Acesso a Recursos Partilhados em Sistemas de Tempo Real

Rui Santos, Arnaldo Oliveira, Luís Almeida

***Abstract* – Real-time systems are typically reactive systems implemented with a set of concurrent tasks running on top of an executive or operating system. Due to concurrent task execution and access to shared resources, it is fundamental that the execution of critical regions is performed in a safe way, i.e. controlled by appropriate synchronization primitives (e.g. semaphores). To reduce the sources of unbounded priority inversions the semaphores must be managed by protocols adequate for real-time systems. This paper describes some protocols used for such purposes, namely the Priority Inheritance Protocol (PIP), the Priority Ceiling Protocol (PCP) and the Stack Resource Policy (SRP).**

***Resumo* – Os sistemas de tempo-real são em geral sistemas reactivos cujo comportamento é implementado através de um conjunto de tarefas que executam concorrentemente sobre um executivo ou sistema operativo. Devido à execução concorrente das tarefas e ao acesso a recursos partilhados, é fundamental que a entrada em regiões críticas seja feita de forma segura, isto é, controlada por primitivas de sincronização, e.g. semáforos, e protocolos adequados a sistemas de tempo-real que reduzam o tempo de bloqueio de tarefas prioritárias. Neste artigo apresentamos algumas técnicas para controlo das inversões de prioridade resultantes do acesso a recursos partilhados, focando sobre os seguintes protocolos de gestão de semáforos: *Priority Inheritance Protocol* (PIP), o *Priority Ceiling Protocol* (PCP) e o *Stack Resource Policy* (SRP).**

***Keywords* – Real-time Systems, Shared Resource Access, Semaphore Management Protocols, Priority Inheritance Protocol, Priority Ceiling Protocol, Stack Resource Policy.**

***Palavras chave* – Sistemas de Tempo-Real, Acesso a Recursos Partilhados, Protocolos de Gestão de Semáforos, Protocolo de Herança de Prioridades, Protocolo de Tecto de Prioridades, Política de Pilha de Recursos.**

I. INTRODUÇÃO

Muitas aplicações de tempo-real consistem na execução de tarefas concorrentes que acedem a recursos partilhados. Para assegurar a integridade destes recursos o acesso a estes deve ser efectuado em regime de exclusão mútua. Porém, quando tarefas de diferentes prioridades concorrem pelo acesso a recursos partilhados podem ocorrer inversões de prioridade. Isto significa que tarefas de maior prioridade podem ficar à espera (bloqueadas) da libertação de recursos anteriormente adquiridos por tarefas menos prioritárias.

Consideremos o cenário da figura 1, onde é apresentado um conjunto de tarefas $\{T_1, T_2, T_3\}$ com prioridades fixas de tal forma que T_1 é a mais prioritária e T_3 a menos prioritária. A tarefa T_3 começa a executar e entra na região crítica adquirindo assim o recurso partilhado. De seguida, T_1 inicia a sua execução no instante *a*, ficando bloqueada a partir do momento em que requisita o recurso (no instante *b*) partilhado com a tarefa T_3. No instante *c*, T_2 inicia a sua execução causando a preempção de T_3. Assim, até ao instante *d*, T_1 é bloqueada pela execução não apenas da tarefa T_3 mas também da tarefa T_2, com a qual T_1 não partilha qualquer recurso. O bloqueio causado por T_3 designa-se por *bloqueio directo* enquanto o causado por T_2 se designa por *bloqueio indirecto*. Este cenário caracteriza o problema da inversão de prioridades e da limitação da sua duração. Note-se que quaisquer tarefas com prioridade entre as de T_1 e T_3 poderão efectuar preempção sobre T_3 e alongar indeterminadamente o bloqueio indirecto causado a T_1.

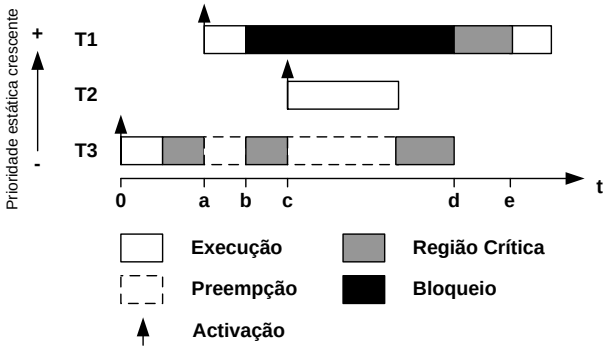

Fig. 1 - O problema da inversão de prioridades.

