Comunicação entre Processos

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

Dividem-se entre processos independentes e cooperantes.

• processos independentes – quando são criados, têm o seu 'tempo de vida' e terminam sem interactuarem de um modo explícito; a interacção que ocorre é implícita e tem origem na sua competição pelos recursos do sistema computacional; trata-se tipicamente dos processos lançados pelos diferentes utilizadores num ambiente interactivo e/ou dos processos que resultam do processamento de jobs num ambiente de tipo batch;

• processos cooperantes – quando partilham informação ou comunicam entre si de um modo explícito; a partilha exige um espaço de endereçamento comum, enquanto que a comunicação pode ser feita tanto através da partilha de um espaço de endereçamento, como da existência de um canal de comunicação que interliga os processos intervenientes.

No processo independente eles querem aceder ao mesmo recurso mas só um é que pode. Isto é da responsabilidade do SO gerir.

No processo cooperante existe uma memoria partilhada onde eles partilham a informação mas só um e que pode aceder a esse espaço. É da responsabilidade dos próprios processos garantir a exclusão mútua neste processo.

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

A competição por um recurso acontece quando dois processos distintos necessitam de aceder ao mesmo recurso. Acesso a discos rígidos, acesso ao mesmo periférico, acesso a memoria partilhada e aos diversos dispositivos de I/O.

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?

Quando falamos em acesso a um recurso, ou a uma região partilhada, estamos a referir-nos a execução de código por parte do processador, assim sendo estamos a falar de uma região de código. Uma região critica é um pedaço de código que usa recursos partilhados e que a execução desse código exige que o recurso partilhado não possa ser utilizado por outro processo com risco de alteração de informação. Logo este código não pode ser interrompido até terminar a região critica.

As propriedades são: garantir a imposição de exclusão mútua, um processo fora da região critica nao pode impedir outro de lá entrar, nao pode ser adiada indefinidamente a possibilidade de acesso a região critica a qualquer processo que o requeira, o tempo de permanência de um processo na região critica e 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 dois processos que cooperam entre si.

Condições de corrida ocorrem quando vários processos tentam aceder e manipular dados partilhados concorrentemente em que o valor final desse dados partilhados dependem do ultimo processo a lá terminar. Se estes processos cooperam quer dizer que estão a usar o acesso a região de memoria de dados partilhada e não ao recurso, logo se por exemplo um processo carregar em memoria partilhada os dados de um ficheiro e os alterar, um processo que aceda posteriormente a esses dados e os alterar, vai desfazer o que o processo anterior fez.

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

Deadlock acontece quando dois/+ processos ficam a espera eternamente de acesso a regiões criticas, e de acontecimentos que irão resultar de outros processos em espera;

Adiamento indefinido acontece quando um/+ processos tentam aceder a regiões criticas respectivas mas são ultrapassados por processos com prioridade maior sendo assim o seu acesso sucessivamente adiado.

No caso do deadlock existe problema de produtor/consumidor quando um produtor quer adicionar novos produtos na fila e a fila está cheia, como os produtores tem prioridade sobre os consumidores, consumidores não vão conseguir tirar produtos da fila criando um ciclo em que os produtores estão sempre a tentar adicionar mas não conseguem.

No caso do adiamento indefinido poderá ser causado por um aparecimento constante de acesso a região critica por parte dos produtores. Uma vez que os produtores tem sempre prioridade sobre os consumidores, estes ficam em adiamento indefinido.

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

Solução software: esta pode funcionar em qualquer processador[(mono/multi)processador], com memoria partilhada nem que em ultimo recurso se use o conjunto básico de instruções. Quando temos operações em multiprocessador torna-se necessário o uso de um arbitro, de modo a controlar a tentativa de acesso simultâneo a mesma posição de memoria.

Solução hardware: esta solução não pode funcionar em qualquer processador pois usa instruções especiais do processador, logo tem de ser implementada para o mesmo, garantindo atomicidade na leitura e subsequente escrita de uma posição de memoria. Estas são a maior parte das vezes suportadas pelo SO.

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?

O algoritmo de Dijkstra cria um array com N posições, em que cada pode ter 3 estados diferentes(NÃO, INTERESSADO, DECIDIDO). O processo que quer entrar é marcado como INTERESSADO e é feita a verificação da prioridade sobre os outros. Caso a prioridade seja sua este é marcado como DECIDIDO, caso contrario verifica se o processo com prioridade quer/não entrar passando DECIDIDO caso o processo com prioridade não queira entrar. Existe a possibilidade de este não resolver o adiamento indefinido, pois um processo pode perder sempre a disputa pela prioridade se um processo que já usou o processador desejar voltar a entrar.

8. O que é que distingue a solução de Dekker da solução de Peterson no acesso de dois 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).

