos de tarefas de teste, que serão avaliados com base nas equações de tempo de resposta.

Os conjuntos de tarefas de teste foram gerados de três maneiras. Na primeira, foram geradas tarefas aleatórias (entre 6 e 10 tarefas por CPU), variando apenas a percentagem de recursos locais (0%, 25%, 50%, 75% e 100% de recursos locais). Na segunda foram geradas tarefas nas mesma configurações das anteriores, mas utilizando períodos harmônicos. Finalmente na terceira situação, foram geradas tarefas aleatórias com base em uma aplicação de controle de *power-train* automotivo real.

Para a primeira configuração de tarefas, o MSRP sempre apresentou melhor desempenho, que em partes pode ser explicado pelo limite de utilização do EDF, já que os conjuntos começam a perder escalonabilidade na faixa dos 70%. Escolhendo períodos harmônicos (para a segunda alternativa) os resultados mostraram que nenhum protocolo superou o outro no caso geral, visto que agora a utilização do RM é igual a 1. Na terceira situação, onde foram gerados conjuntos não genéricos, mas com parâmetros cuidadosamente ajustados, não houve conjuntos escalonáveis com RM. O trabalho aponta que o MPCP apresenta melhor desempenho quando a predominância dos recursos é global e o MSRP apresenta desempenho melhor quando a maioria dos recursos é local. Outra questão apontada é que para o caso do *power train*, como as seções críticas foram curtas, os mecanismos de *busy-wait* do MSRP foram favorecidos, indicando que seções críticas de tamanhos diferentes podem exigir protocolos diferentes. Uma das contribuições do artigo é que a utilização do *busy-wait* do MSRP pode levar a sistemas não escalonáveis, pois o pior caso no tempo de resposta aumenta enquanto se mantém o processador *idle*

3.2.9.2 Comparação do FMLP com o MPCP

Em (BRANDENBURG; ANDERSON, 2008a) é apresentada uma comparação entre o MPCP distribuído, o MPCP para memória compartilhada e FMLP. O trabalho considera somente escalonamento particionado e prioridade fixa. A versão do MPCP distribuído é implementada com o uso de um estratégia parecida com RPC entre as tarefas e o agente que efetivamente efetua o acesso no *syncronization processor*. Para realizar as comparações eles implementaram os protocolos em uma plataforma de testes baseada no Linux chamada LITMUS^{RT}. Através das implementações eles levantaram os *overheads* e custos de escalonamento dos protocolos. Os custos foram utilizados com os conjuntos de tarefas em testes de escalonabilidade. Para avaliar a escalonabilidade, foram utilizados testes baseados em demanda estendidos com os custos e *overheads*.

Os conjuntos de tarefas foram gerados com a utilização individual de cada tarefa distribuída uniformemente no intervalo [0,001, 0,1]. Os períodos foram gerados em quatro intervalos: [3ms-33ms], [10ms-100ms], [33ms-100ms], [100ms-1000ms]. Para cada processador foram geradas tarefas até completar o limite de utilização ou o número de tarefas chegar a 30. Em seguida são geradas as requisições de acesso. Eles alegam que gerando conjuntos de tarefas desta forma eles evitam o desvios de resultado causados pelas heurísticas de *bin-packing*. Os tempos de acesso aos recursos também foram gerados de forma aleatória, com cada tarefa tendo entre 0 e 9 recursos (uniformemente), sendo que cada recurso possui tempo de acesso uniformemente distribuído no intervalo [0,1μs, L], com L no intervalo [0,5μs, 15,5μs]. O número de processadores considerado nos experimentos varia de 4 a 16. Uma das limitações dos trabalho é efetivamente o levantamento de parâmetros através de uma arquitetura real, visto que para outra arquitetura (por exemplo, uma arquitetura de grande porte), estes parâmetros podem assumir valores completamente diferentes, devido a contenção de barramento e comportamento diferente de *cache*. O número de tarefas foi utilizado para determinar o número de recursos através da fórmula $\frac{K.N}{\alpha.m}$, onde α é o grau de compartilhamento, e é escolhido no intervalo $\{0,5, 1, 2, 3, 4\}$. O valor de m não tem a sua semântica

explicada.

Utilizando um cenário de quatro CPUs, eles observam que suspensão nunca apresenta melhor performance que *spin*. Segundo o trabalho, todos os conjuntos de tarefas foram escalonados com *spin*, mas nem todos o foram com métodos baseados em suspensão. Suspensão foi viável com um número pequeno de recursos (K pequeno, baixa utilização e α grande). No artigo também é avaliada a escalabilidade dos protocolos. Para escalabilidade a versão *short* do FMLP apresentou melhores resultados, o que pode ser explicado pelas seções críticas extremamente curtas. Em termos de *overhead*, todas as abordagens baseadas em suspensão obtiveram piores resultados. Partindo das conclusões do artigo, em comparação com o artigo anterior, não se deve generalizar os resultados sobre a adequação de cada protocolo, pois tais resultados são claramente dependentes do conjunto de tarefas, que são gerados de forma *pseudo*-aleatória, utilizando alguns critérios pré definidos. Outra questão é que o artigo utiliza análises muito pessimistas para protocolos baseados em suspensão, sendo que já foram desenvolvidas técnicas mais precisas de análise.

3.2.9.3 Comparação de Diferentes Políticas de Controle de Execução no MPCP

Em (LAKSHMANAN et al., 2009) também é efetuada uma comparação entre o MPCP e uma versão alterada deste, que utiliza *spin* ao invés de suspensão. Esta segunda versão apresenta um comportamento próximo ao FMLP, exceto pelo enfileiramento FIFO (no FMLP). Neste trabalho é utilizada uma abordagem mais holística, onde o particionamento é efetuado de forma consciente dos recursos, e não uma alocação aleatória de tarefas em processadores, assim como foi feito no artigo anterior. Com a análise proposta neste artigo, foi mostrado que protocolos baseados em suspensão se comportam melhor quando os custos de preempção são baixos (menores que $160\mu s$) e seções criticas são maiores que as abordadas em (BRANDENBURG; ANDERSON, 2008a). A métrica utilizada neste trabalho, é o número de CPUs necessárias pra comportar um conjunto de tarefas dada a presença de recursos e utilizando as duas políticas de controle de execução. O artigo considera os 3 fatores

que podem afetar as políticas de controle de execução: o tamanho das seções críticas, número de tarefas por CPUs e o número de usuários de cada recurso.

O primeiro cenário é um contexto de *overhead* zero e variando o tamanho das seções críticas ($5\mu s$ até $1280\mu s$) em um sistema com 8 processadores com utilização completa, com 5 tarefas cada e 2 seções críticas por tarefa e 2 usuários para cada recurso. Inicialmente as duas abordagens apresentam performance similar, mas, conforme o tamanho da seção crítica aumenta, a abordagem baseada em *spin* começa a exigir mais e mais processadores. Isto ocorre em parte pela perda de tempo de processamento durante o *spin*, que no caso da suspensão, este tempo pode ser efetivamente utilizado por outras tarefas. Entretanto, as duas abordagens demandam mais processadores conforme o tamanho das seções críticas aumenta.

Em um segundo cenário, é aumentado o número de tarefas por processador, visto que isto tende a aumentar o número de preempções e de inversões de prioridade. São utilizadas seções críticas com duração de $500\mu s$. Neste caso, partindo dos 8 processadores e adicionando tarefas, o número de processadores exigidos também deverá aumentar. Também neste caso, o número de processadores aumenta consideravelmente mais para a abordagem baseada em *spin*, embora ambos os esquemas exijam mais processadores.

Se forem considerados os *overheads*, o artigo mostra, assim como os outros trabalhos, que a abordagem baseada em *spin* pode ser útil para situações onde as seções críticas são pequenas.

O que cada trabalho faz, é mostrar que um determinado protocolo apresenta performance superior, considerando uma metodologia de geração de conjuntos de tarefas e uma técnica de análise. A geração do conjunto de tarefas e o critério de avaliação certamente geram um *viés* nos resultados.

3.3 CONCLUSÕES

Este capítulo apresentou uma visão geral sobre sincronização em sistemas de tempo real monoprocessados e multiprocessados. Para sistemas monoprocessados, foram apresentados protocolos como *Priority Ceiling Protocol*, *Priority Inheritance Protocol* e *Stack Resource Policy*. Para sistemas mul-

tiprocessados, os tipos de bloqueio foram ilustrados, bem como as políticas de enfileiramento de tarefas e de controle de execução. Também foram apresentados os dois algoritmos de particionamento mais conhecidos que levam recursos em consideração. Uma análise de escalonabilidade por tempo de resposta para tarefas envolvendo recursos também foi apresentada, como uma ferramenta importante para utilização dos protocolos.

É apresentado na Tabela 3 um resumo das características de cada protocolo multiprocessado apresentado. A primeira coluna enumera os protocolos enquanto a segunda, as características. As células marcadas com “x” indicam verdadeiro para uma determinada característica.

É dito na tabela o seguinte para o MPCP: para recursos globais a contenção é feita por suspensão e enfileiramento por prioridades, sendo que há possibilidade de preempção (somente por tarefas que também estejam em seção crítica) e para recursos locais, a lógica é parecida, com a diferença que as tarefas podem ser preemptadas por outras executando fora de seções críticas.

Para o MSRP é dito que: a contenção é realizada por espera ocupada (*busy-wait*) com enfileiramento FIFO para acesso ao recurso. A lógica para acesso a recursos locais é a mesma. Outra característica é que seções críticas não são preemptáveis.

Por fim para o FMLP é dito que: como há dois tipos de recursos, há duas formas de tratar recursos globais. Para recursos *short* o protocolo trata a contenção com espera ocupada e enfileiramento FIFO, sendo a seção crítica não preemptável. Para recursos *long* o protocolo trata a contenção com suspensão e enfileiramento FIFO para acesso ao recurso. Recursos *long* também não são preemptáveis. O protocolo não trata especificamente de recursos locais, mas indica possibilidades, como o uso do SRP.

O FMLP tem características herdadas tanto do MPCP quanto do MSRP. Uma das herdadas do MPCP é a suspensão na contenção por disputa de recursos *long*, com a diferença de utilizar enfileiramento FIFO. Com relação ao MSRP, ele utiliza o enfileiramento FIFO de forma semelhante, para ambos os tipos de recurso. Os recursos *short* são acessados de uma forma muito similar ao MSRP. De certa forma o FMLP para recursos *short* opera identicamente ao MSRP.

Com relação ao aninhamento de recursos, a estratégia adotada no FMLP pode ser aplicada aos demais protocolos. Entretanto esta estratégia aumenta a granularidade das seções críticas. Por exemplo, pode-se supor três recursos R_0, R_1 e R_2 e duas tarefas τ_0 e τ_1. A tarefa τ_0 acessa R_0 e R_1 e a tarefa τ_1 acessa R_1 e R_2. Segundo o FMLP, os três recursos devem ser agrupados através do mesmo *group lock* e mesmo τ_0 não utilizando R_2, este recurso será bloqueado via *group lock*.

De um modo geral, os trabalhos não são conclusivos, pois dependem muito da condução dos testes. Em cada trabalho, tomando uma abordagem diferente, é mostrado como é possível fazer com que um protocolo se sobressaia sobre os demais. Para conjuntos aleatórios de tarefas, não é observada muita diferença entre o MPCP e o MSRP. Entretanto, o mesmo não pode ser dito sobre conjuntos cuidadosamente criados. Outra conclusão é que, de um modo geral, não existe um protocolo que seja melhor em todos os casos, pois cada protocolo responde de uma maneira a determinados conjuntos de teste. Por exemplo, o FMLP e o MSRP tentam distribuir de forma mais igualitária os bloqueios, enquanto o MPCP tenta privilegiar as tarefas de mais alta prioridade. Fazer com que uma tarefa de mais alta prioridade tenha as mesmas penalidade de bloqueio que uma de baixa pode não ser uma boa escolha quando os períodos sejam muito diferentes, pois a tarefa mais prioritária perderá mais facilmente seu *deadline* por conta do seu menor período.

Tabela 3: Comparação entre os protocolos

Características / Protocolo	**Controle de execução**		**Ordem de acesso**		**Preemptável**
	Suspensão	**Busy-wait**	**Prioridade**	**FIFO**	
MPCP:					
global	x		x		x
local	x		x		x
MSRP:					
global		x		x	
local		x		x	
FMLP:					
short		x		x	
long	x			x	
local	x	x	x	x	
FMLP+:	x			x	x

4 PROPOSTAS DE VARIAÇÕES PARA O PROTOCOLO *MULTIPROCESSOR PRIORITY CEILING*

No capítulo anterior foi apresentada uma revisão sobre conceitos relacionados à sincronização para sistemas de tempo real monoprocessados e multiprocessados com escalonamento particionado e prioridade fixa. Também foram apresentados os protocolos existentes na literatura para esta classe de sistemas. As características de cada protocolo são sumarizadas na Tabela 4. Nesta tabela são apresentadas características como política de enfileiramento em caso de bloqueio (ordem de acesso ao recurso), política de controle de execução, que pode ser suspensão ou *spin* e por fim, se o protocolo permite preempção de seções críticas. Por exemplo, o MPCP permite preempção (somente por tarefas que também irão executar seções críticas) porque cada seção crítica pode ser executada com um *ceiling* diferente. No FMLP e no MSRP seções críticas são executadas de forma totalmente não preemptiva. Neste capítulo serão apresentadas propostas de variações para o MPCP criadas no escopo deste mestrado, sendo a primeira uma versão que executa as seções críticas de forma não preemptiva (no MPCP, seções podem preemptar outras seções críticas) denominada *Multiprocessor Priority Ceiling* Não Preemptivo (MPCPNP). A segunda é uma variação baseada em enfileiramento FIFO no acesso a recursos (no MPCP, o enfileiramento é realizado utilizando a prioridade nominal da tarefa) denominada *Multiprocessor Priority Ceiling* Com Enfileiramento FIFO (MPCPF). Estas variações também são apresentadas na Tabela 4.

4.1 VARIAÇÃO PROPOSTA 1: MPCP NÃO PREEMPTIVO

Esta primeira variação, denominada MPCP Não Preemptivo (MPCPNP) consiste em executar seções críticas de forma não preemptiva, ou seja, de forma semelhante ao MSRP e ao FMLP. Para este protocolo, também foram propostas versões baseadas em *spin* como política de controle de execução. Para a versão baseada em *spin*, este foi adotado como sendo não preemptivo.

Tabela 4: Comparação entre os protocolos incluindo as propostas

Características / Prot.	Controle de execução: Suspensão	Controle de execução: Spin	Ordem de acesso: Ordem de prio.	Ordem de acesso: FIFO	Preemptável
Protocolos existentes					
MPCP:					
MPCP-Susp	x		x		x
MPCP-Spin		x	x		x
MSRP:		x		x	
FMLP:					
short		x		x	
long	x			x	
FMLP+:	x			x	x
Propostos neste trabalho					
MPCPNP:					
MPCPNP-Susp	x		x		
MPCPNP-Spin		x	x		
MPCPF:					
MPCPF-Susp	x			x	x
MPCPF-Spin		x		x	x

Como as seções críticas são não preemptivas, tratar bloqueios com *spin* não preemptivo neste caso apresenta uma solução mais homogênea do ponto de vista do protocolo. Com *spin* não preemptivo, em um determinado instante, poderá haver somente uma tarefa bloqueada em cada processador, de forma semelhante ao FMLP *short*. A seguir, serão apresentadas as regras que as tarefas devem seguir quando da utilização de recursos protegidos por este protocolo.

4.1.1 Regras de Aquisição de Recursos

Para este protocolo, são definidas as seguintes regras para aquisição de recursos:

Requisição de recursos (versão com suspensão): Uma tarefa τ_i executa uma requisição para um recurso global R_j. Se o recurso estiver disponível, a tarefa se torna não preemptável (via prioridade máxima) e entra na seção crítica, caso contrário ela será bloqueada (suspensa). A ordem de atribuição do recurso às tarefas é pela prioridade nominal (acessa antes quem tiver a maior prioridade). A tarefa volta a executar de forma preemptável quando liberar o recurso (volta a executar com a prioridade nominal). Quando uma tarefa libera um recurso, ela deverá desbloquear a tarefa (e inserir-la na fila de aptos com prioridade máxima) que estiver no início da fila, se houver.