A inversão de prioridades é um fenómeno que não pode ser completamente evitado na presença de recursos partilhados de acesso exclusivo. Contudo, é fundamental determinar a sua duração e minimizá-la. Neste artigo apresentamos várias técnicas para limitação do fenómeno de inversão de prioridades que variam em termos de dificuldade de concretização e de impacto sobre a pontualidade do sistema. A informação apresentada neste artigo foi essencialmente compilada de [1] e [2].

II. TÉCNICAS BÁSICAS DE SINCRONIZAÇÃO

Conforme terá ficado claro no exemplo da secção anterior, as inversões de prioridade surgem associadas à preempção de tarefas. De facto, num sistema sem preempção, considerando que uma tarefa deverá, no final de cada instância, libertar todos os recursos utilizados nessa instância (uma questão de boas práticas no desenvolvimento de software), apenas uma tarefa de cada vez poderá aceder a cada recurso partilhado, resolvendo implicitamente o problema de sincronização no respectivo acesso. Assim, as técnicas utilizadas para limitação da inversão de prioridades são, de facto, técnicas para controlo de preempção em sistemas ditos *preemptivos*.

Estas técnicas podem, grosso modo, ser divididas em dois

grupos. No primeiro constam as técnicas chamadas básicas, as quais são de fácil implementação mas não são selectivas relativamente às tarefas que sofrem bloqueio. Isto quer dizer que bloqueiam igualmente todas as tarefas de maior prioridade, quer utilizem o recurso partilhado em questão ou não. São por isso pouco eficientes e causadoras de considerável perturbação na pontualidade do sistema. Num segundo grupo encontram-se as técnicas baseadas em semáforos que, embora mais complexas, permitem reduzir o conjunto das tarefas afectadas pelos bloqueios associados às inversões de prioridade. Nesta secção debruçar-nos-emos apenas sobre as técnicas ditas básicas.

A. Inibição de interrupções

Uma das formas de inibição da preempção causada pelo executivo é a inibição das interrupções. De facto, a activação de tarefas de maior prioridade é, regra geral, efectuada ou por interrupções assíncronas (tarefas aperiódicas) ou pelo mecanismo de gestão temporal do executivo (tarefas periódicas) inserido na rotina de atendimento da interrupção periódica do sistema, vulgo *tick*. Assim, enquanto as interrupções estiverem inibidas não poderá haver activação de outras tarefas e, como tal, fica garantido o acesso exclusivo a quaisquer recursos. Para utilizar esta técnica basta desligar as interrupções quando uma tarefa entra na sua região crítica, tomando posse de um recurso partilhado, e voltar a activá-las assim que de lá sair. É, pois, uma técnica de muito simples concretização.

Como consequência todas as restantes actividades do sistema ficam bloqueadas, não apenas todas as tarefas de maior prioridade mas também o atendimento a interrupções externas e, principalmente, à interrupção periódica do sistema. Neste último caso, se o bloqueio for suficientemente longo poderão perder-se *ticks*, levando a um atraso do relógio do sistema e, conforme o modo de operação, a um atraso na activação de todas as tarefas periódicas. Por esta razão, esta técnica só deve ser usada em regiões críticas muito curtas.

No que diz respeito à duração do bloqueio causado, com esta técnica uma tarefa só pode ser bloqueada uma vez e por um tempo que, no pior caso, é igual à maior região crítica das tarefas de menor prioridade.

B. Inibição de preempção

Uma solução menos drástica do que a anterior consiste em inibir directamente a preempção de tarefas sempre que estas estejam a executar numa região crítica mas sem inibir o atendimento a interrupções. Alguns executivos disponibilizam funções para este fim, e.g. *set_preempt()* e *set_no_preempt()*, que actuam sobre uma *flag* interna. Esta técnica é também fácil de concretizar ao nível do executivo e fácil de utilizar ao nível da aplicação. Contudo, embora já não bloqueie a actividade associada ao atendimento de interrupções, incluindo o atendimento do relógio do sistema, esta técnica bloqueia igualmente todas as tarefas mais prioritárias, independentemente de usarem ou não quaisquer recursos.

A duração do bloqueio causado é semelhante à da técnica anterior. De facto, também com esta técnica cada tarefa só pode ser bloqueada uma vez e por um tempo, no pior caso, igual à maior região crítica das tarefas de menor prioridade.

