ANEXO A

Extensões em Esquemas de Prioridade Dinâmica

A.1 Testes para Escalonamentos com Prioridades Dinâmicas

A análise de escalonabilidade do EDF baseada no conceito de utilização, embora corresponda a um teste exato, é limitada na sua aplicação pela simplicidade do modelo de tarefas de sua definição. Os testes que se seguem apresentam como finalidade a extensão do uso do EDF para modelos mais complexos.

A.1.1 Deadline igual ao período

Quando considerados uma política baseada em deadlines e um conjunto de tarefas periódicas com deadlines relativos iguais aos seus respectivos períodos, qualquer análise em uma janela de tempo deve levar em conta a carga computacional representada pelas ativações dessas tarefas que devem ser executadas dentro dessa janela. Ou seja, essa carga corresponde às ocorrências de tarefas que apresentam deadlines absolutos menores ou iguais ao limite superior desse intervalo de tempo. Por exemplo, a necessidade de processador que uma tarefa T_i possui em um intervalo $[t,t+l]$ para que suas ativações completem antes ou em $t+l$ é dado por $\lfloor l/P_i \rfloor C_i$.

Para um conjunto de n tarefas periódicas, as instâncias liberadas e que completam no intervalo $[0, t]$ correspondem à *demanda de processador C(t)* fornecido por:

$$C(t) = \sum_{P_i \le t}^{n} \left\lfloor \frac{t}{P_i} \right\rfloor C_i .$$

A escalonabilidade de um conjunto de tarefas executadas sob o EDF (política dirigida a deadlines), é garantida se e somente se, em qualquer intervalo $[0, t]$, a demanda de processador ($C(t)$) das instâncias que devem se completar nesse intervalo é menor ou igual ao tempo disponível t [JeS93]:

$$t \geq \sum_{P_i \leq t} \left\lfloor \frac{t}{P_i} \right\rfloor C_i . \qquad [A1]$$

Para aplicar o teste [A1], a inspeção de valores de *t* pode ser limitada aos tempos de liberação das tarefas dentro do *hiperperíodo H* do conjunto de tarefas considerado[1]. Mas isto ainda pode representar um grande número de valores de *t* a serem verificados.

O conceito de *"busy period"* pode ser útil na determinação de um conjunto mais reduzido de valores de *t*. Retomando então um *período ocupado* como um intervalo de tempo com execuções contínuas de tarefas e considerando o instante crítico em *t=0*, o pior caso de execução de uma tarefa T_i ocorre no primeiro *i-busy period* que inicia na origem (*t=0*) e termina com a execução da instância considerada. A idéia é que o *completion time* de uma instância de T_i com deadline *d* deva estar no fim do *i-busy period* no qual se executarão instâncias de outras tarefas que tenham deadlines menores ou iguais a *d*.

O intervalo de tempo *L* que ocorre entre *t=0* e o primeiro *"idle time"* do processador, é assumido como o maior *"busy period"* no sistema. O valor de *L* é obtido a partir de [Spu96]:

$$\begin{cases} L^{(0)} = \sum_{i=1}^{n} C_i , \\ L^{(m+1)} = W(L^{(m)}), \end{cases} \qquad [A2]$$

onde *W(t)* corresponde a carga cumulativa no intervalo [*0, t*], ou seja a soma dos tempos de computação de todas as instâncias que chegaram antes do tempo *t* :

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{P_i} \right\rceil C_i . \qquad [A3]$$

Quando $L^{(m+1)} = L^{(m)}$ os cálculos em [A2] estão terminados[2].

Os valores de *t* que devem servir na inspeção de [A1] são dados então por $t \in \mathfrak{I}$, onde:

---

[1] *hiperperíodo* de um conjunto de tarefas periódicas corresponde ao mínimo múltiplo comum dos períodos das tarefas do conjunto.