Requisição de recursos (versão com *spin*): As regras são as mesmas da versão baseada em suspensão, entretanto, quando uma tarefa bloqueia ela permanece em execução com *spin* não preemptivo (prioridade máxima). A tarefa somente se tornará preemptável novamente no momento em que liberar o recurso o qual utilizava.

A motivação por trás desta alteração consiste na observação de que isto só afeta, no pior caso, bloqueios remotos. Isto se deve ao fato de, toda tarefa de baixa prioridade, independente do *ceiling* do recurso solicitado, poder potencialmente bloquear tarefas de mais alta prioridade no mesmo processador. Isto fica claro com o seguinte exemplo, com 4 tarefas e 3 processadores e 2

recursos, que é ilustrado na Figura 11 (utilizando a versão baseada em suspensão):

$t = 0$: todas as tarefas estão aptas a execução. A tarefa τ_0 inicia sua execução na CPU 0 (mais prioritária), as tarefas τ_2 e τ_3 iniciam suas execuções nas CPUs 1 e 2, respectivamente;

$t = 0,5$: a tarefa τ_2 adquire o recurso representado pela região hachurada xadrez;

$t = 1$: a tarefa τ_0 bloqueia (suspensão) no recurso detido pela tarefa τ_2, permitindo que a tarefa τ_1 execute no mesmo processador. Neste mesmo instante, a tarefa τ_3 adquire o recurso hachurado em preto;

$t = 2$: a tarefa τ_1 bloqueia (suspensão) no recurso detido pela tarefa τ_3;

$t = 3$: a tarefa τ_2 libera o recurso que estava utilizando para a tarefa τ_0, que imediatamente retoma sua execução;

$t = 4$: a tarefa τ_3 libera o recurso que estava utilizando para a tarefa τ_1, mas não pode executar (τ_0 em modo não preemptivo);

$t = 5$: a tarefa τ_0 libera o recurso que estava utilizando. Como a tarefa volta a executar de modo preemptivo, ela sofre uma preempção adicional por parte da tarefa τ_1 quando esta executa sua seção crítica;

$t = 9$: a tarefa τ_1 libera o recurso o qual estava utilizando, sendo preemptada por τ_0;

$t = 10$: τ_0 termina sua execução, permitindo que τ_1 também termine.

No exemplo anterior o protocolo MPCP clássico apresentaria o mesmo resultado (em termos de tempo de resposta), pois ou a tarefa τ_1 preemptaria τ_0 no interior de sua seção crítica (caso o *ceiling* fosse maior), ou imediatamente na saída desta (quando τ_0 voltar a ser preemptável). Em suma, um segmento normal de execução de uma tarefa de mais alta prioridade sempre poderá ser preemptado por seções críticas de qualquer outra tarefa de mais baixa prioridade, não importando o *ceiling* dos recursos desta. Neste caso, executar de

modo não preemptivo pode ser visto como executar todas as seções com a mesma prioridade *ceiling*.

Sob o ponto de vista de um processador específico, no pior caso, a real diferença entre o MPCPNP e o MPCP (ambos baseados em suspensão) é o instante de preempção: No MPCP ela pode ocorrer no interior de uma seção crítica (a seção da tarefas recém desbloqueada possui um *ceiling* mais alto do que a da seção em execução atualmente) ou na saída desta (a seção da tarefas recém desbloqueada possui um *ceiling* mais baixo ou igual, neste caso). No MPCPNP esta preempção somente poderá ocorrer na saída de uma seção crítica (mesmo efeito de *ceilings* iguais do MPCP). Uma forma de transformar o MPCP em MPCPNP é definir os *ceiling* de todos os recursos com o mesmo valor.

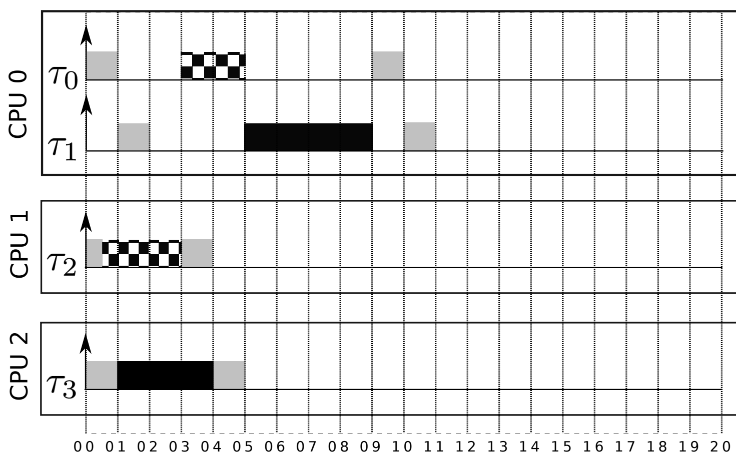


Figura 11: Exemplo de tarefas compartilhando recurso pelo MPCPNP

Já para o caso de bloqueios remotos, no pior caso, o tempo de espera por um recurso será aumentado, pois agora a ordem de execução de seções críticas diferentes em um mesmo processador será FIFO (tarefas mutuamente não preemptivas). Em vez de uma tarefa aguardar apenas a execução da seção critica em si adicionada das seções críticas de maior *ceiling*, ela terá que

aguardar (no pior caso) todas as seções críticas no processador da tarefa que a bloqueia.

Esta variação pode ser vista na Tabela 4 onde é mostrado que ela não equivale a nenhum outro protocolo apresentado anteriormente. A diferença entre a variação e o FMLP/MSRP está justamente na política de enfileiramento e com relação MPCP, a diferença está na não preemptibilidade.

4.1.2 Bloqueios do Protocolo

Para o MPCPNP, existem diversas situações de bloqueios possiveis:

Bloqueio local por tarefas de mais baixa prioridade: todos os protocolos envolvem algum tipo de bloqueio local causado por tarefas de mais baixa prioridade. Este tipo de bloqueio não está relacionado a recursos locais, e sim ao fato das tarefas de prioridade mais baixa acessarem recursos globais. No MPCP não preemptivo, este bloqueio se manifesta de maneira diferente nas diferentes variações (suspensão e *spin*).

Na versão baseada em suspensão, uma tarefa (τ_i) poderá sofrer bloqueios causados por tarefas de mais baixa prioridade acessando seções críticas. Estas tarefas, cujas prioridades são mais baixas, podem ter sido bloqueadas em duas situações distintas: antes da ativação da tarefa em questão, ou quando executaram enquanto esta estava bloqueada, esperando o acesso a algum outro recurso. Quando uma tarefa é desbloqueada em uma seção crítica, ela sempre poderá preemptar tarefas de mais alta prioridade que não estejam executando dentro de seções críticas.

Na versão baseada em *spin* (*spin* não preemptivo), quando uma tarefa de prioridade mais alta (τ_i por exemplo) inicia sua execução, ela poderá ser bloqueada (bloqueio manifestado em forma de *release jitter*) por uma tarefa de mais baixa prioridade que está executando dentro de uma seção crítica (em modo não preemptivo) ou aguardando em *spin* por alguma seção crítica. No pior caso, a tarefa em questão (τ_i) será ativada um instante infinitesimal após uma tarefa de mais baixa prioridade iniciar o *spin* (a tarefa com a maior soma de tempo de *spin* com tempo de seção

crítica). Ou seja, no pior caso este bloqueio será restrito a apenas uma tarefa de mais baixa prioridade, pois quando esta terminar o acesso à sua seção crítica, a tarefa em questão (τ_i) imediatamente iniciará sua execução. Se as tarefas de mais baixa prioridade não acessarem recursos, este bloqueio não existirá para tarefas com prioridades imediatamente mais altas.

Bloqueio remoto: sempre que uma tarefa solicitar um recurso, ela possivelmente será bloqueada. Este bloqueio ocorre pois, no pior caso, todas as tarefas desejam acessar o recurso em um determinado momento. Este bloqueio sofre influência da política de enfileiramento, que é por prioridades. Este bloqueio também sofre influência direta da não preemptividade, como será mostrado mais adiante na análise de escalonabilidade. Se uma tarefa não acessar recursos, ela não sofrerá bloqueios remotos.

Execução *back-to-back*: uma tarefa poderá sofrer interferência adicional por conta de tarefas de mais alta prioridade que se auto suspendem (recurso ocupado), para retomar a execução em um momento posterior (quando recurso estiver disponível). Os testes de tempo de resposta normais somente capturam interferência de tarefas que não se auto suspendem. Este tipo de bloqueio somente ocorre com a versão baseada em suspensão. Se uma tarefa de mais alta prioridade não acessar recursos, então este fator não existe (em relação a esta tarefa) para as tarefas de mais baixa prioridade.

Em resumo, bloqueios remotos serão uma consequência dos recursos acessados pela própria tarefa, enquanto bloqueios locais e execução *back-to-back* são consequência de recursos acessados por tarefas de mais baixa e mais alta prioridade, respectivamente.

4.1.3 Análise de Escalonabilidade

Utilizando a abordagem de análise de escalonabilidade do capítulo anterior, pode-se calcular o tempo de resposta das tarefas utilizando o protocolo

MPCPNP. Primeiramente, deve-se definir os parâmetros das equações dependentes do protocolo. A notação utilizada será a mesma apresentada no capítulo anterior.

Para calcular o pior caso no tempo de resposta de uma seção crítica em particular, têm-se duas abordagens, uma para a versão baseada em suspensão e outra para a baseada em *spin*:

Baseada em suspensão: O pior caso no tempo de resposta de uma seção crítica $C'_{i,k}$ da versão baseada em suspensão é dado por:

$$W'_{i,k} = C'_{i,k} + \sum_{\tau_u \in P(\tau_i)} \max_{1 \leq v \leq s(u)} C'_{u,v} \tag{4.1}$$

Como o protocolo é não preemptivo, isto equivale a dizer que todas as seções críticas são executadas com o mesmo nível de prioridade (prioridade máxima neste caso). O pior caso de uma seção crítica de uma tarefa ocorre quando todas as outras tarefas também estão aptas a executar suas seções críticas, sendo a tarefa em questão a última a ser desbloqueada (todas com a mesma prioridade implica ordenamento FIFO de acesso ao processador).

Baseada em *spin*: Para a versão baseada em *spin*, o pior tempo de resposta de uma seção crítica $C'_{i,k}$ deve ser calculado de forma diferente:

$$W'_{i,k} = C'_{i,k} \tag{4.2}$$

Na Equação 4.2, o pior caso de uma seção crítica é o próprio tempo de execução desta, de forma idêntica ao FMLP *short*. Isto ocorre pelo fato da versão baseada em *spin* utilizar *spin* não preemptivo como mecanismo de controle de execução. Desta maneira, somente uma tarefa pode estar bloqueada em algum recurso em um determinado instante, logo, quando desbloqueada a tarefa não precisará competir com nenhuma outra na utilização do processador em sua seção crítica.

Para o cálculo do tempo de bloqueio ($B^r_{i,j}$) para cada seção crítica $C^{'}_{i,j}$, a seguinte convergência deve ser utilizada para ambas as versões (baseadas em suspensão ou *spin*), com valor inicial $B^{r,0}_{i,j} = \max_{l>i\&(\tau^{'}_{l,u})\in R(\tau_i,j)} (C^{'}_{l,u})$ e sendo $R(\tau_{i,j})$ o conjunto de segmentos críticos (seções críticas) referentes ao mesmo recurso utilizado no segmento $\tau_{i,j}$:

$$B^{r,n+1}_{i,j} = \max_{l>i\&(\tau^{'}_{l,u})\in R(\tau_{i,j})} (W^{'}_{l,u}) + \sum_{h<i\&(\tau^{'}_{h,v}\in R(\tau_{i,j}))} \left(\left\lceil \frac{B^{r,n}_{i,j}}{T_h} \right\rceil + 1 \right) (W^{'}_{h,v}) \tag{4.3}$$

Na Equação 4.3, no pior caso, da mesma maneira que no MPCP, a tarefa τ_i pode ser bloqueada por somente uma tarefa de mais baixa prioridade τ_l (primeiro termo da equação), que tem a maior seção crítica relativa ao recurso $R(\tau_{i,j})$, entre todas as tarefas de mais baixa prioridade, e que já estava executando no interior da seção crítica quando a tarefa τ_i efetuou a tentativa de acesso a este recurso. Entretanto, a tarefa τ_i pode ser bloqueada por muitas tarefas de mais alta prioridade que acessam $R(\tau_{i,j})$, varias vezes cada uma (segundo termo da equação). Isto ocorre devido ao fato das filas de acesso aos recursos serem ordenadas pelas prioridades das tarefas, como no MPCP. Vale ressaltar que, na Equação 4.3, deve-se utilizar os valores de $W^{'}_{i,k}$ adequados a versão do protocolo.

Para o cálculo do tempo de resposta total das tarefas, deve-se utilizar a abordagem adequada a cada versão do protocolo:

Para a versão baseada em suspensão: para o cálculo do tempo de resposta utilizando a versão baseada em suspensão, deve-se utilizar a Equação de interferência 3.11 e a de bloqueio local 3.12, ambas do Capítulo 3, utilizando os tempos de bloqueio representados pela convergência da Equação 4.3. A equação final fica:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n + B_h^r}{T_h} \right\rceil C_h \tag{4.4}$$
$$+ s(i) \times \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C_{l,k}^{'}$$

Para a versão baseada em *spin*: para o cálculo do tempo de resposta utilizando a versão baseada em *spin*, deve-se utilizar a Equação de interferência 3.13 e a de bloqueio local para protocolos baseados em *spin* não preemptivo 3.15 (ambas do Capítulo 3) utilizando também os tempos de bloqueio apresentados anteriormente, ou seja, a convergência da equação 4.3, considerando *spin*. A equação final resulta em:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n}{T_h} \right\rceil (C_h + B_h^r) \tag{4.5}$$
$$+ \max_{l>i \& \tau_l \in P(\tau_i) \& 1 \leq k < s(l)} (C_{l,k}^{'} + B_{l,k}^r)$$

4.1.4 Exemplo

O exemplo numérico apresentado para os protocolos MPCP e FMLP no capítulo anterior pode ser utilizado como exemplo para o protocolo MPCPNP. Abaixo estão as configurações do conjunto de tarefas, repetido aqui por questões de comodidade.

$\tau_0 : \ C_{0,1} = 1, C_{0,1}^{'} = 2, C_{0,2} = 1$

$R_{0,1} = S_0$

$T_0 = 50$

$P(0) = P_1$

$\tau_1 : \ C_{1,1} = 1, C_{1,1}^{'} = 1, C_{1,2} = 2$

$R_{1,1} = S_2$

$T_1 = 85$

$$P(1) = P_1$$

$$\tau_2 : \; C_{2,1} = 2, C'_{2,1} = 1, C_{2,2} = 2$$
$$R_{2,1} = S_2$$
$$T_2 = 105$$
$$P(2) = P_1$$

$$\tau_3 : \; C_{3,1} = 1, C'_{3,1} = 1, C_{3,2} = 1, C'_{3,2} = 1, C_{3,3} = 1$$
$$R_{3,1} = S_0, R_{3,2} = S_1$$
$$T_3 = 45$$
$$P(3) = P_2$$

$$\tau_4 : \; C_{4,1} = 1$$
$$T_4 = 70$$
$$P(4) = P_2$$

$$\tau_5 : \; C_{5,1} = 1, C'_{5,1} = 2, C_{5,2} = 1, C'_{5,2} = 1, C_{5,3} = 1$$
$$R_{5,1} = S_3, R_{5,2} = S_0$$
$$T_5 = 85$$
$$P(5) = P_2$$

$$\tau_6 : \; C_{6,1} = 1, C'_{6,1} = 2, C_{6,2} = 1$$
$$R_{6,1} = S_3$$
$$T_6 = 135$$
$$P(6) = P_2$$

$$\tau_7 : \; C_{7,1} = 2, C'_{7,1} = 2, C_{7,2} = 2$$
$$R_{7,1} = S_1$$
$$T_7 = 75$$
$$P(7) = P_3$$

$$\tau_8 : \; C_{8,1} = 2, C'_{8,1} = 3, C_{8,2} = 2$$
$$R_{8,1} = S_1$$
$$T_8 = 100$$
$$P(8) = P_3$$

A Tabela 5 apresenta os piores casos nos tempos de resposta obtidos para o conjunto de tarefas apresentado, utilizando as equações anteriores. É possível notar que não existe dominância da versão com suspensão sobre a versão com *spin* ou vice-versa, no que diz respeito ao tempo de resposta no pior caso.