III. PROTOCOLOS DE SINCRONIZAÇÃO BASEADA EM SEMÁFOROS

Ao invés das técnicas básicas, as técnicas baseadas na utilização de semáforos permitem reduzir o conjunto das tarefas que sofrem bloqueio por via de inversões de prioridade inerentes ao acesso a recursos partilhados. Nestes casos, apenas as tarefas que acedem a recursos poderão ser bloqueadas. A utilização de semáforos requer um conjunto adicional de mecanismos e estruturas de dados, sendo, por isso, mais complexa de concretizar e de utilizar. Por outro lado, a redução do número de situações de bloqueio leva a menores tempos de resposta e logo a uma melhor pontualidade do sistema.

Basicamente, estas técnicas constam de associar um semáforo a cada recurso e forçar cada tarefa a passar pelo controlo do semáforo respectivo antes de entrar numa região crítica. Se o semáforo estiver aberto o acesso é concedido de imediato. Se estiver fechado, a tarefa fica bloqueada e é colocada numa fila de espera até que o semáforo abra e a tarefa tenha prioridade suficiente. A forma como a abertura e fecho dos semáforos é gerida, bem como as respectivas filas de espera, é determinada por um protocolo específico. Do ponto de vista dos sistemas de tempo-real, alguns dos efeitos que um protocolo de gestão de semáforos deverá evitar são os bloqueios indeterminados, os bloqueios em cadeia e a situação de *deadlock*.

Nesta secção abordaremos três protocolos típicos em executivos de tempo-real, nomeadamente o Protocolo de Herança de Prioridades (*PIP - Priority Inheritance Protocol* [3]), o Protocolo de Tecto de Prioridades (*PCP - Priority Ceiling Protocol* [3]) e o Política de Pilha de Recursos (*SRP - Stack Resource Policy* [4], [5]).

A. Protocolo de Herança de Prioridades (PIP - Priority Inheritance Protocol)

O Protocolo de Herança de Prioridades foi desenvolvido como uma solução simples mas eficaz para eliminar o indeterminismo associado ao bloqueio indirecto ilustrado na figura 1, causado nas tarefas de maior prioridade pelas tarefas de prioridade intermédia. Conforme se explica em baixo, a opção inerente ao protocolo PIP foi transferir esse bloqueio para as próprias tarefas de prioridade intermédia.

Este protocolo aplica-se a sistemas baseados em prioridades fixas, i.e., em que as tarefas têm uma prioridade atribuída em fase de projecto. Contudo, em tempo de execução o protocolo PIP eleva temporariamente a prioridade de uma tarefa sempre que esta se encontra a bloquear outra, deste modo reduzindo a possibilidade de preempção da tarefa bloqueante. Suponhamos que uma tarefa T_i é bloqueada num semáforo. Nesse instante, a tarefa bloqueante T_j herda a prioridade de T_i, mais alta, e retoma a execução da sua região crítica. No momento em que T_j completa a sua região crítica, liberta o semáforo em questão e a sua prioridade volta a assumir o seu valor inicial. Na ausência de acessos encadeados a recursos partilhados, cada tarefa só pode bloquear outra uma vez e só pode ficar bloqueada uma

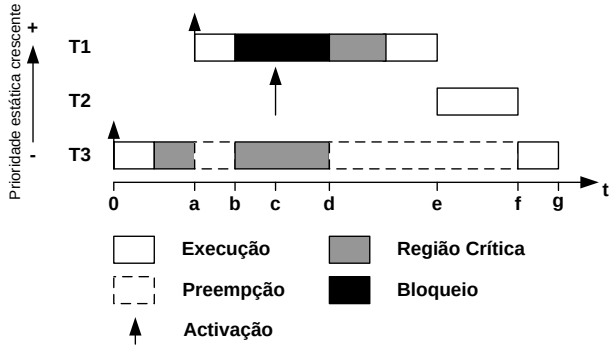

Fig. 2 - Exemplo do uso do Protocolo de Herança de Prioridades.

vez em cada semáforo. Assim, uma tarefa mais prioritária pode sofrer dois tipos de bloqueio:

- Bloqueio directo: ocorre quando a tarefa mais prioritária tenta aceder a um recurso partilhado já na posse de uma tarefa menos prioritária (figura 2, entre *b* e *d*).
- Bloqueio por herança (*push-through blocking*): ocorre quando uma tarefa de prioridade intermédia é impedida de continuar a sua execução por uma tarefa que tenha herdado a prioridade de uma tarefa mais prioritária (figura 2, entre *c* e *d*).

A aplicação do Protocolo de Herança de Prioridades no exemplo da figura 1, implica que a tarefa T_3 não sofrerá preempção pela tarefa T_2 porque herda a prioridade de T_1 no instante *b* (figura 2). Ao sair da sua região crítica (no instante *d*), T_3 volta a ter a sua prioridade original e é interrompida pelas tarefas mais prioritárias T_1 e T_2.

