Cap. 3 – COMUNICAÇÃO ENTRE PROCESSOS

1. Como caracteriza os processsos em termos de interacção? Mostre como em cada categoria se coloca o problema da exclusão mútua.

Processos independentes: quando são criados, têm o seu tempo de vida e terminam sem interactuarem de um modo explícito. A interacção é implícita e tem origem na sua competição pelos recursos do S.C.. Trata-se tipicamente de processos lançados pelos diferentes utilizadores num ambiente interactivo e/ou dos processos que resultam do processamento de "jobs" num ambiente de tipo "batch". Neste caso, os processos competem pelo acesso a um recurso comum do S.C.. O S.O. deve intervir de modo a que a atribuição do recurso seja feita de forma controlada para que não haja perda de informação, o que impõe que apenas 1 processo tenha acesso (de cada vez) ao recurso – exclusão mútua.

Processos cooperantes: quando há partilha de informação ou comunicam entre si de um modo explícito. A partilha exige um espaço de endereçamento comum, enquanto que a comunicação pode ser feita através da partilha de um espaço de endereçamento, como da existência de um canal de comunicação que interliga os processos intervenientes. Neste caso, os processos partilham informação ou comunicam entre si, sendo da sua responsabilidade a garantia de que o acesso à região partilhada seja feita controladamente e sem que haja perda de informação. Por isso é imposto que só 1 processo (de cada vez) possa ter acesso à região partilhada.

2. O que é a competição por um recurso? Dê exemplos concretos de competição em, pelos menos, 2 situações distintas.

Fala-se em competição por um recurso quando determinados processos (independentes ou cooperantes) disputam o acesso a um recurso comum (independentes), à região (estrutura e dados) partilhada (cooperantes), ou ao canal de comunicação (cooperantes) que é visto como um recurso comum. Deste modo, 2 exemplos concretos de competição são: o "scheduler" e um gravador de CD's.

3. Quando se fala em região crítica, há por vezes alguma confusão em estabelecer-se se se trata de uma região crítica de código, ou de dados. Esclareça o conceito. Que tipo de propriedades devem apresentar as primitivas de acesso com exclusão mútua a uma região crítica?

Entende-se por região crítica como sendo as zonas de código que existem quando há processos produtor / consumidor, isto é, quando há um processo a consumir aquilo que outro produz. As regiões críticas são precisamente as regiões de código correspondentes ao produzir e consumir destes processos, pois tem de haver um sincronismo entre o produtor e o consumidor, de modo que nem o produtor produza demais nem o consumidor consuma de menos. Se houverem N processos, cada um deles com a sua região crítica, é importante que num sistema, quando um processo é executado na sua região crítica, não haja mais nenhum processo a ser executado nessa sua secção.

As propriedades que devem ser apresentadas são:

- Garantia efectiva de imposição de exclusão mútua: o acesso à região crítica associada a um mesmo recurso ou região partilhada só pode ser permitido a 1 processo de cada vez, de entre todos os processos que competem pelo acesso;
- Independência da velocidade de execução relativa dos processos intervenientes, ou do seu número;
- Um processo fora da região crítica não pode impedir outro de lá entrar;
- Não pode ser adiada indefinidamente a possibilidade de acesso à região crítica a qualquer processo que o requeira;
- O tempo de permanência de um processo na região crítica é necessariamente finito.

4. Não havendo exclusão mútua no acesso a uma região crítica, corre-se o risco de inconsistência de informação devido à existência de condições de corrida na manipulação dos dados internos a um recurso ou a uma região partilhada. O que são condições de corrida? Exemplifique a sua ocorrência numa situação simples em que coexistem 2 processos que cooperam entre si.

Têm-se condições de corrida quando, por exemplo, no caso dos produtores / consumidores, onde um produtor produz para o consumo do consumidor, em vez de se ter um produtor terem-se 2 e acontecer o seguinte: o produtor 1 produz e espera que o consumidor consuma, só que antes de isso acontecer vem o produtor 2, sobrepõe-se ao produtor 1 e passa a ser o produtor 2 quem espera pelo consumidor, passando o produto do produtor 1 a ficar inválido.

5. Distinga "deadlock" de adiamento indefinido. Tomando como exemplo o problema dos produtores / consumidores, descreva uma situação de cada tipo.