Tabela 5: Piores casos no tempo de resposta (exemplo com MPCPNP)

Tarefa	W_i **com Suspensão**	W_i **com *Spin***
τ_0	22	8
τ_1	10	12
τ_2	13	16
τ_3	26	15
τ_4	6	16
τ_5	26	23
τ_6	16	25
τ_7	22	13
τ_8	23	17

4.2 VARIAÇÃO PROPOSTA 2: MPCP COM ENFILEIRAMENTO FIFO

A segunda variação (MPCPF) apresentada nesta dissertação altera a ordem com que as tarefas bloqueadas acessam um determinado recurso. Esta variação é mais próxima do MPCP original que a anterior. No MPCP original, as tarefas acessam recursos por ordem de prioridade, sendo que uma tarefa bloqueada em algum recurso poderá esperar por várias tarefas de prioridade mais alta e apenas uma de prioridade mais baixa, sendo que cada tarefa de mais alta prioridade poderá acessar a seção crítica mais de uma vez antes da tarefa em questão acessar. Nesta segunda variação, as tarefas bloqueadas acessam os recursos em ordem FIFO. Isto não quer dizer que esta variação se iguala

ao FMLP para prioridade fixa, pois no último, as tarefas além de acessarem os recursos em ordem FIFO, o fazem de forma não preemptiva. Esta variação também é diferente do FMLP+, pois este utiliza uma regra de desempate diferente. No FMLP+ o desempate entre seções críticas ativas em um processador (tarefas desbloqueadas em suas respectivas seções críticas) ocorre com o uso de *timestamps* do instante de solicitação dos recursos (o primeiro a solicitar é o primeiro a executar). No MPCPF o desempate é efetuado da mesma maneira que no MPCP, ou seja, pelo *ceiling* do recurso associado à seção crítica. A comparação desta variação com os outros protocolos também pode ser vista na Tabela 4.

Para o MPCPF, também são propostas versões baseadas em suspensão e *spin*. Para a versão baseada em *spin*, o mesmo foi adotado como sendo preemptivo. Se o *spin* fosse não preemptivo, esta variação (com *spin*) seria equivalente ao FMLP *short*, apresentando os mesmo tempos de bloqueio, o que não justificaria sua proposta. Se o protocolo utilizasse *spin* não preemptivo, então seria possível haver somente uma tarefa bloqueada por processador em um dado instante, logo uma tarefa desbloqueada nunca seria preemptada por outra que também foi desbloqueada (pois não existiria uma segunda tarefa sendo desbloqueada), exatamente como ocorre no FMLP. Antes de apresentar as regras de aquisição de recursos, deve-se definir as regras para o cálculo do *ceiling* dos recursos.

4.2.1 Cálculo dos *Ceilings* dos Recursos

O protocolo possui a mesma definição de *ceiling* do MPCP original. Seja P_H a mais alta das prioridades entre todas as prioridades das tarefas do sistema. O *ceiling* de recursos globais é definido por uma composição de dois *ceilings*:

1. P_G tal que $P_G > P_H$;

2. PS_{G_i} que é definida para cada processador P_i como sendo a mais alta das prioridades entre as tarefas que acessam o recurso R_i e estão alocadas a processadores diferentes de P_i.

O *ceiling* final será $P_G + PS_{G_i}$, sendo a segunda componente variável para cada processador. Quando uma tarefa acessa um recurso R_i, deve fazê-lo com sua prioridade ajustada a $P_G + PS_{G_i}$.

4.2.2 Regras de Aquisição de Recursos

As seguintes regras devem ser utilizadas na aquisição de recursos:

Requisição de recursos (versão com suspensão): uma tarefa τ_i executa uma requisição para um recurso global R_j. Se o recurso estiver disponível a tarefa toma posse deste e têm sua prioridade ajustada ao *ceiling* deste recurso, caso contrário ela será bloqueada (suspensa). A ordem de atribuição do recurso às tarefas é em ordem FIFO. A tarefa volta a executar com sua prioridade nominal no momento que liberar o recurso. Quando uma tarefa libera um recurso, ela deverá desbloquear a tarefa que estiver no início da fila de acesso, se houver. A prioridade de uma tarefa desbloqueada deverá ser ajustada ao *ceiling* para que esta obtenha o processador imediatamente, se possível.

Requisição de recursos (versão com *spin*): a versão baseada em *spin* segue as mesmas regras da versão baseada em suspensão. A diferença é que quando uma tarefa bloqueia ela permanece em *spin* preemptivo, permitindo que tarefas de mais alta prioridade executem. A prioridade da tarefa também será ajustada ao *ceiling* quando o recurso for concedido. A tarefa também voltará a executar com sua prioridade nominal no momento em que liberar o recurso (e conceder este a outra tarefa que, porventura, esteja esperando).

4.2.3 Bloqueios do Protocolo

Este protocolo possui os mesmos tipos de bloqueios do MPCPNP, mas a forma como estes ocorrem é diferente:

Bloqueio local por tarefas de mais baixa prioridade: para o caso da versão baseada em suspensão, este bloqueio será exatamente igual ao do MPCP

e do MPCP Não Preemptivo, pois envolve auto-suspensão. Para o caso da versão baseada em *spin*, como este foi adotado como sendo preemptivo, tarefas podem sofrer bloqueios por parte de outras tarefas de mais baixa prioridade que tenham bloqueado em recursos antes da ativação desta (igual ao MPCP).

Bloqueio remoto: o bloqueio remoto poderá ocorrer sempre que uma tarefa tentar acessar um recurso que não estiver disponível. Como o enfileiramento é FIFO, no pior caso, a tarefa em questão estará no final da fila de acesso, estando todos as outras tarefas aptas a utilizar o recurso. Diferentemente do MPCP, uma tarefa poderá ser bloqueada somente uma vez por cada tarefa que também acessa o mesmo recurso, independentemente das prioridades.

Execução *back-to-back*: neste protocolo, uma tarefa poderá sofrer interferência adicional por conta de tarefas de mais alta prioridade que se auto suspendem (recurso ocupado), para retomar a execução em um momento posterior (quando recurso estiver disponível). Este tipo de bloqueio ocorre também somente com a versão baseada em suspensão, pois na versão baseada em *spin* não ocorre auto-suspensão de tarefas.

Da mesma forma que no MPCPNP, bloqueios remotos serão uma consequência dos recursos acessados pela própria tarefa, enquanto bloqueios locais e execução *back-to-back* são consequência de recursos acessados por tarefas de mais baixa e mais alta prioridades, respectivamente.

4.2.4 Análise de Escalonabilidade

Utilizando a abordagem de análise de escalonabilidade do capítulo anterior, para este protocolo, também pode-se calcular o tempo de resposta das tarefas. Primeiramente, deve-se definir os parâmetros das equações específicos para este protocolo.

O primeiro parâmetro a ser definido é o pior caso no tempo de resposta de um seção crítica $C^{'}_{i,k}$, que é $W^{'}_{i,k}$. Para ambos as versões (suspensão e *spin*),

a equação é a mesma, onde $gc(i,k)$ é o *ceiling* da k-ésima seção crítica da tarefa τ_i:

$$W'_{i,k} = C'_{i,k} + \sum_{\tau_u \in P(\tau_i)} \max_{1 \leq v \leq s(u) \& gc(u,v) > gc(i,k)} C'_{u,v} \tag{4.6}$$

No pior caso (Equação 4.6), quando uma tarefa estiver pronta para executar uma seção crítica (recebeu o acesso a algum recurso), ela deverá esperar a maior seção crítica de cada tarefa que possua *ceiling* igual ou superior ao *ceiling* do recurso em questão, de forma idêntica ao MPCP.

Por fim, $B^r_{i,j}$ representa o bloqueio remoto para uma tarefa τ_i na aquisição da seção crítica global $C'_{i,j}$. Como o enfileiramento é FIFO, no pior caso, a tarefa terá que esperar todas as outras tarefas que acessam o recurso (somatório dos piores tempos de resposta de seções críticas), e é dado por:

$$B^r_{i,j} = \sum_{h \neq i \& (\tau'_{h,v} \in R(\tau_{i,j}))} W'_{h,v} \tag{4.7}$$

A Equação 4.7 também vale para ambas as versões (baseadas em suspensão e *spin*).

Para o cálculo do tempo de resposta total das tarefas, deve-se utilizar a abordagem adequada a cada versão do protocolo:

Para a versão baseada em suspensão: da mesma forma que na proposta anterior, para o cálculo do tempo de resposta utilizando a versão baseada em suspensão, deve-se utilizar a Equação de interferência 3.11 e a de bloqueio local 3.12, ambas do Capítulo 3, utilizando os tempos de bloqueio apresentados anteriormente através da Equação 4.7. A equação final fica:

$$W_i^{n+1} = C_i + B^r_i + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n + B^r_h}{T_h} \right\rceil C_h + s(i) \times \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \leq k < s(l)} C'_{l,k} \tag{4.8}$$

Para a versão baseada em *spin*: para o cálculo do tempo de resposta utilizando a versão baseada em *spin*, deve-se utilizar a Equação de interferência 3.13 e a de bloqueio local para protocolos baseados em *spin* preemptivo 3.14 utilizando também os tempos de bloqueio calculados pela convergência da Equação 4.7). A equação final resulta em:

$$W_i^{n+1} = C_i + B_i^r + \sum_{h<i \& \tau_h \in P(\tau_i)} \left\lceil \frac{W_i^n}{T_h} \right\rceil (C_h + B_h^r) + \sum_{l>i \& \tau_l \in P(\tau_i)} \max_{1 \le k < s(l)} C_{l,k}^{'} \quad (4.9)$$

4.2.5 Exemplo

Utilizando o mesmo exemplo numérico apresentado para os outros protocolos, têm-se os piores casos nos tempo de resposta mostrados na Tabela 6 para as versões baseadas em suspensão e *spin* do protocolo MPCPF. Novamente, as versões baseadas em suspensão e *spin* podem favorecer e prejudicar diferentes tarefas.

Tabela 6: Piores casos no tempo de resposta (exemplo com MPCPF)

Tarefa	W_i **com Suspensão**	W_i **com *Spin***
τ_0	17	15
τ_1	10	18
τ_2	13	22
τ_3	31	23
τ_4	6	24
τ_5	22	32
τ_6	16	34
τ_7	17	14
τ_8	18	23

4.3 CONCLUSÕES

Este capítulo apresentou duas propostas de variação para o *Multiprocessor Priority Ceiling*, no contexto de sistemas com escalonamento particionado e prioridade fixa. A primeira variação consiste em executar as seções críticas de forma não preemptiva, com versões baseadas em suspensão e *spin*. Para a versão com *spin*, o mesmo foi adotado como sendo não preemptivo, pois a seções críticas também são não preemptivas. A segunda variação consiste em alterar a ordem com que as tarefas acessam recursos. No MPCP original, as tarefas acessam recursos por ordem de prioridade (nominal), sendo que na variação proposta esta ordem é FIFO. Para esta variação também foram propostas variações baseadas em suspensão e *spin*. Para a versão baseada em *spin*, este foi adotado como preemptivo. Se o *spin* fosse não preemptivo nesta variação, o protocolo seria de fato o FMLP *short*, não havendo possibilidade de tarefas serem preemptadas por seções críticas de mais alta prioridade. Isto ocorre pois, com *spin* não preemptivo, só pode haver uma tarefa bloqueada em um processador em um determinado instante. Questões com aninhamento não foram consideradas em relação as propostas, entretanto, é perfeitamente possível aplicar técnicas como *group locking* (BLOCK et al., 2007) aos protocolos aqui apresentados. Como este trabalho se foca em compartilhamento de recursos globais, não foi dado tratamento aos locais. Para recursos locais, a solução é a mesma utilizada no MPCP, que é a redução do problema para o domínio dos sistemas monoprocessados, com a utilização do PCP.

5 AVALIAÇÃO EMPÍRICA DA PROPOSTA

O capítulo anterior apresentou as propostas de variações para o protocolo *Multiprocessor Priority Ceiling Protocol* baseadas em não preemptividade e enfileiramento FIFO. Este capítulo tem como objetivo apresentar uma comparação de tais propostas com as já existentes, que são o MPCP clássico e o FMLP (*Flexible Multiprocessor Locking Protocol*). Os primeiros experimentos visam comparar a escalonabilidade de conjuntos de tarefas gerados de forma sintética perante cada um dos protocolos (considerando as versões baseadas em suspensão e *spin*). A seguir, será apresentado um estudo experimental comparando o *overhead* de uma das propostas com o FMLP em uma plataforma baseada em Linux/PREEMPT-RT(MOLNAR; GLEIXNER, 2005). Para este último experimento, tais protocolos foram implementados no *kernel* do sistema operacional de forma a serem visíveis para *drivers* e módulos carregáveis (pedaços do *kernel* que podem ser carregados e descarregados dinamicamente) em geral. É importante salientar que a implementação serve também como prova de conceito para a proposta. Como o objetivo deste trabalho não é comparar políticas de controle de execução (existem muitos trabalhos que fazem este tipo de comparação na literatura, como por exemplo (BRANDENBURG et al., 2008), (LAKSHMANAN et al., 2009) e (BRANDENBURG; ANDERSON, 2008a)), os resultados deste capítulo serão apresentados separadamente para protocolos baseados em suspensão e *spin*.

5.1 COMPARAÇÃO EMPÍRICA DA ESCALONABILIDADE

Para a comparação da escalonabilidade dos protocolos, as equações das análises de escalonabilidade foram todas implementadas em um *framework* único. Desta forma, o mesmo conjunto de tarefas pode ser submetido à análise perante cada protocolo. Por questões de nomenclatura, será utilizada a seguinte convenção ao longo dos experimentos empíricos:

- PLAIN: denota a simples análise do tempo de resposta, desconside-

rando bloqueios, será utilizada apenas de forma comparativa. Esta análise é apresentada no capítulo 2, na Equação 2.6;

- MPCP_SUSP: representa a versão baseada em suspensão do MPCP. Para este protocolo, o tempo de resposta é obtido pela Equação 3.19 do Capítulo 3;

- MPCP_SPIN: representa a versão baseada em *spin* do MPCP. Para este protocolo, o tempo de resposta é obtido pela Equação 3.20 do Capítulo 3;

- MPCPNP_SUSP: representa a versão baseada em suspensão do MPCP não preemptivo. Para este protocolo, o tempo de resposta é obtido pela Equação 4.4 do Capítulo 4;

- MPCPNP_SPIN: representa a versão baseada em *spin* do MPCP não preemptivo. Para este protocolo, o tempo de resposta é obtido pela Equação 4.5 do Capítulo 4;

- MPCPF_SUSP: representa a versão baseada em suspensão do MPCP com enfileiramento FIFO. Para este protocolo, o tempo de resposta é obtido pela Equação 4.8 do Capítulo 4;

- MPCPF_SPIN: representa a versão baseada em *spin* do MPCP com enfileiramento FIFO. Para este protocolo, o tempo de resposta é obtido pela Equação 4.9 do Capítulo 4;

- FMLP_LONG: representa FMLP *long* (baseado em suspensão). Para este protocolo, o tempo de resposta é obtido pela Equação 3.26 do Capítulo 3;

- FMLP_SHORT: representa FMLP *short* (baseado em *spin*). Para este protocolo, o tempo de resposta é obtido pela Equação 3.27 do Capítulo 3.