Este protocolo é relativamente fácil de implementar, pois apenas necessita de mais um campo na estrutura de controlo da tarefa (*Task Control Block - TCB*) para armazenar a prioridade herdada. No caso de se permitir o acesso encadeado a múltiplos recursos a implementação fica um pouco mais complexa já que é necessário garantir a transitividade da herança, i.e., 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. Para além da maior complexidade, o acesso encadeado com semáforos PIP apresenta ainda alguns efeitos nefastos, nomeadamente a possibilidade de ocorrência de bloqueios em cadeia e de *deadlocks*.

É também interessante perceber os compromissos que se estabelecem ao utilizar herança de prioridades relativamente a um semáforo sem esse mecanismo. Note-se que o protocolo PIP gera um novo tipo de bloqueio que afecta todas as tarefas de prioridade intermédia quer usem ou não recursos partilhados. Isto leva a atrasos suplementares que não existem com semáforos simples (sem herança de prioridades). O que se ganha é um menor e mais definido bloqueio das tarefas mais prioritárias. Este compromisso nem sempre é fácil de resolver mesmo por programadores experientes como sucedeu no célebre caso da sonda de Marte, Pathfinder [6].

B. Protocolo de Tecto de Prioridades (PCP - Priority Ceiling Protocol)

A ideia central do Protocolo de Tecto de Prioridades é limitar o número de bloqueios por inversão de prioridades, evitar a ocorrência de bloqueios em cadeia e *deadlocks* durante a execução das tarefas. Este protocolo é uma extensão do Protocolo de Herança de Prioridades ao qual se adicionou uma regra de controlo no acesso aos semáforos livres baseada no conceito de tecto de prioridade. É também dirigido aos sistemas de prioridades fixas.

Todos os recursos (semáforos) acedidos em regime de exclusão mútua possuem um valor de prioridade tecto (*ceiling priority C(S_k)*) que corresponde à prioridade mais elevada do conjunto de tarefas que acedem ao recurso. A regra que define a entrada na região crítica é a seguinte: uma tarefa só acede a um recurso partilhado se este estiver livre e a prioridade da tarefa for maior que a prioridade tecto (*ceiling*) de qualquer recurso que nesse momento esteja bloqueado. São excluídos dessa comparação os recursos bloqueados pela tarefa que tenta o acesso. Se S_1 for o semáforo com maior prioridade tecto entre todos os semáforos bloqueados, então uma tarefa T_i só entrará na sua região crítica se a sua prioridade p_i for maior que o *ceiling C(S_1)*. Se p_i for menor ou igual a *C(S_1)*, o acesso é negado a T_i mesmo que o recurso a que está a tentar aceder esteja livre. Este é um novo tipo de bloqueio que se designa por *bloqueio de tecto*.

Esta particularidade do protocolo PCP, que parece ser estranha à primeira vista, destina-se a garantir que quando uma tarefa acede a um recurso partilhado todos os recursos de que poderá ainda necessitar durante a sua execução estão livres. De facto, se a prioridade da tarefa for superior à maior prioridade tecto dos semáforos bloqueados, isso quer dizer que essa tarefa não utiliza os recursos desses semáforos. Caso contrário, alguma das prioridades tecto teria de ser pelo menos igual à prioridade da tarefa. Desta forma demonstra-se que o protocolo evita *deadlocks*, i.e., é *deadlock-free*.

A propriedade explicada atrás também permite inferir que, para além do bloqueio indirecto do tipo *push-through* inerente à herança de prioridades, com o protocolo PCP uma tarefa só pode ser bloqueada uma vez e logo no seu primeiro acesso a um recurso partilhado. A duração máxima de bloqueio que uma tarefa pode sofrer corresponde à duração da maior região crítica das tarefas de menor prioridade que partilham recursos com a tarefa em questão ou com outras tarefas de maior prioridade.

