Capítulo 2

O Escalonamento de Tempo Real

Em sistemas de tempo real que seguem a abordagem assíncrona os aspectos de implementação estão presentes mesmo na fase de projeto. Na implementação de restrições temporais, é de fundamental importância o conhecimento das propriedades temporais do suporte de tempo de execução usado e da escolha de uma abordagem de escalonamento de tempo real adequada à classe de problemas que o sistema deve tratar. Neste sentido, este capítulo e o próximo apresentam aspectos da teoria de escalonamento e de sistemas operacionais sob a ótica de tempo real.

Este capítulo trata sobre escalonamento de tempo real de um modo geral. Conceitos, objetivos, hipóteses e métricas são claramente apresentados no sentido de introduzir o que chamamos de um *problema de escalonamento*. Posteriormente, diferentes classes de problemas de escalonamento são examinadas em suas soluções algorítmicas.

2.1 Introdução

Em sistemas onde as noções de tempo e de concorrência são tratadas explicitamente, conceitos e técnicas de escalonamento formam o ponto central na previsibilidade do comportamento de sistemas de tempo real. Nos últimos anos, uma quantidade significativa de novos algoritmos e de abordagens foi introduzida na literatura tratando de escalonamento de tempo real. Infelizmente muitos desses trabalhos definem técnicas restritas e conseqüentemente de uso limitado em aplicações reais. Esse capítulo se concentra em algumas técnicas gerais e em suas extensões, visando também à perspectiva de um uso mais prático.

O foco desse capítulo é sobre técnicas para escalonamentos dirigidos a prioridades. Essa escolha é devido à importância da literatura disponível e, porque cobre diversos aspectos de possíveis comportamentos temporais em aplicações de tempo real. A grande difusão de suportes (núcleos, sistemas operacionais), na forma de produtos, que baseiam seus escalonamentos em mecanismos dirigidos a prioridade é sem dúvida outra justificativa bastante forte para a escolha do enfoque dado nesse capítulo. Essa difusão é tão significativa que organismos de padronização têm adotado essa abordagem de escalonamento de tempo real. Os padrões POSIX [Gal95] - referência para produtos comerciais - enfatizam em suas especificações para tempo real escalonamentos dirigidos a prioridades. Alguns dos algoritmos apresentados nesse capítulo são recomendados pelas especificações POSIX.

Na seqüência é introduzido um conjunto de conceitos que permitem a caracterização de um *problema de escalonamento*.

2.2 Modelo de Tarefas

O conceito de *tarefa* é uma das abstrações básicas que fazem parte do que chamamos um problema de escalonamento. Tarefas ou processos formam as unidades de processamento seqüencial que concorrem sobre um ou mais recursos computacionais de um sistema. Uma simples aplicação de tempo real é constituída tipicamente de várias tarefas. Uma tarefa de tempo real, além da correção lógica (*"correctness"*), deve satisfazer seus prazos e restrições temporais ou seja, apresentar também uma correção temporal (*"timeliness"*).

As restrições temporais, as relações de precedência e de exclusão usualmente impostas sobre tarefas são determinantes na definição de um *modelo de tarefas* que é parte integrante de um problema de escalonamento. Nas seções subseqüentes descrevemos essas formas de restrições que normalmente estão presentes em processamentos de tempo real.

2.2.1 Restrições Temporais

Aplicações de tempo real são caracterizadas por restrições temporais que devem ser respeitadas para que se tenha o comportamento temporal desejado ou necessário. Todas as tarefas de tempo real tipicamente estão sujeitas a prazos: os seus *"deadlines"*. A princípio, uma tarefa deve ser concluída antes de seu *"deadline"*. As conseqüências de uma tarefa ser concluída após o seu *"deadline"* define dois tipos de tarefas de tempo real:

- *Tarefas Críticas* (tarefas *"hard"*) : Uma tarefa é dita crítica quando ao ser completada depois de seu *"deadline"* pode causar falhas catastróficas no sistema de tempo real e em seu ambiente. Essas falhas podem representar em danos irreversíveis em equipamentos ou ainda, em perda de vidas humanas.

- *Tarefas Brandas ou Não Críticas* (tarefas *"soft"*): Essas tarefas quando se completam depois de seus *"deadlines"* no máximo implicam numa diminuição de desempenho do sistema. As falhas temporais nesse caso são identificadas como benignas onde a conseqüência do desvio do comportamento normal não representa um custo muito significativo.

Outra característica temporal de tarefas em sistemas de tempo real está baseada na regularidade de suas ativações. Os modelos de tarefa comportam dois tipos de tarefas segundo suas freqüências de ativações:

- *Tarefas Periódicas*: Quando as ativações do processamento de uma tarefa ocorrem, numa seqüência infinita, uma só ativação por intervalo regular chamado de *Período*, essa tarefa é identificada como periódica. As ativações de uma tarefa periódica formam o conjunto de diferentes *instâncias* da tarefa. Nesse texto assumimos a primeira ativação de uma tarefa periódica ocorrendo na origem dos tempos considerados na aplicação (em *t=0*).

- *Tarefas Aperiódicas ou Tarefas Assíncronas*: Quando a ativação do processamento de uma tarefa responde a eventos internos ou externos definindo uma característica aleatória nessas ativações, a tarefa é dita aperiódica.

As tarefas periódicas pela regularidade e portanto pela previsibilidade, usualmente são associadas a *"deadlines hard"*, ou seja, são tarefas críticas. As tarefas aperiódicas pela falta de previsibilidade em suas ativações, normalmente, tem *"deadlines soft"* associados a suas execuções, compondo portanto as tarefas brandas de um sistema de tempo real. *Tarefas esporádicas* que correspondem a um subconjunto das tarefas aperiódicas, apresentam como característica central a restrição de um intervalo mínimo conhecido entre duas ativações consecutivas e por isso, podem ter atributos de tarefas críticas. As tarefas esporádicas portanto são também associadas a *"deadlines hard"*. As figuras 2.1 e 2.2 apresentam características temporais de tarefas periódicas e aperiódicas, respectivamente.

Outras restrições temporais são importantes na definição do comportamento temporal de uma tarefa:

- *Tempo de computação* (*"Computation Time"*): O tempo de computação de uma tarefa é o tempo necessário para a execução completa da tarefa.

- *Tempo de início* ("*Start Time"*): Esse tempo corresponde ao instante de início do processamento da tarefa em uma ativação.

- *Tempo de término* (*"Completion Time"*): É o instante de tempo em que se completa a execução da tarefa na ativação.

- *Tempo de chegada* (*"Arrival Time"*): O tempo de chegada de uma tarefa é o instante em que o escalonador toma conhecimento de uma ativação dessa tarefa. Em tarefas periódicas, o tempo de chegada coincide sempre com o início do período da ativação. As tarefas aperiódicas apresentam o tempo de chegada coincidindo com o tempo da requisição do processamento aperiódico.

- *Tempo de liberação* (*"Release Time"*): O tempo de liberação de uma tarefa coincide com o instante de sua inclusão na fila de Pronto (fila de tarefas prontas) para executar.

Dependendo do modelo de tarefas assumido o tempo de liberação pode ou não

coincidir com o tempo de chegada da tarefa. Em geral é assumido que tão logo uma instância de uma tarefa chegue, a mesma é liberada na fila de Pronto. Mas, nem sempre esse é o caso; uma tarefa pode ser retardada na sua liberação pelo *"polling"* de um escalonador ativado por tempo (*"tick scheduler"*) ou talvez pelo bloqueio na recepção de uma mensagem (tarefas ativadas por mensagem). Essa não coincidência dos tempos de chegada com as liberações da tarefa conduz ao que é identificado como *"Release Jitter"*, que representa a máxima variação dos tempos de liberação das instâncias da tarefa.

Diante das restrições temporais citadas temos então o comportamento temporal de uma tarefa periódica T_i descrito pela quádrupla (J_i, C_i, P_i, D_i) onde C_i representa o tempo de computação da tarefa, P_i é o período da tarefa, D_i é o *"deadline"* e J_i é o *"Release Jitter"* da tarefa que, de certa maneira, corresponde a pior situação de liberação da tarefa. Nessa representação de tarefas periódicas, J_i e D_i são grandezas relativas (intervalos), medidas a partir do início do período P_i. O *"deadline"* absoluto e o tempo de liberação da $k^{ésima}$ ativação da tarefa periódica T_i são determinados a partir dos períodos anteriores:

$$d_{ik} = (k\text{-}1)P_i + D_i$$
$$r_i = (k\text{-}1)P_i + J_i \qquad \text{(pior situação de liberação).}$$

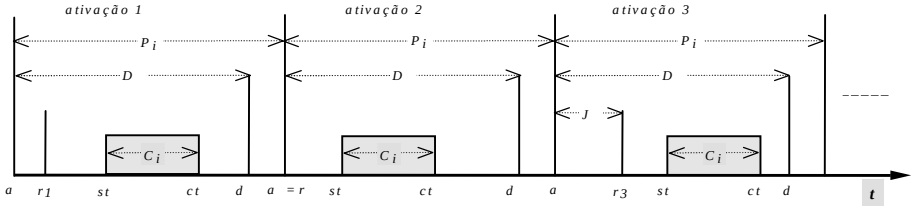


Figura 2.1: Ativações de uma tarefa periódica

Na figura 2.1 é ilustrado alguns dos parâmetros descritos acima. Porém, cada ativação da tarefa periódica (J_i, C_i, P_i, D_i) é definida a partir de tempos absolutos: os tempos de chegada (a_i), os tempos de liberação (r_i), os tempos de início (st_i), os tempos de término (ct_i) e os *"deadlines"* absolutos (d_i).

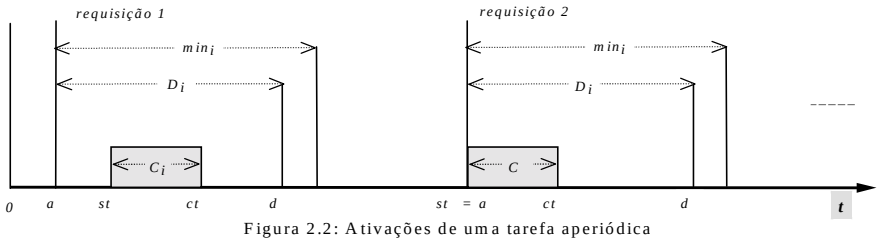


Figura 2.2: Ativações de uma tarefa aperiódica

Uma tarefa esporádica é descrita pela tripla (C_i, D_i, min_i) onde C_i é o tempo de computação, D_i é o *"deadline"* relativo medido a partir do instante da requisição do processamento aperiódico (chegada da tarefa esporádica) e min_i corresponde ao mínimo intervalo entre duas requisições consecutivas da tarefa esporádica. A descrição de uma tarefa aperiódica pura se limita apenas às restrições C_i e D_i. Na figura 2.2, a tarefa aperiódica esporádica (C_i, D_i, min_i) é apresentada com duas requisições. Tomando o tempo de chegada da requisição esporádica 2 como a_2, o *"deadline"* absoluto desta ativação assume o valor dado por: $d_2=a_2+D_i$.

2.2.2 Relações de Precedência e de Exclusão

Em aplicações de tempo real, muitas vezes, os processamentos não podem executar em ordem arbitrária. Implicações semânticas definem *relações de precedência* entre as tarefas da aplicação determinando portanto, ordens parciais entre as mesmas. Uma tarefa T_j é precedida por uma outra T_i ($T_i \rightarrow T_j$), se T_j pode iniciar sua execução somente após o término da execução de T_i. Relações de precedência podem também expressar a dependência que tarefas possuem de informações (ou mesmo sinais de sincronização) produzidas em outras tarefas. As relações de precedência em um conjunto de tarefas usualmente são representadas na forma de um grafo acíclico orientado, onde os nós correspondem às tarefas do conjunto e os arcos descrevem as relações de precedência existentes entre as tarefas.

O compartilhamento de recursos em exclusão mútua define outra forma de relações entre tarefas também significativas em escalonamentos de tempo real: as *relações de exclusão*. Uma tarefa T_i exclui T_j quando a execução de uma seção crítica de T_j que manipula o recurso compartilhado não pode executar porque T_i já ocupa o recurso. Relações de exclusão em escalonamentos dirigidos a prioridade podem levar a *inversões de prioridades* onde tarefas mais prioritárias são bloqueadas por tarefas menos prioritárias.

As relações de precedência e de exclusão serão retomadas no item que trata sobre os algoritmos de escalonamento dirigidos a prioridades.

2.3 Escalonamento de Tempo Real

2.3.1 Principais Conceitos

O termo *escalonamento* (*"scheduling"*) identifica o procedimento de ordenar tarefas na fila de Pronto. Uma escala de execução (*"schedule"*) é então uma ordenação ou lista que indica a ordem de ocupação do processador por um conjunto de tarefas disponíveis

na fila de Pronto. O *escalonador* (*"scheduler"*) é o componente do sistema responsável em tempo de execução pela gestão do processador. É o escalonador que implementa uma *política de escalonamento* ao ordenar para execução sobre o processador um conjunto de tarefas.

Políticas de escalonamento definem critérios ou regras para a ordenação das tarefas de tempo real. Os escalonadores utilizando então essas políticas produzem escalas que se forem *realizáveis* (*"feasible"*), garantem o cumprimento das restrições temporais impostas às tarefas de tempo real. Uma escala é dita *ótima* se a ordenação do conjunto de tarefas, de acordo com os critérios pré-estabelecidos pela política de escalonamento, é a melhor possível no atendimento das restrições temporais.

Tendo como base a forma de cálculo da escala (ordenação das tarefas), algumas classificações são encontradas para a grande variedade de algoritmos de escalonamento de tempo real encontrados na literatura [AuB90, But97, CSR88]. Os algoritmos são ditos *preemptivos* ou *não preemptivos* quando em qualquer momento tarefas se executando podem ou não, respectivamente, ser interrompidas por outras mais prioritárias. Algoritmos de escalonamento são identificados como *estáticos* quando o cálculo da escala é feito tomando como base parâmetros atribuídos às tarefas do conjunto em tempo de projeto (parâmetros fixos). Os dinâmicos, ao contrário, são baseados em parâmetros que mudam em tempo de execução com a evolução do sistema. Os algoritmos de escalonamento que produzem a escala em tempo de projeto são identificados como algoritmos *"off-line"*. Se a escala é produzida em tempo de execução o algoritmo de escalonamento é dito de *"on-line"*. A partir dessas classificações podemos ter algoritmos off-line estáticos, on-line estáticos e on-line dinâmicos. Essas classificações serão revistas na apresentação de algoritmos de escalonamento.

Um *problema de escalonamento*, na sua forma geral, envolve um conjunto de processadores, um conjunto de recursos compartilhados e um conjunto de tarefas especificadas segundo um modelo de tarefas definindo restrições temporais, de precedência e de exclusão. O escalonamento de tempo real, na sua forma geral, é identificado como um problema intratável (NP-completo [GaJ79], [AuB90]). Muito freqüentemente os algoritmos existentes representam uma solução polinomial para um problema de escalonamento particular, onde um conjunto de hipóteses podem expressar simplificações no modelo de tarefas ou ainda na arquitetura do sistema, no sentido de diminuir a complexidade do problema. Quando nenhuma simplificação é usada para abrandar a complexidade no escalonamento, uma heurística é usada para encontrar uma escala realizável ainda que não sendo ótima mas que garanta as restrições do problema.

Os algoritmos de escalonamento estão então ligados a classes de problemas de escalonamento de tempo real. Um algoritmo é identificado como *ótimo* se minimiza algum custo ou métrica definida sobre a sua classe de problema. Quando nenhum custo ou métrica é definido, a única preocupação é então encontrar uma escala realizável. Nesse caso, o algoritmo é dito *ótimo* quando consegue encontrar uma escala realizável para um conjunto de tarefas sempre que houver um algoritmo da mesma classe que também chega a uma escala realizável para esse mesmo conjunto; se o algoritmo ótimo

falha em um conjunto de tarefas na determinação de uma escala realizável então todos os algoritmos da mesma classe de problema também falharão.

2.3.2 Abordagens de Escalonamento

Uma aplicação de tempo real é expressa na forma de um conjunto de tarefas e, para efeito de escalonamento, o somatório dos tempos de computação dessas tarefas na fila de Pronto determina a carga computacional (*"task load"*) que a aplicação constitui para os recursos computacionais em um determinado instante. Uma carga toma características de *carga estática ou limitada* quando todas as suas tarefas são bem conhecidas em tempo de projeto na forma de suas restrições temporais, ou seja, são conhecidas nas suas condições de chegada (*"arrival times"* das tarefas). O fato de conhecer *a priori* os tempos de chegada torna possível a determinação dos prazos a que uma carga está sujeita. As situações de pico (ou de pior caso) nestas cargas são também conhecidas em tempo de projeto. Cargas estáticas são modeladas através de tarefas periódicas e esporádicas.