5.1.1 Condições dos Experimentos

Para a realização dos testes, foram gerados conjuntos de tarefas segundo um critério específico (que será apresentado mais adiante). Com o conjunto de tarefas, efetua-se o particionamento (através de um algoritmo de alocação) levando em consideração as análises apresentadas para cada protocolo. O parâmetro de comparação utilizado, assim como em (LAKSHMANAN et al., 2009), é o número de processadores necessários para comportar um determinado sistema. Para estes experimentos, será considerado *overhead* zero. Muitos estudos consideram *overhead* zero em experimentos (BLOCK et al., 2007) (LAKSHMANAN et al., 2009), ou seja, esta é uma prática usual. Também é comum considerar o *overhead* como parte do tempo de computação, ou incorporado nas equações de análise de escalonabilidade, partindo de observações experimentais, como tempos de chaveamentos de contexto e manipulação de filas, por exemplo.

Todos os programas referentes ao experimento foram implementados em linguagem C, sem a utilização de bibliotecas adicionais. O programa gerador de conjuntos de tarefas tem como saída arquivos de texto com a descrição das mesmas. Tais arquivos servem de entrada para o programa que implementa os experimentos propriamente ditos.

5.1.2 Algoritmo de Alocação

A distribuição de tarefas em sistemas multiprocessados com escalonamento particionado é um problema equivalente ao *bin-packing*, que é NP-*hard*. Tal problema deve ser atacado com heurísticas sub-ótimas, mesmo estas estando sujeitas a anomalias de escalonamento. Apesar de terem sido apresentados algoritmos de particionamento cientes da utilização de recursos no capítulo anterior, estes não foram utilizados no contexto destes experimentos. Tais algoritmos tem por objetivo primário reduzir o número de seções críticas globais agrupando tarefas que utilizam os mesmos recursos em um mesmo processador. Como este trabalho foca basicamente recursos globais, é interessante que o algoritmo utilizado mantenha os recursos sempre globais para

uma melhor comparação dos protocolos. Outro impedimento para utilização destes algoritmos é que eles podem ser guiados por características exclusivas de um determinado protocolo, produzindo resultados pobres para outros.

No contexto deste trabalho, para efetuar a alocação das tarefas nos processadores, foi utilizada uma variação do algoritmo RM-FFDU. A função de *fit* (verificar se a inserção de uma tarefa mantém o sistema escalonável) utiliza os testes de escalonabilidade dos capítulos anteriores e indicados na seção 5.1 deste capítulo. Esta variação do RM-FFDU considera somente a dependência entre tarefas, não a utilização de recursos.

Em sua maneira mais simples, o algoritmo *First-fit* parte de um conjunto não alocado de tarefas e um conjunto de processadores. Isto não vale para tarefas com dependências (como exclusão mútua por exemplo), pois se o algoritmo decidir colocar as tarefas “A” e “B” no mesmo processador em um primeiro momento (o sistema é escalonável), isto poderá fazer com que o sistema nunca atinja um particionamento escalonável, pois pode haver uma tarefa “C” que, independente do processador colocado, faça com que “B” perca o *deadline* por compartilhar recursos com esta (indução de bloqueio). Para o exemplo anterior, é necessário que “A” e “B” não fiquem no mesmo processador, e isto só é possível se o algoritmo de particionamento tiver uma visão global do sistema. De forma resumida, quando tarefas não são independentes (pode ocorrer em tarefas independentes também, devido a anomalias), qualquer alocação de uma tarefa anterior poderá influenciar a alocação de tarefas futuras, e mais, qualquer decisão que não considere o sistema de um ponto de vista global, poderá ser otimista, ou seja, uma alocação viável agora poderá tornar o sistema não escalonável no futuro. Isto vale também para os outros algoritmos de alocação, como BF (*Best-fit*) e WF (*Worst-fit*) por exemplo.

Para contornar este problema, o algoritmo utilizado parte do pior caso possível, que é uma tarefa por processador (contando que com uma tarefa por processador o sistema seja escalonável) com as tarefas/processadores ordenados por ordem não crescente de utilização. Partindo deste ponto, o algoritmo tenta mover a tarefa do processador 1 para o processador 0, depois a tarefa do processador 2 para o processador 0 ou 1 (via *first-fit*) e assim sucessivamente. Após este passo, o algoritmo deve remover os processadores vazios. A entrada

do algoritmo é um conjunto de tarefas $\mathcal{T}$, não alocadas que para efeito de implementação pode ser visto como um vetor. A saída do algoritmo é um super conjunto $\mathcal{T}_\pi$ composto por conjuntos de tarefas alocadas a cada processador. O pseudo-código é apresentado no algoritmo 1.

No algoritmo 1 tarefas são realocadas somente quando o sistema permanecer escalonável. O algoritmo termina quando todas as tarefas sofrerem tentativas de realocação.

5.1.3 Geração dos Conjuntos de Tarefas

As experiências foram efetuadas desconsiderando efeitos de *overhead*. Os conjuntos de tarefas foram gerados de acordo com o artigo (LAKSHMANAN et al., 2009). Segundo o artigo, a geração dos conjuntos é sempre segmentada em subconjuntos com utilização 1. Isto significa dizer que, se a utilização do conjunto a ser gerado é igual a 8, então são gerados 8 subconjuntos com utilização 1, os quais são unidos ao final do processo. Todos os parâmetros específicos de cada teste foram previamente descritos em (LAKSHMANAN et al., 2009). Nesta dissertação, foi utilizado o UUniFast(BINI; BUTTAZZO, 2005) para geração das utilizações, pois é provado que ele gera uma distribuição de utilizações imparciais para estudos empíricos de testes de escalonabilidade. Os períodos das tarefas foram gerados no intervalo [10ms, 100ms] uniformemente. Os tempos de computação foram calculados com base nos valores gerados anteriormente. Os conjuntos gerados servem de entrada para o algoritmo de particionamento.

5.1.4 Realização dos Experimentos

Para cada experimento, 30 conjuntos de tarefas para cada configuração foram submetidos ao particionamento utilizando os testes de escalonabilidade de cada um dos protocolos. Cada configuração é apresentada no eixo x do gráfico de cada experimento. Como não é o objetivo deste trabalho comparar políticas de controle de execução e sim novas propostas, os resultados serão apresentados separadamente para os protocolos baseados em suspensão e *spin*.

Algoritmo 1 Algoritmo de Alocação; Entrada: $\mathcal{T}$; Saída: $\mathcal{T}_\pi$

$m \leftarrow 0$
{ordena as tarefas em ordem não crescente de utilização}
$sort_du(\mathcal{T})$
{insere cada tarefa em um processador}
for all $\tau \in \mathcal{T}$ **do**
  $insert \quad \tau \quad in \quad \mathcal{T}_{\pi_m}$
  $m \leftarrow m+1$
end for
{para cada tarefa inserida}
for $i \leftarrow 1, i < n_tasks(\mathcal{T}), i \leftarrow i+1$ **do**
  $\tau \leftarrow \mathcal{T}(i)$
  $p \leftarrow P(\tau)$
  $remove \quad \tau \quad from \quad \mathcal{T}_{\pi_p}$
  $allocated \leftarrow FALSE$
  {tenta mover a tarefa para um}
  {processador de menor índice (via first-fit)}
  for $m \leftarrow 0, m < i, m \leftarrow m+1$ **do**
    if $u(\tau) + u(\mathcal{T}_{\pi_m}) < 1$ **then**
      $insert \quad \tau \quad in \quad \mathcal{T}_{\pi_m}$
      if $schedulable(\mathcal{T}_\pi)$ **then**
        $allocated \leftarrow TRUE$
        $exit \quad for$
      else
        $remove \quad \tau \quad from \quad \mathcal{T}_{\pi_m}$
      end if
    end if
  end for
  {se a realocação não é possível}
  {devolve a tarefa para o seu processador}
  {de origem}
  if $not \quad allocated$ **then**
    $insert \quad \tau \quad in \quad \mathcal{T}_{\pi_p}$
  end if
end for
{o último passo é remover os processadores}
{vazios (conjuntos de tarefas vazios)}
$remove \quad all \quad empty \quad \mathcal{T}_{\pi_m} \quad in \quad \mathcal{T}_\pi$

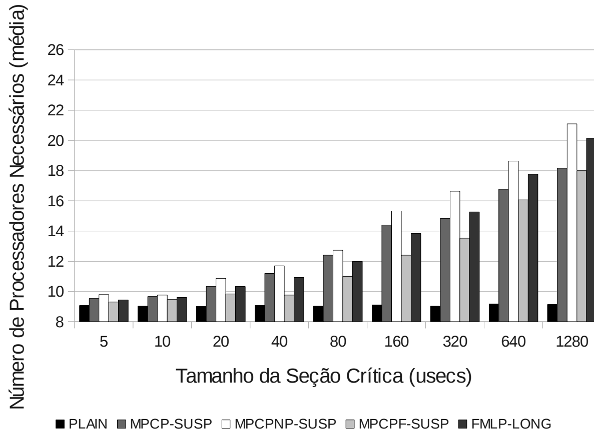
De fato, a comparação de políticas de controle de execução já foi estudada em trabalhos como (BRANDENBURG et al., 2008) e (LAKSHMANAN et al., 2009). Embora o resultado apresentado seja o número de processadores necessários para escalonar um sistema, este é um parâmetro indicativo da eficiência de escalonamento de cada protocolo. Os resultados numéricos referentes aos gráficos dos experimentos podem ser vistos no Apêndice A.

5.1.4.1 Experimento 1: Variação do Tamanho das Seções Críticas

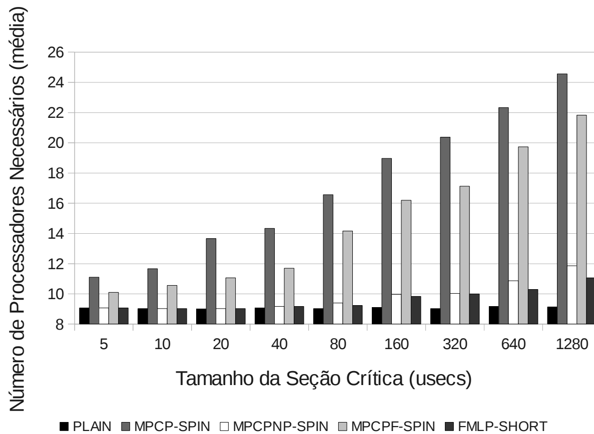
O primeiro experimento visa avaliar o comportamento dos protocolos para diferentes tamanhos de seção crítica. Todos os conjuntos gerados são compostos de 40 tarefas, com utilização total igual a 8. Nos sistemas gerados, cada tarefa acessa duas seções críticas globais, sendo que cada recurso é compartilhado por duas tarefas. As seções variaram no intervalo [5, 1280]μs, dobrando o tamanho a cada teste. Os resultados deste experimento (média de processadores necessários) são apresentados na Figura 12. Os desvios-padrões relativos do experimento são apresentados na Figura 13.

Para este experimento, as versões dos protocolos baseadas em suspensão (ver Figura 12(a)) e em *spin* (ver Figura 12(b)) apresentaram resultados muito diferentes. Para as versões baseadas em suspensão. os melhores resultados foram obtidos para o MPCPF (melhor). Em segundo lugar, estão o FMLP (melhor no intervalo de [5, 160]) e o MPCP (melhor no intervalo [320, 1280]). Para as versões baseadas em *spin*, os melhores resultados foram obtidos para o FMLP (melhor) e para o MPCPNP (segundo melhor). Isto mostra que um mesmo protocolo pode apresentar comportamento bem distinto dependendo da política de controle de execução em caso de bloqueio. Nota-se que estes experimentos refletem o comportamento de pior caso, que depende de pessimismos diferentes para cada protocolo. Em relação aos desvios-padrões relativos (Figura 13), pode-se ver que estes foram em geral pequenos, sendo no máximo 15% para um protocolo baseado em *spin*, como pode ser visto na Figura 13(b). Para o caso dos protocolos baseados em suspensão, este desvio não passou de 14%, conforme pode ser observado na Figura 13(a).

De fato, os desvios-padrões podem ter diversas origens, como particu-



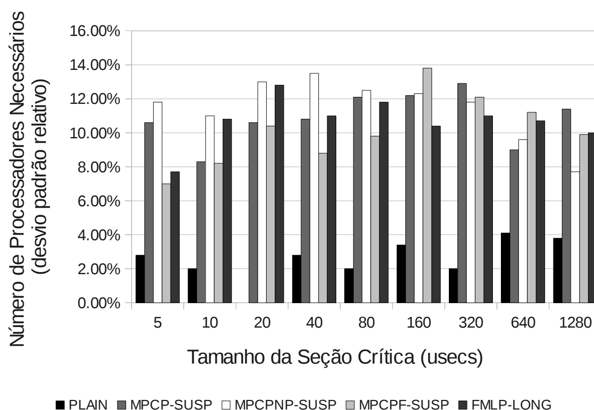
(a) Utilizando suspensão



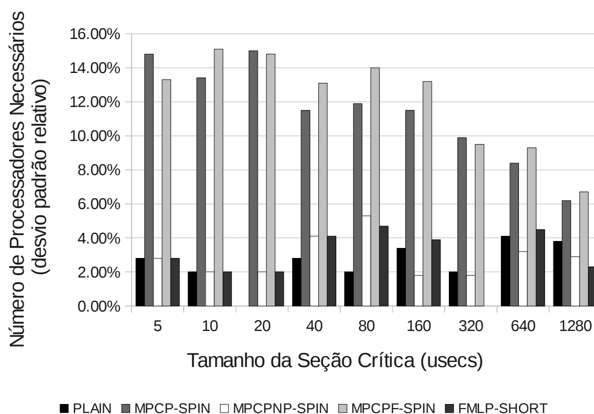
(b) Utilizando *spin*

Figura 12: Variação do tamanho das seções críticas

laridades do próprio protocolo ou até mesmo nos conjuntos de tarefas. Para entender a influência do conjunto de tarefas, considera-se o primeiro cenário do experimento (seções críticas de $5\mu s$) e o cenário mais extremo ($1280\mu s$). Para cada um destes cenários, foram plotados os resultados (número do conjunto de tarefas pelo número de processadores necessários utilizando cada pro-



(a) Utilizando suspensão

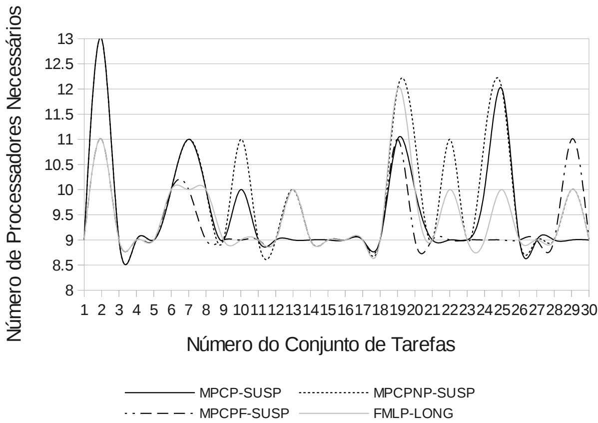


(b) Utilizando *spin*

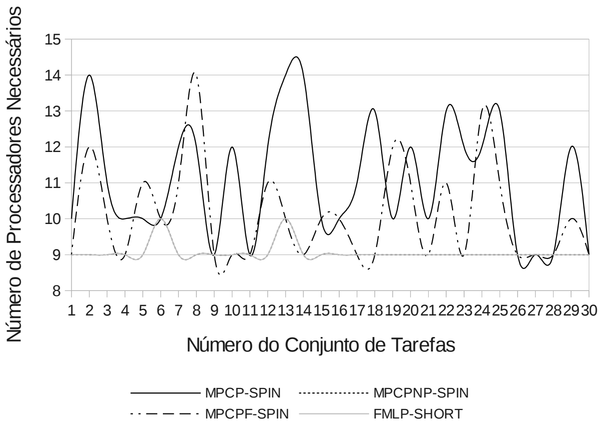
Figura 13: Variação do tamanho das seções críticas (desvio padrão relativo)

tocolo), de forma a se observar o comportamento real do experimento. Para o primeiro cenário, conforme a Figura 14, nota-se que alguns conjuntos de tarefas impuseram certa dificuldade de particionamento para todos os algoritmos (por exemplo, o conjunto 2 foi difícil para todos os protocolos em ambas as políticas de controle de execução). Para o segundo cenário (Figura 15), ape-

sar do maior espalhamento dos resultados, conjuntos de tarefas mais difíceis ainda podem ser percebidos, embora de forma mais sutil no caso das versões baseadas em *spin*. Estes resultados corroboram a intuição de que os conjuntos de tarefas possuem influência significativa sobre a variância/desvio-padrão do experimento do que os protocolos propriamente ditos.