"Deadlock": quando 2 ou mais processos ficam a aguardar eternamente o acesso às regiões críticas respectivas, esperando acontecimentos que nunca vão acontecer, resultando no bloqueio das operações. Uma situação deste tipo é: o produtor produz e espera que o consumidor consuma, só que entretanto o consumidor "morre", o que implica que o produto do produtor nunca chega a ser consumido.

Adiamento indefinido: quando 1 ou mais processos competem pelo acesso a uma região crítica e, devido a um conjunto de circunstâncias em que surgem sempre novos processos que os ultrapassam, o acesso é sucessivamente adiado, estando-se por isso perante um impedimento real à continuação deles. Uma situação deste tipo é: se estiver sempre a aparecer sistematicamente produtores que se sobrepõem uns aos outros, fazendo com que o consumidor não tenha nunca oportunidade de consumir, não se realizando, portanto, o processo.

6. A solução do problema de acesso com exclusão mútua a uma região crítica pode ser enquadrada em 2 categorias: soluções ditas "software" e soluções ditas "hardware". Quais são os pressupostos em que cada uma se baseia?

Soluções "software": são soluções que, quer sejam implementadas num monoprocessador quer num multiprocessador com memória partilhada, supõem o recurso em último lugar ao "instruction set" básico do processador, ou seja, as instruções de transferência de dados de e para a memória são do tipo "standard" (leitura e escrita de um valor). A única suposição adicional diz respeito ao multiprocessador, em que a tentativa de acesso simultâneo a uma mesma posição de memória por parte de diferentes processadores é necessariamente serializada por intervenção de um árbitro.

Soluções "hardware": são soluções que supõem o recurso a instruções especiais do processador para garantir a atomicidade na leitura e escrita de uma mesma posição de memória. São muitas vezes suportadas pelo mesmo S.O. e podem mesmo estar integradas na linguagem de programação utilizada.

7. Dijkstra propôs um algoritmo para estender a solução de Dekker a N processos. Em que consiste este algoritmo? Será que ele cumpre todas as propriedades desejáveis? Porquê?

No alguritmo de dijktra, o processo que pretende aceder a R.C. fica em interessado, quando o processo pretender aceder a R.C este outro fica a espera, quando o processo cm prioridade já n pretender aceder a R.C., o processo que quer entrar na R.C. passa a ser o processo com prioridade e moda o estado para decidido.Qando este sair passa a ter prioridade o processo de índice seguinte e o que saiu fica com o estado não. Tem adiamento indefenido sempre que o processo com + prioridade quiser aceder a R.C. os outros tem que ficar a espera que esta saia.

8. O que é que distingue a solução de Dekker da solução de Peterson no acesso de 2 processos com exclusão mútua a uma região crítica? Porque é que uma é passível de extensão a N processos e a outra não (não é conhecido, pelo menos até hoje, qualquer algoritmo que o faça)?

O algoritmo de Dekker resolve o conflito de acesso entre os 2 processos a uma mesma região crítica usando o mecanismo de alternância estrita, não sendo portanto possível a sua extensão a N processos, de modo a serem cumpridas todas as propriedades, sendo que mesmo hoje em dia não se conhece nenhum algoritmo que respeite todas as propriedades desejáveis.

O algoritmo de Peterson estabelece uma prioridade baseada na ordem de chegada, forçando cada processo a escrever a sua identificação na mesma variável, e uma subsequente leitura permite por comparação determinar qual foi o último que aí escreveu. A sua generalização a N processos é quase directa, se se tiver em conta a analogia de formação de uma fila de espera.

9. O tipo de fila de espera que resulta da extensão do algoritmo de Peterson a N processos é semelhante àquela materializada por nós quando aguardamos o acesso a um balcão para pedido de uma informação, aquisição de um produto ou obtenção de um serviço. Há, contudo, uma diferença subtil. Qual é ela?

Sim é parecida so que na fila normal as pessoas ocupam todos os lugares da fila e vão avançando conforme vão sendo atendidas. Na fila de Peterson enquanto um esta a ser atendido temos os lugares da fila que conforme vão xegando vão avançando na fila, 1º chega e fica no ultimo lugar da fila, 2º chega ocupa o ultimo lugar da fila e o 1º avança para ser atendido, assim sucessivamente ate terem sido todos atendidos.

10. O que é o "busy waiting"? Porque é que a sua ocorrência se torna tão indesejável? Haverá algum caso, porém, em que não é assim? Explique detalhadamente.