Cargas dinâmicas ou ilimitadas ocorrem em situações onde as características de chegada das tarefas não podem ser antecipadas. Essas cargas são modeladas usando tarefas aperiódicas, ou seja, basta que se tenha uma tarefa aperiódica no conjunto cujo intervalo mínimo entre requisições seja nulo e teremos as condições de pico dessa carga desconhecida em tempo de projeto.

O escalonamento é muitas vezes dividido em duas etapas. Um *teste de escalonabilidade* é inicialmente executado, no sentido de determinar se as restrições temporais de um conjunto de tarefas são atendidas considerando os critérios de ordenação definidos no algoritmo de escalonamento. A segunda etapa envolve então o cálculo da escala de execução.

Diferentes abordagens de escalonamento são identificadas na literatura, tomando como base o tipo de carga tratado e as etapas no escalonamento identificados acima. Em [RaS94] é definida uma taxonomia que identifica três grupos principais de abordagens de escalonamento de tempo real: as abordagens com garantia em tempo de projeto (*"off-line guarantee"*), com garantia em tempo de execução (*"on-line guarantee"*) e abordagens de melhor esforço (*"best-effort"*).

O grupo *garantia em tempo de projeto* é formado por abordagens que tem como finalidade a previsibilidade determinista. A garantia em tempo de projeto é conseguida a partir de um conjunto de premissas:

- a carga computacional do sistema é conhecida em tempo de projeto (carga estática);
- no sistema existe uma reserva de recursos suficientes para a execução das tarefas, atendendo suas restrições temporais, na condição de pior caso.

O fato de se conhecer *a priori* a carga (conhecer os tempos de chegada das tarefas), permite que se possa fazer testes de escalonabilidade, determinando se o conjunto de tarefas considerado é escalonável e, portanto, garantindo suas restrições temporais ainda em tempo de projeto. O conhecimento prévio da situação de pior caso (situação de pico) permite até que se possa redimensionar o sistema de modo a garantir as restrições temporais mesmo nessa situação de pico. O grupo de abordagens com *garantia em tempo de projeto* é próprio para aplicações deterministas ou críticas onde a carga é conhecida previamente, como em aplicações embarcadas, controle de tráfego ferroviário, controle de processos em geral, etc.

São basicamente dois os tipos de abordagens com garantia em tempo de projeto: *o executivo cíclico* e *os escalonamentos dirigidos a prioridades*. No *executivo cíclico* ([Kop97] [XuP93]) ambos, o teste de escalonabilidade e a produção da escala, são realizados em tempo de projeto. Essa escala (ou grade), definida em tempo de projeto, é de tamanho finito e determina a ocupação dos *slots* do processador em tempo de execução. O teste de escalonabilidade fica implícito no processo de montagem da escala. A complexidade dos algoritmos ou heurísticas usadas no cálculo da escala depende da abrangência do modelo de tarefas usado. Modelos mais complexos envolvem o uso de funções heurísticas para dirigir técnicas de *"branch and bound*" na procura de escalas realizáveis [Pin95].

Essa abordagem é chamada de *executivo cíclico* porque o escalonador em tempo de execução se resume a um simples dispachante, ativando tarefas segundo a grade, ciclicamente. A escala calculada nessa abordagem, em tempo de projeto, reflete o pior caso[1]. Como o pior caso é distante do caso médio e nem sempre ocorre, a ativação das tarefas segundo essas grades, embora satisfaça às restrições temporais, implica em desperdício de recursos que são sempre reservados para o pior caso.

Abordagens com garantia em tempo de projeto baseadas em escalonadores *dirigidos a prioridades* são mais flexíveis. O teste de escalonamento é realizado em tempo de projeto enquanto a escala é produzida *"on-line"* por um escalonador dirigido a prioridades. As tarefas têm suas prioridades definidas segundo políticas de escalonamento que envolvem atribuições estáticas (prioridades fixas) ou dinâmicas (prioridades variáveis), caracterizando escalonadores on-line estáticos ou dinâmicos. Nessa abordagem, o pior caso se reflete apenas no teste de escalonabilidade que é executado em tempo de projeto, decidindo se o conjunto de tarefas é escalonável ou não. Por não existir uma reserva de recursos em tempo de execução como no *executivo cíclico*, os *escalonamentos dirigidos a prioridades* definem soluções mais flexíveis, porém, mantendo também garantias de recursos para o pior caso.

Os grupos de abordagens *garantia dinâmica* (*"on-line guarantee"*) e *de melhor*

[1] No cálculo da escala o pior caso é considerado, levando em conta a pior situação de chegada das tarefas, as tarefas se executando com os seus piores tempos de computação, as tarefas esporádicas ocorrendo na máxima freqüência possível definida por seus intervalos mínimos entre ativações (min_i), etc.

esforço (*"best-effort"*), ao contrário do grupo anterior, não tratam com uma carga computacional previsível; na verdade, a carga tratada por essas abordagens é dinâmica: os tempos de chegada das tarefas não são conhecidos previamente. Quando o pior caso não pode ser antecipado em tempo de projeto, não se consegue prever recursos para todas as situações de carga. Não existindo portanto, a possibilidade de se antecipar que todas as tarefas, em qualquer situação de carga, terão sempre seus *"deadlines"* respeitados. Essas abordagens que tratam com carga dinâmica devem então lidar com situações, em tempo de execução, onde os recursos computacionais são insuficientes para os cenários de tarefas que se apresentam (situações de sobrecarga). Tanto a escala como os testes de escalonabilidade são realizados em tempo de execução nessas abordagens (escalonadores *"on-line"* e dinâmicos).

O grupo de abordagens *com garantia dinâmica* ([RaS94]) se utilizam de um teste para verificar a escalonabilidade do conjunto formado por uma nova tarefa que chega no sistema e das tarefas que já existiam previamente na fila de pronto. Esses testes são chamados de *testes de aceitação* e estão baseados em análises realizadas com hipóteses de pior caso sobre alguns parâmetros temporais. Se o teste indica o conjunto como não escalonável, a nova tarefa que chegou é então descartada. Esse mecanismo de garantia dinâmica em qualquer situação preserva as tarefas já previamente garantidas como escalonáveis. Essas abordagens que oferecem *garantia dinâmica* são próprias para aplicações que possuam restrições críticas mas que operam em ambientes não deterministas. Sistemas militares, sistemas de radar e controle aéreo exemplificam alguns desses sistemas que estão sujeitos a cargas dinâmicas e necessitam atender restrições temporais.

As abordagens de melhor esforço são constituídas por algoritmos que tentam encontrar uma escala realizável em tempo de execução sem realizar testes ou ainda, realizando testes mais fracos. Não existe a garantia de execuções de tarefas atendendo suas restrições temporais. Essas abordagens são adequadas para aplicações não críticas, envolvendo tarefas *"soft"* onde a perda de *"deadlines"* não representa custos além da diminuição do desempenho nessas aplicações. As abordagens de melhor esforço são adequadas para aplicações de tempo real brandas como sistemas de multimídia.

O desempenho de esquemas de melhor esforço, na ocorrência de casos médios, é melhor que os baseados em garantia. As hipóteses pessimistas feitas em abordagens com garantia dinâmica podem desnecessariamente descartar tarefas. Nas abordagens de melhor esforço tarefas são abortadas somente em condições reais de sobrecarga, na ocorrência de falhas temporais (*"deadline"*s não podem ser atendidos).

Algumas técnicas foram introduzidas no sentido de tratar com sobrecargas. A *Computação Imprecisa* [LSL94], o *"Deadline (m,k) Firm"* [HaR95], *Tarefas com Duplo "Deadlines"* [GNM97] são exemplos dessas técnicas adaptativas. Essas técnicas de escalonamento adaptativo são usadas em abordagens de melhor esforço para tratar com sobrecargas em cargas dinâmicas. A figura 2.3 sintetiza as abordagens descritas nesse item.

Nesse capítulo, o estudo de escalonamento de tempo real é concentrado sobre a

abordagem de *escalonamento dirigido a prioridades.*

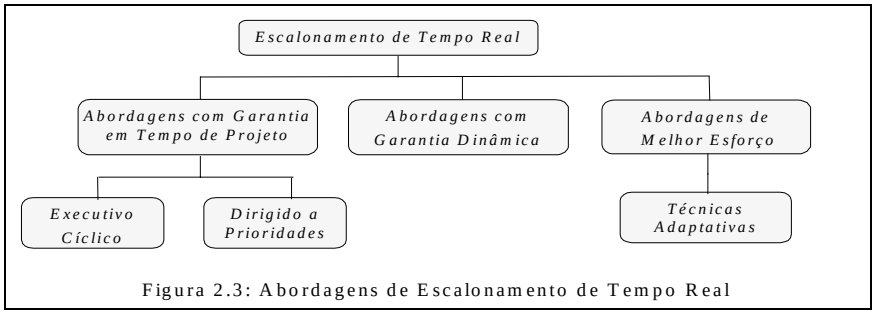


Figura 2.3: Abordagens de Escalonamento de Tempo Real

2.3.3 Teste de Escalonabilidade

Testes de escalonabilidade são importantes no processo de escalonamento de tarefas de tempo real no sentido de determinar se um conjunto de tarefas é escalonável, ou seja, se existe para esse conjunto de tarefas uma escala realizável. Esses testes variam conforme os modelos de tarefas e políticas definidas em um problema de escalonamento. Normalmente, correspondem a análises de pior caso que em certas classes de problemas, apresentam complexidade NP.

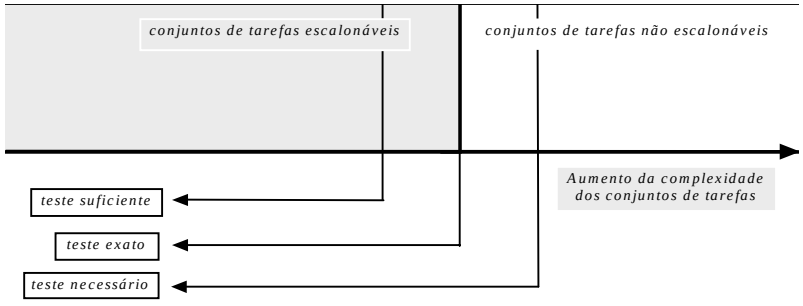


Figura 2.4 : Tipos de testes de escalonabilidade

Na literatura são identificados alguns tipos de testes [Kop92c]:

- *Testes exatos* são análises que não afastam conjuntos de tarefas que apresentam escalas realizáveis. São precisos na medida em que identificam também conjuntos não escalonáveis. Em muitos problemas são impraticáveis os testes exatos.

- *Testes suficientes* são mais simples na execução porém apresentam o custo do descarte de conjuntos de tarefas escalonáveis. É um teste mais restritivo onde

conjuntos de tarefas aceitos nesses testes certamente são escalonáveis; porém entre os descartados podem existir conjuntos escalonáveis.

- *Testes necessários* correspondem também a análises simples porém não tão restritivas. O fato de um conjunto ter passado por um teste necessário não implica que o mesmo seja escalonável. A única garantia que esse tipo de teste pode fornecer é que os conjuntos descartados de tarefas certamente são não escalonáveis.

A figura 2.4 ilustra os três tipos de testes descritos acima.

A *utilização de uma tarefa* T_i que serve como uma medida da ocupação do processador pela mesma, é dado por:

$U_i = C_i / P_i$ se a tarefa T_i é periódica, ou
$U_i = C_i / Min_i$ se a tarefa T_i é esporádica,

onde C_i, P_i e Min_i são respectivamente o tempo máximo de computação, o período e o intervalo mínimo entre requisições da tarefa T_i. A utilização de um processador (U) dá a medida da ocupação do mesmo por um conjunto de tarefas { T_1, T_2, T_3, ..., T_n}:

$$U = \sum_{i}^{n} U_i . \qquad [1]$$

O conceito de *utilização* serve de base para testes simples e bastante usados. A idéia central nesses testes é que a soma dos fatores de utilização (U_i) das tarefas do conjunto não ultrapasse o número de processadores disponíveis para suas execuções:

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

onde m é o número de processadores. Testes baseados na utilização podem ser exatos, necessários ou suficientes, dependendo da política usada e do modelo de tarefas assumido [Kop92c].

2.4 Escalonamento de Tarefas Periódicas

Nesse item é tratado o problema do escalonamento de tarefas periódicas. Se considerarmos aplicações de tempo real de uma maneira geral, sejam controle de processos ou mesmo aplicações de multimídia, as atividades envolvidas nessas aplicações se caracterizam basicamente pelo comportamento periódico de suas ações. As características de tarefas periódicas que determinam o conhecimento *a priori* dos tempos de chegada e, por conseqüência, da carga computacional do sistema, permitem que se obtenha garantias em tempo de projeto sobre a escalonabilidade de um conjunto de tarefas periódicas.

O escalonamento de tarefas periódicas, nesse texto, é discutido em esquemas

dirigidos a prioridades. Nesses esquemas de escalonamento, as prioridades atribuídas às tarefas do conjunto são derivadas de suas restrições temporais, e não de atributos outros como a importância ou o grau de confiabilidade das tarefas. Escalonamentos baseados em prioridades, além de apresentarem melhor desempenho e flexibilidade, se comparados a abordagens como o executivo cíclico, são objeto de uma literatura relativamente recente e abrangente que estende os trabalhos iniciais introduzidos em [LiL73].

Neste capítulo, é feita uma revisão dos algoritmos de prioridade fixa Taxa Monotônica ("*Rate Monotonic*") [LiL73] e *"Deadline"* Monotônico ("*Deadline Monotonic*") [LeW82] e do algoritmo *"Earliest Deadline First"* [LiL73] que apresenta atribuição dinâmica de prioridades. Esses algoritmos clássicos são tidos como ótimos para suas respectivas classes de problemas, sendo caracterizados por modelos de tarefas simplificados. O conceito de utilização e os testes de escalonabilidade nesses algoritmos são apresentados como forma de análise *a priori* para determinar se existe uma escala realizável que garanta as restrições temporais impostas sobre um conjunto de tarefas periódicas. Na seqüência, após o estudo destes modelos clássicos, são aprofundados os esquemas de prioridade fixa, nas suas extensões aos trabalhos de Liu [LiL73]. Um estudo similar para esquemas de prioridade dinâmica é apresentado no *Anexo A*.

2.4.1 Escalonamento Taxa Monotônica [LiL73]

O escalonamento Taxa Monotônica ("*Rate Monotonic*") produz escalas em tempo de execução através de escalonadores preemptivos, dirigidos a prioridades. É um esquema de prioridade fixa; o que define então, o RM como escalonamento estático e *on-line* segundo os conceitos apresentados no item 2.3.1. O RM é dito ótimo entre os escalonamentos de prioridade fixa na sua classe de problema, ou seja, nenhum outro algoritmo da mesma classe pode escalonar um conjunto de tarefas que não seja escalonável pelo RM.

As premissas do RM que facilitam as análises de escalonabilidade, definem um modelo de tarefas bastante simples :

a. As tarefas são periódicas e independentes.
b. O *"deadline"* de cada tarefa coincide com o seu período ($D_i=P_i$).
c. O tempo de computação (C_i) de cada tarefa é conhecido e constante ("*Worst Case Computation Time*").
d. O tempo de chaveamento entre tarefas é assumido como nulo.

As premissas *a* e *b* são muito restritivas para o uso desse modelo na prática, contudo essas simplificações são importantes para que se tenha o entendimento sobre o escalonamento de tarefas periódicas.

A política que define a atribuição de prioridades usando o RM, determina uma

ordenação baseada nos valores de períodos das tarefas do conjunto: as prioridades decrescem em função do aumento dos períodos, ou seja, quanto mais freqüente a tarefa maior a sua prioridade no conjunto. Como os períodos das tarefas não mudam, o RM define uma atribuição estática de prioridades (prioridade fixa).

A análise de escalonabilidade no RM, feita em tempo de projeto, é baseada no cálculo da utilização. Para que *n* tarefas tenham o atendimento de suas restrições temporais quando escalonadas pelo RM, deve ser satisfeito o teste abaixo que define uma condição *suficiente*:

$$U = \sum_{i}^{n} C_i / P_i \leq n \left(2^{1/n} - 1\right). \qquad [2]$$

A medida que *n* cresce, nesse teste, a utilização do processador converge para 0,69. Uma utilização de aproximadamente 70% define uma baixa ocupação do processador que, certamente, implica no descarte de muitos conjuntos de tarefas com utilização maior e que, mesmo assim, apresentam escalas realizáveis ("*feasible*").