[2] $L^{(m)}$ converge para *L* em um número finito de iterações se [Spu96]:

$\sum^{n}$ $\frac{C_i}{}$ ≤ 1.

$$\Im = \left\{ d_{ik} \,\middle|\, d_{ik} = kT_i \ \wedge \ d_{ik} \le min(L,H),\ 1 \le i \le n,\ k \ge 1 \right\}. \qquad [A4]$$

Um modelo mais realista onde ocorrem *"jitters"*, determina que uma tarefa T_i seja retida depois de sua chegada até um tempo máximo J_i para a sua liberação. As equações acima devem ser modificadas no sentido de expressar esses tempos. O efeito de *"jitters"* no pior caso pode provocar em [*0, t*] a interferência de instâncias que em situações normais teriam suas liberações fora desse intervalo. Com isto, a carga cumulativa *W(t)* correspondente ao intervalo [*0,t*] (carga que ocorre em [0, t]) deve, no pior caso, aumentar [Spu96]:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t + J_i}{P_i} \right\rceil C_i .$$

No mesmo sentido, a demanda de processador *C(t)* (carga com deadlines menor ou igual a *t*, ou seja, concluída em [0,t]) passa a ser fornecida por:

$$C(t) = \sum_{P_i \le t + J_i} \left\lfloor \frac{t + J_i}{P_i} \right\rfloor C_i .$$

A equação [A1] é modificada igualmente:

$$t \ge \sum_{P_i \le t + J_i} \left\lfloor \frac{t + J_i}{P_i} \right\rfloor C_i .$$

Os valores de *t* continuam sendo definidos pelo conjunto $\Im$ de [A4].

A.1.2 Deadlines Arbitrários

Ao considerarmos um conjunto de tarefas periódicas com deadlines relativos (D_i) arbitrários e escalonadas segundo o EDF, o teste anterior deixa de ser suficiente. O teste definido pelas condições [A1] e [A4] pode ser facilmente estendido para tratar tarefas com deadlines arbitrários [Spu96].

A carga de trabalho acumulada em [*0, t*] deve levar em consideração esses deadlines arbitrários. Considerando $t - D_i$ o instante de liberação da última instância de T_i com garantia de execução em [*0, t*], a *demanda de processador C(t)* proposto por esse conjunto de tarefas é dada por :

$$C(t) = \sum_{D_i \le t} \left(1 + \left\lfloor \frac{t - D_i}{P_i} \right\rfloor \right) C_i . \qquad [A5]$$

Com isto uma condição necessária e suficiente para tarefas periódicas possuindo deadlines arbitrários e escalonadas pelo EDF, é obtida através de:

$$t \ge \sum_{D_i \le t} \left(1 + \left\lfloor \frac{t - D_i}{P_i} \right\rfloor \right) C_i \qquad [A6]$$

$$e \qquad \Im = \left\{ d_{ik} \,\middle|\, d_{ik} = kT_i + D_i ,\ d_{ik} \le min(L, H), 1 \le i \le n, k \ge 0 \right\}, \qquad [A7]$$

onde *t* assume valores de $\Im$ para a inspeção da condição do teste. A ocorrência de *"jitter"* no modelo redefine as equações [A5] e [A6] [Spu96]:

$$C(t) = \sum_{D_i \le t + J_i} \left(1 + \left\lfloor \frac{t + J_i - D_i}{P_i} \right\rfloor \right) C_i \qquad [A8]$$

$$t \ge \sum_{D_i \le t + J_i} \left(1 + \left\lfloor \frac{t + J_i - D_i}{P_i} \right\rfloor \right) C_i \qquad [A9]$$

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

Tabela I: Exemplo da figura 2.7 (capítulo 2)

Para ilustrar esse teste para modelos de tarefas com deadlines arbitrários em esquemas de prioridade dinâmica, considere o conjunto de tarefas descrito na *tabela I*. Usando a equação [A5] podemos determinar para essas tarefas a demanda de processador *C(t)*:

$$C(t) = \left(1 + \left\lfloor \frac{t-6}{10} \right\rfloor \right).2 + \left(1 + \left\lfloor \frac{t-8}{10} \right\rfloor \right).2 + \left(1 + \left\lfloor \frac{t-16}{20} \right\rfloor \right).8$$