O "busy waiting" é um problema comum às soluções "software" e ao uso de "flags" de "locking" implementadas a partir de instruções do tipo READ-MODIFY-WRITE, e acontece quando os processos intervenientes aguardam a entrada na região crítica no estado activo. A sua ocorrência origina:

- Perda de eficiência: a atribuição do processador a um processo que pretende acesso a uma região crítica, associada a um recurso ou região partilhada em que um segundo processo se encontra na altura no seu interior, faz com que o intervalo de tempo de atribuição do processador se esgote sem que qualquer trabalho útil tenha sido realizado;
- Constrangimentos no estabelecimento do algoritmo de "scheduling": numa política "preemptive" onde os processos que competem por um mesmo recurso, ou partilham uma mesma região de dados, têm prioridades diferentes, existe o risco de "deadlock" se for possível ao processo de mais alta prioridade interromper a execução do outro

Em S.C.'s multiprocessador com memória partilhada (mais concretamente, no caso de um multiprocessador simétrico), o problema de "busy waiting" não é tão crítico.

11. O que são "flags" de "locking"? Em que é que elas se baseiam? Mostre como é que elas podem ser usadas para resolver o problema de acesso a uma região crítica com exclusão mútua.

Quando a execução do código das regiões críticas tem uma duração relativamente curta, a alternativa de bloquear o processo enquanto aguarda uma oportunidade de acesso à região crítica, exige uma mudança de contexto para calendarizar outro processo para execução nesse processador e pode tornar-se, por isso, menos eficiente. É aqui que as "flags" de "locking" se tornam importantes, sendo também referidas por "spinlocks", em que o processo gira em torno da variável enquanto aguarda o "locking".

Ou seja, são um género de indicador de ocupação ou não da região crítica. Isto é, quando um processo acede à região crítica, a "flag" que deve estar a 0, ficando a 1 quando for ocupada. Quando ficar livre deverá voltar a 0. Se estiver a 1, outro processo não pode lá entrar.

12. O que são semáforos? Mostre como é que eles podem ser usados para resolver o problema de acesso a uma região crítica com exclusão mútua e para sincronizar processos entre si.

Um semáforo é um dispositivo de sincronização que pode ser concebido como uma variável do tipo

```
typedef struct
{
        unsigned int        val;
        NODE                *queue;
} SEMAPHORE;
```

Trata-se portanto de uma estrutura de dados complexa que tem um campo (*val*) que pode assumir 0 ou 1 (semáforo binário). É uma variável protegida que só pode ser acedida pelas operações UP e DOWN – estas operações são primitivas, pelo que são realizadas do princípio ao fim sem serem interrompidas (operações atómicas).

- DOWN:
  - Se *val* for diferente de 0, o seu valor é decrementado;
  - Se *val* for igual a 0, o processo que executou a operação é bloqueado e a sua identificação é colocada na fila de espera *queue*;
- UP:
  - Se houverem processos bloqueados na fila *queue*, um deles é acordado, senão o seu valor é incrementado.

Se vários processos tentarem um DOWN em simultâneo, apenas 1 é atendido, enquanto os outros ficam em espera, sem haver "starvation". Um semáforo só pode ser manipulado desta forma e é para garantir isso que qualquer referência a um semáforo particular é feita de forma indirecta.

13. O que são monitores? Mostre como é que eles podem ser usados para resolver o problema de acesso a uma região crítica com exclusão mútua e para sincronizar processos entre si.

Monitores são módulos especialmente construídos para a implementação da exclusão mútua. Contêm estruturas de dados e procedimentos necessários para a atribuição, com exclusão mútua, de recursos partilhados. O acesso às estruturas de dados é feito exclusivamente através de procedimentos. Embora os procedimentos de acesso às estruturas de dados internos possam ser invocados concorrentemente por diferentes processos é garantido que só 1 de cada vez pode efectivamente executá-los, ficando os restantes bloqueados.

A 1ª instrução de cada procedimento é o DOWN do semáforo de acesso à estrutura de dados correspondente. A última instrução é o UP do semáforo. Os monitores só existem em linguagens especiais, próprias para programação concorrente. A sincronização entre processos é conseguida pela introdução de variáveis de condição, que são filas de espera de processos que pretendem aceder à região crítica – solução de alto nível.