Essa condição suficiente pode ser relaxada quando as tarefas do conjunto apresentam períodos múltiplos do período da tarefa mais prioritária. Nesse caso a utilização alcançada sob o RM se aproxima do máximo teórico, coincidindo o teste abaixo com uma condição *necessária e suficiente* [Kop92c]:

$$U = \sum_{i}^{n} C_i / P_i \leq 1.$$

Tarefas Periódicas	***Periodo*** (P_i)	***Tempo de Computação*** (C_i)	***Prioridade RM*** (p_i)	***Utilização*** (U_i)
tarefa A	100	20	1	0,2
tarefa B	150	40	2	0,267
tarefa C	350	100	3	0,286

tabela 2.1: Utilização de tarefas periódicas

A tabela 2.1 mostra um exemplo de conjunto de tarefas periódicas onde o objetivo é verificar a escalonabilidade desse conjunto sob a política Taxa Monotônica. A utilização do processador por esse conjunto de tarefas corresponde a 0,753. Aplicando a equação [2] é concluído que esse conjunto é escalonável sob o RM:

$$0{,}753 \leq n \left(2^{1/n} - 1\right) = 0{,}779$$

A figura 2.1 apresenta – na forma de um *diagrama de Gantt* – a escala RM correspondente à tabela 2.1. As tarefas *A*, *B* e *C* chegam em *t=0*. Por ser mais freqüente (maior prioridade segundo o RM), *A* assume o processador. Em *t=20*, a tarefa *A* conclui e *B* toma posse do processador por ser a mais prioritária na fila de Pronto. A tarefa *C*

assume em *t=60* e é interrompida quando da chegada da nova ativação de *A* em *t=100* que, por sua vez, passa a se executar (preempção de *C* por *A*). *C* sofre mais interrupções de novas ativações das tarefas *B* e *A* em *t=150* e *t=200*, respectivamente.

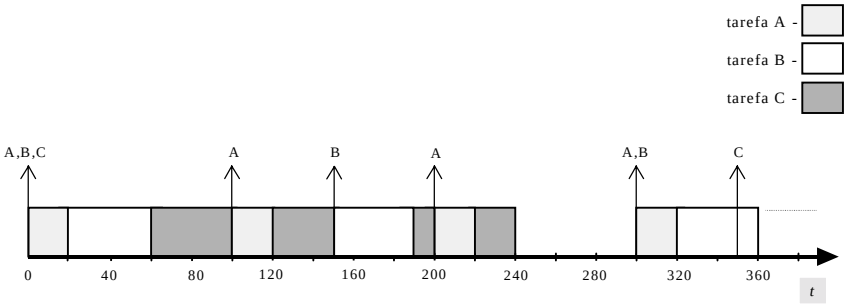


Figura 2.5: Escala RM produzida a partir da Tabela 2.1

2.4.2 Escalonamento *"Earliest Deadline First"* (EDF) [LiL73]

O *"Earliest Deadline First"* (EDF) define um escalonamento baseado em prioridades: a escala é produzida em tempo de execução por um escalonador preemptivo dirigido a prioridades. É um esquema de prioridades dinâmicas com um escalonamento *"on-line"* e dinâmico (item 2.3.1.). O EDF é um algoritmo ótimo na classe dos escalonamentos de prioridade dinâmica. As premissas que determinam o modelo de tarefas no EDF são idênticas às do RM:

a. As tarefas são periódicas e independentes.
b. O *"deadline"* de cada tarefa coincide com o seu período ($D_i=P_i$).
c. O tempo de computação (C_i) de cada tarefa é conhecido e constante ("*Worst Case Computation Time*").
d. O tempo de chaveamento entre tarefas é assumido como nulo.

A política de escalonamento no EDF corresponde a uma atribuição dinâmica de prioridades que define a ordenação das tarefas segundo os seus *"deadlines"* absolutos (d_i). A tarefa mais prioritária é a que tem o *"deadline"* d_i mais próximo do tempo atual. A cada chegada de tarefa a fila de prontos é reordenada, considerando a nova distribuição de prioridades. A cada ativação de uma tarefa T_i, seguindo o modelo de tarefas periódicas introduzido no item 2.2., um novo valor de *"deadline"* absoluto é determinado considerando o número de períodos que antecede a atual ativação (k): $d_{ik}=kP_i$.

No EDF, a escalonabilidade é também verificada em tempo de projeto, tomando como base a utilização do processador. Um conjunto de tarefas periódicas satisfazendo as premissas acima é escalonável com o EDF se e somente se :

$$U = \sum_{i=1}^{n} C_i / P_i \leq 1. \qquad [3]$$

Esse teste é *suficiente* e *necessário* na classe de problema definida para o EDF pelas premissas *a*, *b*, *c* e *d*. Se qualquer uma dessas premissas é relaxada (por exemplo, assumindo $D_i \neq P_i$), a condição [3] continua a ser *necessária* porém não é mais *suficiente*.

No exemplo mostrado na figura 2.6, o mesmo conjunto de tarefas é submetido a escalonamentos EDF e RM. A utilização do conjunto de tarefas é de 100%. Com isto pela equação [2] o conjunto não é escalonável pelo RM. Entretanto, a equação [3] garante a produção de uma escala pelo EDF. No caso (b) da figura 2.6, no tempo *t = 50* a tarefa B perde seu *"deadline"*. Além da melhor utilização, uma outra diferença, normalmente citada na literatura, é que o EDF produz menos preempções que o RM. A favor do RM está a sua simplicidade que facilita a sua implementação.

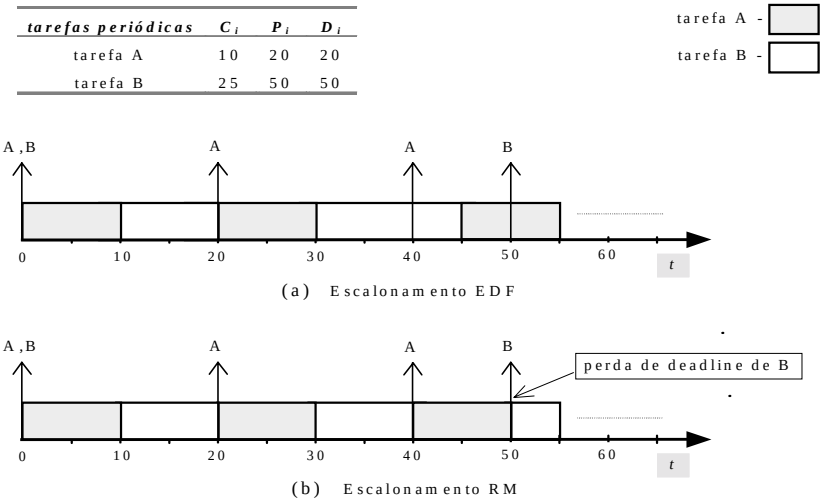


Figura 2.6: Escalas produzidas pelo (a) EDF e (b) RM

2.4.3 Escalonamento *"Deadline"* Monotônico [LeW82]

O *"Deadline Monotonic"* introduzido em [LeW82] estende o modelo de tarefas do Taxa Monotônica. A premissa do RM que limitava os valores de *"deadlines"* relativos aos valores dos períodos das tarefas ($D_i = P_i$), em muitas aplicações, pode ser considerada bastante restritiva. O modelo de tarefas do DM também define tarefas periódicas independentes, o pior caso de tempo de processamento das tarefas (C_i) e o

chaveamento entre tarefas de duração nula. Porém assume *"deadlines"* relativos menores ou iguais aos períodos das tarefas ($D_i \leq P_i$).

A política do DM define uma atribuição estática de prioridades, baseada nos *"deadlines"* relativos das tarefas (D_i). As prioridades são atribuídas na ordem inversa dos valores de seus *"deadlines"* relativos. A produção da escala, portanto, é feita em tempo de execução por escalonador preemptivo dirigido a prioridades. O esquema de prioridades fixas do DM também define um escalonamento estático e *"on-line"*. O *"Deadline"* Monotônico é também um algoritmo ótimo na sua classe de problema. A figura 2.7 mostra a escala produzida pelo DM para um conjunto de tarefas periódicas que apresentam *"deadlines"* menores que seus respectivos períodos.

tarefas periódicas	C_i	P_i	D_i	p_i
tarefa A	2	10	6	1
tarefa B	2	10	8	2
tarefa C	8	20	16	3

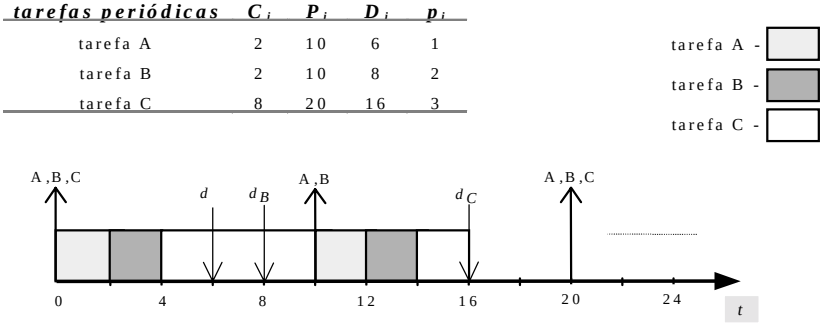


Figura 2.7: Escala produzida pelo DM

Na introdução desse algoritmo os autores não apresentaram em [LeW82] um teste correspondente. Em [ABR91] é definido um teste suficiente e necessário para o DM, baseado no conceito de *tempo de resposta* de uma tarefa. Uma evolução deste teste é descrito na seqüência, no item 2.5.2.

2.5 Testes de Escalonabilidade em Modelos Estendidos

Os testes de escalonabilidade apresentados até aqui são baseados em limites na utilização do processador. Os modelos de tarefas nesses algoritmos preemptivos, baseados em prioridades, são mantidos simples o que facilita as análises nesses testes. Qualquer extensão nos modelos permitindo, por exemplo, que tarefas possam ter relações de precedência entre si ou que os *"deadlines"* assumam valores arbitrários, define modelos mais próximos de aplicações concretas de tempo real. Porém, com essas extensões, os testes fundamentados na noção de utilização, como apresentados anteriormente, passam a ser *condições necessárias* e novos testes mais restritivos são desejáveis.

São muitas as propostas na literatura de extensões dos modelos de [LiL73]. Estes modelos apresentados no item 2.4 podem ser estendidos, por exemplo, para incluírem também tarefas esporádicas. Nos testes que se seguem podemos interpretar o valor P_i também como o mínimo intervalo entre requisições (min_i) de uma tarefa esporádica T_i [ABR91], [TBW94]. Garantir uma tarefa esporádica na sua maior freqüência implica em ter ativações menos freqüentes também respeitando suas restrições temporais.

Como visto anteriormente, a análise (ou teste) de escalonabilidade tenta responder as questões sobre o atendimento dos requisitos de correção temporal de um conjunto de tarefas tempo real quando uma determinada política de escalonamento, e portanto uma atribuição de prioridades é feita. Nessas análises são considerados cenários de pior caso. Além de considerarem as tarefas executando com os seus piores casos de tempo de computação (C_i) e com as suas maiores freqüências de ativação (no caso de tarefas esporádicas), esses testes são construídos levando em conta o *Instante Crítico*. O pior caso de ocorrência de tarefas na fila de Pronto do processador é identificado como instante crítico, ou seja, é o instante onde todas as tarefas do sistema estão prontas para a ocupação do processador. Se nesses cenários de pior caso um conjunto de tarefas consegue recursos suficientes para atender suas restrições, em situações mais favoráveis certamente também atenderá.

Na literatura, tanto para escalonamentos com prioridades fixas como para prioridades dinâmicas são descritos testes de escalonabilidade exatos ou suficientes, que exploram modelos de tarefas mais complexos. Esses testes podem ainda ser construídos usando a noção de *utilização* ou estarem baseados em conceitos como *tempo de resposta* e *demanda de processador*. Nesse item exploramos alguns testes para políticas de prioridade fixa. Os testes referentes a políticas de prioridade dinâmica seguem o mesmo estilo de apresentação e, foram colocados no *Anexo A* no sentido de simplificar a apresentação deste capítulo.

2.5.1 *"Deadline"* Igual ao Período

Ainda não estendendo o modelo do algoritmo Taxa Monotônica (*"Rate Monotonic"*), mas tentando apresentar um teste com melhor desempenho do que o original, em [LSD89] é apresentado um teste onde a utilização do processador não tem mais um sentido estático, dependente somente das restrições temporais das tarefas, mas é também função de uma janela de tempo *t* considerada no seu cálculo. As tarefas de prioridade maior ou igual a *i* que estão disponíveis para execução no processador no intervalo *t* constituem a carga cumulativa (*"workload"*) $W_i(t)$ deste intervalo e é dada por:

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil . C_j , \qquad [4]$$

onde $\lceil t/P_j \rceil$ corresponde ao máximo número de ocorrências de uma tarefa T_j no intervalo t e $\lceil t/P_j \rceil . C_j$ define as necessidades de processador dessa mesma tarefa no intervalo considerado. A divisão do *"workload"* $W_i(t)$ pelo valor de t dá a percentagem de utilização do processador nesse intervalo de tempo t, considerando apenas tarefas de prioridade maior ou igual a Ti :

$$U_i(t) = \frac{W_i(t)}{t}. \qquad [5]$$

A condição necessária e suficiente para que a tarefa Ti seja escalonável é que exista um valor de t em que a sua utilização $U_i(t)$ seja menor ou igual a 1, isto é, que a carga de trabalho $W_i(t)$ não supere o intervalo de tempo t ($W_i(t) \leq t$) [LSD89]. A dificuldade deste teste está em se determinar este valor de t.

Como o modelo define tarefas periódicas com instante crítico ocorrendo em $t = 0$, se cada tarefa T_i respeitar o seu primeiro *"deadline"* então os *"deadlines"* subseqüentes, referentes a suas outras ativações, também serão atendidos. Com isto, os valores de t podem se limitar ao intervalo $(0,P_i]$. Neste caso a procura de t pode se resumir a testar os valores de mínimo de $U_i(t)$ no intervalo considerado:

$$\forall i, \min_{0<t\leq Pi} U_i(t) \leq 1,$$

No lugar de fazer uma procura entre todos os valores de t no intervalo $(0, P_i]$, os cálculos podem se resumir a testar os valores de :

$$S_i = \left\{ kP_j \;\middle|\; 1 \leq j \leq i, \quad k = 1,2,\ldots,\left\lfloor \frac{P_i}{P_j} \right\rfloor \right\}. \qquad [6]$$

Os pontos descritos em S_i correspondem aos tempos de chegada das ativações das tarefas T_j de prioridade maior ou igual a T_i, ocorrendo dentro do período P_i. Estes pontos correspondem aos mínimos de $U_i(t)$, uma vez que, esta função é monotônica decrescente nos intervalos entre chegadas de tarefas T_j. Logo, um conjunto de tarefas periódicas independentes, com instante crítico em $t=0$ e escalonadas segundo o RM, terá seus *"deadlines"* respeitados se :

$$\forall i, \min_{t \in Si} U_i(t) \leq 1. \qquad [7]$$

O teste acima quando comparado com [2], embora mais complexo, pode aprovar conjuntos de tarefas que seriam rejeitados pelo teste suficiente proposto originalmente para o RM. O teste composto a partir de [5] e [7] permite uma maior utilização do processador, formando uma condição suficiente e necessária [LSD89], [Fid98].

Como exemplo, considere o conjunto de tarefas da figura 2.6 (tabela 2.2) em que ocorre na escala do RM, a perda do *"deadline"* da tarefa T_2 em $t=50$. Para a verificação

das condições de escalonabilidade do conjunto de tarefas usamos o teste formado por [5] e [7].

tarefas periódicas	C_i	P_i	D_i
tarefa T_1	10	20	20
tarefa T_2	25	50	50

Tabela 2.2: Exemplo da figura 2.6

Nessa verificação, inicialmente temos que calcular as utilizações $U_i(t)$ para todas as tarefas do conjunto. Começamos então calculando a carga de trabalho correspondente de cada tarefa (equação [4]):

$$W_1(t) = \left\lceil \frac{t}{20} \right\rceil .10 \qquad e \qquad W_2(t) = \left\lceil \frac{t}{50} \right\rceil .25 + \left\lceil \frac{t}{20} \right\rceil .10,$$

determinamos as utilizações (equação [5]):

$$U_1(t) = \frac{W_1(t)}{t} \quad e \quad U_2 = \frac{W_2(t)}{t}.$$

A dificuldade é fazer a inspeção na procura de valores de t que determinam $U_i(t) \bullet 1$. Para a tarefa T_1 esses valores são obtidos a partir de S_1 ([6]): $S_1 = \{20\}$. Então, para o valor de $t=20$, a utilização $U_1(20)$ é determinada como sendo de 0,5. Logo, pela condição [7], a tarefa T_1 é escalonável.