Na solução de Dekker é usado um mecanismo de alternância para garantir que os processos acedam a região critica um a seguir ao outro e na solução de Dekker foi obrigar todos os processos a escrever a sua identificação na mesma variável de modo a que seja feita uma comparação de quem foi o ultimo que usou a região critica, colocando todos os processos em fila de espera por ordem de chegada. Esta permite uma implementação mais directa para para N processos que o Dekker.

Ambos são passiveis a extensão a N processos, excepto que a tentativa de expansão do de Dekker para vários processos(Dijkstra), não resolve a possibilidade de haver adiamento indefinido, não cumprindo assim todas as propriedades. Enquanto que a expansão de Dekker para N processos(“”Lamport””) não tem o problema do adiamento indefinido.

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?

A diferença é que o ultimo a chegar coloca-se na ultima posição da fila e vai verificando se a posição seguinte esta livre, avançando caso esteja.

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 o problema causado pelas soluções de software ao problema de acesso a uma região critica com exclusão mútua e pelo uso da flag de locking que indica que aquela posição de memoria esta bloqueada aquela processo podendo mesmo não estar a ser acedida. Isto implica que os processos aguardam a entrada da região critica em estado activo. Esta solução é mais indesejada nos sistemas monoprocessador, pois atribuição do processador a um processo que pretende acesso a uma região critica que esta associada a um recurso ou uma região partilhada que esta a ser acedida por outro processo, fazendo que o tempo de atribuição do processador se esgote. Esta solução e interessante e desejável em sistemas multiprocessador, em que os processadores são dedicados, tornando o busy waiting vantajoso numa situação de bloqueio.

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.

Servem para garantir atomicidade (a sua execução não pode ser interrompida) na leitura e escrita sucessivas duma variável. Como foi dito baseiam-se no uso duma variável. Os processadores tem a instrução test-and-set (tas) que permite duma maneira atómica a leitura duma posição memória, este tipo de operação usam variáveis flags de locking que após serem testados com sucesso, mudam o seu estado, qualquer processo q a seguir tentar entrar nessa região entrará em busy waiting até q o processo que fez o lock faça unlock da flag de locking.

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 inventado por Dijkstra. Este é uma estrutura de dados composta por um valor unsigned int que funciona como flag de acesso e de posição na fila e um NO para uma fila. O sistema de sincronização é feito por rotinas de sleep e de wakeup. Este utiliza estas rotinas para controlar o acesso a região critica colocando a dormir quando estes precisam de informação que outros processos ainda não disponibilizaram e acordando-os quando a informação já esta disponível.

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 dispositivos de sincronização que pode ser construído como sendo um modulo especial e pela linguagem de programação concorrente. Este é constituído por estruturas de dados interna, código de inicialização e um conjunto de primitivas de acesso.

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 um 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 DETI

imediatamente o monitor (signal é a última instrução executada); 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 pros- segue a sua execução, o thread acordado mantém-se fora do monitor e compete pelo acesso a ele; é simples de implementar, mas pode originar situ- ações 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

– suporte ao nível do sistema de operação – porque a sua implementação é feita pelo kernel, as

operações sobre semáforos estão directamente disponíveis ao programador de aplicações, constituindo-se como uma biblioteca de chamadas ao sistema que podem ser usadas em qualquer linguagem de programação;

– universalidade – são construções de muito baixo nível e podem, portanto, devido à sua versatilidade, ser usadas no desenho de qualquer tipo de soluções;

• desvantagens

– conhecimento especializado – a sua manipulação directa exige ao programador um domínio completo dos princípios da programação concorrente, pois é muito fácil cometer erros que originam condições de corrida e, mesmo, deadlock.

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?

O paradigma da troca de mensagens é feito pelo uso de um canal de comunicação. Quando um processo A pretende enviar uma mensagem ao processo B coloca-a no canal de comunicação. O processo B para receber a mensagem apenas tem de ler o canal de comunicação. Como o acesso ao canal de comunicação é feito através do processo este não sabe se lá existe mais informação de outro processo. Quando um processo envia informação pelo canal de comunicação tem de indicar o processo a quem quer enviar assim como o processo que escuta o canal de comunicação tem de indicar o seu id.

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.

Para implementar uma arquitectura que use este tipo de paradigma pode ser feito de duas formas, uma estrutura central (processa) controla o acesso informando os súbitos de se podem ou não entrar na região através de após o pedido enviar uma mensagem (ficando o processo que pediu acesso em bloqueio na recepção da resposta). Outra forma seria um sistema perguntar a toda a gente se está lá alguém a entrar. Quando receber resposta, quando sair avisa toda gente que sai, perde-se rentabilização na medida que tudo fica à espera que saia em sincronização bloqueante.

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).