14. Que tipos de monitores existem? Distinga-os.

Monitor de Hoare: o "thread" que invoca a operação de SIGNAL é colocado fora do monitor para que o "thread" acordado possa prosseguir. É muito geral, mas a sua implementação exige a existência de uma "stack" onde são colocados os "threads" postos fora do monitor por invocação de SIGNAL.

Monitor de Brinch / Hansen: o "thread" que invoca a operação de SIGNAL liberta imediatamente o monitor. É simples de implementar mas pode tornar-se bastante restritivo porque só há possibilidade de execução de um SIGNAL em cada invocação de uma primitiva de acesso.

Monitor de Lampson / Redell: o "thread" que invoca a operação de SIGNAL prossegue a sua execução, o "thread" acordado mantém-se fora do monitor e compete pelo acesso a ele. É simples de implementar, mas pode originar situações em que alguns "threads" são colocados em adiamento indefinido.

15. Que vantagens e inconvenientes as soluções baseadas em monitores trazem sobre soluções baseadas em semáforos?

Vantagens: o principal problema do uso dos semáforos é que eles servem simultaneamente para garantir o acesso com exclusão mútua a uma região crítica e para sincronizar os processos intervenientes. Assim, porque se trata de primitivas de muito baixo nível, a sua aplicação é feita segundo uma perspectiva "bottom-up" (os processos são bloqueados antes de entrarem na região crítica, se as condições à sua continuação não estiverem reunidas) e não em "top-down". A primeira abordagem é muito confusa e sujeita a erros, sobretudo em interacções complexas, porque as primitivas de sincronização podem estar dispersas pelo programa. Os monitores são vantajosos, pois introduzem uma construção concorrente, ao nível da própria linguagem de programação, que trata separadamente o acesso com exclusão mútua a uma dada região de código e a sincronização entre processos. Quando as estruturas de dados são implementadas com monitores, a linguagem garante que a execução de uma primitiva do monitor é feita em regime de exclusão mútua. Assim, o compilador, ao compilar um monitor, gera o código necessário para impôr esta situação.

Desvantagens: o conhecimento dos monitores pelo programador deve ser especializado (tal como os semáforos), que deve ter um domínio completo dos princípios da programação concorrente. Além disso, os monitores são pouco versáteis (são construções de alto nível), são não-universais.

16. Em que é que se baseia o paradigma da passagem de mensagens? Mostre como se coloca aí o problema do acesso com exclusão mútua a uma região crítica e como ele está implicitamente resolvido.

A passagem de mensagens é um mecanismo de comunicação entre processos onde há troca de mensagens. Não exige partilha do espaço de endereçamento e é válida em monoprocessamento, multiprocessamento e processamento distribuído. Baseia-se no seguinte: sempre que o processo remetente pretende comunicar com o processo destinatário, envia-lhe uma mensagem através do canal de comunicação. O destinatário, para receber a mensagem, tem que aceder ao canal de comunicação e aguardar a chegada. Este processo só é fiável se houver sincronização. Para que a mensagem possa ser encaminhada, as operações de envio e de recepção devem ter um parâmetro que faça referência à identidade do processo interlocutor. Se a referência é explícita, tem-se endereçamento directo, senão o que é referenciado explicitamente é o canal de comunicação e o endereçamento é indirecto. Além disso, o canal de comunicação pode permitir o armazenamento intercalar da mensagem, tipicamente em caixas de correio que são estruturas de dados partilhadas que implementam filas de espera onde a ordem cronológica de chegada é, em princípio, respeitada.

17. O paradigma da passagem de mensagens adequa-se de imediato à comunicação entre processos. Pode, no entanto, ser também usado no acesso a uma região em que se partilha informação. Conceba uma arquitectura de interacção que torne isto possível.

???????

18. Que tipo de mecanismos de sincronização podem ser introduzidos nas primitivas de comunicação por passagem de mensagens? Caracterize os protótipos das funções em cada caso (suponha que são escritas em Linguagem C).

Sincronização não-bloqueante: a sincronização é da responsabilidade dos processos intervenientes. A operação de envio envia a mensagem e regressa sem qualquer informação sobre se a mensagem foi recebida. A operação de recepção regressa independentemente de ter sido ou não recebida uma mensagem.

Protótipos:
```
void msg_send_nb (unsigned int destid, MESSAGE msg);
void msg_receive_nb (unsigned int srcid, MESSAGE *msg, BOOLEAN *msg_arrival);
```

Sincronização bloqueante: quando as operações de envio e recepção contêm em si mesmas elementos de sincronização. A operação de envio envia a mensagem e bloqueia até que esta seja recebida. A operação de recepção só regressa quando uma mensagem tiver sido recebida.