O conjunto referente a T_2, usando também [6], é formado por: $S_2 = \{20, 40, 50\}$. A partir deste conjunto é determinada $U_2(t)$:

$$t = 20 \Rightarrow U_2(20) = 1,75$$

$$t = 40 \Rightarrow U_2(40) = 1,12$$

$$t = 50 \Rightarrow U_2(50) = 1,1.$$

O máximo valor de utilização $U_2(t)$ ocorre em $t = 20$ (o menor valor de t em S_2) e, usando a condição [7], é determinado que o conjunto de tarefas da tabela 2.2 é de fato não escalonável pelo RM.

2.5.2 *"Deadline"* Menor que o Período

O modelo de tarefas, no teste que se segue, relaxa em escalonamentos de prioridade fixa a condição que todas as tarefas periódicas independentes devem ter seus

"deadlines" relativos iguais aos seus respectivos períodos. O modelo assume também tarefas com *"deadlines"* menores aos seus períodos ($D_i \leq P_i$). Esse teste, diferentemente dos anteriores, está fundamentado no conceito de *tempo de resposta*. Em [JoP86], *tempo de resposta máximo* de uma tarefa foi introduzido como sendo o tempo transcorrido entre a chegada e o término de sua execução, considerando a máxima interferência que a tarefa pode sofrer de outras tarefas de maior ou igual prioridade. Uma análise de escalonabilidade que se utilize desse conceito deve representar então o pior caso de interferências (em termos de tempo) sobre cada tarefa do sistema.

Para o cálculo do tempo de resposta máximo de uma tarefa é necessário que se defina uma janela de tempo R_i que corresponda ao intervalo de tempo máximo transcorrido da liberação de uma tarefa T_i até o término de sua execução. A largura R_i corresponde ao tempo necessário, em situação de *instante crítico*, para a execução de T_i e de todas as tarefas com prioridades maiores ou iguais a i (p_i=i). Nestas condições, o tempo de resposta máximo R_i da tarefa T_i é dado por :

$$R_i = C_i + \sum_{j \in hp(i)} I_j$$

onde *hp(i)* é o conjunto de prioridades maior que i e, I_j é a interferência que a tarefa T_i pode sofrer de uma tarefa T_j de prioridade maior, durante a largura R_i. A interferência I_j é calculada por:

$$I_j = \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

onde $\lceil R_i / P_j \rceil$ representa o número de liberações de T_j em R_i. A expressão do tempo de resposta R_i pode ser reescrita como:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j \qquad [8]$$

Uma tarefa T_i mantém suas propriedades temporais sempre que $R_i \leq D_i$. Nesta verificação é necessário a determinação de R_i a partir de [8]. Porém a largura R_i aparecendo em ambos os lados dessa equação, implica na necessidade de um método iterativo para essa determinação. Em [JoP86] é apresentado um método para solucionar [8]:

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{P_j} \right\rceil \cdot C_j$$

onde a solução é conseguida a partir do cálculo de R_i^n, a $n^{ésima}$ aproximação de R_i, quando $R_i^{n+1} = R_i^n$. Nesse método iterativo é assumido, como condição de partida, $R_i^0 = C_i$. O método não converge quando a utilização do conjunto de tarefas for maior que 100%.

O teste fundamentado no conceito de tempo de resposta determina então um

conjunto de *n* tarefas de tempo real como escalonável sempre que a condição $\forall i: 1 \leq i \leq n$ $R_i \leq D_i$ for verificada. Os cálculos dos tempos de respostas formam a base para um teste suficiente e necessário.

tarefas periódicas	C_i	P_i	D_i
tarefa A	2	10	6
tarefa B	2	10	8
tarefa C	8	20	16

Tabela 2.3: Exemplo da figura 2.7

Para clarificar o uso desse teste de escalonabilidade, considere o exemplo que foi apresentado na figura 2.7 onde era apresentada uma escala produzida pelo *"Deadline Monotonic"*. As características das tarefas são reproduzidas na tabela 2.3.

A política DM pelos valores da tabela define uma atribuição de prioridades onde $p_A > p_B > p_C$. Os tempos de resposta das tarefas consideradas são determinados a partir da aplicação da equação [8] nos valores da tabela 2.3. A tarefa T_A, por ser a mais prioritária, não sofre interferência das demais e o seu tempo de resposta é dado por $R_A = C_A = 2$. T_A é escalonável porque seu tempo de resposta máximo é menor que seu *"deadline"* relativo ($D_A = 6$) O cálculo de R_B, ao contrário, envolve mais passos devido a interferência que T_B sofre de T_A. Aplicando [8] é obtido:

$$R_B^0 = C_B = 2$$
$$R_B^1 = 2 + \left\lceil \frac{2}{10} \right\rceil .2 = 4$$
$$R_B^2 = 2 + \left\lceil \frac{4}{10} \right\rceil .2 = 4$$

A tarefa T_B que apresenta $R_B = 4$ é também escalonável ($R_B \leq D_B$). O tempo R_C, por sua vez, envolve as interferências de T_A e T_B em T_C. A partir de [8]:

$$R_C^0 = C_C = 8$$
$$R_C^1 = 8 + \left\lceil \frac{8}{10} \right\rceil .2 + \left\lceil \frac{8}{10} \right\rceil .2 = 12$$
$$R_C^2 = 8 + \left\lceil \frac{12}{10} \right\rceil .2 + \left\lceil \frac{12}{10} \right\rceil .2 = 16$$
$$R_C^3 = 8 + \left\lceil \frac{16}{10} \right\rceil .2 + \left\lceil \frac{16}{10} \right\rceil .2 = 16$$

A tarefa T_C é também escalonável apresentando um tempo de resposta (16 unidades de tempo) no limite máximo para o seu *"deadline"* relativo ($R_C = D_C$). Os resultados obtidos confirmaram a escala obtida pelo DM para o conjunto de tarefas da figura 2.7.

Nos modelos apresentados até aqui as tarefas são assumidas como periódicas e liberadas sempre no início de cada período. Contudo isto nem sempre corresponde a uma hipótese realista. Escalonadores ativados por tempo (*"ticket scheduler"* [TBW94]) podem ser fonte do atraso na liberação de tarefas: as tarefas tem as suas chegadas detectadas nos tempos de ativação do escalonador, determinando atrasos nas suas liberações. Esses atrasos podem ser expressados no pior caso como *"release jitters"* (*J*).

Para determinar as modificações necessárias no sentido de representar no cálculo dos tempos de resposta as variações na liberação das tarefas, é necessário que se introduza o conceito de *período ocupado* ("*busy period*"). Um período ocupado *i* corresponde a uma janela de tempo W_i onde ocorre a execução contínua de tarefas com prioridade maior ou igual a *i* ($p_i = i$). O *"i-busy period"* associado com a tarefa T_i começa quando da liberação da instância de T_i e parte do pressuposto que todas as tarefas de prioridades maior que *i* estão na fila de pronto.

Se considerarmos a janela W_i, o limite máximo das ocorrências de T_j nesse intervalo é dado por $\lceil W_i / P_j \rceil$. Porém ao se assumir que uma instância de T_j, anterior ao início de W_i, experimenta um atraso máximo J_j na sua liberação determinando a interferência dessa instância sobre T_i associada com *Wi*, o número de ativações de T_j que interferem com T_i passa a ser $\lceil (W_i+J_j) / P_j \rceil$. Nessas condições o cálculo de W_i é dado por [TBW94] :

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil \cdot C_j \qquad [9]$$

onde a solução de [9] é conseguida pelo mesmo método iterativo usado para resolver [8]. *Wi* é o intervalo entre a liberação e o termino de T_i. Para o cálculo do tempo de resposta máximo (R_i), correspondendo ao intervalo de tempo entre a chegada e o término da instância da tarefa T_i, é necessário que se considere também o atraso máximo experimentado por T_i na sua liberação:

$$R_i = W_i + J_i . \qquad [10]$$

O teste de um conjunto de tarefas experimentando atrasos em suas liberações leva então em consideração [9], [10] e a verificação da condição $\forall i: 1 \le i \le n \; R_i \le D_i$.

2.5.3 Deadline Arbitrário

O teste baseado em tempos de resposta calculados a partir de [9] e [10] é exato quando aplicado em modelos com $D_i \le P_i$. Porém em modelos de *"deadlines"* arbitrários onde *"deadlines"* podem assumir valores maiores que os respectivos períodos ($D_i > P_i$), esse teste deixa de ser uma condição suficiente [TBW94]. Tarefas que apresentam seus *"deadlines"* maiores que o seus períodos, sofrem o que se pode chamar de

interferências internas, ou seja, uma vez que se pode ter $D_i > P_i$, ocorrências anteriores de uma mesma tarefa podem interferir numa ativação posterior.

É necessário que se estenda o conceito de *"busy period"* para se determinar o pior caso de interferência em uma tarefa T_i onde suas ativações podem se sobrepor. O *"i-busy period*" não começa mais com a liberação da instância de T_i que se deseja calcular o tempo de resposta. O período ocupado i continua sendo a janela de tempo W_i que inclui essa instância de T_i e que corresponde a maior execução contínua de tarefas com prioridade maior ou igual a i ($p_i = i$). Porém, o início desse período de execução contínua de tarefas se dá com a liberação de outra ativação anterior de T_i. Nessa extensão é assumido que a execução de qualquer liberação de uma tarefa será atrasada até que liberações anteriores da mesma tarefa sejam executadas completamente.

Considera-se então o *"i-busy period"* incluindo $q+1$ ativações de T_i que podem se sobrepor na concorrência ao processador pois $D_i > P_i$. O valor da janela de tempo $W_i(q)$ correspondente é dado por [TBW94]:

$$W_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(q)}{P_j} \right\rceil C_j . \qquad [11]$$

O valor $q.C_i$ em [11] representa a interferência interna que a última instância de T_i sofre em $W_i(q)$. Por exemplo, se $q=2$ então T_i terá três liberações em $W_i(q)$. O tempo de resposta da $(q+1)^{ésima}$ ativação de T_i é dada por:

$$R_i(q) = W_i(q) - qP_i \qquad [12]$$

ou seja, o tempo de resposta é dado considerando o desconto do número de períodos de T_i em $W_i(q)$, anteriores a $(q+1)^{ésima}$ ativação de T_i. Para que se obtenha o *tempo de resposta máximo* R_i da tarefa T_i é necessário que se faça a inspeção de todos *"i-busy periods"*, na ordem de valores crescentes de q ($q=0, 1, 2, 3,..$), até um valor de janela que verifique a condição:

$$W_i(q) \leq (q+1)P_i .$$

Esta janela corresponde ao maior valor de *"i-busy period"* que se consegue, antes da execução de uma tarefa menos prioritária que i [TBW94]. Nessas condições, o tempo de resposta máximo R_i é calculado por:

$$R_i = \max_{q=0,1,2,\ldots} R_i(q).$$

Se o efeito de *"deadlines"* arbitrários for incluído em modelos onde as liberações não coincidem com os tempos de chegada das instâncias da tarefa T_i, a janela de tempo $W_i(q)$ e o tempo de resposta R_i passam a ser dados, respectivamente, por:

$$W_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(q) + J_j}{P_j} \right\rceil C_j \qquad [13]$$

$$e \qquad R_i = \max_{q=0,1,2,\ldots} (J_i + W_i(q) - qP_i). \qquad [14]$$

O teste de um conjunto de tarefas com *"deadlines"* arbitrários e experimentando atrasos em suas liberações leva então em consideração [13], [14] e a verificação da condição $\forall i: 1 \leq i \leq n \; R_i \leq D_i$. Este teste funciona como uma análise *a priori* (em tempo de projeto) válida para qualquer política de prioridade fixa. A solução iterativa da equação [13] tem uma complexidade pseudo-polinomial que para propósitos práticos pode ser assumida como polinomial [TBW94].

tarefas periódicas	J_i	C_i	P_i	D_i
tarefa T_1	1	10	40	40
tarefa T_2	3	10	80	25
tarefa T_3	-	5	20	40

Tabela 2.4: Tarefas com Deadlines Arbitrários

Como um exemplo de aplicação deste teste para modelos com *"deadlines"* arbitrários, considere o conjunto de tarefas dado pela tabela 2.4. Suponha que queremos determinar o tempo de resposta máximo da tarefa T_3. Considerando uma atribuição de prioridades onde $p_1 > p_2 > p_3$. A tarefa T_3 sofre interferência das outras duas e pela equação [13], obtemos:

$$W_3(q) = (q+1).5 + \left\lceil \frac{W_3(q) + 1}{40} \right\rceil .10 + \left\lceil \frac{W_3(q) + 3}{80} \right\rceil .10$$

Para que se obtenha o *tempo de resposta máximo* R_3 da tarefa T_3 é necessário que se faça a inspeção de todos *"3-busy periods"*, na ordem de valores crescentes de q ($q=0, 1, 2, 3,..$), até que o valor de janela seja limitado por: $W_3(q) \bullet (q+1).P_3$. Então, para $q = 0$ e usando a equação [13], é obtido:

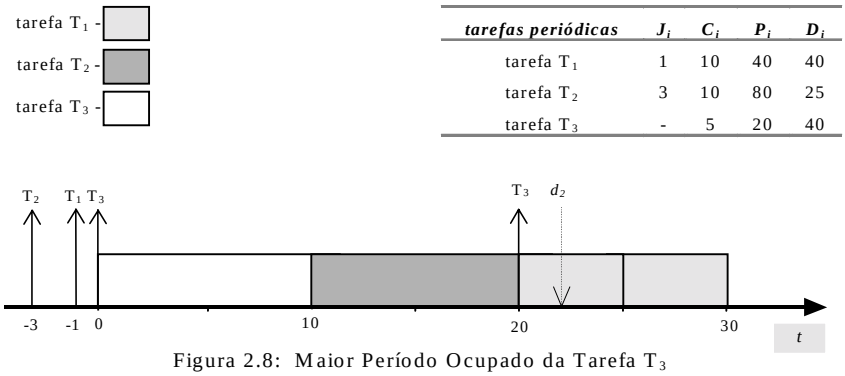
$$W_3^0(0) = C_3 = 5$$
$$W_3^1(0) = 5 + \left\lceil \frac{5+1}{40} \right\rceil .10 + \left\lceil \frac{5+3}{80} \right\rceil .10 = 25$$
$$W_3^2(0) = 5 + \left\lceil \frac{25+1}{40} \right\rceil .10 + \left\lceil \frac{25+3}{80} \right\rceil .10 = 25$$

O valor do tempo de resposta $R_3(0)$ é dado por [12] e corresponde a 25. Com $W_3(0)$ superando o período P_3 ($P_3 = 20$) então, essa janela não corresponde ao maior *"busy*

period" de prioridade 3. Assumindo então *q =1*:

$$W_3^0(1) = C_3 = 5$$
$$W_3^1(1) = 10 + \left\lceil \frac{5+1}{40} \right\rceil .10 + \left\lceil \frac{5+3}{80} \right\rceil .10 = 30$$
$$W_3^2(1) = 10 + \left\lceil \frac{30+1}{40} \right\rceil .10 + \left\lceil \frac{30+3}{80} \right\rceil .10 = 30$$

Com isto obtemos pela equação [12] $R_3(1) = W_3(1) - P_3 = 10$. Como $W_3(1) < 2P_3$, temos então o máximo *"3-busy period"* envolvendo duas ativações da tarefa T_3. Logo o tempo de resposta máximo da tarefa T_3 é o maior dos tempos de resposta obtidos, ou seja, é de valor 25, correspondendo a primeira ativação de T_3 no maior *"3-busy period"*. A figura 2.8 mostra esse período ocupado de prioridade 3 onde T_3 é liberada em suas duas ativações em $t = 0$ e $t = 20$. Por sua vez, devido aos seus *"releases jitters"*, as tarefas T_1 e T_2 que chegam em –1 e –3, respectivamente, só são liberadas em $t = 0$. A primeira ativação de T_3 é empurrada para fora de seu período interferindo com a ativação seguinte da mesma tarefa.



tarefas periódicas	J_i	C_i	P_i	D_i
tarefa T_1	1	10	40	40
tarefa T_2	3	10	80	25
tarefa T_3	-	5	20	40

Figura 2.8: Maior Período Ocupado da Tarefa T_3

2.6 Tarefas Dependentes:Compartilhamento de Recursos

Nos modelos discutidos até a presente seção, as tarefas eram apresentadas como independentes o que, se considerarmos a grande maioria de aplicações de tempo real, não corresponde a uma premissa razoável. Em um ambiente multitarefas o compartilhamento de recursos é implícito e determina alguma forma de relação de exclusão entre tarefas. Comunicações entre tarefas residindo no mesmo processador, por exemplo, podem se dar através de variáveis compartilhadas, usando mecanismos como semáforos, monitores ou similares para implementar a exclusão mútua entre as

tarefas comunicantes.