Para ilustrar o protocolo PCP considere-se o exemplo apresentado na figura 3. As tarefas T_1, T_2 e T_3 acedem de forma encadeada e em regime de exclusão mútua aos recursos R_1, R_2, R_3. A prioridade tecto (*ceiling*) de cada semáforo é definida segundo as tarefas que acedem ao recurso: *C(S_1)* = 3, *C(S_2)* = 3 e *C(S_3)* = 2. Para um melhor entendimento do funcionamento do protocolo explicamos de seguida os passos mais relevantes na execução das tarefas (figura 3):

- Instante *c*: T_2 sofre bloqueio directo: o semáforo S_3 está fechado, logo T_3 reassume a execução e herda a prioridade de T_2;
- Instante *d*: T_3 entra em mais uma região crítica e fecha

o semáforo S_2;
- Instante *e*: T_1 inicia a execução. T_1 é mais prioritária que T_3 mesmo com a prioridade herdada de T_2;
- Instante *f*: T_1 tenta fechar o semáforo S_1, é bloqueada por *ceiling*; possui uma prioridade igual ao maior *ceiling* dos semáforos já fechados por outras tarefas (*C(S_2)* = 3);
- Instante *g*: T_3 liberta semáforo S_2. T_1 interrompe a execução de T_3 e fecha o semáforo S_1;
- Instante *k*: T_1 termina a execução. T_3 reassume a execução ainda com a prioridade herdada de T_2;
- Instante *l*: T_3 liberta S_3 retomando assim a sua prioridade inicial. T_2 interrompe a execução de T_3 e fecha o semáforo S_3.

O Protocolo de Tecto de Prioridades tem como vantagem relativamente ao Protocolo de Herança de Prioridades o facto de ser livre de bloqueios em cadeia e de *deadlocks*. Porém, apresenta um bloqueio adicional, bloqueio de tecto, que poderá atrasar tarefas que não usam os recursos ocupados nesse instante e que não seriam bloqueadas com o protocolo PIP. Para além disso, não é transparente para o programador, já que este necessita de especificar as prioridades tecto dos semáforos, e é mais difícil de concretizar pois requer no TCB um campo para a prioridade herdada e outro para o semáforo onde a tarefa está bloqueada. Para facilitar a transitividade da herança, requer ainda uma estrutura para os semáforos com os respectivos tectos e identificação das tarefas que os estão a usar. Finalmente, refira-se ainda que também existe uma versão do protocolo PCP para prioridades dinâmicas, nomeadamente para EDF, conhecida como DPCP [7].

C. Política de Pilha de Recursos (SRP - Stack Resource Policy)

A Política de Pilha de Recursos é um protocolo criado para poder ser aplicado indistintamente em sistemas de prioridades fixas ou dinâmicas. Tem como principal característica o facto de impor bloqueios apenas antes das tarefas iniciarem execução. Após início de execução e até terminação as tarefas já não sofrerão qualquer bloqueio. Note-se o antagonismo face ao protocolo PCP, ou mesmo PIP, com os quais uma tarefa só é bloqueada quando tenta entrar numa região crítica. Esta característica reduz a necessidade de comutações de contexto e simplifica a implementação do protocolo pois deixa, na prática, de existir o estado de *bloqueio*. Se também não for possível a auto-suspensão das tarefas, então deixa de ser necessário salvaguardar o estado em situações de inversão de prioridade. Nestas circunstâncias não será necessário manter um *stack* separado para cada tarefa e o sistema poderá funcionar com um *stack* único, reduzindo substancialmente os requisitos de memória.

De acordo com o protocolo SRP, a cada tarefa T_i é atribuído um *nível de preempção* π_i que é um parâmetro estático e que representa, grosso modo, a capacidade de uma tarefa efectuar preempção sobre outras. Em sistemas de prioridades fixas o nível de preempção corresponde à prioridade. No caso de prioridades dinâmicas ditadas pela proximidade à *deadline*, tal como no algoritmo de escalonamento EDF (*Earliest Deadline First*), o nível de preempção é definido na ordem inversa das *deadlines* relativas.

O protocolo SRP, tal como o PCP, atribui a cada semáforo (*R*) uma prioridade tecto (C_R) constituída pelo máximo nível de preempção de entre as tarefas que usam esse semáforo. O protocolo SRP define ainda a prioridade tecto do sistema (*system ceiling*, $\prod(t)$) que é simplesmente a máxima prioridade tecto de todos os semáforos que estão fechados num dado instante:

$$\prod(t) = max(C_R : R\,est\acute{a}\ fechado\ no\ instante\ t) \quad (1)$$

Neste caso, quando activada uma tarefa poderá iniciar execução se tiver a prioridade adequada (dinâmica ou fixa) e o seu nível de preempção for superior ao tecto do sistema nesse instante. Com um raciocínio semelhante ao efectuado no caso do protocolo anterior, facilmente se demonstra que nessas circunstâncias os recursos de que uma tarefa necessita estão todos livres e a tarefa executará sem bloqueios.