Protótipos:
```
void msg_send (unsigned int destid, MESSAGE msg);
void msg_receive (unsigned int srcid, MESSAGE *msg);
```

A sincronização bloqueante pode ser dividida em 2 tipos: "rendez-vous" – quando, só após uma sincronização prévia entre os processos interlocutores, a transferência tem efectivamente lugar (ligações ponto-a-ponto); remota – quando a operação de envio envia a mensagem e bloqueia, aguardando confirmação de que a mensagem foi efectivamente recebida pelo destinatário.

19. O que são sinais? O "standard" Posix estabelece que o significado de 2 deles é mantido em aberto para que o programador de aplicações lhes possa atribuir um papel específico. Mostre como poderia garantir o acesso a uma região crítica com exclusão mútua por parte de 2 processos usando um deles.
Sugestão – Veja no manual "on-line" como se pode usar neste contexto a chamada ao sistema *wait.*

Um *sinal* constitui uma interrupção produzida no contexto de um processo onde lhe é comunicada a ocorrência de um acontecimento especial. Pode ser despoletado pelo *kernel*, em resultado de situações de erro ao nível hardware ou software, pelo próprio processo, por outro processo, ou pelo utilizador através do dispositivo de entrada / saída.

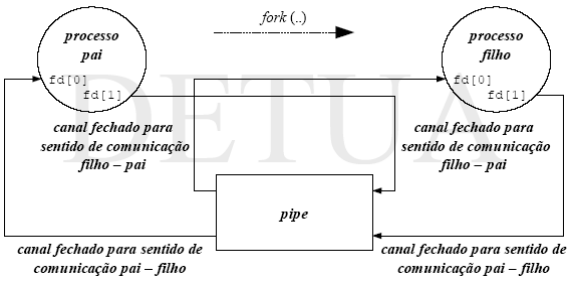
Tal como o processador no tratamento de excepções, o processo assume uma de três atitudes possíveis relativamente a um sinal; *ignorá-lo* – não fazer nada face à sua ocorrência; *bloqueá-lo* – impedir que interrompa o processo durante intervalos de processamento bem definidos; *executar uma acção associada* – pode ser a acção estabelecida por defeito quando o processo é criado ou uma acção específica que é introduzida pelo próprio processo em *runtime*.

Chamadas ao sistema que possibilitam o processamento de sinais. Destacam-se entre elas: *kill,raise* – no primeiro caso, um sinal (especificado em argumento) é enviado para um processo ou grupo de processos (especificado em argumento); no segundo, um caso particular do primeiro, um sinal (especificado em argumento) é enviado para o próprio processo; *sigaction* – registo de um função para serviço a um sinal específico; *sigprocmask,sigpending,sigsuspend* – no primeiro caso, manipulação da máscara que inibe ou activa o processamento de sinais específicos; no segundo, determinação dos sinais pendentes; no terceiro, suspensão da execução do processo até à chegada de um sinal.

20. Mostre como poderia criar um canal de comunicação bidireccional ("full-duplex") entre 2 processos parentes usando a chamada ao sistema *pipe.*

Um *pipe* é um canal de comunicação elementar que é estabelecido entre dois ou mais processos. O mecanismo de *piping* constitui a forma tradicional de comunicação entre processos em Unix, tendo sido inclusivamente introduzido por ele. Apresenta, contudo, algumas limitações: *o canal de comunicação é half-duplex* – só permite comunicação num sentido; *só pode ser usado entre processos que estão relacionados entre si* – um *pipe* é criado por um processo, que a seguir se duplica uma ou mais vezes e a comunicação é então estabelecida entre o *pai* e o *filho*, ou entre dois *irmãos*.

A sua principal utilização é na composição de comandos complexos a partir de uma organização em cadeia de comandos mais simples em que o *standard output* de um dado comando é redireccionado para a entrada de um *pipe* e o *standard input* do comando seguinte é redireccionado para a saída do mesmo *pipe*.



21. Como classifica os recursos em termos do tipo de apropriação que os processos fazem deles? Neste sentido, como classificaria o canal de comunicações, uma impressora e a memória de massa?

"Preemptable": quando os recursos podem ser retirados aos processos que os detêm, sem que daí resulte qualquer consequência irreparável à boa execução dos processos. São, por exemplo, o processador ou regiões da memória principal onde o espaço de endereçamento de um processo está alojado.