Falta obtermos o valor *t* a ser usado no teste [A6]. Para isto, a carga cumulativa (tarefas que chegam antes e em *t*) é necessária:

$$W(t) = \left\lceil \frac{t}{10} \right\rceil .2 + \left\lceil \frac{t}{10} \right\rceil .2 + \left\lceil \frac{t}{20} \right\rceil .8 \cdot$$

Aplicando [A2] no sentido de determinar L (o maior *"busy period"*) :

$$L^{(0)} = \sum C_i = 12$$
$$L^{(1)} = W(L^{(0)}) = 16$$
$$L^{(2)} = W(L^{(1)}) = 16$$

Logo $L=16$, com isto obtemos o conjunto dos valores possíveis de t [A7]:

$$\Im = \left\{ d_{ik} \,\middle|\, d_{ik} \leq 16 \right\}$$

Considerando as tarefas da *tabela I* e a figura 2.7 (capítulo 2) verificamos que $\Im = \{6, 8, 16\}$. Com isto obtemos as seguintes demandas de processador: $C(6) = 2$, $C(8) = 6$ e $C(16) = 16$. Em todos esses valores de t é verificado o teste [A6], ou seja, $t \geq C(t)$. Logo o conjunto apresentado na *tabela I* e figura 2.7 é escalonável também em esquemas de prioridade dinâmica.

A.2 Compartilhamento de Recursos em Políticas de Prioridade Dinâmica

A técnica chamada de *Stack Resource Policy* (SRP) foi introduzida em [Bak91] para controlar em políticas de prioridade dinâmica, o compartilhamento de recursos de uma forma previsível. Basicamente, esse método estende os resultados do *"Priority Ceiling Protocol"* para esquemas de prioridade dinâmica. Ou seja, as inversões de prioridades se limitam a um bloqueio por ativação da tarefa.

A.2.1 Política de Pilha (*Stack Resource Policy*)

O SRP é similar ao IPCP na característica de que uma tarefa é bloqueada só no instante em que tenta a preempção. Esse bloqueio antecipado – diferente do PCP onde uma tarefa só é bloqueada quando tenta entrar em uma seção crítica – reduz a necessidade de trocas de contextos de tarefas, simplificando a implementação do protocolo.

Descrição do Protocolo

Em escalonamentos dirigidos por prioridades dinâmicas, a prioridade p_i de uma tarefa T_i indica a sua urgência em um determinado instante *t*. Quando usada a atribuição EDF, a prioridade p_i para refletir a urgência de T_i no instante *t*, é função direta ou assume o valor do deadline absoluto d_i desta tarefa. Além da prioridade p_i, quando usado o SRP, a tarefa T_i deve apresentar um *nível de preempção* π_i que é um parâmetro estático. Os níveis de preempção estabelecem regras para preempções de tarefas: uma tarefa T_j só poderá ser interrompida por uma tarefa T_i se $(\pi_i > \pi_j) \wedge (p_i > p_j)$. A utilidade dos níveis de preempção é que, por serem estáticos, esses parâmetros permitem a prevenção de bloqueios em esquemas de prioridades dinâmicas.

No escalonamento EDF, usando o SRP, os valores de π são definidos na ordem inversa dos deadlines relativos. Supondo as tarefas T_i e T_j ilustradas com dois casos de chegada na figura A.1. É tirado desta figura que π_i é maior que π_j pois $D_i < D_j$. No caso (a) da figura, a tarefa T_i que chega em t_2 consegue a preempção porque $\pi_i > \pi_j$ ($D_i < D_j$) e $p_i > p_j$ ($d_i < d_j$). No caso (b) da figura A.1 a preempção não ocorre por que uma das duas condições necessárias não se verifica: T_i que chega em t_2 não possui prioridade maior que T_j ($d_i > d_j$).