Existe ainda um aspecto particular do protocolo SRP que é o facto de permitir lidar com situações em que os recursos têm múltiplas unidades, e.g., uma *pool* de *buffers* num interface de comunicação. Em situações dessas não basta que um semáforo esteja aberto, é necessário que haja um número de unidades do recurso livres suficiente para as necessidades da tarefa em questão. Por exemplo, imagine-se uma tarefa que necessita de 3 buffers livres. Não lhe servirá de nada poder aceder à *pool* de *buffers* se só estiverem dois livres nesse momento.

Neste caso a prioridade tecto de um semáforo torna-se dinâmica sendo definida como o nível de preempção mais elevado de entre as tarefas que num dado instante não têm unidades livres do recurso respectivo suficientes para os seus requisitos. Se n_R representar o número de unidades de recurso disponíveis para alocação e $\mu_R(T)$ representar o número máximo de unidades requerido pela tarefa *T* então, o tecto C_R do respectivo semáforo *R* é dado por:

$$C_R(n_R) = max[\{0\} \bigcup \{\pi(T) : n_R < \mu_R(T)\}] \quad (2)$$

Por outras palavras quando todas as unidades de alocação de *R* estão disponíveis C_R = 0. No entanto, se as unidades de *R* não são suficientes para que uma ou mais tarefas executem, i.e., as tarefas podem ficar bloqueadas nesse recurso, então C_R é igual ao maior nível de preempção dessas tarefas. Também neste caso uma tarefa só pode iniciar execução quando tiver prioridade adequada e o seu nível de preempção for superior ao tecto do sistema, $\pi_i > \prod$.

Este método impõe que, aquando da criação de uma tarefa T_i, esteja disponível a informação das suas necessidades em termos de recursos de modo a se determinar previamente os valores da prioridade tecto de cada recurso *R* em diferentes situações de alocação.

A fim de ilustrar o funcionamento do protocolo SRP, considere o conjunto de tarefas apresentado na tabela I, onde são apresentados os recursos requeridos por cada tarefa. A estrutura de execução das tarefas é apresentada na figura 4, onde *wait(R_i,n)* denota o pedido de n unidades do recurso R_i, e *signal(R_i)* representa a libertação do recurso.

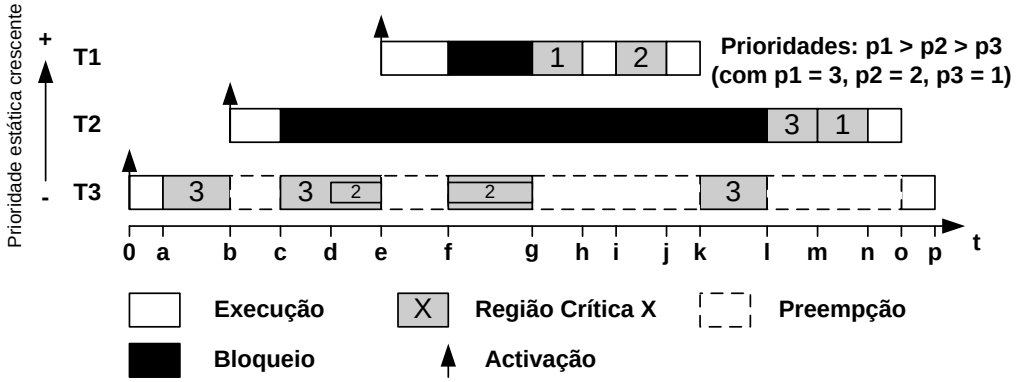

Fig. 3 - Exemplo do uso do Protocolo de Tecto de Prioridades.

	D_i	π_i	μ_{R1}	μ_{R2}	μ_{R3}
T_1	5	3	1	0	1
T_2	10	2	2	1	3
T_3	20	1	3	1	1

TABELA I

PARÂMETROS DAS TAREFAS E RECURSOS REQUERIDOS.

	Max n_R	$C_R(3)$	$C_R(2)$	$C_R(1)$	$C_R(0)$
R_1	3	0	1	2	3
R_2	1	-	-	0	2
R_3	3	0	2	2	3

TABELA II

PRIORIDADES TECTO DOS RECURSOS EM FUNÇÃO DO NÚMERO DE UNIDADES DE RECURSO DISPONÍVEIS.