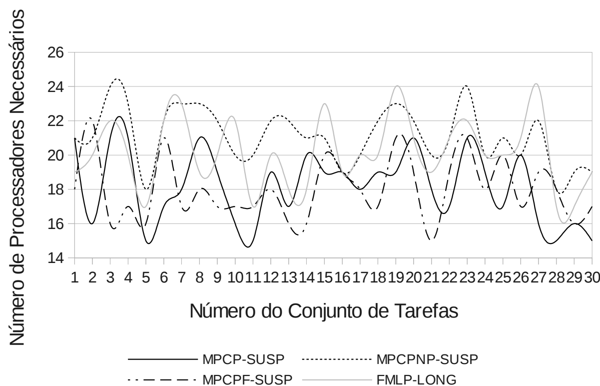


(a) Utilizando suspensão

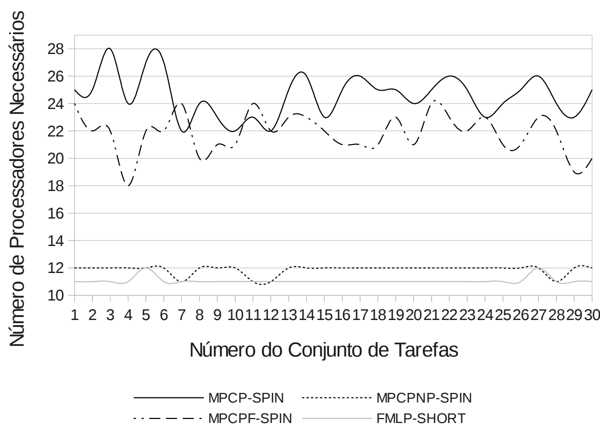


(b) Utilizando *spin*

Figura 14: Resultados individuais para cada um dos 30 conjuntos de tarefas do Experimento 1 (seções críticas de 5μs)



(a) Utilizando suspensão



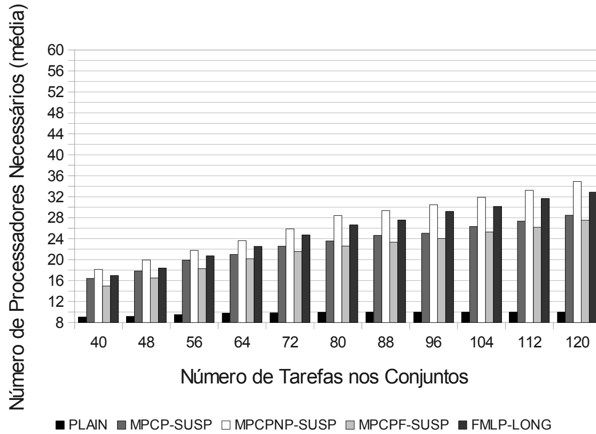
(b) Utilizando *spin*

Figura 15: Resultados individuais para cada um dos 30 conjuntos de tarefas do Experimento 1 (seções críticas de 1280μs)

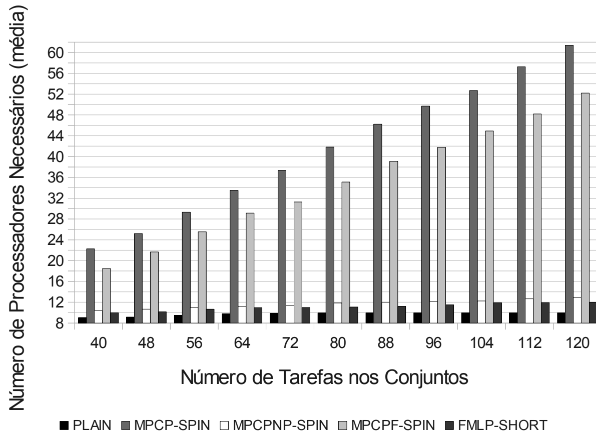
5.1.4.2 Experimento 2: Variação do Número de Tarefas nos Conjuntos

O segundo experimento busca avaliar o comportamento quando se varia o número de tarefas nos conjuntos utilizados no experimento. Inicialmente

começa-se com 40 tarefas e aumenta-se sucessivamente este número até 120. As seções críticas foram fixadas em $500\mu s$. Para este experimento também são utilizadas duas seções críticas por tarefa e duas tarefas por recurso. Os resultados deste experimento são apresentados na Figura 16.



(a) Utilizando suspensão



(b) Utilizando *spin*

Figura 16: Variação do número de tarefas por processador

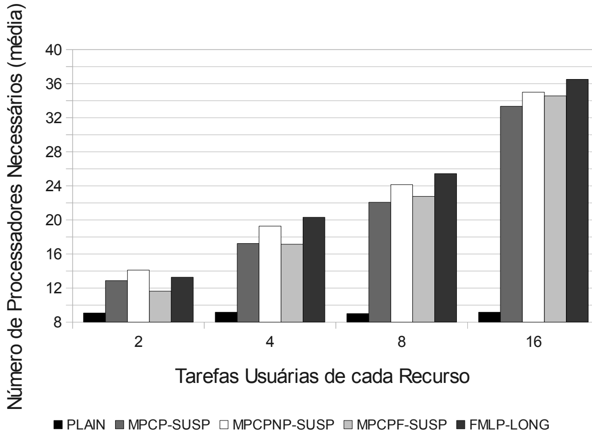
Este experimento apresentou comportamento muito semelhante ao an-

terior (mas sem empate no caso da suspensão) em ambas as versões (baseadas em suspensão na Figura 16(a) e baseadas em *spin* na Figura 16(b)). Para as versões baseadas em suspensão, o melhor resultado também ficou por conta do MPCPF (melhor) e do MPCP (segundo melhor). Para as versões baseadas em *spin* o melhores também foram para o FMLP (melhor) e o MPCPNP (segundo melhor).

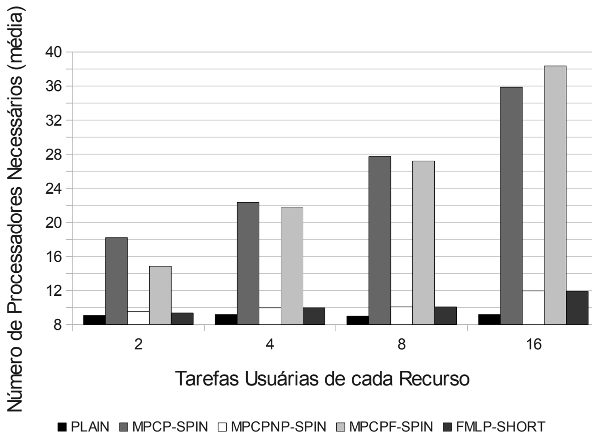
5.1.4.3 Experimento 3: Variação do Número de Tarefas Usuárias por Recurso

No terceiro experimento, o parâmetro que é variado é o número de tarefas usuárias de cada recurso (mantendo 2 seções críticas por tarefa, com 5 tarefas em cada um dos 8 subconjuntos intermediários). O número varia de 2 a 16 usuários para cada recurso, sempre multiplicando por dois a cada teste. As seções críticas foram fixadas em $100\mu s$. Os resultados deste são apresentados na Figura 17.

Para este experimento, no caso das versões baseadas em suspensão (Figura 17(a)) existem dois extremos: com 2 usuários por recurso o melhor resultado foi obtido para o MPCPF enquanto o segundo melhor foi obtido para o MPCP tendo o FMLP resultados muito próximos a este, estando o MPCPNP em última colocação. No outro extremo, com 16 usuários por recurso, o melhor resultado ficou por conta o MPCP, sendo o segundo melhor atribuído ao MPCPF, vindo em seguida o MPCPNP e o FMLP. Este experimento indica que enfileiramento FIFO para protocolos baseados em suspensão tende a degradar o resultado conforme o número de usuários por recurso aumenta. Para as versões baseadas em *spin*, os melhores resultados foram obtidos para o MPCPNP e para o FMLP. O enfileiramento FIFO nao afeta o FMLP *short*, pois neste, o bloqueio é limitado pelo número de processadores (constante no experimento) e não pelo número de usuários (só pode haver uma tarefa bloqueada em cada processador em um determinado instante). Para o MPCPNP-SPIN, o resultado foi semelhante aos experimentos anteriores.



(a) Utilizando suspensão



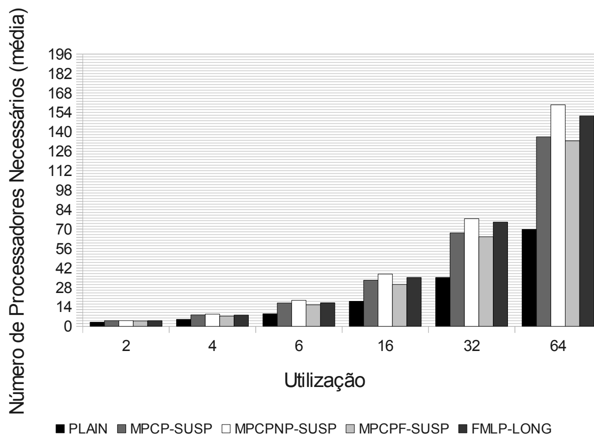
(b) Utilizando *spin*

Figura 17: Variação do número de tarefas usuárias por recurso

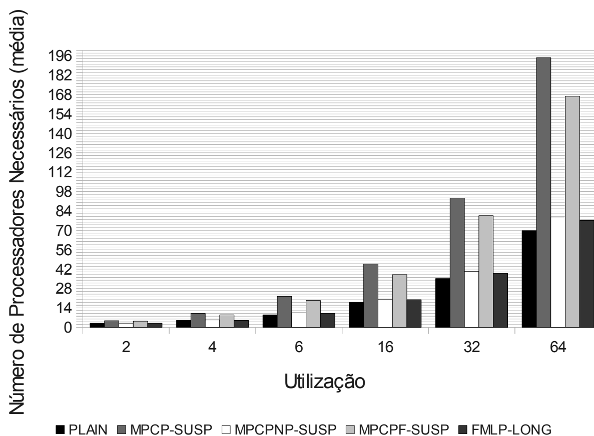
5.1.4.4 Experimento 4: Variação da Utilização dos Conjuntos

No quarto experimento o parâmetro que é variado é a utilização dos conjuntos gerados. A utilização foi variada de 2 até 64, dobrando a cada teste, diferentemente dos experimentos anteriores, onde a utilização foi fixada em

8. Neste cenário, os parâmetros constantes são o tamanho das seções críticas (500 usecs), o número de usuários por recurso (2) e o número de seções críticas por tarefa (2). O resultado deste experimento é apresentado na Figura 18.



(a) Utilizando suspensão



(b) Utilizando *spin*

Figura 18: Variação da utilização dos conjuntos

Segundo a Figura 18, para o caso dos protocolos baseados em suspensão , os melhores resultados foram obtidos para o MPCPF (melhor) e para

o MPCP (segundo melhor), onde para os quais houveram particionamentos com número menor de processadores (Figura 18(a)). O pior resultado ficou por conta do MPCPNP. Para os protocolos baseadas em *spin* os resultados foram inversos aos obtidos com suspensão. Neste caso os melhores resultados foram obtidos para o FMLP (melhor) e o MPCPNP (segundo melhor). Este resultados foram semelhantes aos dos experimentos 1 e 2.

5.1.4.5 Resultados

Tabela 7: Análise subjetiva dos resultados

Protocolos baseadados em suspensão

Rank / **Exp**	1º	2º	3º	4º
1	MPCPF	MPCP/FMLP	MPCPNP	
2	MPCPF	MPCP	FMLP	MPCPNP
3	MPCPF/MPCP	MPCPNP/FMLP		
4	MPCPF	MPCP	FMLP	MPCPNP

Protocolos baseadados em *spin*

Rank / **Exp**	1º	2º	3º	4º
1	FMLP	MPCPNP	MPCPF	MPCP
2	FMLP	MPCPNP	MPCPF	MPCP
3	FMLP	MPCPNP	MPCPF/MPCP	
4	FMLP	MPCPNP	MPCPF	MPCP

Os resultados dos experimentos anteriores foram resumidos na Tabela 7. Nesta tabela, os protocolos foram ordenados em ordem decrescente de desempenho nos testes. Pode-se ver que, de forma geral o MPCPF apresentou os melhores resultados em relação às versões baseadas em suspensão. O MPCPNP apresentou os segundos melhores resultados para as versões dos protocolos baseados em *spin*, perdendo somente para o FMLP. Também houveram situações de empate nos testes, onde um protocolo foi melhor até uma certa faixa dos testes, e outro protocolo foi melhor após esta faixa.

De forma geral, os resultados obtidos foram diferentes daqueles apresentados em (LAKSHMANAN et al., 2009) pelos seguintes fatores: gerador de conjunto de tarefas diferente, embora com os mesmo parâmetros (neste

trabalho foi utilizado UUnifast para geração das utilizações), algoritmo de alocação diferente (neste trabalho, foi utilizada uma variação do RM-FFDU, no artigo citado foi utilizado uma variação do BFD, apresentada no capítulo 3 como *Synchronization-Aware Partitioning Algorithm*) e por último, outros protocolos também foram avaliados. De fato, avaliar protocolos com base em alocação está sujeito aos efeitos causados por anomalias de particionamento. Avaliando os resultados com base no particionamento, pode se ver que, para os cenários de teste avaliados, a diferença entre os protocolos pode ser significativa.

Para o caso das versões de protocolos baseadas em *spin*, a variação não preemptiva MPCPNP (juntamente com o FMLP) induziu a particionamentos com menor número de processadores. Esta variação do protocolo simplifica possíveis implementações reais, por não exigir o cálculo de *ceiling*, sem contar a viabilidade de uso em sistemas de tempo real *soft*. Com relação a variação com enfileiramento FIFO (MPCPF), foi evidenciado que, para o conjunto de testes efetuados, ela apresentou melhores resultados que o MPCP clássico, com exceção da última configuração do experimento 3. Neste último, o enfileiramento FIFO pode ter sido prejudicado pela alta taxa de compartilhamento de recursos (filas de espera grandes para todas as tarefas, no pior caso) similarmente ao que ocorreu na versão baseada em suspensão.

5.2 IMPLEMENTAÇÃO DOS PROTOCOLOS NO LINUX PREEMPT-RT

Os protocolos foram implementados no *kernel* do Linux/PREEMPT-RT, versão 3.0.18-rt34+. O PREEMPT-RT é um *patch* para o *kernel* do Linux, desenvolvido primariamente por Ingo Molnar, tendo como um dos principais objetivos permitir a execução determinística de tarefas de tempo real no *kernel* do Linux. Ele gera um *kernel* com baixas latências e preemptável no núcleo, com exceção do código de interrupção e alguns segmentos, como código de escalonamento, por exemplo.

Embora existam versões para tempo real acadêmicas do Linux, como o LITMUS^{RT} (CALANDRINO et al., 2006), este trabalho foi baseado no PREEMPT-RT devido a sua arquitetura totalmente preemptiva e também ao

fato deste sistema já ser utilizado em produção. A implementação proposta foi feita primariamente para uso em *device-drivers* dedicados a aplicações de tempo real, ou seja, em espaço de *kernel*. A implementação proposta não substitui a implementação atual, sendo somente uma alternativa para certas aplicações que exijam mais determinismo nos tempos de bloqueio e principalmente, como prova de conceito.