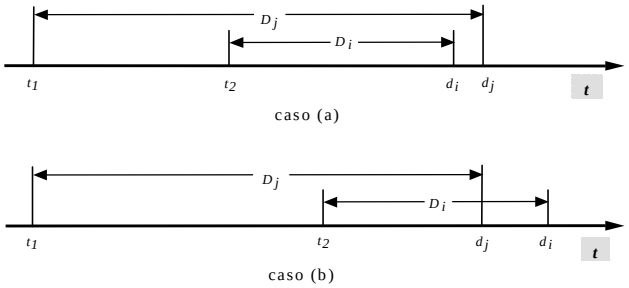


Figura A.1: Níveis de Preempção no EDF

Cada recurso compartilhado possui um valor máximo de unidades de recurso para alocação (*max* n_r). O SRP também, a exemplo do PCP, atribui a cada recurso compartilhado uma prioridade teto ($\lceil R_r \rceil$). A diferença é que, neste caso, este parâmetro é dinâmico: o *"ceiling"* corrente de R_r tem o valor determinado a partir do número de unidades de alocação disponíveis no recurso R_r. Se n_r representa o número de unidades de R_r disponíveis em um dado instante e, $\mu_{r,i}$ corresponde as necessidades de uma tarefa T_i em termos do recurso R_r então o *ceiling* é dado por:

$$\lceil R_r \rceil_{(n_r)} = max\left[\{0\} \cup \left\{\pi_i \mid n_r < \mu_{r,i}\right\}\right].$$

Na expressão acima, quando todas as unidades de R_r estão disponíveis, o valor

corrente de *"ceiling"* de R_r é mínimo ($\lceil R_r \rceil = 0$). Se as unidades de alocação de R_r não são suficientes para que algumas tarefas se executem, então o valor do *"ceiling"* assumirá o mais alto nível de preempção dentre estas tarefas ($\lceil R_r \rceil = \pi_j$). O *"ceiling"* do sistema (Π) é assumido como o máximo *"ceiling"* entre todos os *m* recursos compartilhados no sistema:

$$\Pi = max\left(\lceil R_r \rceil \,\middle|\, r = 1,2,\ldots,m\right).$$

No SRP a regra que garante a não ocorrência de múltiplas inversões de prioridade determina que uma tarefa T_i não pode se executar até que os recursos necessários para a sua execução estejam disponíveis, ou seja, $\pi_i > \Pi$. De uma maneira geral, uma tarefa T_i pode preemptar uma tarefa T_j em execução quando: (i) for a tarefa mais prioritária ($p_i > p_j$); (ii) o seu nível de preempção for maior do que a tarefa se executando ($\pi_i > \pi_j$); e por fim, (iii) se o seu nível de preempção for superior ao *"ceiling"* do sistema ($\pi_i > \Pi$).

Esse método impõe que, quando da ativação de uma tarefa T_i, fique disponível as informações de suas necessidades em recursos (ou seja, os seus valores de $\mu_{r,i}$ para cada recurso R_r) de modo a se determinar previamente os valores de *"ceiling"* de cada recurso R_r em diferentes situações de alocação. Quando as condições (i), (ii) e (iii) são verificadas, a tarefa T_i tem satisfeita as suas necessidades em recursos disponíveis e poderá, portanto, iniciar sua execução sem ser bloqueada por tarefas menos prioritárias. Muito embora, os recursos só venham a ser alocados quando requisitados durante a execução de T_i.

A figura A.2 e suas tabelas apresentam uma aplicação do *"Stack Resource Policy"*. Os parâmetros das tarefas T_1, T_2 e T_3 e suas necessidades de recursos são descritos nas tabelas ($\mu_{r,i}$). Os recursos R_1, R_2 e R_3 são mostrados com suas máximas unidades para alocação ($max\ n_r$). Os *"ceilings"* são determinados em tabela a partir das unidades de alocação disponíveis nos recursos.