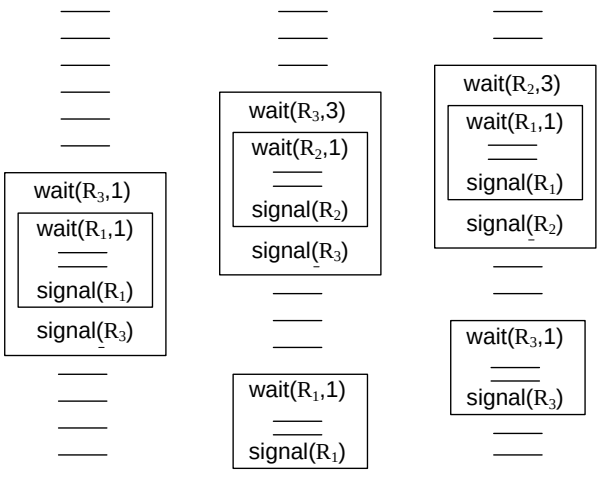

Fig. 4 - Estrutura de execução das tarefas.

Os tectos dos recursos em função do número de unidades de recurso disponíveis são apresentados na tabela II, e um possível escalonamento com o algoritmo EDF para o conjunto de tarefas é apresentado na figura 5 (nesta figura é ainda mostrada a evolução do tecto do sistema ao longo da execução das tarefas).

No instante zero, a tarefa T_3 começa a execução. O tecto do sistema mantém o valor zero porque todos os recursos estão completamente disponíveis. Quando T_3 entra na primeira região crítica apodera-se de apenas uma unidade de R_2, assim, o tecto do sistema assume o maior nível de preempção de todas as tarefas que possam bloquear em R_2 (ver tabela II) e que é $\prod = \pi_2 = 2$. Consequentemente, no instante de activação da tarefa T_2 (instante *a*), esta é bloqueada por não ter um nível de preempção superior ao tecto do sistema, o que leva T3 a continuar a sua execução. No instante *b* a tarefa T_3 entra na região crítica associada ao recurso R_1, apoderando-se de todas as suas unidades de recurso. O tecto do sistema passa a ter o valor $\prod = \pi_1 = 3$, o que provoca o bloqueio da tarefa T_1 instante *c*, visto ter um nível de preempção de 3, não superior ao tecto do sistema. Quando T_3 liberta o recurso R_1 (instante *d*), o tecto do sistema passa a ter o valor $\prod = 2$ o que permite a T_1 interromper T_3 e iniciar a sua execução. De notar que, após iniciar execução, T_1 nunca é bloqueada pois a condição $\pi_1 > \prod$ garante que todos os recursos necessários para a sua execução estão disponíveis. Quando T_1 termina, T_3 retoma a execução e liberta o recurso R_2. Ao libertar o recurso R_2, o tecto do sistema volta a zero o que provoca a preempção de T_3 por T_2 que inicia execução até terminar, sem sofrer qualquer tipo de bloqueio, pois também todos os recursos de que necessita estão disponíveis.

Relativamente aos protocolos PIP e PCP, o protocolo SRP apresenta a vantagem de ser mais versátil no sentido de possuir adequação intrínseca a prioridades fixas ou dinâmicas e a poder ser utilizado com recursos que possuem múltiplas unidades. Para além disso, concentra todos os bloqueios que uma tarefa pode sofrer no início, entre a activação e a entrada em execução, de forma que a execução se processa sem mais bloqueios. Esta propriedade, por um lado, garante a ausência de *deadlocks* e de bloqueios em cadeia, como no PCP, e por outro, permite reduzir o número de preempções, logo causando menos *overhead* no sistema. Permite ainda, como explicado no início desta secção, evitar a utilização de um *stack* individual por tarefa, podendo usar-se apenas um *stack* único resultando em menores requisitos de memória. As desvantagens são semelhantes às do protocolo PCP, i.e., não é transparente ao programador e exige mecanismos e estruturas de dados relativamente complexos.