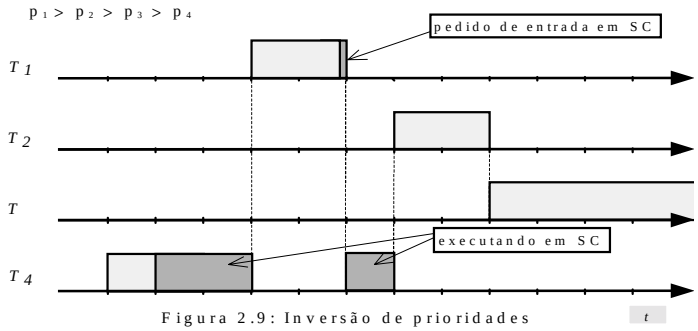


Figura 2.9: Inversão de prioridades

O compartilhamento de recursos e as relações de exclusão decorrentes do mesmo, determinam bloqueios em tarefas mais prioritárias. Esses bloqueios são identificados na literatura de tempo real como *inversões de prioridades*. Considere o cenário da figura 2.9, formado pelas tarefas periódicas T_1, T_2, T_3 e T_4, apresentadas na ordem crescente de seus períodos. Uma escala é construída sobre o conjunto de tarefas baseada no algoritmo *"Rate Monotonic"*. T_1 e T_4 compartilham um recurso guardado por um mecanismo de exclusão mútua. Na escala da figura 2.9, o bloqueio que T_1 sofre pelo acesso anterior de T_4 ao recurso compartilhado caracteriza um exemplo de *inversão de prioridade*: mesmo liberada a tarefa T_1 não consegue evoluir devido ao bloqueio. A tarefa T_1, durante o bloqueio, sofre também interferências de T_2 e de T_3. Esse fato ocorre porque T_4 é a tarefa menos prioritária do conjunto e sofre preempções dessas tarefas intermediárias. As preempções de T_2 e de T_3 sobre a tarefa T_4 podem caracterizar um bloqueio de T_1 com duração de difícil determinação.

Quando as tarefas se apresentam como dependentes, a inversão de prioridades é inevitável em um escalonamento dirigido a prioridades. O que seria desejável é que as inversões de prioridades que eventualmente possam ocorrer nas escalas produzidas sejam limitadas. É nesse contexto que alguns métodos para controlar o acesso em recursos compartilhados foram introduzidos. Esses métodos impõem certas regras no compartilhamento dos recursos de modo que o pior caso de bloqueio experimentado por uma tarefa no acesso a uma variável compartilhada possa sempre ser conhecido *a priori*.

Nesse item examinamos duas destas técnicas: o *Protocolo Herança de Prioridade* e o Protocolo de Prioridade Teto (“*Priority Ceiling Protocol”*) desenvolvidos para esquemas de prioridades fixas [SRL90]. A técnica Política de Pilha (“*Stack Resource Policy*” [Bak91]) própria para escalonamentos de prioridades dinâmicas é apresentada no *anexo A*.

2.6.1 Protocolo Herança de Prioridade

Uma solução simples para o problema de inversões prioridades não limitadas seria ter desabilitada a preempção quando tarefas entrassem em seções críticas [AuB90]. Esse método de escalonamento híbrido (preemptivo e não preemptivo) evita as interferências de tarefas intermediárias, porém, é bastante penalizante com tarefas mais prioritárias que não usam os recursos compartilhados, quando as seções críticas envolvidas não são pequenas.

O método de *Herança de Prioridade* apresentado por [SRL90] corresponde a uma solução eficiente para tratar o problema de inversões de prioridades provocadas pelas relações de exclusão. Nesse protocolo as prioridades deixam de ser estáticas; toda vez que uma tarefa menos prioritária bloqueia uma de mais alta prioridade em um recurso compartilhado, a menos prioritária ascende à prioridade da tarefa bloqueada mais prioritária.

- **Descrição do Protocolo**

O uso do Protocolo Herança de Prioridade (PHP) determina que as tarefas sejam definidas possuindo uma prioridade *nominal ou estática*, atribuída por alguma política de prioridade fixa (RM, DM, etc.) e uma prioridade *dinâmica ou ativa* derivada das ações de bloqueio que ocorrem no sistema. Inicialmente, numa situação sem bloqueio no sistema, todas as tarefas apresentam suas prioridades estáticas coincidindo com suas prioridades ativas. As tarefas são escalonadas tomando como base suas prioridades ativas.

Quando uma tarefa T_i é bloqueada em um semáforo, sua prioridade dinâmica ou ativa é transferida para a tarefa T_j que mantém o recurso bloqueado. Quando reassume, T_j executa o resto de sua seção crítica com a prioridade herdada de T_i ($p_j = p_i$). Uma tarefa *herdada*, ao se executar, sempre a mais alta das prioridades das tarefas que mantenha sob bloqueio. No momento em que T_j ao completar sua seção crítica, libera o semáforo associado, a sua prioridade ativa retorna à prioridade nominal ou assume a mais alta prioridade das tarefas que ainda estejam sob seu bloqueio.

A aplicação do Protocolo Herança de Prioridade no exemplo anterior (da figura 2.9) implica que T_4 não mais sofrerá de interferências intermediárias (preempções de T_2 e T_3) porque herda a prioridade de T_1 em *t=5* (figura 2.10). E, ao sair da sua seção crítica (em *t=7*, figura 2.10), T_4 volta ao nível de sua prioridade original.

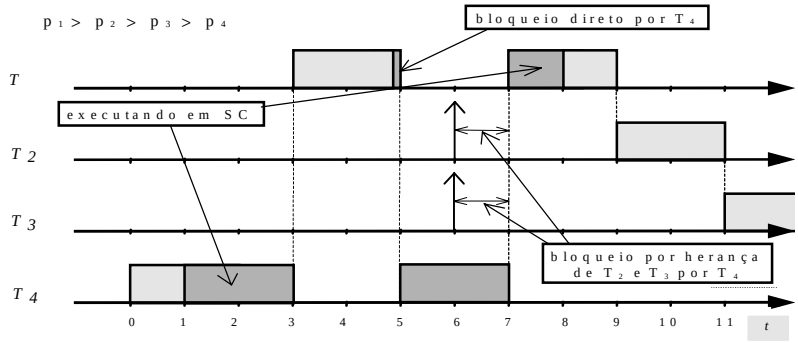


Figura 2.10: Exemplo do uso do PHP

Uma tarefa mais prioritária quando executando sob o PHP pode sofrer dois tipos de bloqueios [But97]:

- *Bloqueio direto*: que ocorre quando a tarefa mais prioritária tenta acessar o recurso compartilhado já bloqueado pela tarefa menos prioritária.

- *Bloqueio por herança*: ocorre quando uma tarefa de prioridade intermediária é impedida de continuar sua execução por uma tarefa que tenha herdado a prioridade de uma tarefa mais prioritária.

No exemplo da figura 2.10, as tarefas T_2 e T_3 sofrem bloqueios por herança em $t = 6$ (T_2 e T_3 chegam em $t = 6$), e T_1 está sujeita a um bloqueio direto em $t = 5$.

O PHP define um limite superior para o número de bloqueios que uma tarefa pode sofrer de outras menos prioritárias. Se uma tarefa T_i pode ser bloqueada por *n* tarefas menos prioritárias, isto significa que, em uma ativação, T_i pode ser bloqueada por *n* seções críticas, uma por cada tarefa menos prioritária. Por outro lado, se houverem *m* distintos semáforos (recursos compartilhados) que podem bloquear diretamente T_i, então essa tarefa pode ser bloqueada no máximo a duração de tempo correspondente às *m* seções críticas, sendo uma por cada semáforo. Em [SRL90] é então assumido que sob o Protocolo Herança de Prioridade, uma tarefa T_i pode ser bloqueada no máximo a duração de *min (n, m)* seções críticas.

A ocorrência de seções críticas aninhadas permite o surgimento de um terceiro tipo de bloqueio: o *transitivo*. A figura 2.11, mostra quatro tarefas (T_1, T_2, T_3 e T_4) onde T_2 e T_3 possuem seções aninhadas. T_1 é mostrada bloqueada por T_2; por sua vez, a tarefa T_2 é bloqueada por T_3 e, por fim, T_3 é bloqueada por T_4. Nessa cadeia de bloqueios, a tarefa T_1 sofre um bloqueio indireto ou transitivo de T_4. A tarefa T_1 só retoma o seu processamento quando houver a liberação, na seqüência, das seções críticas de T_4, T_3 e T_2, respectivamente. Bloqueios transitivos portanto, criam a possibilidade de que se formem cadeias de bloqueios que podem levar até mesmo a situações de *deadlocks*.

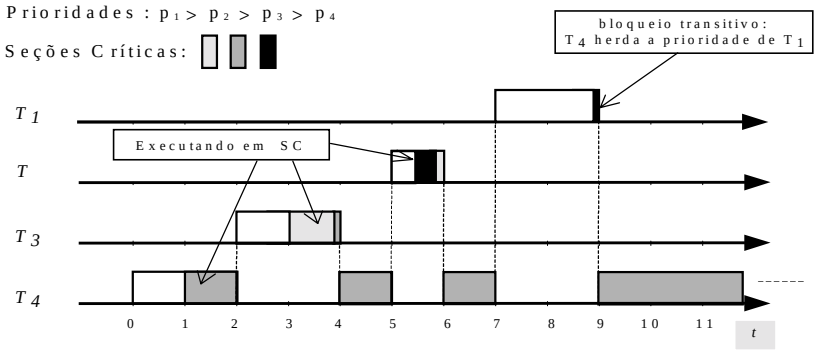


Figura 2.11: Bloqueio transitivo

- **Extensões de Testes de Escalonabilidade Tomando como Base o PHP**

Uma determinação precisa do valor de bloqueio máximo B_i que uma tarefa T_i pode sofrer quando do uso do PHP é certamente bem difícil, uma vez que, seções críticas de tarefas menos prioritárias podem interferir com T_i através de diferentes tipos de bloqueios. Dependendo da complexidade do modelo de tarefas, fica impraticável a determinação precisa de B_i. Alguns autores apresentam métodos para estimativas desse tempo de bloqueio ([BuW97], [But97] e [Raj91]). Um cálculo mais preciso de B_i envolve procuras exaustivas que considerando a complexidade do conjunto de tarefas pode ser impraticável.

O limite imposto pelo PHP no bloqueio máximo (B_i) que uma tarefa T_i pode sofrer de tarefas menos prioritárias, tem que se refletir nas análises de escalonabilidade de esquemas baseados em prioridades fixas. Em [SRL90] e [SSL89] o teste do RM (equação [**2**]) é estendido no sentido de incorporar as relações de exclusão de um conjunto de tarefas:

$$\left(\sum_{j=1}^{i} \frac{C_j}{P_j}\right) + \frac{B_i}{P_i} \leq i\,(2^{1/i} - 1), \quad \forall i. \qquad [15]$$

O somatório do teste acima considera a utilização de tarefas com prioridade maior ou igual a p_i e o termo B_i / P_i corresponde à utilização perdida no bloqueio de T_i por tarefas menos prioritárias. Para que um conjunto de *n* tarefas seja considerado escalonável pelo *"Rate Monotonic"*, é necessário que as *n* condições geradas a partir desse teste sejam verificadas.

tarefas	C_i	P_i	B_i
T_1	6	18	2
T_2	4	20	4
T_3	10	50	0

Tabela 2.5

A tabela 2.5 apresenta um conjunto de tarefas periódicas com seus respectivos bloqueios máximos quando executadas sob o PHP. O uso do teste [15] nesse conjunto de tarefas implica nas relações abaixo:

$$\frac{C_1}{P_1} + \frac{B_1}{P_1} \leq 1$$
$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{B_2}{P_2} \leq 0,82$$
$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} \leq 0,78$$

Todas essas relações acima se verificam para os valores de indicados na tabela 2.5; o que indica que o conjunto é escalonável e todas as tarefas se executarão dentro de seus *"deadlines"*.

Uma outra variante desse teste onde a escalonabilidade pode ser verificada apenas por uma equação só, é também apresentado em [SRL90]:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + max\left(\frac{B_1}{P_1}, \ldots, \frac{B_n}{P_n}\right) \leq n.\left(2^{\frac{1}{n}} - 1\right). \qquad [16]$$

O novo teste é mais simples que o anterior porém é mais restritivo e menos preciso. Como exemplo, considere o uso do teste [16] na verificação da escalonabilidade do mesmo conjunto de tarefas da tabela 2.5. A equação [16] aplicada às condições desse mesmo conjunto implica em:

$$\frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3} + max\left(\frac{B_1}{P_1}, \frac{B_2}{P_2}\right) \leq 3\left(2^{\frac{1}{3}} - 1\right)$$

onde, se substituirmos os valores da tabela 2.5, chegaremos a conclusão que o conjunto é não escalonável. Como esse teste é mais restritivo, todo o conjunto descartado em relação ao teste [16] deve ser verificado com o teste [15] no sentido de confirmar o descarte. Porém o conjunto que passar pelo teste [16] certamente é escalonável.

Os testes para políticas de prioridades fixas, apresentados na seção 2.5, podem ser facilmente estendidos no sentido de incluir os bloqueios que sofrem cada tarefa no conjunto. O teste proposto em [LSD89] baseado em utilização, na sua versão estendida

toma a seguinte forma, ([Fid98]) :

$$U_i(t) = \frac{\sum_{j=1}^{i} \left\lceil \frac{t}{P_j} \right\rceil C_j + B_i}{t}, \quad \forall i, \ min_{0<t\le P_i} U_i(t) \le 1. \qquad [17]$$

O bloqueio B_i nessa equação é também apresentado como utilização perdida no bloqueio de T_i por tarefas menos prioritárias. O teste baseado em tempo de resposta para modelos envolvendo *"deadlines"* arbitrários apresentado em [TBW94] é também facilmente estendido:

$$W_i(q) = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(q) + J_j}{P_j} \right\rceil C_j. \qquad [18]$$

O bloqueio B_i é apresentado na equação [18] como uma interferência sofrida por T_i. Todos os testes apresentados com as extensões referentes ao limite máximo de bloqueio definido sob o PHP, deixam de ser exatos e passam a ser condições suficientes. Isto porque, o cálculo de B_i conforme citado acima, não é exato, refletindo um pessimismo por vezes exagerado.

2.6.2 Protocolo de Prioridade Teto (“*Priority Ceiling Protocol*”)

A idéia central no *"Priority Ceiling Protocol"* (PCP), introduzido em [SRL90], é limitar o número de bloqueios ou inversões de prioridades e evitar a formação de cadeias de bloqueios e *"deadlocks"* em uma ativação de tarefa. O PCP é dirigido para escalonamentos de prioridade fixa, como o *"Rate Monotonic"*. Esse protocolo é uma extensão do Protocolo Herança de Prioridade ao qual se adiciona uma regra de controle sobre os pedidos de entrada em exclusão mútua.

Em essência, o PCP assegura no máximo uma inversão de prioridades por ativação. Ou seja, se uma tarefa menos prioritária T_j tiver uma seção crítica executando em um recurso compartilhado com T_i, então nenhuma outra tarefa menos prioritária que T_i conseguirá entrar em seção crítica que possa também bloquear T_i. Essa regra evita também que uma tarefa possa entrar em uma seção crítica se já houverem semáforos que podem levá-la a bloqueios.

- **Descrição do Protocolo**

Nesse protocolo, todas as tarefas apresentam também uma prioridade *nominal* ou *estática*, definida pelo RM. Uma prioridade *ativa* ou *dinâmica* que incorpora o mecanismo de herança do PHP, é também usada para definir a inclusão da tarefa nas escalas em tempo de execução. Sempre que uma tarefa menos prioritária bloquear uma mais prioritária, sua prioridade ativa assume a prioridade da tarefa mais prioritária. A

herança de prioridades é transitiva, ou seja, se uma tarefa menos prioritária T_3 bloqueia uma tarefa T_2 de prioridade média e, por sua vez, T_2 bloqueia uma tarefa mais prioritária T_1, então T_3 herda a prioridade de T_1.

Todos os recursos acessados em exclusão mútua possuem um valor de *prioridade teto* (“*ceiling*” $C(S_k)$) que corresponde à prioridade da tarefa mais prioritária que acessa o recurso.