Uma escala obtida sob o SRP é descrita na figura A.2. Em *t=2*, a tarefa menos prioritária T_3 começa a executar porque possui seu nível de preempção maior que o *"ceiling"* do sistema ($\Pi = 0$). T_3 entra na sua primeira seção crítica em *t =3* e ocupa uma unidade do recurso R_3. Com isto o *"ceiling"* do sistema passa ao valor de π_2, porque T_2 não consegue ter suas necessidades atendidas ($\mu_{3,2} = 2$). Logo, em *t = 4*, T_2 é bloqueada pelo teste de preempção e não consegue executar. A tarefa T_3 continua a sua execução e em *t = 4* entra em sua seção crítica aninhada ocupando uma unidade de R_2. Com esta nova seção de T_3, a execução de T_1 é postergada em *t=5* pela não disponibilidade de suas necessidades no recurso R_2 ($\mu_{2,1} = 3$). Como conseqüência, o *"ceiling"* do sistema cresce para π_1. Quando T_3 libera o recurso R_2, o *"ceiling"* do sistema torna se π_2 o que permite que T_1 interrompa T_3. T_1 é executada completamente porque seu nível de preempção é maior que o *"ceiling"* do sistema. Com a conclusão de T_1, T_3 reassume e quando libera o recurso R_3 em *t=15*, o *"ceiling"* do sistema baixa para $\Pi = 0$ e a tarefa T_2 pode interromper T_3.

R_r	*Max* n_r	$\mu_{r,1}$	$\mu_{r,2}$	$\mu_{r,3}$	$\lceil R_r \rceil_{(3)}$	$\lceil R_r \rceil_{(2)}$	$\lceil R_r \rceil_{(1)}$	$\lceil R_r \rceil_{(0)}$
R_1	**1**	**1**	**1**	-	-	-	**0**	π_1
R_2	**3**	**3**	-	**1**	0	π_1	π_1	π_1
R_3	**2**	-	**2**	**1**	-	0	π_2	π_2

T_1	T_2	T_3
....	....	....
pedido (R_1,1)	pedido (R_3,2)	pedido (R_3,1)
libera (R_1)	libera (R_3)	pedido (R_2,1)
pedido (R_2,3)	pedido (R_1,1)	libera (R_2)
libera (R_2)	libera (R_1)	libera (R_3)
....	....	....

Prioridades : $p_1 > p_2 > p_3$

Níveis de preempção : $\pi_1 > \pi_2 > \pi_3$ ($D_1 < D_2 < D_3$)

Semáforos : S_1 - S_2 - S_3 -

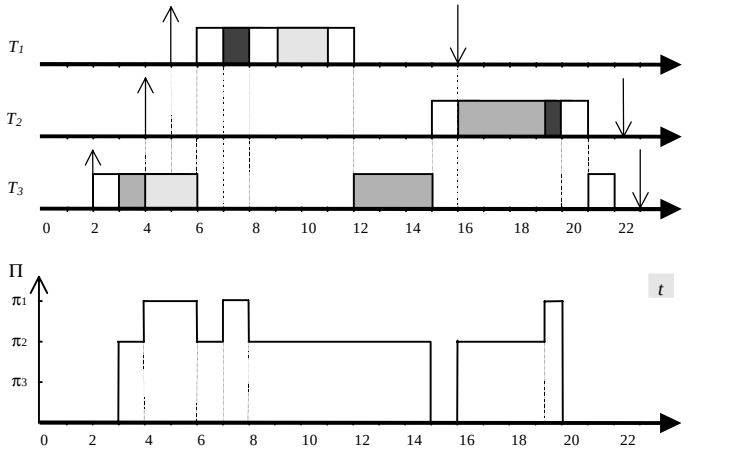


Figura A.2 : Exemplo de escala com o SRP

Extensões de Testes de Escalonabilidade Tomando como Base o SRP

O SRP foi concebido para ser usado com políticas de prioridades dinâmicas como o EDF. Em [Bak91], o teste de escalonabilidade do EDF (equação [3], capítulo 2) foi estendido considerando o SRP:

$$\sum_{j=1}^{i} \frac{C_j}{P_j} + \frac{B_i}{P_i} \leq 1, \quad \forall i, 1 \leq i \leq n.$$

O somatório da equação acima representa a utilização de T_i e de todas as tarefas T_j que possuam nível de preempção maior que o seu ($\pi_j > \pi_i$). Esta condição considera o instante crítico na origem o que implica nas tarefas ordenadas segundo valores decrescentes de seus níveis de preempção, ou seja, no sentido crescente de seus deadlines relativos. O termo B_i/P_i corresponde a utilização perdida com bloqueio por tarefa de nível de preempção menor que π_i.