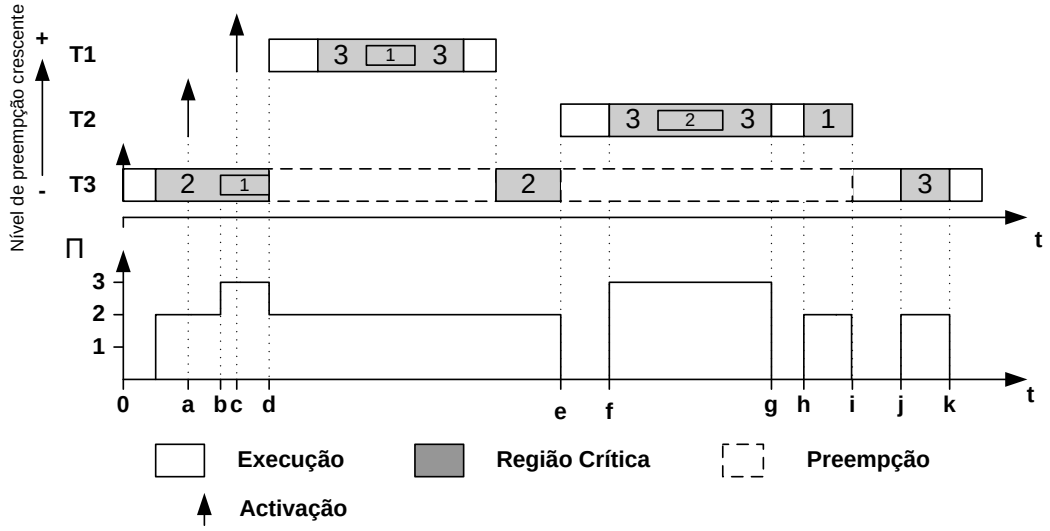

Fig. 5 - Exemplo do uso da Politica da Pilha de Recursos.

IV. CONCLUSÃO

Este artigo abordou a problemática do acesso a recursos partilhados em sistemas multi-tarefa de tempo-real, bem como as técnicas normalmente usadas para garantir exclusão mútua. Foram apresentadas algumas técnicas básicas, de fácil aplicação mas excessivamente penalizadoras, assim como algumas técnicas baseadas em semáforos, mais complexas mas também mais eficientes. Em particular, discutiu-se o funcionamento de três protocolos relativamente bem conhecidos de gestão de semáforos, nomeadamente o *Priority Inheritance Protocol*, *Priority Ceiling Protocol* e *Stack Resource Policy*. Na tabela III são apresentadas algumas características importantes destes protocolos o que facilita a respectiva comparação. Na coluna de atribuição de prioridade é de salientar o facto do protocolo SRP funcionar tanto com prioridades fixas como com prioridades dinâmicas. De notar que este protocolo efectua bloqueios apenas antes da tarefa iniciar execução, ao contrário dos outros dois, o que reduz as comutações de contexto e permite usufruir dos benefícios que daí advêm. Quanto à transparência, o PIP é o único protocolo transparente ao programador, i.e., em que o programador não tem de declarar nenhuma informação adicional para além da utilização dos semáforos respectivos, mas também o único que não previne *deadlocks*. Finalmente, note-se que todas as técnicas mencionadas têm prós e contras, devendo a sua aplicação resultar de um exercício crítico cuidado por parte do programador tendo em conta os requisitos da aplicação.

Para finalizar, é importante referir que existem outros protocolos de sincronização não abordados neste artigo, sendo alguns deles evoluções dos protocolos apresentados.

	PIP	**PCP**	**SRP**
Atribuição de Prioridade	Fixas	Fixas	Fixas ou Dinâmicas
Instante de Bloqueio	No acesso ao recurso	No acesso ao recurso	Na preempção
Transparência para o Programador	Sim	Não	Não
Prevenção de *Deadlocks*	Não	Sim	Sim

TABELA III
PRINCIPAIS CARACTERÍSTICAS DOS PROTOCOLOS DE SINCRONIZAÇÃO APRESENTADOS.

REFERÊNCIAS

[1] Giorgio C. Buttazo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Real-Time Systems Series. Springer, Berlin - Germany, 2. edição, 2004.

[2] Luís Almeida e Paulo Pedreiras, *Página da disciplina de Sistemas de Tempo-real*, DET-UA, http://sweet.ua.pt/~ lda/str/str.htm, 2005-06 edição, Set. de 2005.

[3] S. Davari e L. Sha, "Sources of unbounded priority inversions in real-time systems and a comparative study of possible solutions", *Operating Systems Review - ACM Press*, vol. 26, no. 2, pp. 110–120, Abr. de 1992.

[4] T. P. Baker, "A stack-based resource allocation policy for realtime processes", em *Proceedings of the IEEE Real-Time Systems Symposium*, Lake Buena Vista, FL, USA, Dez. de 1990, pp. 191– 200, IEEE.

[5] T. P. Baker, "Stack-based scheduling of realtime processes", *The Real-Time Systems Journal*, vol. 3, no. 1, pp. 67–100, Mar. de 1991.

[6] M. B. Jones, *What really happened on Mars?*, Microsoft, http://research.microsoft.com/research/os/mbj/Mars_ Pathfinder, Dez. de 1997.

[7] M. Chen e K. Lin, *Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems*, Real-Time Systems. Kluwer Academic Press, Norwell, MA, USA, 2. edição, 1990.