Atualmente, o *kernel* PREEMPT-RT utiliza herança de prioridade para limitação dos tempos de bloqueio em seções críticas. Porém, este protocolo foi desenvolvido para sistemas monoprocessados, sendo utilizado como um melhor esforço no caso do PREEMPT-RT (que suporta multiprocessadores), como tentativa de aliviar inversões de prioridade descontroladas. No caso de aplicações críticas em sistemas multiprocessados que acessam hardware através de *device-drivers*, é necessário que os tempos de bloqueio sejam conhecidos. Com estes tempos conhecidos, é possível garantir a escalonabilidade do sistema como um todo. Isto só é possível com a utilização de protocolos adequados para sincronização em sistemas multiprocessados, implementados em monitores ou sistemas operacionais para tempo real. Protocolos adequados para sistemas multiprocessados com escalonamento particionado e prioridade fixa são o *Multiprocessor Priority Ceiling* e o *Flexible Multiprocessor Locking Protocol*, bem como as propostas apresentadas no capítulo anterior.

A proposta escolhida para implementação foi o MPCP não preemptivo (MPCPNP) por ser um protocolo auto-suficiente, no sentido de não precisar de pré-configuração *offline*, como é o caso do MPCP. No MPCP, todos os recursos devem ter seus *ceilings* configurados partindo de uma análise prévia de todas as tarefas que os acessam. Outra questão é que o MPCPNP pode ser implementado sem alterações no escalonador do sistema operacional Linux/PREEMPT-RT, que foi escolhido como plataforma de testes. O MPCP original precisaria da implementação de níveis adicionais de prioridade, o que incluiria mais filas de escalonamento por processador. A implementação do MPCP com enfileiramento FIFO exigiria as mesmas alterações necessárias ao MPCP.

Como protocolo de comparação, o FMLP também foi implementado.

O FMLP também é um protocolo auto-suficiente que não precisa de alterações no escalonador, como as esperadas para o MPCP original e para o MPCP com enfileiramento FIFO.

A implementação da proposta fornece um tipo de dado *struct mpcpnp_mutex*, uma família de funções e uma macro:

- DEFINE_MPCPNP_MUTEX(mutexname, type), onde type pode ser SUSP para controle de execução baseado em suspensão e SPIN para baseado em espera ocupada (*spin*).

- mpcpnp_mutex_lock(struct mpcpnp_mutex *lock) que efetua a aquisição do *mutex* cujo identificador é nomeado de "lock". Esta função possivelmente bloqueia a tarefa chamadora caso o *mutex* esteja de posse de uma tarefa em outro processador.

- mpcpnp_mutex_unlock(struct mpcpnp_mutex *lock) que efetua a liberação do *mutex* cujo identificador é "lock". Esta função deve acordar a próxima tarefa a utilizar o recurso, caso haja alguma.

Adicionalmente, para o FMLP, também é fornecido um tipo de dado *struct fmlp_mutex*, e as devidas funções de manipulação, que são muito semelhantes as apresentadas anteriormente, exceto pelo campo "type", que aqui deve ser LONG ou SHORT para as variações baseadas em suspensão e *spin*, respectivamente.

As implementações propostas tanto para a variação (MPCPNP) quanto para o FMLP, diferentemente da que contempla os *rt_mutexes* com herança de prioridade no *kernel* PREEMPT-RT, não apresenta um *fastpath*. *Fastpaths* são úteis por questões de desempenho, mas no caso deste trabalho, o ganho seria constante ao custo de uma implementação que efetuasse mudanças do escalonador de forma intrusiva. Esta mudança intrusiva significa ajustar de forma postergada a prioridade das tarefas que adquirem recursos via *fastpath*, e os pontos onde ela deve ocorrer estão espalhados por diversos locais do código de escalonamento, como o de reescalonamento por exemplo. *Fastpaths* só apresentam ganhos em situações especificas, como quando o *mutex* está completamente livre para aquisição e/ou o *mutex* a ser liberado não apresenta

nenhuma tarefa em sua fila de espera. De qualquer forma, *fastpath* não afetam o pior caso, somente otimizam o caso comum.

5.3 AVALIAÇÃO EMPÍRICA DOS *OVERHEADS*

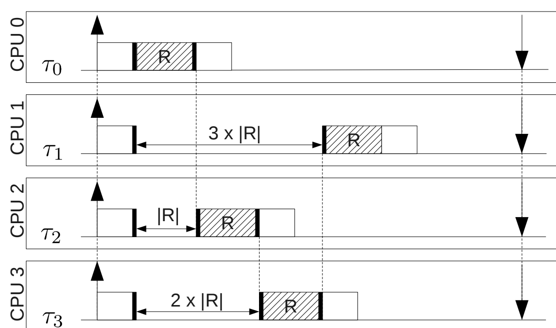
A maioria dos experimentos que envolvem medição de *overhead* de protocolos de sincronização utilizam testes de escalonabilidade inflados com medições pontuais (tempos de eventos como chaveamentos de contexto, por exemplo) aplicados a conjuntos de tarefas gerados de forma sintética. Este trabalho se propõe a efetuar a comparação do *overhead* partindo da medição fim-a-fim de ativações de tarefas que compartilham recursos. Como *overhead*, será considerado o tempo de execução do protocolo (tempo das primitivas de aquisição/liberação de recursos). Este *overhead* não inclui atrasos das tarefas de baixa prioridade.

Para avaliar o *overhead* da implementação proposta, foi desenvolvido um cenário de testes no qual algumas tarefas acessam um *device-driver*. Este *device-driver* possui uma seção crítica protegida com os protocolos implementados. Foi utilizada uma máquina com 4 *cores* (núcleos Intel Xeon 5130 com frequência 2.00GHz), sendo que em cada *core* foi utilizada uma tarefa que acessa o *device-driver*. Estas tarefas foram configuradas com o mesmo tempo de liberação e períodos iguais, de forma que todas elas irão sempre tentar acessar o recurso no mesmo tempo, ou tempos extremamente próximos. Utiliza-se aqui o termo “próximos” devido ao fato das fontes de tempo dos processadores não serem perfeitamente sincronizadas, (*Local APIC timer*, no caso da máquina utilizada).

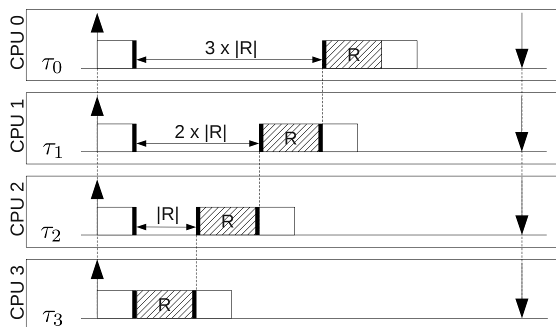
Para o cenário de testes desenvolvido, tanto o MPCPNP quanto o FMLP irão apresentar os mesmos tempos totais de bloqueio (somatório dos tempos de bloqueio de todas as tarefas para uma determinada ativação). Quanto todas as tarefas são idênticas (mesmo período, mesmo recurso e mesmo tempo de computação), independentemente do protocolo utilizado, o somatório dos tempos de bloqueio é igual, sendo que o único fator que é alterado no sistema é a ordem de acesso ao recurso.

É apresentado na Figura 19 dois exemplos de 4 tarefas idênticas, uma

em cada processador. Estas tarefas tentam acessar um determinado recurso ao mesmo tempo utilizando dois protocolos arbitrários. Para o primeiro protocolo (mostrado na Figura 19(a)), a ordem de acesso foi τ_0, τ_2, τ_3 e finalmente τ_1. Para este protocolo arbitrário, o somatório dos tempo de bloqueio foi $6 \times |R|$, onde $|R|$ representa o tamanho da seção crítica relacionada ao recurso R. Para o segundo protocolo (mostrado na Figura 19(b)), foi produzida a sequência de acesso τ_3, τ_2, τ_1 e finalmente τ_0, mas mantendo o mesmo somatório dos tempo de bloqueio, que é $6 \times |R|$.



(a) Ordem arbitrária 1



(b) Ordem arbitrária 2

Figura 19: Exemplos de escala de acesso a um determinado recurso

Naturalmente, em sistemas reais, não se deve desprezar o tempo de execução das primitivas de *lock* e *unlock* necessárias na entrada e na saída das seções críticas, respectivamente. Este tempo é ilustrado como uma pequena

região escura nas figuras 19(a) e 19(b) antes e após as seções críticas, bem como na precedência de bloqueios.

Para uma dada ativação conjunta das 4 tarefas, considera-se como δ o somatório dos tempos das primitivas de *lock* e *unlock* nesta ativação, desconsiderando tempos de bloqueio. A ideia apresentada é generalizada na Equação 5.1.

$$\begin{aligned} soma_tempos &= \sum_{0 \leq i < 4} (tempo_lock(i) + tempo_unlock(i)) \\ &\approx 6 \times |R| + \delta \end{aligned} \tag{5.1}$$

Na Equação 5.1, o somatório dos tempos de *lock*/*unlock* se aproxima da soma de 6 seções críticas adicionado do δ. Se um protocolo apresentar maior *overhead* que outro, então seu δ também será maior que o deste outro. Desta forma, pode-se comparar protocolos em função do δ. Em outras palavras, para o cenário proposto, quanto maior o δ, maior o tempo *lock* e/ou *unlock*.

5.3.1 Condições dos Experimentos

Para a realização dos experimentos, as quatro tarefas foram configuradas como mostrado na Tabela 8.

Tabela 8: Configuração das tarefas utilizadas no experimento

Tarefa	**Período**	**Prioridade**	**Processador**	**Ativações**
τ_0	40 ms	60	1	10000
τ_1	40 ms	61	3	10000
τ_2	40 ms	62	0	10000
τ_3	40 ms	63	2	10000

É mostrado na Tabela 8 que todas as tarefas possuem o mesmo período, e são escalonadas com a política de escalonamento SCHED_FIFO. Esta é uma política para tempo real em sistemas compatíveis com POSIX, admitindo prioridades estáticas de 0 a 99 (maior o número, maior a prioridade). Desta maneira, as ativações e os acessos ao recurso podem ser sincronizados (pela sincronização dos períodos). Para cada ativação ocorrida (ativações sin-

cronizadas), foram somados os tempos de *lock* e *unlock* das 4 tarefas, em tal ativação. O experimento foi efetuado para seções críticas (internas ao *device-driver*) com os tamanhos 10 μs, 20 μs, 40 μs, 80 μs e 160 μs. Para cada tamanho de seção crítica, o experimento foi repetido para cada protocolo e suas variações. Os tempos foram medidos com o uso do registrador *tsc* (*time-stamp counter*) devido ao seu baixo *overhead* de leitura e precisão. Este registrador é incrementado monotonicamente a cada ciclo de *clock*, desta forma, os resultados também são apresentados em ciclos de *clock*. Embora o registrador *tsc* possa não ser sincronizado entre processadores em diversas microarquiteturas, se for utilizado localmente em cada processador, não haverá discrepância de tempo (tempos negativos por exemplo). Na máquina utilizada, um ciclo de *clock* corresponde aproximadamente à 0,5*ns*.

Os resultados (média e desvio padrão) dos dados medidos (correspondem a variável *soma_tempos*) são apresentados na Tabela 9. Estes resultados foram previamente filtrados para a remoção de *outliers*, por motivos que serão discutidos mais adiante. Conforme dito anteriormente, estas medições referem-se aos tempos que são descritos pela Equação 5.1.

Tabela 9: Dados estatísticos básicos do experimento (media +- desvio padrão) com 10000 amostras para cada configuração e resultados em ciclos de *clock*

Prot. \ **Tam. S.C.**	10 μs	20 μs	40 μs	80 μs	160 μs
MPCPNP-SUSP	240038 +- 40537	358564 +- 20504	597789 +- 41903	1069387 +- 23289	2025579 +-23282
FMLP-LONG	235548 +- 40023	358784 +- 20694	595497 +- 49440	1074668 +- 32243	2048230 +- 29625
MPCPNP-SPIN	174380 +- 17162	295085 +- 13494	585654 +- 32127	1008573 +- 15230	1969719 +- 14804
FMLP-SHORT	172884 +- 14457	293065 +- 14919	533552 +- 84209	1011168 +- 13957	1967608 +- 17006

5.3.2 Análise dos Dados

Para análise dos dados, deve-se comparar estatisticamente as amostras obtidas com o intuito de se verificar qual protocolo apresenta maior ou

menor *overhead*. Como os dados medidos não seguem uma distribuição normal de probabilidades, deve-se utilizar um teste estatístico não paramétrico na comparação. No contexto deste trabalho, foi utilizado o teste estatístico de Mann-Whitney-Wilcoxon (MANN; WHITNEY, 1947) para comparação das distribuições dos dados medidos.

Embora o teste escolhido apresente robusteza em relação a *outliers* nas amostras, os dados medidos foram previamente filtrados para a remoção dos mesmos. Neste trabalho, os *outliers* ocorrem quando há interrupções no código que está sob medição. Por exemplo, uma interrupção ocorre quando se está executando uma primitiva de *lock* mas que não pode ser atendida pois esta executa com interrupções desabilitadas. No entanto, quando a primitiva reabilita interrupções, a interrupção é finalmente atendida. Esta interrupção será percebida nas medição pois as chamadas para as primitivas de *lock*/*unlock* são envoltas por código de medição. A identificação destes *outliers* como sendo gerados por interrupções pode ser facilmente efetuada com a utilização de ferramentas para o Linux, como o *ftrace*(ROSTEDT, 2008) por exemplo. Estes *outliers* devem ser removidos da análise pois refletem um comportamento independente e não relativo à implementação dos protocolos. Como não faz parte deste trabalho a comparação de políticas de controle de execução, só serão comparados protocolos sob a mesma política.

É apresentado na Tabela 10 a comparação das amostras, utilizando o teste estatístico citado e um nível de significância $\alpha = 0.05$ para as versões dos protocolos baseados em suspensão. É apresentado na Figura 11 o mesmo para as versões baseadas em *spin*. Na tabela, os operadores relacionais indicam que obteve maior ou menor *overhead* para uma dada configuração.

Segundo a Tabela 10, para as variações baseadas em suspensão, o FMLP *long* apresentou menor *overhead* para seções críticas de 10 μs, sendo que o mesmo se aplica para seções de 40 μs. Para seções críticas de 20 μs, o *overhead* foi equivalente. Para seções de 80μs e de 160 μs, o MPCPNP com suspensão apresentou *overhead* menor. Embora estatisticamente exista esta diferença na distribuição, pode-se observar pelas médias (Tabela 9) que esta é muito pequena, podendo ser considerado *overheads* equivalentes na prática. Como estes protocolos utilizam suspensão como política de controle de execu-

Tabela 10: Comparação das amostras obtidas em cada configuração do experimento através do teste estatístico Mann-Whitney-Wilcoxon (protocolos baseados em suspensão)

10 μs (tamanho da seção crítica)	
Protocolo	FMLP-LONG
MPCPNP-SUSP	$>$
20 μs (tamanho da seção crítica)	
Protocolo	FMLP-LONG
MPCPNP-SUSP	$=$
40 μs (tamanho da seção crítica)	
Protocolo	FMLP-LONG
MPCPNP-SUSP	$>$
80 μs (tamanho da seção crítica)	
Protocolo	FMLP-LONG
MPCPNP-SUSP	$<$
160 μs (tamanho da seção crítica)	
Protocolo	FMLP-LONG
MPCPNP-SUSP	$<$

Tabela 11: Comparação das amostras obtidas em cada configuração do experimento através do teste estatístico Mann-Whitney-Wilcoxon (protocolos baseados em *spin*)

10 μs (tamanho da seção crítica)	
Protocolo	FMLP-SHORT
MPCPNP-SPIN	$>$
20 μs (tamanho da seção crítica)	
Protocolo	FMLP-SHORT
MPCPNP-SPIN	$>$
40 μs (tamanho da seção crítica)	
Protocolo	FMLP-SHORT
MPCPNP-SPIN	$>$
80 μs (tamanho da seção crítica)	
Protocolo	FMLP-SHORT
MPCPNP-SPIN	$<$
160 μs (tamanho da seção crítica)	
Protocolo	FMLP-SHORT
MPCPNP-SPIN	$<$

ção, estão sobre influência direta do escalonador. Tanto suspender quanto retomar tarefas são atribuições do escalonador, estando o comportamento deste fora do controle do experimento. Protocolos baseados em *spin*, por sua vez, não sofrem influência do escalonador.