Quando é assumido deadlines relativos menores que os correspondentes períodos no modelo de tarefas, o teste proposto por Baker toma a seguinte forma:

$$\sum_{j=1}^{i} \frac{C_j}{D_j} + \frac{B_i}{D_i} \leq 1, \quad \forall i, 1 \leq i \leq n.$$

Os testes acima, conseguidos a partir de extensões de [3] (capítulo 2), dependem apenas de parâmetros estáticos e portanto, podem ser verificados em tempo de projeto.

Outros testes mais precisos e gerais para políticas de prioridade dinâmica, apresentados neste *Anexo*, devem também ser modificado para expressar os bloqueios que tarefas podem sofrer de tarefas menos prioritárias. É o caso dos testes baseados em *demanda de processador* [Spu96], onde tarefas com deadlines relativos arbitrários e sofrendo de *release "jitters"*, quando compartilham recursos sob o SRP, podem ter suas escalonabilidades verificadas a partir de:

$$t \geq \sum_{D_j \leq t + J_j} \left(1 + \left\lfloor \frac{t + J_j - D_j}{P_j} \right\rfloor \right) C_j + \left(\left\lfloor 1 + \frac{t + J_i - D_i}{P_i} \right\rfloor \right) B_i \qquad \forall i,\ 1 \leq i \leq n$$

onde t é definido em $\Im$ (conjunto [A.7] indicado no item A.1.2.):

$$\Im = \left\{ d_{ik} \,\middle|\, d_{ik} = kT_i + D_i,\ d_{ik} \leq min(L, H),\ k \geq 0 \right\}.$$

No teste acima o somatório determina a demanda de carga de tarefas de nível de prioridade maior ou igual a i, ou seja, tarefas que apresentem $D_j - J_j \leq D_i - J_i$. O termo que envolve B_i corresponde ao pior caso de bloqueio imposto sobre T_i, durante o intervalo t, por tarefas de menor nível de preempção.

O máximo bloqueio B_i que uma tarefa T_i pode experimentar em uma ativação, deve corresponder a duração da maior seção crítica entre as que podem bloquear T_i. A exemplo do PCP, um bloqueio só pode ser caracterizado quando uma tarefa T_j menos prioritária executando em uma seção crítica de duração $D_{j,r}$, guardada pelo semáforo S_r, tiver o seu nível de preempção menor que o de T_i ($\pi_i > \pi_j$) e o recurso guardado apresentar o *"ceiling"* máximo igual ou maior que π_i. O *"ceiling"* máximo de um semáforo é assumido quando o número de unidades livres do recurso for nulo. O bloqueio máximo é dado então por:

$$B_i = \max_{j,r}\left\{ D_{j,r} \,\middle|\, \left(\pi_j < \pi_i\right) \wedge \left(\left\lceil R_{S_r} \right\rceil_{(0)} \geq \pi_i \right) \right\}$$

A.3 Escalonamento de tarefas aperiódicas com políticas de prioridade dinâmica

O escalonamento de tarefas aperiódicas e periódicas pode também ser feito sob atribuições dinâmica de prioridades. As tarefas periódicas neste caso são escalonadas usando o algoritmo *"Earliest Deadline First"* (EDF). A justificativa encontrada para a extensão do EDF de modo a permitir esquemas híbridos de escalonamento é fundamentada nos maiores limites de escalonabilidade desta política se comparada com as de prioridade fixa. Em [Spu96] a título de exemplo é apresentada uma carga periódica com utilização $U_p=0,6$. Se a atribuição de prioridades é feita pelo RM e o servidor de prioridade fixa usado é o SS, a máxima Utilização do servidor é dada por $U_S=0,1$. Se o EDF é usado a utilização do processador vai para 100% e a máxima utilização do servidor acaba sendo de $U_S=0,4$.