A regra que define as entradas ou não em seções críticas é enunciada como se segue: *Uma tarefa só acessa um recurso compartilhado se sua prioridade ativa for maior que a prioridade teto (“ceiling”) de qualquer recurso já previamente bloqueado*. São excluídos dessa comparação recursos bloqueados pela tarefa requerente. Se S_l for o semáforo com maior prioridade teto entre todos os semáforos bloqueados, então uma tarefa T_i só entrará em sua seção crítica se sua prioridade dinâmica p_i for maior que o "ceiling" $C(S_l)$. Se $p_i \leq C(S_l)$ o acesso é negado a T_i.

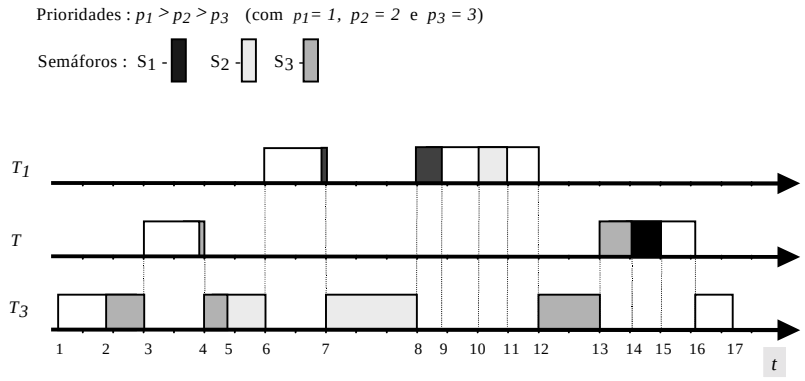
Quando nenhum recurso estiver bloqueado então o acesso ao primeiro recurso será sempre permitido. A conseqüência do uso do *Protocolo de Prioridade Teto* (PCP) é que uma tarefa mais prioritária só pode ser bloqueada por tarefas menos prioritárias uma só vez por ativação [SRL90].

O exemplo apresentado em [Kop92c] é reproduzido aqui no sentido de ilustrar o efeito do PCP sobre um conjunto de tarefas. As tarefas T_1, T_2 e T_3 cujas evoluções são apresentadas na figura 2.12, acessam em exclusão mútua os recursos R_1, R_2 e R_3. A prioridade teto de cada semáforo é definido segundo o compartilhamento dos recursos indicado na figura: $C(S_1)=1$, $C(S_2)=1$ e $C(S_3)=2$. Os eventos na evolução das tarefas, sinalizados na figura, são também descritos na própria figura 2.12. Nesse exemplo, a tarefa T_1, em cada ativação, é bloqueada no máximo uma vez por uma seção crítica de uma tarefa menos prioritário.

Além dos bloqueios diretos e por herança, o PCP introduz uma outra forma de bloqueio conhecida como bloqueio de *"ceiling"* onde uma tarefa fica bloqueada porque não possui prioridade dinâmica superior a maior *prioridade teto* dentre os recursos ocupados. Esse bloqueio é necessário para evitar as cadeias de bloqueios e os *"deadlocks"* [But97]. Na figura 2.12 a tarefa T_1 sofre um bloqueio de *"ceiling"* no tempo do evento 7.

O *"Immediate Priority Ceiling Protocol*” (IPCP) é uma versão do PCP cuja finalidade principal é a de apresentar um melhor desempenho. A herança de prioridade no IPCP deixa de se dar quando a seção crítica bloqueia a tarefa mais prioritária. A tarefa menos prioritária tem sua prioridade ativa elevada, assumindo logo no início da seção crítica a prioridade teto do recurso acessado [BuW97]. Uma conseqüência dessa mudança é que uma tarefa pode sofrer bloqueio somente no início de sua execução. Uma vez que comece a executar, a tarefa terá todos os recursos que necessite para o seu processamento. Sua execução só poderá ser postergada pelas interferências de tarefas mais prioritárias. O IPCP é mais fácil de se implementar que o PCP e envolve menos troca de contextos de tarefas. O *"Priority Protect Protocol"* apresentado nas

especificações POSIX é baseado no IPCP.



Eventos	*Descrição*
1	T_3 começa sua execução
2	T_3 entra em seção crítica (fecha o Semáforo S_3)
3	T_2 inicia seu processamento interrompendo T_3 (preempção de T_3).
4	T_2 sofre bloqueio direto: S_3 "fechado" por T_3 que reassume e herda prioridade de T_2.
5	T_3 entra na seção crítica aninhada ao fechar semáforo S_2
6	T_1 inicia e interrompe T_3. T_1 é mais prioritária que T_3 com a prioridade herdada de T_2.
7	T_1 tenta fechar semáforo S_1, é bloqueado por "ceiling"; possui prioridade igual ao maior "ceiling" de semáforo já fechado pelas outras tarefas ($C(S_2)=1$).
8	T_3 libera semáforo S_2. T_1 é reativado e interrompe T_3. T_1 entra em S_1.
9	T_1 libera S_1.
10	T_1 fecha Semáforo S_2.
11	T_1 libera S_2.
12	T_1 termina e T_3 reassume na prioridade herdada de T_2.
13	T_3 libera S_3 retornando a sua prioridade estática. T_2 interrompe T_3 e fecha S_3.
14,15,16,17	T_2 libera S_3 ; fecha e libera S_1; e completa. T_3 reassume e completa.

Figura 2.12: Exemplo de uso do PCP

- **Extensões de Testes de Escalonabilidade Tomando como Base o PCP**

Os testes de escalonabilidade mostrados anteriormente – quando da aplicação do PHP sobre um conjunto de tarefas – continuam válidos na aplicação do Protocolo de Prioridade Teto (PCP). A diferença está no valor limite de bloqueio máximo B_i que pode experimentar uma tarefa T_i. No PCP esse valor limite corresponde a duração da maior seção crítica de tarefas menos prioritárias que podem bloquear T_i. Logo, nas equações [16], [17] e [18], quando se considera o uso do PCP, B_i assume sempre o

valor da maior seção crítica que bloqueia T_i.

Uma seção crítica pertencente a uma tarefa T_j, guardada pelo semáforo S_k e de duração $D_{j,k}$ pode bloquear por *"ceiling"* uma tarefa mais prioritária T_i se e somente se [SRL90]: $p_i > p_j$ e $C(S_k) \geq p_i$. O máximo bloqueio B_i que T_i pode sofrer é dado pela duração da maior seção crítica que pode bloquear por *"ceiling"* essa tarefa ([AuB90], [But97]):

$$B_i = \max_{j,k} \left\{ D_{j,k} \mid \left(p_j < p_i\right) \wedge \left(C(S_k) \geq p_i\right) \right\}.$$

tarefas	S_1	S_2	S_3
tarefa T_1	1	1	0
tarefa T_2	1	0	1
tarefa T_3	0	4	8

Tabela 2.6

Para ilustrar o cálculo de B_i sob o uso do PCP, considere a tabela 2.6 que descreve as durações de seções críticas ($D_{j,k}$) a partir de condições do problema apresentado na figura 2.12. De acordo com a equação acima os bloqueios máximos por tarefas são dados por:

$$B_1 = max\ (1,4) = 4$$
$$B_2 = max\ (8) = 8$$
$$B_3 = 0$$

2.7 Tarefas Dependentes: Relações de Precedência

Em muitas aplicações, alguns processamentos não podem ser executados em ordens arbitrárias mas sim, em ordens previamente definidas, o que determina o surgimento de *relações de precedência* entre tarefas do conjunto. As escalas produzidas devem refletir as ordens parciais definidas através destas relações.

Alguns autores preferem expressar as relações de precedência entre tarefas com o uso de *"offsets"* [Aud93]. Neste caso, a tarefa sucessora de uma precedência é liberada pela passagem do tempo (liberada por valor de tempo correspondente ao *"offset"* que garante o tempo de resposta da predecessora). O uso de *"offsets"* pode representar em sub-utilização de recursos, uma vez que a sucessora é sempre liberada por tempo em situação de pior caso: um *"offset"* é calculado para a pior situação possível em termos

de tempo de resposta da tarefa predecessora.

Relações de precedência podem ser definidas através das necessidades de comunicação e sincronização entre as tarefas. As tarefas tipicamente, recebem mensagens, executam seus processamentos e por fim, enviam seus resultados ou sinais de sincronização na forma de mensagens. Com isto, a liberação de uma tarefa sucessora pode se dar por meio de mensagem [TBW94]. Uma conseqüência direta destas liberações por mensagem é a existência de *"release jitters"* nas liberações de tarefas sucessoras.

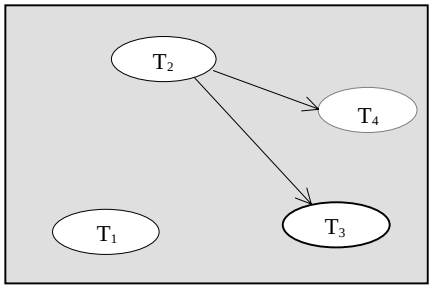
Conforme a técnica usada para a liberação de sucessoras, seja por passagem de tempo ou por mensagem, as relações de precedência são representadas nas análises de escalonabilidade, por valores de *"offsets"* ou de *"jitters"*. Em ambos os casos, esses valores devem garantir o pior caso de tempo de resposta da tarefa predecessora na relação de precedência. Ou seja, em termos de análise de escalonabilidade, os dois métodos são equivalentes pois tanto *"offsets"* como *"jitters"* devem assumir valores que garantam a execução da predecessora no seu pior caso de tempo de resposta, antes da liberação da sucessora. A diferença está em tempo de execução; enquanto, a liberação por passagem de tempo é uma técnica estática onde o *"offset"* é definido previamente, impondo sempre o pior caso de liberação, a liberação por mensagem é dinâmica e o pior caso de liberação eventualmente pode acontecer.

O conceito de *atividade* é usado como a entidade encapsuladora de *tarefas* que se comunicam e/ou se sincronizam. Cada atividade é representada por um grafo orientado acíclico onde os nodos representam tarefas e os arcos identificam as relações de precedência. As atividades são ditas *síncronas* (**"***loosely synchronous activity*") quando as tarefas liberam suas sucessoras pelo envio de mensagens; no outro caso, onde a liberação envolve *"offsets"*, as atividades são identificadas como *assíncronas* ("*asynchronous activity*") [BNT93]. Um exemplo de atividades assíncronas pode ser encontrada no sistema MARS [Kop97], onde as entidades encapsuladoras das tarefas dependentes são identificadas como *transações* e apresentam suas tarefas liberadas por passagem de tempo no sentido de implementar as relações de precedência.

Na seqüência deste item, concentramos nossas descrições em atividades síncronas (liberações por mensagem) para esquemas de prioridades fixas e nas relações de precedência que, para efeito de análise, são representadas como *"jitters"*. Nessas condições, o modelo de tarefas assume carga estática e, portanto, uma aplicação é constituída por *atividades periódicas*. Cada atividade periódica A_i corresponde a uma seqüência infinita de ativações ocorrendo em intervalos regulares de tempo P_i (período da atividade). Uma atividade A_i é caracterizada por um *"deadline"* D_i: limite máximo associado à conclusão de todas as suas tarefas. As tarefas de uma mesma atividade possuem os mesmos tempos de chegada, porém suas liberações dependem dos tempos de resposta de suas predecessoras.

A figura 2.13 mostra duas atividades: a primeira constituída de apenas uma tarefa T_1 e uma segunda, formada pelas tarefas T_2, T_3 e T_4. As relações de precedência nesta

segunda atividade implicam na ordem $T_2 \rightarrow T_3 \rightarrow T_4$ [2].



tarefas	J_i	C_i	P_i	D_i
T_1	1	10	40	40
T_2	3	10	80	25
T_3	-	5	80	40
T_4	-	10	80	80

Figura 2.13: Uma aplicação constituída por duas atividades

Quando se considera um grafo de precedências (uma atividade), a maneira natural de se atribuir prioridades é seguindo as relações do grafo com um decréscimo nas prioridades das tarefas envolvidas, ou seja, as prioridades são decrescentes ao longo do grafo de precedências seguindo as orientações dos arcos. Este tipo de atribuição, respeitando as relações de precedência, se aproxima da política *"Deadline Monotonic"*.

As comunicações usando variáveis compartilhadas – conforme visto no item 2.6 – podem levar a inversões de prioridades (bloqueios). Se a atribuição de prioridades é feita segundo as orientações dos grafos e as liberações de tarefa obedecem às relações de precedência, diminuem as possibilidades de bloqueios e inversões de prioridades, uma vez que as tarefas mais prioritárias são liberadas antes nas relações de precedência.

Considere como exemplo o conjunto de tarefas ilustrado na figura 2.13. Tomando as relações de precedência e as restrições temporais indicadas na figura, queremos verificar a escalonabilidade do conjunto. A atribuição de prioridades é feita segundo as orientações dos grafos; os índices das tarefas representam as suas respectivas prioridades (se T_i é mais prioritária que T_j então $i < j$).

O modelo introduzido coloca as atividades como síncronas o que implica em tratar precedências como *"release jitters"*. Como as tarefas possuem *"deadlines"* relativos menores que seus respectivos períodos, a verificação de escalonabilidade pode ser feita usando as equações [9] e [10] do item 2.5, onde os tempos de resposta são obtidos a partir de :

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil \cdot C_j \qquad e \qquad R_i = W_i + J_i ,$$

[2] As precedências entre tarefas nas atividades podem ser expressas pela relação de ordem parcial "$\rightarrow$", definida sobre o conjunto de tarefas. Se Ti precede uma outra tarefa Tj (ou seja, Tj é sucessora de Ti), esta relação é representada por Ti$\rightarrow$Tj, indicando que Tj não pode iniciar sua execução antes de Ti terminar. A relação "$\rightarrow$" é transitiva.

onde a escalonabilidade é verificada por : $\forall i \;\; R_i \leq D_i$.

No cálculo destes tempos devem ser consideradas as relações de precedência no conjunto de tarefas. O teste acima permite a consideração de precedências na forma de *"jitters"*. O valor de *"jitter"* de uma tarefa é determinado a partir do tempo de resposta máximo da sua predecessora (pior situação de liberação).

O conjunto de tarefas com relações de precedências, passa a ser tomado como um conjunto de tarefas independentes com *"jitters"* associados. Mas as interferências assim calculadas, a partir de tarefas com *"jitters"* associados e tomadas como independentes, resultam em tempos de respostas extremamente grandes e muitas vezes, irreais. Na verdade, tarefas sujeitas a precedências determinam cenários mais restritos de interferência e bem distante do *instante crítico* [OlF97]. Por exemplo, na figura 2.13, a tarefa T_2 embora mais prioritária, não interfere com T_3 e T_4 porque ambas são liberadas após a sua conclusão; a influência de T_2 sobre estas duas tarefas se dá só na forma de *"jitter"*.

No caso da figura 2.13, T_1 é a mais prioritária e não sofre interferência de outras tarefas. O seu tempo de resposta é dado por seu tempo de computação acrescentado pelo *"jitter"* que sofre: $R_1 = C_1 + J_1 = 11$. A tarefa T_2 sofre interferência só da tarefa T_1 e o seu tempo de resposta máximo é calculado facilmente a partir das equações [9] e [10]:

$$W_2^0 = C_2 = 10$$
$$W_2^1 = 10 + \left\lceil \frac{10+1}{40} \right\rceil \times 10 = 20$$
$$W_2^2 = 10 + \left\lceil \frac{20+1}{40} \right\rceil \times 10 = 20$$

Com $W_2 = 20$ e tomando $J_2 = 3$, o valor do tempo de resposta é dado por $R_2 = 23$. A tarefa T_3, por sua vez, sofre interferências de T_1 e um *"jitter"* porque sua liberação depende da conclusão de T_2 ($J_3 = R_2$):

$$W_3^0 = C_3 = 5$$
$$W_3^1 = 5 + \left\lceil \frac{5+1}{40} \right\rceil \times 10 = 15$$
$$W_3^2 = 5 + \left\lceil \frac{15+1}{40} \right\rceil \times 10 = 15$$

O tempo de resposta de T_3 é dado por: $R_3 = W_3 + J_3 = 38$. A tarefa T_4, por sua vez, sofre interferências de T_1 e T_3 e um *"jitter"* de T_2 ($J_4 = R_2$):

$$W_4^0 = C_4 = 10$$
$$W_4^1 = 10 + \left\lceil \frac{10+1}{40} \right\rceil \times 10 + \left\lceil \frac{10+23}{80} \right\rceil \times 5 = 25$$

$$W_4^2 = 10 + \left\lceil \frac{25+1}{40} \right\rceil \times 10 + \left\lceil \frac{25+23}{80} \right\rceil \times 5 = 25$$

A tarefa T_4 tem o seu pior tempo de resposta portanto em 48 ($R_4=W_4+J_4$). Se compararmos os tempos de resposta encontrados com os *"deadlines'* relativos das respectivas tarefas na figura 2.13, verificamos que as tarefas são escalonáveis..