Para as variações baseadas em *spin* (Tabela 11), o FMLP *short* apresentou menor *overhead* para seções críticas de 10, 20 e 40 μs. Para seções de 80 μs e 160 μs o *overhead* menor ficou por conta do MPCPNP *spin*. Para este caso também vale a observação de que estatisticamente há diferença nas distribuições mas, na prática, os valores são muito próximos. Para este experimento, os resultados foram mais bem comportados que para os protocolos baseados em suspensão (possivelmente, pelo que já foi comentado, pela não influência do escalonador).

De forma geral, mesmo apresentando *overhead* estatisticamente igual em apenas uma configuração para as versões baseadas em suspensão e em apenas uma para as baseadas em *spin*, pode-se notar semelhanças quando observadas apenas as médias e desvios padrão. Empiricamente pode-se observar que o protocolo proposto é tão adequado à implementação em termos de *overhead* quanto o FMLP. É possível que efetivamente os *overheads* dos protocolos sejam idênticos, sendo as diferenças apresentadas em função dos ruídos de medição/hardware e do comportamento do sistema operacional, estando este muitas vezes fora do controle do experimento.

5.4 CONCLUSÕES

Este capítulo apresentou uma avaliação empírica entre as variações do MPCP apresentadas no capítulo anterior e os protocolos presentes na literatura para o mesmo modelo de sistema.

O primeiro experimento empírico consistiu em avaliar a escalonabilidade de conjuntos de tarefas gerados de forma sintética. Em relação a comparação das variações propostas, é sempre difícil fazer experimentos conclusivos sobre protocolos de sincronização em sistemas multiprocessados, devido a anomalias de particionamento, viés induzidos na seleção de parâmetros e geração pseudo-aleatória de utilizações. Então, neste trabalho foram utiliza-

dos cenários de teste com utilizações geradas via UUniFast, cuja distribuição é provada ser imparcial para avaliação de sistemas. Os outros parâmetros foram os mesmos do artigo (LAKSHMANAN et al., 2009). Com relação aos resultados, O MPCPF apresentou os melhores resultados para as versões baseadas em suspensão enquanto o MPCPNP apresentou bons resultados para as versões dos protocolos baseados em *spin*.

A implementação analisou a viabilidade de utilização de um dos protocolos propostos em um sistema operacional real. Trata-se de uma prova de conceito, no caso de que o protocolo proposto MPCPNP é viável no Linux PREEMPT-RT(MOLNAR; GLEIXNER, 2005). O MPCPF, assim como o MPCP, possuem uma limitação no que tange a implementação. Esta limitação se deve ao fato destes protocolos exigirem níveis adicionais de prioridade, além das prioridades normais do sistema, para a implementação do *ceiling*.

O segundo experimento apresentado neste capítulo buscou comparar o *overhead* de implementação de umas das variações (MPCPNP) com o *overhead* do FMLP, utilizando a implementação apresentada anteriormente. Neste estudo, foi comparado o *overhead* quando da alteração do tamanho das seções críticas. Os resultados levaram em consideração somente as tarefas de interesse, e não o impacto para o restante do sistema causado pela utilização de um ou de outro protocolo. Embora ambas as versões baseadas em *spin* tenham apresentado menor *overhead*, não se pode afirmar que estas apresentam desempenho superior as baseadas em suspensão, pois existem outras questões não abordadas neste trabalho, como o fato de *spin* reduzir a utilização do sistema. Como resultado deste experimento, para o cenário de teste apresentado, em linhas gerais pode-se dizer que os protocolos apresentaram *overhead* muito semelhante, embora não sendo possível provar isto estatisticamente em todos os cenários medidos. Isto pode ser explicado pelo fato de nunca se ter controle absoluto sobre o comportamento do sistema operacional, onde pequenas atividades do *kernel* podem impor ruídos sobre os resultados que serão utilizados em testes estatísticos.

De fato, não existe dominância de um único protocolo sobre os demais, no sentido dele sempre ser melhor. Isto não ocorre universalmente, sequer em termos probabilistas, pois o perfil da aplicação é que decide qual protocolo é

melhor ou pior.

6 CONSIDERAÇÕES FINAIS

Com o crescente uso dos sistemas multiprocessados, ocorreu o movimento natural do foco da teoria e pesquisa sobre tempo real para esses sistemas. Para a indústria, os fatores preponderantes para a adoção de arquiteturas multiprocessadas foram econômicos e tecnológicos. Em relação a tempo real neste tipo de sistema, um dos maiores desafios encontrados foi justamente como fazer a sincronização eficaz quanto ao acesso a recursos a nível de seções críticas. Os protocolos de sincronização existentes para sistemas monoprocessados não podem ser diretamente estendidos para multiprocessadores. Ou seja, soluções para monoprocessadores não são capazes de limitar os tempos de bloqueio (controle de inversões de prioridade descontroladas) em sistemas multiprocessados.

Em sincronização para monoprocessadores, soluções simples como o desabilitar de preempções no interior de seções críticas já são capazes de controlar inversões de prioridade. Desabilitar preempções de forma seletiva é justamente o que é feito por protocolos como *Priority Ceiling*. Outro protocolo bem conhecido para monoprocessadores é o *Priority Inheritance*, que atua dinamicamente ajustando as prioridades das tarefas quando estas bloqueiam outras de prioridade mais alta. Contudo, estas técnicas, quando aplicadas a sistemas multiprocessados, não garantem a limitação de bloqueio nem para cenários bastante simples. Para ilustrar este cenário considera-se o seguinte exemplo: a tarefa τ_1 no processador “A” consegue o acesso a um determinado recurso utilizando o protocolo *Priority Ceiling*. Em seguida a tarefa τ_2 no processador “B” emite uma solicitação de acesso a este mesmo recurso e bloqueia (este recurso está com a tarefa τ_1). Então, uma tarefa τ_0 (que não utiliza recursos) preempta a tarefa τ_1 no processador “A”. Esta preempção aumentará o tempo de espera da tarefa τ_2, que agora além de ter que esperar pela tarefaτ_1 acessar a seção crítica, terá que esperar pela tarefa τ_0 executar sua ativação. Este raciocínio leva a um requisito fundamental que deve ser respeitado em todos os protocolos de sincronização para sistemas multiprocessados: tempos de bloqueio devem estar sempre em função de tempos de seções críticas. No

exemplo anterior, o tempo de bloqueio da tarefa τ_2 estava condicionado tanto à duração da seção crítica da tarefa τ_1 quanto ao tempo de execução completo da tarefa τ_0.

Para que este requisito seja atendido, é necessário que, para cada protocolo haja uma combinação correta de caraterísticas como política de acesso a recursos (prioridades ou FIFO, por exemplo), mecanismo de controle de execução em caso de bloqueio (suspender a tarefa ou mantê-la em *busy-wait*/*spin*) e ainda se é possível a preempção de seções críticas. Se o protocolo for baseado em *spin*, este ainda pode ser preemptivo (preempção por tarefas de mais alta prioridade) ou não preemptivo.

Com este requisito em mente, alguns protocolos para escalonamento particionado em multiprocessadores foram propostos, como o *Multiprocessor Priority Ceiling Protocol* (MPCP), *Flexible Multiprocessor Locking Protocol* (FMLP) e o *Multiprocessor Stack Resource Policy* (MSRP). No MPCP (RAJKUMAR et al., 1988), as tarefas acessam os recursos em ordem de prioridade e seções críticas podem ser preemptadas conforme os *ceilings* dos recursos associados. Para o MPCP, também existe uma versão baseada em *spin* preemptivo (LAKSHMANAN et al., 2009). O protocolo *Flexible Multiprocessor Locking Protocol* (BLOCK et al., 2007) foi proposto para escalonamento particionado e global. A versão para escalonamento particionado é apresentada em (BRANDENBURG; ANDERSON, 2008b). Este protocolo se baseia na classificação de recursos em *short*, que apresentam seções críticas curtas, e *long*, que apresentam seções críticas mais longas, sendo que a classificação em sí é deixada à cargo do desenvolvedor da aplicação. O protocolo trata recursos *short* com *spin* (em caso de bloqueios) e suspensão para o caso de recursos *long*. No FMLP as tarefas acessam os recursos em ordem FIFO e as seções críticas não sofrem preempções. O MSRP (GAI et al., 2001a) por sua vez, equivale ao FMLP *short*, com a adição de pseudo-recursos necessários para o reaproveitamento de pilha.

Neste contexto, foram identificadas combinações de características ainda não exploradas pelos protocolos existentes. Partindo destas características, foram apresentados neste trabalho de dissertação de mestrado duas propostas de variações para o protocolo MPCP, com as devidas análises de escalonabilidade

e fatores de bloqueios associados. No entanto, ambas as variações podem ser encaradas também como variações do FMLP, dependendo do ponto de vista, pois abrangem características comuns a ambos os protocolos. A primeira variação, que é o MPCP Não Preemptivo, considera todas as características do MPCP original, exceto pela não preemptividade de seções críticas, que é uma característica do FMLP. Este efeito pode ser conseguido no MPCP com a definição de *ceilings* iguais para todos os recursos. Para a versão baseada em *spin* desta variação, este é efetuado de forma semelhante ao FMLP, ou seja, de forma não preemptiva. A segunda variação, chamada de MPCP Com Enfileiramento FIFO, preserva todas as características do MPCP original, exceto pela alteração da política de enfileiramento, que foi adotada como sendo a mesma do FMLP (que é FIFO). Seções críticas podem ser preemptadas normalmente, de acordo com os *ceilings* dos recursos, como no MPCP original. Para a versão baseada em *spin*, este foi adotado como sendo preemptivo, assim como na versão proposta por (LAKSHMANAN et al., 2009).

Partindo das propostas apresentadas, juntamente com as já existentes na literatura, foi efetuado um estudo empírico com o objetivo de comparar a escalonabilidade de cada protocolo perante alguns cenários de teste. Os cenários de teste foram compostos de conjuntos de tarefas gerados de forma sintética. Tais conjuntos foram posteriormente particionados em um número suficiente de processadores, levando em conta cada um dos protocolos (considerando os já existentes e as variações propostas). O algoritmo de particionamento utilizado foi uma variação do RM-FFDU que considera dependências genéricas entre tarefas. O parâmetro utilizado como comparação foi o número de processadores necessários para comportar um determinado conjunto de tarefas, dada a utilização de um determinado protocolo.

Segundo os resultados obtidos, o protocolo MPCP Com Enfileiramento FIFO apresentou os melhores resultados em relação às versões de protocolo baseadas em suspensão. Com relação às versões baseadas em *spin*, o FMLP *short* apresentou os melhores resultados, seguido do MPCP Não Preemptivo. Entretanto, o MPCP Com Enfileiramento FIFO apresentou resultados pobres em sua versão baseada em *spin* e o MPCP Não Preemptivo também apresentou resultados pobres em sua versão baseada em suspensão. Nenhum protocolo

apresentou melhores resultados em ambas as versões.

Com os resultados obtidos neste trabalho, é possível notar que as estratégias de *spin* preemptivo possuem desvantagens em relação ao *spin* não preemptivo. *Spin* preemptivo congrega os pessimismos tanto da suspensão quanto do *spin* não preemptivo. Este trabalho também corrobora alguns resultados como o fato de *spin* preemptivo ser inferior à suspensão (LAKSHMANAN et al., 2009), embora comparar políticas de controle de execução esteja fora do escopo do trabalho. Também foi corroborado o fato do *spin* não preemptivo ser em termos gerais, superior à suspensão (BRANDENBURG; ANDERSON, 2008a). Com suspensão, aproveita-se o processador na suspensão, porém aumenta-se os tempos de bloqueio, devido aos pessimismos incorporados. Por outro lado, com *spin*, a utilização do sistema é reduzida (pela espera ocupada propriamente dita), mas os bloqueios são menos pessimistas. Em relação aos protocolos baseados em suspensão, notou-se que a combinação de preemptividade de seções críticas (via *ceiling*) com enfileiramento de acesso FIFO se mostrou mais eficiente em relação aos protocolos existentes avaliados sob o cenário de teste adotado. De forma geral, enfileiramento FIFO sempre posicionou como sendo uma melhor alternativa à enfileiramento por prioridades.

Em relação à implementação foi mostrada a viabilidade de implementação de um dos protocolos (mais especificamente o MPCP Não Preemptivo) em ambas as versões (baseada em suspensão e *spin*). A plataforma de implementação foi o Linux/PREEMPT-RT, que é uma versão do Linux voltada para *soft real-time*, focada na redução das latências do sistema e respeito mais rigoroso às prioridades (controle de inversões de prioridades, por exemplo). Como comparação, também foi implementado o FMLP em suas versões short (baseada em *spin*) e *long* (baseada em suspensão).

Para o estudo do *overhead*, os protocolos anteriormente implementados foram utilizados em um experimento, cujos resultados foram obtidos pela medição fim-a-fim de seções críticas de tarefas com prioridade de tempo real. Para o caso dos protocolos baseados em suspensão, embora não se possa afirmar de forma estatística que os *overheads* medidos foram iguais, os valores próximos indicam que, para fins práticos o são. A instabilidade/ruídos do ex-

perimento envolvendo protocolos baseados em suspensão pode ser explicada pela maior influência do escalonador no experimento, visto que este é responsável por suspender e retomar as tarefas do sistema. Utilizando *spin*, estes ruídos são drasticamente reduzidos, pois as tarefas bloqueadas permanecem em *busy-wait*, não executando código de escalonamento. Para estas versões dos protocolos, pode-se ver que o FMLP teve menor *overhead* para seções críticas menores e o MPCP Pão Preemptivo para seções críticas maiores.

Em resumo, as propostas apresentadas se mostraram competitivas tanto em escalonabilidade quanto em *overhead* de implementação. O MPCP Com Enfileiramento FIFO se posicionou muito bem em sua versão baseada em suspensão. O MPCP Não Preemptivo, em sua versão baseada em *spin* obteve resultados muito próximos ao FMLP *short*. Do ponto de vista prático, as variações propostas facilitam a utilização em sistemas reais. Lidar com *ceilings* exige pre-processamento *offline* e níveis de prioridade adicionais, requisitos que são eliminados com a utilização do MPCP Não Preemptivo. Outra característica é que as filas de prioridades exigidas pelo MPCP não podem ser implementadas por algoritmos com complexidade $O(1)$ como filas FIFO, situação solucionada pelo MPCP Com Enfileiramento FIFO.

Como resultados desta dissertação de mestrado foram publicados até o momento 3 artigos em conferências (CARMINATI; OLIVEIRA, 2011) (CARMINATI; OLIVEIRA, 2012b) (CARMINATI; OLIVEIRA, 2012a), e outro foi submetido ao *28th ACM Symposium On Applied Computing* de 2013.

Como perspectiva de trabalhos futuros, pretende-se entender como os protocolos de sincronização podem ajudar na ocupação de processadores, através da possível elevação da taxa de utilização em cada processador individual, pela exposição de paralelismos. Isto fica claro quando se pensa que o problema da alocação é agravado quando se tem um número m de processadores e apenas $m+1$ tarefas. Do ponto de vista teórico, possivelmente não haverá uma alocação viável. Entretanto, na prática, pode-se sempre contornar esta limitação quebrando as tarefas em subtarefas menores, facilitando o processo de *bin-packing*. Contudo, a quebra de tarefas pode induzir ao aparecimento de seções críticas, sendo estas abordadas pelos protocolos apresentados. Isto somente será possível pelo estudo da estrutura das tarefas e não somente do

modelo de tarefas.

REFERÊNCIAS

ANDERSON, J.; SRINIVASAN, A. Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks. In: *Proceedings of the13th Euromicro Conference on Real-Time Systems*. [S.l.]: IEEE Comput. Soc, 2001. p. 76–85. ISBN 0-7695-1221-6.

ANDERSSON, B.; BARUAH, S.; JONSSON, J. Static-priority scheduling on multiprocessors. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2001. p. 193–202. ISBN 0-7695-1420-0.

ANDERSSON, B.; BLETSAS, K.; BARUAH, S. Scheduling Arbitrary-Deadline Sporadic Task Systems on Multiprocessors. In: *Proceedings of the Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2008. p. 385–394. ISBN 978-0-7695-3477-0.