A.3.1 Servidores de Prioridade Dinâmica [SpB96]

Dentre as muitas abordagens introduzidas em [SpB96] para servidores de prioridade dinâmica (*"Dynamic Priority Exchange"*, *"Dynamic Sporadic Server"*, *"Improved Priority Exchange"*, etc.), uma grande parte é extensão dos servidores de prioridade fixa apresentados no item 2.8 (capítulo 2). Neste item, no sentido de evitar textos repetitivos, é apresentado como uma ilustração destes servidores o *"Dynamic Sporadic Server"* (DSS); os leitores que quiserem um estudo mais exaustivo sobre o assunto devem recorrer a [SpB96].

Servidor Esporádico Dinâmico

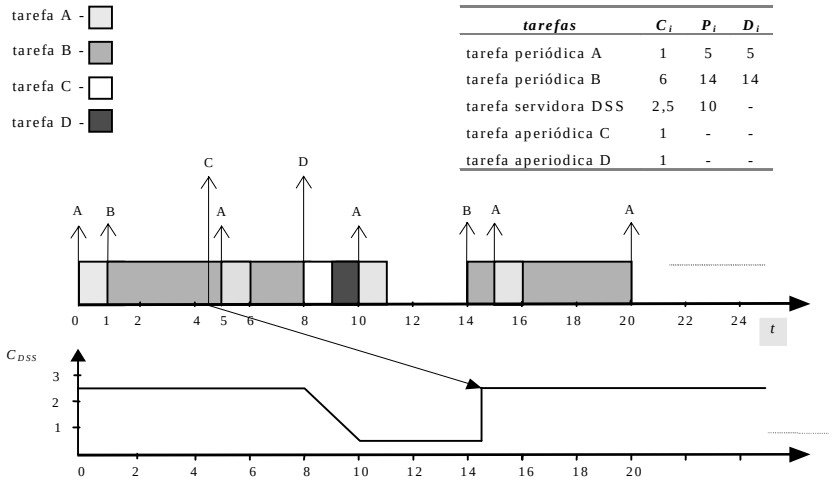
O *"Dynamic Sporadic Server"* (DSS) estende o algoritmo do SS para atuar com políticas de prioridade dinâmica – mais precisamente, o EDF. Esta abordagem híbrida abre espaço para execuções aperiódicas em escalas ordenadas pelo EDF também usando o conceito de servidor: uma tarefa servidora é então criada com capacidade C_S e período P_S. A servidora DSS preserva a sua capacidade enquanto não ocorrem requisições aperiódicas. Como a sua homônima de prioridade fixa a capacidade consumida não é preenchida no início do período da servidora DSS, mas no tempo de preenchimento RT correspondente.

A prioridade e o preenchimento da capacidade da servidora DSS são determinados pelas seguintes ações [SpB96, But97]:

- Quando a servidora é criada, a sua capacidade C_S é iniciada no seu máximo valor.
- O tempo de preenchimento e o deadline d_S da tarefa servidora são determinados quando temos $C_S > 0$ e existe uma requisição aperiódica pendente. Se t_a é o instante de tempo em que se verificam estas condições, então $RT = d_S = t_a + P_S$.
- A quantidade de capacidade a ser preenchida em RT é obtida quando a última requisição aperiódica é completada ou a capacidade C_S da servidora é totalmente consumida. Se no tempo t_f ocorre uma dessas duas situações, então a quantidade de capacidade a ser preenchida é igual a capacidade consumida em $[t_a, t_f]$.

A limitação imposta pela servidora DSS sobre a utilização da carga periódica do sistema é provado em [Spu96] como idêntica a de uma tarefa periódica de período P_S e tempo de computação C_S, ou seja: $U_p \leq 1 - U_S$.

A figura A.3 ilustra um exemplo de uso da técnica DSS – que é o mesmo utilizado quando da apresentação do servidor SS de prioridade fixa (seção 2.8.1). O conjunto de tarefas descrito na tabela da figura é para ser escalonado segundo uma atribuição EDF (dinâmica). a tarefa servidora DSS é também definida com capacidade $C_S=2,5$ e período $P_S = 10$.