No modelo de tarefas apresentado, as relações de precedência são implementadas a partir de ativações por mensagens. Os tempos em comunicações locais nos modelos de tarefas ideais são desconsiderados; em situações reais, tempos não desprezíveis podem ser adicionados aos tempos de computação das tarefas predecessoras (emissoras de mensagens), aproximando então o modelos reais de premissas de tempos nulos em comunicações locais.

Os cálculos de tempos de resposta em ambientes distribuídos, envolvendo precedências, não é muito explorado na literatura [Fid98]. As atividades nestes ambientes se estendem por vários nós (vários domínios de escalonamento local) o que implica em precedências remotas. Estas situações exigem considerações especiais. Soluções para problemas distribuídos devem se basear na assim chamada "*holistic schedulability analisys*" para sistemas de tempo real distribuídos [TiC94]: O *"release jitter"* de uma mensagem depende do pior caso de tempo de resposta da tarefa emissora. O pior caso de tempo de resposta de uma tarefa receptora depende do tempo de resposta de suas mensagens.

2.8 Escalonamento de Tarefas Aperiódicas

Todas as técnicas de escalonamento apresentadas até este item eram dirigidas para modelos de tarefas periódicas. Mas aplicações de tempo real, de um modo geral, envolvem tanto tarefas periódicas como aperiódicas. Examinamos neste item o escalonamento de tarefas aperiódicas em abordagens mistas, envolvendo tarefas críticas e não críticas. As tarefas periódicas são assumidas como críticas, necessitando de garantias em tempo de projeto para condições de pior caso. As tarefas aperiódicas podem envolver diferentes requisitos temporais: críticos, não críticos ou ainda sem requisitos temporais.

As aperiódicas apresentando um mínimo intervalo entre suas ativações e um *"deadline hard"* são identificadas como tarefas esporádicas, possuindo um comportamento temporal determinista – o que facilita, portanto, a obtenção de garantias em tempo de projeto. As tarefas aperiódicas que não possuem seus tempos de chegada conhecidos e também não se caracterizam por um intervalo mínimo entre suas ativações, definem o que se pode chamar de uma carga computacional dinâmica. Com estas últimas tarefas é possível a obtenção de garantias dinâmicas ou, ainda, usar técnicas de melhor esforço no sentido de executá-las segundo as disponibilidades do

processador em tempo de execução. As aperiódicas que apresentam *"deadlines hard"* e necessitam de garantias dinâmicas em seus escalonamentos são chamadas de tarefas aperiódicas *"firm"*. As tarefas aperiódicas com requisitos não críticos (*"deadline soft"*) e as sem requisitos temporais (aplicações não de tempo real) necessitam apenas de bons tempos de resposta.

Num quadro misto, uma questão que pode ser colocada está ligada ao tipo de política adequado para tarefas periódicas e que possa ser estendido para carga dinâmica: *são as políticas de prioridade fixa (RM, DM e etc.) ou políticas de prioridade dinâmica (EDF) as mais apropriadas?* As políticas baseadas em prioridade fixa foram sempre as preferidas para esquemas mistos de escalonamento. Embora apresentem melhor fator de utilização, se comparado com esquemas de prioridade fixa, as políticas de prioridade dinâmica como o EDF eram consideradas por alguns autores até pouco tempo como instáveis para tratar com carga dinâmica [SSL89]. Nos últimos anos, a direção dos trabalhos tem mudado e o EDF tem sido também alvo de extensões para escalonamentos mistos. Os algoritmos dinâmicos apresentam os mais altos limites de escalonabilidade o que permite uma maior utilização do processador o que, por sua vez, aumenta a capacidade de processamento da carga aperiódica.

As sobras de processador nas escalas são importantes para o escalonamento de tarefas aperiódicas em modelos híbridos. Existem dois tipos de abordagens para a determinação de sobras de processador: as soluções baseadas em *servidores* [SSL89] e as baseadas em *tomadas de folgas* ("*slack stealing*") [DtB93], [LeR92]. Neste texto são apresentadas unicamente técnicas de escalonamento para tarefas aperiódicas baseadas no conceito de *servidor*. Os escalonamentos híbridos neste capítulo são construídos com políticas de prioridade fixa [SSL89]. No *Anexo A* são apresentados escalonamentos mistos usando políticas de prioridade dinâmica [SpB96].

2.8.1 Servidores de Prioridade Fixa [LSS87, SSL89]

As técnicas examinadas nesse item são para políticas baseadas em prioridades fixas, mais precisamente, o *"Rate Monotonic"*. As sobras nas escalas de carga periódica, são determinadas estaticamente, em tempo de projeto, e posteriormente, em tempo de execução, são atribuídas ao processamento aperiódico usando o conceito de *servidor*.

- **Servidor de *"Background"***

Este servidor é extremamente simples. A idéia central corresponde em atender as requisições aperiódicas quando a fila de prontos envolvendo tarefas periódicas está vazia, ou seja, se tarefas periódicas não estão se executando ou pendentes, o processador é entregue para a carga aperiódica.

A determinação de prioridades nesta abordagem é feita atribuindo - segundo o RM - as prioridades mais altas para as tarefas periódicas. As prioridades mais baixas são destinadas para as tarefas aperiódicas. Como conseqüência, o *"Background Server"*

(BS) apresenta tempos de resposta muito altos para cargas aperiódicas. Se a carga envolvendo as tarefas periódicas é alta, então a utilização deixada para o serviço de *"Background*" é baixa ou não freqüente.

tarefas	C_i	P_i	D_i	p_i
tarefa periódica A	4	10	10	1
tarefa periódica B	8	20	20	2
tarefa aperiódica C	1	-	-	3
tarefa aperiodica D	1	-	-	3

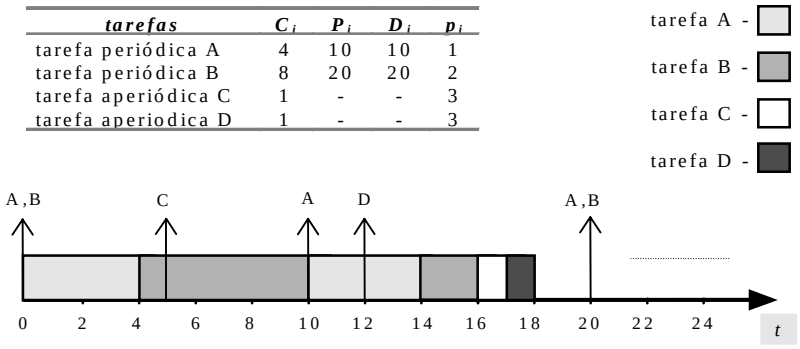


Figura 2.14: Servidora de "*Background*"

A figura 2.14 ilustra um exemplo introduzido em [SSL89] onde duas tarefas periódicas e duas requisições aperiódicas são executadas usando uma atribuição de prioridades RM. Com base no teste do RM, a carga periódica tem garantia em tempo de projeto pois a sua utilização não passa o limite de 0,828 (equação [2], item 2.4). As requisições C e D são executadas no fim da escala da figura 2.16, depois que a carga periódica foi completada.

O BS é bastante simples na sua implementação, porém só é aplicável quando as requisições aperiódicas não são críticas e a carga periódica não é alta.

- ***"Polling Server"***

O esquema do *"Polling Server"* (PS) consiste na definição de uma tarefa periódica para atender a carga aperiódica [SSL89]. Um espaço é aberto periodicamente na escala para a execução da carga aperiódica, através da tarefa Servidora de *"Polling"*. A tarefa servidora possui um período P_{PS} e um tempo de computação C_{PS} e, como as outras tarefas da carga periódica do sistema, tem a sua prioridade atribuída segundo o *"Rate Monotonic"*. Em cada ativação, a tarefa servidora executa as requisições aperiódicas pendentes dentro do limite de sua *capacidade* C_{PS} – o tempo destinado para o atendimento de carga aperiódica em cada período da servidora.

Quando não houver requisições aperiódicas pendentes, a tarefa PS se suspende até a sua nova chegada, no próximo período. Neste caso, a sua capacidade C_{PS} é entregue para a execução de tarefas periódicas pendentes. Se um pedido aperiódico ocorre logo depois da suspensão da tarefa servidora, o pedido deve aguardar até o início do próximo período da tarefa PS.

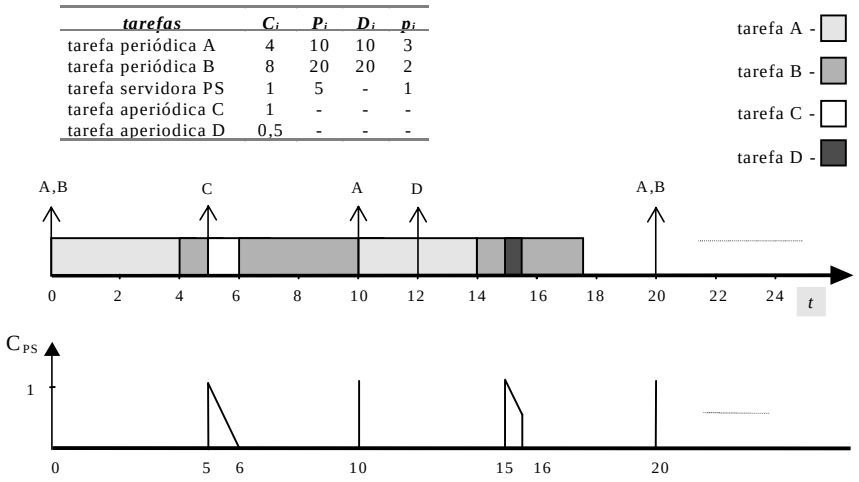


Figura 2.15: Algorítmo "Polling Server"

O mesmo exemplo usado com o BS é mostrado na figura 2.15 onde a carga aperiódica é escalonada segundo o algoritmo PS. Nesse caso, a tarefa servidora é criada com capacidade C_{PS} de uma unidade e o período P_{PS} de 5 unidades. Na ativação da servidora em *t=0* não existe carga aperiódica e a sua capacidade é entregue para a execução das tarefas periódicas. Em *t = 5*, a chegada de uma requisição aperiódica C coincide com a chegada da servidora PS. Com isto, a capacidade C_{PS} é consumida totalmente até *t* = 6. No período seguinte da servidora (*t = 10*), novamente não existe carga aperiódica pendente e a capacidade da servidora, que foi restaurada no seu máximo no início deste período, é entregue a carga periódica. A servidora, por não estar mais ativa, não atende a segunda requisição aperiódica D que chega em *t=12*. No início de seu período seguinte (em *t=15*), esta requisição é executada, consumindo a metade da capacidade da servidora, conforme mostra a figura.

A interferência da tarefa servidora sobre o conjunto de tarefas periódicas do sistema é no pior caso igual a interferência causada por uma tarefa com tempo de computação C_{PS} e período P_{PS} ou seja, dada a utilização do PS, a escalonabilidade do conjunto periódico é garantido por :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \frac{C_{PS}}{P_{PS}} \leq (n+1)\left(2^{1/(n+1)} - 1\right),$$

ou seja

$$U_P \leq (n+1)\left(2^{1/(n+1)} - 1\right) + U_S .$$

A abordagem do *"Polling Server"*, se comparada com a abordagem BS, melhora o tempo de resposta médio de tarefas aperiódicas. O PS porém não fornece serviço de

resposta imediato para processamentos aperiódicos. O tempo de resposta de requisições aperiódicas depende do período e da capacidade da tarefa servidora.

- ***"Deferrable Server"***

O *"Deferrable Server"* (DS) também é baseado na criação de uma tarefa periódica que no conjunto de tarefas da carga estática, recebe uma prioridade segundo uma atribuição RM. Ao contrário do PS, o DS conserva a sua capacidade – tempo destinado para o processamento aperiódico – mesmo quando não existir requisições durante a ativação da tarefa DS. Requisições não periódicas podem ser atendidas no nível de prioridade da tarefa servidora, enquanto a sua capacidade C_{DS} não se esgotar no período correspondente. No início de cada período da tarefa servidora, a sua capacidade processamento é restaurada.

Por preservar sua capacidade, a abordagem DS fornece melhores tempos de resposta para as tarefas aperiódicas que o *"Polling Server"*. Como a tarefa servidora usualmente executa na prioridade mais alta do conjunto periódico, se a capacidade for suficiente, o atendimento de requisições aperiódicas é imediato.

tarefas	C_i	P_i	D_i	p_i
tarefa periódica A	4	10	10	3
tarefa periódica B	8	20	20	2
tarefa servidora PS	1	5	-	1
tarefa aperiódica C	1	-	-	-
tarefa aperiodica D	0,5	-	-	-

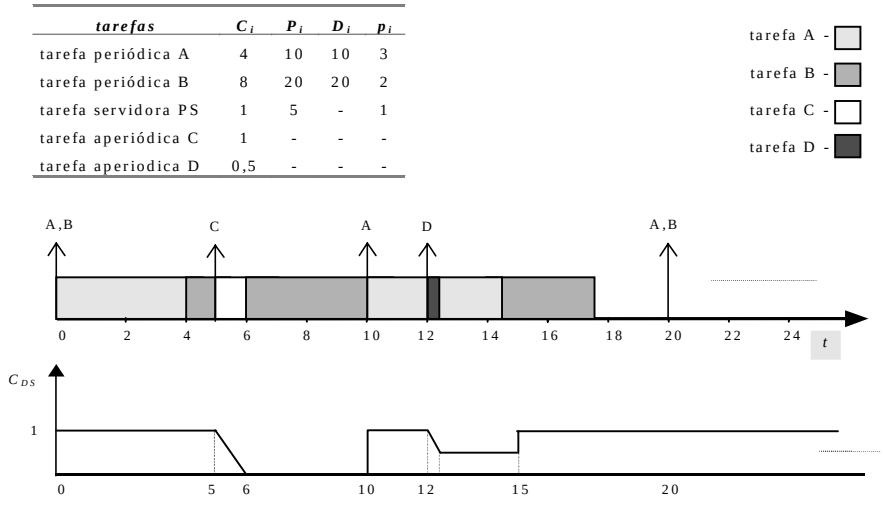


Figura 2.16: Algorítmo "Deferrable Server"

A figura 2.16 ilustra o uso do algoritmo DS no escalonamento da carga aperiódica com o mesmo exemplo introduzido em [SSL89]. Neste exemplo é criada uma tarefa servidora com capacidade $C_{DS}=1$ e de período $P_{DS} = 5$. Na ativação da servidora em $t=0$ não existe carga aperiódica e a sua capacidade é preservada durante todo o período P_{DS}. Em $t=5$, a chegada de uma requisição aperiódica C coincide com a chegada da servidora DS, o que determina o consumo total da capacidade da servidora até $t=6$

(figura 2.16). No período seguinte da servidora (*t=10*), a capacidade C_{DS} é novamente preenchida ao seu máximo. Este valor de capacidade se mantém até a chegada da requisição aperiódica D em *t=12* que então consome 0,5 da capacidade da servidora até *t=12,5*. No início do período seguinte da servidora (*t=15*), a sua capacidade volta ao seu valor máximo ($C_{DS}=1$). A figura 2.16 confirma sobre o mesmo exemplo usado nas técnicas anteriores, o melhor desempenho do servidor DS sobre os anteriores em termos de tempo de resposta e serviço de resposta imediata.

Quando usando a política RM, a influência da tarefa servidora DS sobre a utilização da carga periódica não pode ser determinada de maneira tão simples como no caso do PS. O comportamento da servidora com a prioridade mais alta, podendo se executar em qualquer ponto do seu período não é captada pelo teste do RM (equação [2], item 2.4). Nas condições do teste original do RM, a tarefa periódica de mais alta prioridade necessita executar em seu tempo de chegada; qualquer atraso pode prejudicar as tarefas menos prioritárias. Em [SSL89], derivada do ajuste no teste do RM para captar o comportamento singular da servidora DS, é apresentada uma relação entre a utilização da servidora e a utilização da carga periódica:

$$U_P \leq \ln\left(\frac{U_{DS}+2}{2U_{DS}+1}\right). \quad [19]$$

A equação [19] é válida somente para um muito grande número de tarefas periódicas no sistema.

- **Servidor Troca de prioridade (“*Priority Exchange Server*”)**

Uma outra técnica de escalonamento apresentada em [LSS87] e [SSL89] para o processamento de requisições aperiódicas em escalonamento híbrido é o *"Priority Exchange Server*" (PE). Diferentemente do DS, neste servidor, diante da ausência de requisições aperiódicas, a capacidade de processamento aperiódico C_{PE} (tempo de computação da tarefa servidora) é preservada executando trocas de prioridades da servidora com tarefas periódicas pendentes. Não será discutido neste texto o algoritmo do PE devido a pouca possibilidade de aplicação deste servidor ligada à complexidade do mecanismo de troca de prioridades. Os leitores interessados podem encontrar informações sobre este servidor nas indicações bibliográficas acima.