Existem dois níveis:

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

/* operação de envio */ void msg_send_nb (unsigned int destid, MESSAGE msg);

/* operação de recepção */ void msg_receive_nb (unsigned int srcid, MESSAGE *msg, BOOLEAN *msg_arrival);

sincronização bloqueante - quando as operações de envio e de recepção contêm em si mesmas elementos de sincronização; a operação de envio envia a mensagem e bloqueia até que esta seja efectivamente recebida; a operação de recepção, por seu lado, só regressa quando uma mensagem tiver sido recebida;

/* operação de envio */void msg_send (unsigned int destid, MESSAGE msg);

/* operação de recepção */ void msg_receive (unsigned int srcid, MESSAGE *msg);

19. O que são sinais? O standard Posix estabelece que o significado de dois 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 dois processos usando um deles. Sugestão – Veja no manual on-line como se pode usar neste contexto a chamada ao sistema wait.

Sinal constitui uma interrupção produzida no contexto de um processo onde lhe é comunicada a ocorrência de um acontecimento especial. Este pode ser despoletado pelo kernel, pela ocurrencia de erros de hardware ou por condições especificas por software a partir do processo.

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 (conduz habitualmente à sua terminação ou suspensão de execução), ou uma acção específica que é introduzida (registada) pelo próprio processo em runtime.

Uma forma é de enviar cada sinal de entrada na região critica e saída, seria qualquer programa que tente aceder posta em wait.

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

Na comunicação full-duplex é comum usarem-se 2 pipes. Um pipe redirecciona o stdout do P1 para o stdin do P2. O outro redirecciona o stdout do P2 para o stdin do P1. Podia-se utilizar só um pipe, mas seria mais ineficiente e complexo.

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?

Um recurso é algo que um processo precisa para a sua execução. Os recursos tanto podem ser componentes físicos do sistema computacional (processadores, regiões de memória principal ou de memória de massa, dispositivos concretos de entrada / saída, etc.), como estruturas de dados comuns definidas ao nível do sistema de operação (tabela de controlo de processos, canais de comunicação, etc.), ou entre processos de uma mesma aplicação.

• recursos preemptable – quando podem ser retirados aos processos que os detêm, sem que daí resulte qualquer consequência irreparável à boa execução dos processos; são, por exemplo, em ambientes multiprogramados, o processador, ou as regiões de memória principal onde o espaço de endereçamento de um processo está alojado;

• recursos non-preemptable – em caso contrário; são, por exemplo, a impressora, ou uma estrutura de dados partilhada que exige exclusão mútua para a sua manipulação.

Assim sendo o canal de comunicação e do tipo non-preemtable, as impressoras são do tipo non-preemtable e a memoria de massa pode ser do tipo preemtable ou non-preemtable.

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.

Para evitar a ocorrência de deadlock basta que não ocorra uma das seguintes condições(visto que para acontecer deadlock tem de acontecer as 4 necessariamente):

• condição de exclusão mútua – cada recurso existente, ou está livre, ou foi atribuído a um e um só processo (a sua posse não pode ser partilhada);

• condição de espera com retenção – cada processo, ao requerer um novo recurso, mantém na sua posse todos os recursos anteriormente solicitados

• condição de não libertação - ninguém, a não ser o próprio processo, pode decidir da libertação de um recurso que lhe tenha sido previamente atribuído;

• condição de espera circular (ou ciclo vicioso) - formou-se uma cadeia circular de processos e recursos, em que cada processo requer um recurso que está na posse do processo seguinte na cadeia.

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

O algoritmo do banqueiro é executado pelo sistema operacional quando um processo de computação requisita recursos.

O algoritmo impede o deadlock, ao negar ou adiar o pedido se ele determinar que aceitar o pedido pode colocar o sistema em um estado inseguro (onde um deadlock poderia ocorrer). Quando um novo processo entra em um sistema, ele deve declarar o número máximo de instâncias de cada tipo de recurso que não pode exceder o número total de recursos no sistema.

Exemplo:

cada filósofo, quando pretende os garfos, continua a pegar primeiro no garfo da esquerda e, só depois, no da direita, mas agora mesmo que o garfo da esquerda esteja sobre a mesa, o filósofo nem sempre pode pegar nele;

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?

Define-se neste contexto estado seguro como uma qualquer distribuição dos recursos do sistema, livres ou atribuídos aos processos que coexistem, que possibilita a terminação de todos eles. Por oposição, um estado é inseguro se não for possível fazer-se uma tal afirmação sobre ele.

O principio subjacente e a diminuição da ocorrência de deadlock e para isso sao necessárias as seguintes condições:

• 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 que vai precisar – só assim se pode caracterizar um estado seguro;

• um estado inseguro não é sinónimo de deadlock – vai, contudo, considerar-se sempre o pior caso possível para garantir a sua não ocorrência.