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

Figura A.3: Algorítmo "Dynamic Sporadic Server"

Em $t = 0$, a tarefa A, na sua primeira ativação apresentando deadline absoluto mais próximo ($d_{A,1}= 5$), assume o processador. Na seqüência, quando A conclui, a tarefa periódica B com deadline em $t = 14$ passa a ser executada. Em $t = 4,5$, com a chegada

de uma requisição C a servidora DSS é liberada com o deadline absoluto em $t = 14,5$ ($d_S = RT = t_a + P_S$). No caso, t_a coincide com o tempo de chegada da requisição C porque $C_S > 0$ em $t = 4,5$. Embora a capacidade da servidora seja suficiente, a tarefa B não é interrompida porque esta última possui o deadline mais próximo ($d_{B,1}=14$). A tarefa A, por sua vez, interrompe B em $t = 5$, durante sua segunda ativação ($d_{A,2}= 10$). Em seguida, no término de A, a tarefa B reassume concluindo em $t=8$. No instante $t=8$, a servidora DSS começa a executar a tarefa C com o deadline absoluto em $t=14,5$. Em $t=8$, chega também uma requisição D que é executada em seguida pela tarefa servidora com o mesmo deadline ($RT=14,5$). Isto ocorre porque, quando liberado, o servidor DSS deve executar todas as requisições aperiódicas pendentes enquanto a condição $C_S > 0$ for mantida. Se compararmos a escala obtida na figura A.3 com a escala do servidor esporádico estático, para o mesmo conjunto de tarefas (figura 2.17, capítulo 2, seção 2.8.1), verificamos que o DSS apresenta um desempenho inferior ao seu similar estático. Em termos de resposta imediata o DSS também é pior.

A figura A.4 mostra o mesmo conjunto de tarefas mas com a servidora DSS projetada com capacidade $C_S=1$. Neste caso, a tarefa aperiódica C consome toda a capacidade da servidora em sua execução que inicia em $t=8$. A tarefa D terá a condição de $C_S > 0$ somente em $t=14,5$, portanto com um t_a diferente de seu tempo de chegada. Com isto o deadline da tarefa D passa a ser em $t=24,5$ e a sua execução sob o EDF ocorre em $t=14,5$, interrompendo a tarefa B na sua segunda ativação ($d_{B,2}=28$). A tarefa *A*, por sua vez, interrompe a requisição D em $t=15$ por apresentar deadline mais próximo ($d_{A,3}= 20$). A requisição D reassume e conclui em $t=16,5$. O DSS pode tratar com tarefas aperiódicas firmes e a análise para a garantia dinâmica não difere muito das usadas em servidores de prioridade fixa.

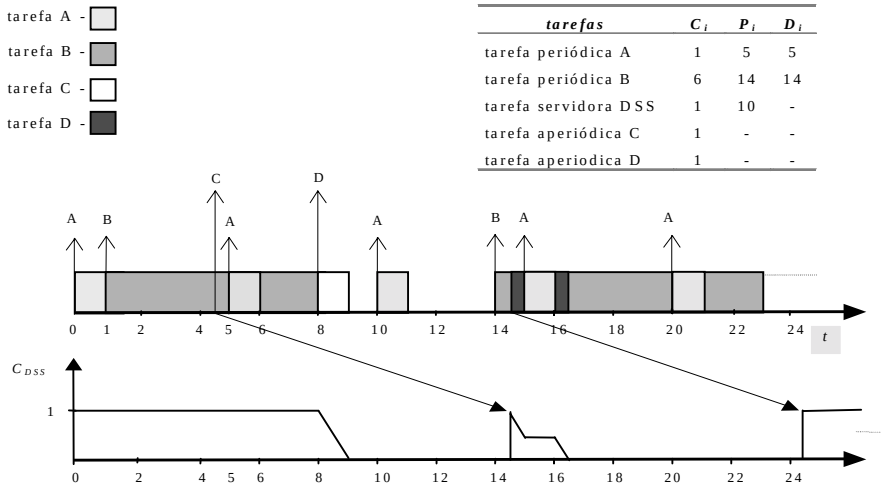


Figura A.4: "Dynamic Sporadic Server" com capacidade menor