- **Servidor Esporádico**

O *"Sporadic Server"* (SS), a exemplo dos algoritmos DS e PE, é outra técnica introduzida em [SSL89] que apresenta bons tempos de resposta e de serviço imediato para requisições aperiódicas. Com características semelhantes a dos anteriores, foi introduzido para possibilitar a execução de tarefas aperiódicas com restrições críticas.

O SS cria uma tarefa periódica que atua em um só nível de prioridade para executar requisições aperiódicas. Para entender o funcionamento do algoritmo do SS é

necessário que se introduza alguns termos:

- p_s : corresponde ao nível de prioridade em execução no processador;
- p_i : é um dos níveis de prioridades do sistema.
- Intervalo Ativo : uma prioridade p_i é dita em um intervalo ativo quando $p_i \leq p_s$.
- Intervalo de Prioridade Desativada: uma prioridade p_i é dita desativada quando $p_i > p_s$.
- Tempo de Preenchimento RT_i : define o instante de tempo em que se dá a restauração da capacidade consumida durante o intervalo em que a prioridade p_i estava ativa.

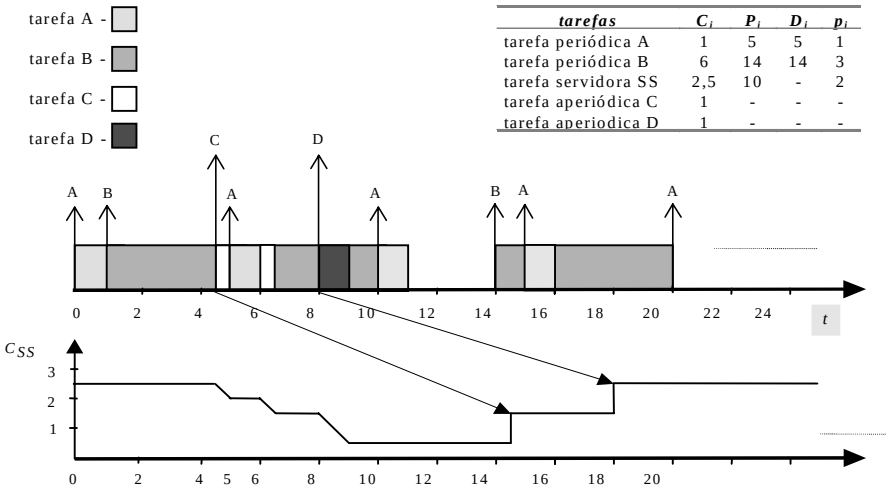
A tarefa servidora no SS preserva sempre a sua capacidade no nível em que foi projetada. Mas difere das outras abordagens anteriores na forma do preenchimento de sua capacidade:

- Se a servidora tem seu tempo computação (capacidade) consumido em um de seus períodos, o preenchimento correspondente ocorrerá no seu tempo de preenchimento (RT_i) que é determinado adicionando o valor do período da servidora ao tempo de início do intervalo onde p_i era ativo e ocorreu o consumo considerado.
- A quantidade a ser preenchida é igual a capacidade do servidor consumida no intervalo ativo.

A figura 2.17 apresenta um exemplo de um escalonamento híbrido com um servidor esporádico possuindo prioridade média no conjunto de tarefas periódicas. Neste exemplo também apresentado em [SSL89], a tarefa servidora SS é definida com capacidade $C_{SS}=2,5$ e período $P_{SS}=10$. Em $t=0$, a tarefa A (a mais prioritária) começa a executar. A prioridade p_s em execução ($p_s = p_A$) é então maior que a prioridade da servidora SS (p_{SS}); o que define o primeiro intervalo ativo da servidora SS nesta execução de A ($p_{SS} < p_A$). Neste intervalo não ocorre consumo de capacidade C_{SS} devido a ausência de requisições aperiódicas.

Em $t=4,5$ uma requisição aperiódica C chega e como a tarefa SS é mais prioritária que B (tarefa em execução), ocorre a preempção da tarefa periódica. O consumo da capacidade da servidora por parte de C vai até $t = 5$ quando, pela chegada da tarefa periódica A, ocorre a interrupção da tarefa aperiódica C. Pelo RM a tarefa A é a mais prioritária. Concluída esta ativação de A, a tarefa C reassume. Em $t = 6,5$ o processamento aperiódico C é concluído. O tempo de preenchimento (RT_i), referente ao consumo de capacidade por parte da requisição C, é programado considerando o intervalo ativo correspondente ($p_{SS} \leq p_s$) que, neste caso, inicia com a chegada da

requisição aperiódica (*t = 4,5*). Portanto, como pode ser visto na figura 2.17, o preenchimento da capacidade consumida neste intervalo ocorre em *t=14,5* ($RT_i=t_a+P_{SS}$). A preempção de C pela tarefa A em *t = 5* não define dois intervalos ativos da servidora para as duas partes de C da figura 2.17. Na verdade, um mesmo intervalo ativo se mantém durante as execuções de C e A porque, entre os tempos *t=4,5* e *t = 6,5*, a condição $p_{SS} \leq p_s$ se mantém como válida.



tarefas	C_i	P_i	D_i	p_i
tarefa periódica A	1	5	5	1
tarefa periódica B	6	14	14	3
tarefa servidora SS	2,5	10	-	2
tarefa aperiódica C	1	-	-	-
tarefa aperiodica D	1	-	-	-

Figura 2.17: Algorítmo "Sporadic Server"

Uma outra requisição aperiódica (tarefa D) chega em outro intervalo ativo da servidora SS. A requisição D também interrompe a tarefa B e consome uma unidade de C_{SS}. O tempo de início do intervalo ativo de D coincide com a sua chegada e, portanto, o tempo de preenchimento (RT_i) correspondente deve ocorrer em *t=18*.

A tarefa servidora SS não apresenta um comportamento convencional de tarefa periódica, uma vez que, a capacidade desta servidora é preservada no mesmo nível como o DS e a sua execução é postergada até a ocorrência de uma requisição aperiódica. Porém em [SSL89], é provado que a técnica de preenchimento da capacidade da servidora compensa este comportamento não convencional, permitindo que, em termos de análise de escalonabilidade, esta tarefa possa assumir um comportamento periódico. Ou seja, a servidora SS pode ser substituída no teste do RM ([2] no item 2.4) por uma tarefa periódica com período P_{SS} e tempo de computação C_{SS}. A limitação que a servidora SS impõe sobre uma carga periódica é dada por :

$$U_P \leq \ln\left(\frac{2}{U_{SS}+1}\right). \qquad [20]$$

A equação [20] é idêntica à obtida para o servidor PE e também só é válida para um número muito grande de tarefas periódicas no sistema.

A tabela 2.7 mostra um exemplo de comparação das utilizações dos servidores DS, PE e SS quando envolvidos com uma mesma carga periódica ([SSL89]). O algoritmo SS possui a simplicidade do DS e a vantagem da maior capacidade do PE para o processamento de requisições aperiódicas. Porém, diferente destes outros algoritmos, o SS pode também ser usado na garantia em tempo de projeto.

tarefas	C_i	P_i	U_i (%)
tarefa 1	2	10	20,0
tarefa 2	6	14	42,9
servidor DS	1,00	5	20,0
servidor PE	1,33	5	26,7
servidor SS	1,33	5	26,7

Tabela 2.7: Utilização dos servidores DS, PE e SS

Na verdade, o SS foi introduzido com o objetivo de garantir a execução de tarefas esporádicas – um caso especial de aperiódicas onde existe um limite conhecido como o mínimo intervalo entre ativações (min_i). Os *"deadlines"* associados a estas tarefas são críticos (*"hard"*) e, portanto, precisam de uma garantia em tempo de projeto. Esta garantia pode ser obtida criando uma servidora SS para tratar exclusivamente uma tarefa esporádica nas suas diversas ativações. A servidora SS assume os *"deadlines"* das requisições da tarefa associada. O período P_{SS}, por sua vez, deve ser no máximo igual ao intervalo mínimo entre ativações da tarefa esporádica (min_i). Esta tarefa servidora conserva a capacidade de processamento aperiódico no seu nível de prioridade até a ocorrência de uma requisição esporádica. A capacidade C_{SS} da servidora deve ser suficiente para atender as necessidades de tempo de computação da tarefa esporádica associada, em cada uma de suas ativações. Nestas condições, os *"deadlines"* críticos podem ser garantidos em tempo de projeto.

O algoritmo SS pode ser usado para garantir tarefas esporádicas apresentando *"deadlines"* relativos (D_i) iguais ou menores que os seus respectivos intervalos mínimos entre ativações (min_i). Nos casos onde os *"deadlines"* relativos são iguais aos respectivos intervalos mínimos, as prioridades da tarefa servidora SS e da carga periódica são determinadas seguindo uma atribuição RM e as escalas são produzidas usando os mesmos algoritmos citados acima.

Para os casos onde tarefas esporádicas apresentam $D_i < min_i$ a atribuição RM não pode ser usada; é necessária uma outra atribuição de prioridades que não seja mais baseada na freqüência de chegada das tarefas periódicas. A figura 2.18 ilustra um exemplo de uma escala com atribuição RM onde ocorre uma sobrecarga com perda de *"deadline"* da tarefa aperiódica C em $t=10$. Neste exemplo, a servidora SS que possui o *"deadline"* relativo mais restritivo ($D_{SS}=10$) apresenta o maior período ($P_{SS}=32$) e portanto a menor prioridade segundo o RM.

tarefas	C_i	P_i	D_i	Min_i	p_i
tarefa periódica A	4	12	12	-	1
tarefa periódica B	4	20	20	-	2
tarefa servidora SS	8	32	10	-	3
tarefa aperiódica C	8	-	10	32	-

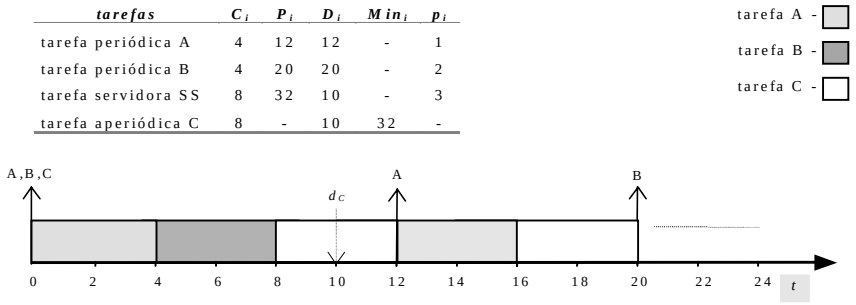
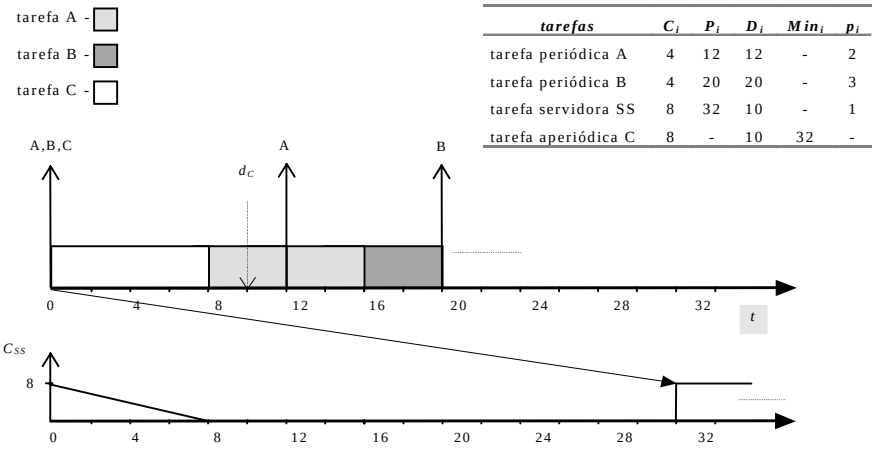


Figura 2.18: Servidor SS e RM usados em carga aperiódica com $D_i = Min_i$

Nos casos de tarefas esporádicas com $D_i<min_i$, são necessárias políticas que sejam dirigidas por *"deadlines"*, permitindo então a atribuição do mais alto nível de prioridade à servidora SS associada. A política de atribuição estática *"Deadline Monotonic"* é a mais apropriada nestes casos. Na figura 2.19 uma escala é mostrada com o mesmo conjunto de tarefas sujeito a uma atribuição DM de prioridades e onde todos os *"deadlines"* são respeitados. A tarefa C se executa em *t=0* (a servidora SS é a tarefa mais prioritária) e, o preenchimento da capacidade consumida correspondente ocorre em *t=32*, portanto, antes de esgotado o intervalo mínimo entre ativações de C (min_C).



tarefas	C_i	P_i	D_i	Min_i	p_i
tarefa periódica A	4	12	12	-	2
tarefa periódica B	4	20	20	-	3
tarefa servidora SS	8	32	10	-	1
tarefa aperiódica C	8	-	10	32	-

Figura 2.19: Servidor SS e DM usados em carga aperiódica com $D_i < Min_i$

2.8.2 Considerações sobre as Técnicas de Servidores

Algumas das premissas assumidas para os servidores seguiram modelos de tarefas originais dos algoritmos de escalonamento usados, mas isto não limita o uso destas técnicas de servidores. Os algoritmos de servidores apresentados neste texto podem ser usados em modelos com tarefas periódicas possuindo *"deadlines"* relativos arbitrários e com recursos compartilhados. Neste caso a análise de escalonabilidade deve levar em consideração as particularidades do modelo de tarefas usado.

Neste texto, as requisições aperiódicas foram apresentadas como processamentos sem prazos (*"deadlines"*), escalonadas segundo abordagens de melhor esforço usando políticas FIFO. Tarefas aperiódicas podem possuir restrições temporais e serem ordenadas segundo estas restrições com políticas diferentes das que conduzem a ordenação das periódicas no escalonamento híbrido. As tarefas aperiódicas necessitando a cada ativação de uma garantia dinâmica são identificadas como *tarefas firmes* (item 2.8). Neste caso, um teste de aceitação é necessário para verificar a escalonabilidade da tarefa aperiódica recém chegada junto com as tarefas previamente garantidas. Se o teste falha a tarefa aperiódica é descartada. Em [But97] é discutido essa verificação de tarefas firmes. Os algoritmos PS, PE e DS são apropriados na verificação dinâmica da escalonabilidade de tarefas aperiódicas firmes. O servidor SS, por sua vez, permite o tratamento de tarefas esporádicas que necessitam de garantias em tempo de projeto para os seus *"deadlines hard"*.

2.9 Conclusão

Em sistemas onde as noções de tempo e de concorrência são tratadas explicitamente, conceitos e técnicas de escalonamentos formam o ponto central na previsibilidade do comportamento de sistemas de tempo real. Esse capítulo se concentrou sobre técnicas para escalonamentos dirigidos a prioridades. Essa escolha na abordagem de escalonamento é porque a mesma cobre diversos aspectos de possíveis comportamentos temporais em aplicações de tempo real e, também, devido a importância da literatura disponível.

Vários *problemas de escalonamento* – que podem ser vistos como extensões aos problemas propostos em [LiL73] – foram examinados neste capítulo. Particularmente, foram apresentados escalonamentos de tarefas periódicas com *"deadlines"* arbitrários, o compartilhamento de recursos e a implementação de relações de precedência. As tarefas aperiódicas são escalonadas usando escalonamentos híbridos baseados no conceito de *servidor*. Todos estes problemas foram discutidos usando atribuições de prioridades fixas neste capítulo. Estes mesmos problemas são revistos com políticas de prioridade dinâmica no *Anexo A*. Escalonamentos com atribuições dinâmicas – como os definidos pelo EDF, embora determinem uma maior utilização apresentam sempre uma complexidade maior em tempo de execução.

A grande difusão de suportes (núcleos, sistemas operacionais), na forma de produtos, que baseiam seus escalonamentos em mecanismos dirigidos a prioridade é sem dúvida uma forte justificativa para o uso das técnicas apresentadas neste capítulo em problemas práticos. Alguns dos algoritmos apresentados nesse capítulo são recomendados por entidades de padronização como a POSIX e a OMG (*"Object Management Group"* [OMG98]).

Leituras complementares recomendadas referente ao assunto tratado neste capítulo são encontradas em: [AuB90], [Bak91], [Fid98], [RaS94], [SRL90], [SSL89], [Spu96], [TBW94].