ANDERSSON, B.; JONSSON, J. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In: *Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications*. [S.l.]: IEEE Comput. Soc, 2000. p. 337–346. ISBN 0-7695-0930-4.

ANDERSSON, B.; TOVAR, E. Multiprocessor Scheduling with Few Preemptions. In: *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. [S.l.]: IEEE Computer Society, 2006. p. 322–334. ISBN 0-7695-2676-4.

AUDSLEY, N. C. et al. Hard Real-time Scheduling Approach: The Deadline-Monotonic Approach. *Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software*, IEEE Computer Society, 1991.

BAKER, T. A stack-based resource allocation policy for realtime processes. In: *Proceedings of the Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 1990. p. 191–200. ISBN 0-8186-2112-5.

BAKER, T. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, v. 16, n. 8, p. 760–768, ago. 2005. ISSN 1045-9219.

BARUAH, S. et al. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, v. 15, n. 6, p. 600–625, jun. 1996. ISSN 0178-4617.

BARUAH, S.; MOK, A.; ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In: *Proceedings of the 11th Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 1990. p. 182–190. ISBN 0-8186-2112-5.

BARUAH, S.; ROSIER, L. E.; HOWELL, R. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, v. 2, n. 4, p. 301–324, nov. 1990. ISSN 0922-6443.

BINI, E.; BUTTAZZO, G. C. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, Springer, v. 30, n. 1-2, p. 129–154, maio 2005. ISSN 0922-6443.

BLOCK, A. et al. A Flexible Real-Time Locking Protocol for Multiprocessors. In: *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. [S.l.]: IEEE Computer Society, 2007. p. 47–56. ISBN 0-7695-2975-5. ISSN 1533-2306.

BRANDENBURG, B. B. *Scheduling and Locking in Multiprocessor Real-time Operating Systems*. Tese (Doutorado) — University of North Carolina at Chapel Hill, 2011.

BRANDENBURG, B. B.; ANDERSON, J. H. A Comparison of the M-PCP , D-PCP , and FMLP on LITMUS RT. *Principles of Distribuited Systems*, v. 5401/2008, p. 105–124, 2008.

BRANDENBURG, B. B.; ANDERSON, J. H. An Implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP Real-Time Synchronization Protocols in LITMUS^RT. In: *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. [S.l.]: IEEE Computer Society, 2008. p. 185–194. ISBN 978-0-7695-3349-0.

BRANDENBURG, B. B. et al. Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin? In: *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.]: IEEE Computer Society, 2008. p. 342–353. ISBN 978-0-7695-3146-5.

BURNS, a. Scheduling hard real-time systems: a review. *Software Engineering Journal*, v. 6, n. 3, p. 116, 1991. ISSN 02686961.

CALANDRINO, J. M. et al. LITMUS^RT : A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers. In: *Proceedings of*

the 27th IEEE International Real-Time Systems Symposium. [S.l.]: IEEE Computer Society, 2006. p. 111–126. ISBN 0-7695-2761-2.

CARMINATI, A.; OLIVEIRA, R. S. D. A Proposal of Change to the Multiprocessor Priority Ceiling Protocol. In: *Proceedings of the Brazilian Symposium on Computing System Engineering*. [S.l.]: IEEE Computer Society, 2011. p. 188–193. ISBN 978-0-7695-4641-4.

CARMINATI, A.; OLIVEIRA, R. S. D. A Study About New Variations For The Spin-Based Multiprocessor Priority Ceiling Protocol. In: *Anais do WTR'2012 - XIV Workshop de Sistemas de Tempo Real*. [S.l.]: IEEE Computer Society, 2012.

CARMINATI, A.; OLIVEIRA, R. S. de. On Variations of the Suspension-Based Multiprocessor Priority Ceiling Syncronization Protocol. In: *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation*. Kraków, Poland: IEEE Computer Society, 2012.

CARPENTER, J. et al. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. *North*, p. 1–30, 2004.

CHEN, C.-m.; TRIPATHI, S. K. Multiprocessor Priority Ceiling Based Protocols. *Tech. Report CS-TR-3252, Univ. of Maryland*, p. 1–22, 1994.

CHO, H.; RAVINDRAN, B.; JENSEN, E. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In: *Proceedings of the27th IEEE International Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2006. p. 101–110. ISBN 0-7695-2761-2.

COFFMAN, E. G.; GAREY, M. R.; JOHNSON, D. S. An Application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal on Computing*, v. 7, n. 1, p. 1–17, fev. 1978. ISSN 0097-5397.

DAVIS, R. I.; BURNS, A. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Computing Surveys*, v. 1, n. 216682, 2009.

DAVIS, R. I.; BURNS, A. FPZL Schedulability Analysis. In: *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.]: Ieee, 2011. p. 245–256. ISBN 978-1-61284-326-1.

DAVIS, R. I.; WELLINGS, A. J. Dual priority scheduling. In: *Proceedings of the 16th IEEE Real-Time Systems Symposium*. [S.l.]: IEEE Comput. Soc. Press, 1995. p. 100–109. ISBN 0-8186-7337-0.

DERTOUZOS, M. Control Robotics: the Procedural Control of Physical Processes. *Information Processing*, 1974.

DHALL, S. K.; LIU, C. L. On a Real-Time Scheduling Problem. *Operations Research*, v. 26, n. 1, p. 127–140, jan. 1978. ISSN 0030-364X.

Di Natale, M. et al. A comparison of mpcp and msrp when sharing resources in the janus multiple-processor on a chip platform. In: *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.]: IEEE Computer Society, 2003. p. 189–198. ISBN 0-7695-1956-3.

EMBERSON, P.; STAFFORD, R.; DAVIS, R. I. Techniques For The Synthesis Of Multiprocessor Tasksets. In: *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. [S.l.: s.n.], 2010. p. 6–11.

FARINES, J.-M.; FRAGA, J.; OLIVEIRA, R. *Sistemas de Tempo Real*. [S.l.: s.n.], 2000.

GAI, P.; LIPARI, G.; Di Natale, M. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2001. p. 73–83. ISBN 0-7695-1420-0.

GAI, P.; LIPARI, G.; Di Natale, M. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2001. p. 73–83. ISBN 0-7695-1420-0.

KEITH, Z. B.; MARZULLO, K.; SCHNEIDER, F. B. Priority Inversion and Its Prevention. *Cornell University. Tech. Rep. 90-1088*, 1990.

KLEIN, M. H.; RALYA, T. An Analysis of Input / Output Paradigms for Real-Time Systems An Analysis of Input / Output Paradigms for Real-Time Systems. Pittsburgh, Pennsylvania, USA, n. July, 1990.

LAKSHMANAN, K.; NIZ, D. D.; RAJKUMAR, R. Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors. In: *Proceedings of the 30th IEEE Real-Time Systems Symposium*. [S.l.]: Ieee, 2009. p. 469–478. ISBN 978-0-7695-3875-4.

LEE, S. K.; ALGORITHM, E. D.; LARITY, L. L. A. L. On-line multiprocessor scheduling algorithms for real-time tasks. In: *Proceedings of TENCON'94 - 1994 IEEE Region 10's 9th Annual International Conference on: 'Frontiers of Computer Technology'*. [S.l.]: IEEE Computer Society, 1994. p. 607–611. ISBN 0-7803-1862-5.

LEHOCZKY, J.; SHA, L.; DING, Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: *Proceedings of the Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 1989. p. 166–171. ISBN 0-8186-2004-8.

LEUNG, J. Y.-T.; WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, v. 2, n. 4, p. 237–250, dez. 1982. ISSN 01665316.

LIU, C. L.; LAYLAND, J. W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, v. 20, n. 1, p. 46–61, jan. 1973. ISSN 00045411.

LORTZ, V.; SHIN, K. Semaphore queue priority assignment for real-time multiprocessor synchronization. *IEEE Transactions on Software Engineering*, v. 21, n. 10, p. 834–844, 1995. ISSN 00985589.

LOVE, R. Linux Kernel Development. Addison-Wesley Professional, 2009.

MANN, H. B.; WHITNEY, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Statist*, v. 18, n. 1, p. 50–60, 1947.

MOK, A. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. Tese (Doutorado) — Massachusetts Institute of Technology, 1983.

MOLNAR, I.; GLEIXNER, T. PREEMPT-RT - Linux with real-time pre-emption patches. 2005. <https://rt.wiki.kernel.org>.

NASSOR, E.; BRES, G. Hard real-time sporadic task scheduling for fixed priority schedulers. In: *Prococeedings of the Intl. Workshop on Responsive Comp. Syst*. [S.l.: s.n.], 1991. p. 44–47.

NEMATI, F.; NOLTE, T.; BEHNAM, M. Blocking-Aware Partitioning for Multiprocessors. *Mälardalen Real-Time Research Centre, Mälardalen University, Sweden. Tech. Rep. 2137*, 2010.

NEMATI, F.; NOLTE, T.; BEHNAM, M. Partitioning Real-Time Systems on Multiprocessors with Shared Resources. *Principles of Distributed Systems*, v. 6490/2010, p. 253–269, 2010.

OH, D.-i.; BAKER, T. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. *Real-Time Systems*, v. 15, p. 183–192, 1998.

RAJKUMAR, R. Real-time synchronization protocols for shared memory multiprocessors. In: *Proceedings of the 10th International Conference on Distributed Computing Systems*. [S.l.]: IEEE Computer Society, 1990. p. 116–123. ISBN 0-8186-2048-X.

RAJKUMAR, R. Dealing With Suspending Periodic Tasks. *IBM Thomas J. Watson Research Center, Yorktown Heights*, 1991.

RAJKUMAR, R.; SHA, L.; LEHOCZKY, J. Real-time synchronization protocols for multiprocessors. In: *Proceedings of the Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 1988. p. 259–269. ISBN 0-8186-4894-5.

REGNIER, P. et al. RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor. In: *Proceedings of the IEEE 32nd Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2011. p. 104–115. ISBN 978-1-4577-2000-0.

ROSTEDT, S. ftrace - Function Tracer. 2008. <http://www.mjmwired.net/kernel/Documentation/trace/ftrace.txt>.

SAKSENA, M. Scalable real-time system design using preemption thresholds. In: *Proceedings of the 21st IEEE Real-Time Systems Symposium*. [S.l.]: IEEE Computer Society, 2000. p. 25–34. ISBN 0-7695-0900-2.

SHA, L.; RAJKUMAR, R.; LEHOCZKY, J. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, v. 39, n. 9, p. 1175–1185, 1990. ISSN 00189340.

Srinivasan, AnandBaruah, S. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, v. 84, n. 2, p. 93–98, out. 2002. ISSN 00200190.

APÊNDICE A – Resultados Numéricos dos Experimentos

Este apêndice contém as tabelas com os resultados numéricos dos experimentos comparativos efetuados entre os protocolos. As tabelas apresentam a variação dos parâmetros do experimento e a média de processadores necessários para escalonar os sistemas utilizando cada um dos protocolos.

A.1 EXPERIMENTO 1: VARIAÇÃO DO TAMANHO DAS SEÇÕES CRÍTICAS

Tabela 12: Experimento 1: utilizando protocolos baseados em suspensão. A configuração representa o tamanho das seções críticas em μs

Protocolo / **Conf.**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
5	9.1	9.5	9.8	9.3	9.4
10	9.0	9.7	9.8	9.5	9.6
20	9.0	10.3	10.9	9.8	10.3
40	9.1	11.2	11.7	9.8	10.9
80	9.0	12.4	12.7	11.0	12.0
160	9.1	14.4	15.3	12.4	13.8
320	9.0	14.8	16.6	13.5	15.3
640	9.2	16.8	18.6	16.1	17.8
1280	9.1	18.2	21.1	18.0	20.1

Tabela 13: Experimento 1: utilizando protocolos baseados em *spin*. A configuração representa o tamanho das seções críticas em μs

Protocolo / **Conf.**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
5	9.1	11.1	9.1	10.1	9.1
10	9.0	11.7	9.0	10.6	9.0
20	9.0	13.7	9.0	11.1	9.0
40	9.1	14.3	9.2	11.7	9.2
80	9.0	16.6	9.4	14.2	9.2
160	9.1	19.0	10.0	16.2	9.8
320	9.0	20.4	10.0	17.1	10.0
640	9.2	22.3	10.9	19.7	10.3
1280	9.1	24.6	11.9	21.8	11.1

A.2 EXPERIMENTO 2: VARIAÇÃO DO NÚMERO DE TAREFAS DOS CONJUNTOS

Tabela 14: Experimento 2: utilizando protocolos baseados em suspensão. A configuração representa o número de tarefas nos conjuntos utilizados

Conf. \ Protocolo	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
40	9.1	16.4	18.1	15.0	17.0
48	9.2	17.8	19.9	16.5	18.4
56	9.5	19.9	21.8	18.3	20.7
64	9.8	21.0	23.6	20.2	22.5
72	9.9	22.6	25.9	21.6	24.7
80	10.0	23.6	28.4	22.6	26.6
88	10.0	24.6	29.4	23.3	27.6
96	10.0	25.0	30.5	24.0	29.2
104	10.0	26.3	31.9	25.3	30.2
112	10.0	27.3	33.2	26.2	31.7
120	10.0	28.5	34.9	27.5	32.9

Tabela 15: Experimento 2: utilizando protocolos baseados em *spin*. A configuração representa o número de tarefas nos conjuntos utilizados

Conf. \ Protocolo	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
40	9.1	22.3	10.4	18.5	10.0
48	9.2	25.2	10.7	21.7	10.2
56	9.5	29.3	11.0	25.5	10.7
64	9.8	33.5	11.2	29.1	11.0
72	9.9	37.3	11.4	31.3	11.0
80	10.0	41.8	11.9	35.1	11.1
88	10.0	46.2	12.0	39.1	11.2
96	10.0	49.7	12.2	41.8	11.5
104	10.0	52.7	12.2	44.9	11.9
112	10.0	57.3	12.7	48.2	11.9
120	10.0	61.4	12.9	52.2	12.0

A.3 EXPERIMENTO 3: VARIAÇÃO DO NÚMERO DE TAREFAS USUÁRIAS DE CADA RECURSO

Tabela 16: Experimento 3: utilizando protocolos baseados em *suspensão*. A configuração representa o número de tarefas usuárias de cada recurso

Conf. \ **Protocolo**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
2	9.1	12.9	14.1	11.6	13.3
4	9.2	17.2	19.3	17.1	20.3
8	9.0	22.1	24.1	22.8	25.4
16	9.2	33.3	35.0	34.6	36.5

Tabela 17: Experimento 3: utilizando protocolos baseados em *spin*. A configuração representa o número de tarefas usuárias de cada recurso

Conf. \ **Protocolo**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
2	9.1	18.2	9.5	14.8	9.4
4	9.2	22.3	10.0	21.7	10.0
8	9.0	27.7	10.1	27.2	10.1
16	9.2	35.9	11.9	38.4	11.9

A.4 EXPERIMENTO 4: VARIAÇÃO DA UTILIZAÇÃO DOS CONJUNTOS

Tabela 18: Experimento 4: utilizando protocolos baseados em suspensão. A configuração representa a utilização total dos conjuntos utilizados

Conf. \ **Protocolo**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
2	3.0	4.1	4.2	3.9	4.1
4	5.0	8.3	8.7	7.4	8.2
6	9.0	16.8	18.7	15.5	16.9
16	18.0	33.2	37.7	30.1	35.2
32	35.2	67.2	77.5	64.5	75.0
64	69.8	136.4	159.4	133.5	151.4

Tabela 19: Experimento 4: utilizando protocolos baseados em *spin*. A configuração representa a utilização total dos conjuntos utilizados

Conf. \ **Protocolo**	PLAIN	MPCP	MPCPNP	MPCPF	FMLP
2	3.0	4.8	3.0	4.5	3.0
4	5.0	10.1	5.3	9.0	5.0
6	9.0	22.4	10.4	19.3	10.0
16	18.0	45.7	20.3	38.1	20.0
32	35.2	93.4	40.3	80.6	39.0
64	69.8	194.7	79.7	166.9	77.1