"Non-preemptable": em caso contrário. São, por exemplo, a impressora ou uma estrutura de dados partilhada que exige exclusão mútua para a sua manipulação.

O canal de comunicações e a impressora são recursos "non-preemptable" e a memória de massa é um recurso "preemptable".

22. Distinga as diferentes políticas de prevenção de deadlock no sentido estrito. Dê um exemplo ilustrativo de cada uma delas numa situação em que um grupo de processos usa um conjunto de blocos de disco para armazenamento temporário de informação.

Na prevenção de "deadlock" no sentido estrito consegue-se anular qualquer possibilidade de "deadlock", desde que se negue uma das 4 condições necessárias para a sua ocorrência:

- Não é desejável que se negue a condição de exclusão mútua;
- Obrigar o processo a requerer todos os recursos necessários antes de o deixar prosseguir a execução é uma solução que pode levar ao desperdício de recursos e pode provocar adiamento indefinido do processo;
- Se um processo necessita de recursos que não estão disponíveis, é obrigado a entregar aqueles que já estão na sua posse. Esta solução pode levar à perda de todo o trabalho realizado pelo processo e à necessidade da sua repetição, podendo também originar adiamento indefinido de um processo que nunca consiga receber todos os recursos que necessita;
- Os recursos são numerados (prioridades) e os recursos só podem pedir um recurso com prioridade superior à do último recurso pedido.

Ou seja, não há "deadlock" quando:

- Não há exclusão mútua no acesso a um recurso;
- Não há espera com retenção;
- Há libertação de recursos;
- Não há espera circular.

23. Em que consiste o algoritmo dos banqueiros de Dijkstra? Dê um exemplo da sua aplicação na situação descrita na questão anterior.

Este algoritmo consiste em atribuir um recurso a um processo se, após esta atribuição, o processo atingir um estado seguro, isto é, uma distribuição dos recursos do sistema de forma a todos eles terem a possibilidade de terminarem, ou seja, quando for possível encontrar pelo menos uma sucessão de atribuição de recursos que conduza à terminação de todos os processos que coexistem.

No caso da memória, em arquitecturas de memória virtual paginada, em que os recursos são páginas em memória, são conhecidos *a priori* os recursos, pois é conhecido o tamanho do espaço de endereçamento do processo. Assim, conhecendo o tamanho da página, é possível calcular o número de páginas necessárias.

24. As políticas de prevenção de "deadlock" no sentido lato baseiam-se na transição do sistema entre estados ditos seguros. O que é um estado seguro? Qual é o princípio que está subjacente a esta definição?

Uma alternativa menos restritiva relativamente à prevenção de "deadlock" no sentido estrito é não negar *a priori* qualquer das condições necessárias à ocorrência de "deadlock", mas monitorar continuamente o estado interno do sistema de modo a garantir que a sua evolução se faz apenas entre estados seguros.

Um estado seguro é uma qualquer distribuição dos recursos do sistema, livres ou atribuídos aos processos que coexistem, que possibilita a terminação de todos eles. Para isso é necessário o conhecimento completo de todos os recursos do sistema e cada processo tem que indicar à cabeça a lista de todos os recursos de que vai precisar, pelo que só assim se pode caracterizar um estado seguro.

25. Que tipos de custos estão envolvidos na implementação das políticas de prevenção de deadlock nos sentidos estrito e lato? Descreva-os com detalhe.

As politicas de prevenção de deadlock no sentido estrito, embora, sejam muito seguras, são muito restritivas, pouco eficientes e difíceis de aplicar em situações muito gerais, por exemplo no caso de negação da condição de exclusão mutua, só pode ser aplicada a recursos possíveis de partilha em simultâneo, na negação da condição de espera com retenção é necessário ter um conhecimento prévio de todos os recursos necessários. Na imposição da condição de não libertação, como é reposta a libertação de todos os recursos anteriores quando o próximo não puder ser atribuído, ocorrem atrasos na execução do processo. No caso da negação da espera circular são desaproveitados recursos eventualmente disponíveis que puderam ser usados na continuação do processo. As políticas de prevenção do deadlock no sentido lato, embora também sejam seguras, não são menos restritivas e ineficientes do que as no sentido estrito, no entanto são mais versáteis, logo mais fáceis de aplicar em situações gerais