P.: a) Que tipo de vantagens se procura retirar da implementação de uma política de 'spooling' da impressão?

b) Qual á a sobrecarga que uma tal política impõe sobre os outros recursos do sistema?

R.: a) Uma política de 'spooling' da impressora procura aumentar o 'throughput' do sistema computacional, mediando a transferência de dados para ela, que é um periférico lento, através do recurso intercalar a um periférico rápido de armazenamento temporário (disco).

b) A implementação desta política impõe sobretudo a necessidade e dar-lhe o controlo sobre a capacidade da memória de armazenamento de massa. No entanto, dado que tem que existir um novo processo, 'spooler', encarregue da transferência de dados entre a memória de massa e a impressora, tem também implicações em termos de memória principal.

P.: Explique as funlções dos vários tipos de *scheduler* do processador.

R.: As funções básicas de qualquer «scheduler» do processador são a escolha do próximo processa a executar, atribuir-lhe um «quantum» de tempo de processador e dar-lhe o controlo sobre o processador. Quando o processa é interrompido deve determinar se está bloqueado ou se terminou a execução e colocá-lo no estado correcto, escolhendo de seguida o próximo processo a ser executado.

Em sistemas com definição de prioridades o «scheduler» pode ainda ter de reajustar a prioridade do processo em função da ocupação do último quantum que lhe foi atribuído, bem como a duração do próximo quantum.

P.: Em que consiste o 'aging' de um processo numa atribuição dinâmica de propriedades?

R.: Em qualquer sistema onde se mantém processos à espera enquanto ele faz a alocação de recursos e decisões de *scheduling* de processos, é possível adiar indefinidamente o *scheduling* de um processo enquanto outros recebem toda a atenção por parte do sistema. Como se sabe chama-se a este fenómeno *adiamento indefinido*. Isto pode ocorrer quando os recursos são organizados com base numa atribuição dinâmica de prioridades, já que é possível que um processo tenha que esperar indefinidamente enquanto outros processos com uma prioridade mais elevada vão sendo servidos.

Sendo assim o *aging* consiste no aumento linear da prioridade de um dado processo enquanto espera por um dado recurso. Assim a prioridade desse processo poderá, eventualmente, exceder a prioridade de todos os outros processos que chegam e será por isso «atendido».

P.: Descreva os objectivos mais importantes que devem ser satisfeitos numa política de *scheduling* do processador. Discuta as diferenças existentes em termos de tempo de resposta entre uma definição estática e dinâmica de prioridades.

O *scheduling* do processador tem com finalidade determinar quando é que um processo deverá ser atribuído e a qual processo. Como o processador é um recurso muito importante do sistema o *scheduller* tenta optimizar a sua utilização, mantendo-o tanto o quanto possível ocupado.

Os objectivos mais importantes são:

- *JUSTIÇA* - garantir que cada processo tem direito à sua parcela de tempo do CPU
- *THROUGHPUT* - maximizar o número de processos complet ados por unidade de tempo
-*TURN AROUND* - minimizar o tempo médio de espera dos utilizadores *batch*
-*TIME DE RESPOSTA* - minimizar o tempo méd de resposta aos utilizadores interactivos
-*EFICIÊNCIA* –minimização do tempo associado com a selecção do proximo processo a que vai ser atribuido o processador e c/ a comutação propriamente dita
-PREVISIBILIDADE – o tempo de execução de um processo deve ser razoavel/ constante e independente da sobrecarga pontual a que o sistema computacional possa estar sujeito.
-DEADLINES – deve procurar garantir-se o máximo de cumprimento possível das "deadlines" impostas pelos processos em execução

Numa definição estática de prioridades é atribuído uma prioridade fixa no inicio da execução e vai-se decrementando esse valor no final de cada *quantum*. Quando chega a zero é reposto o valor inicial. Dado esta característica é de esperar que se um processo tiver uma prioridade inicial baixa então o tempo de resposta será bastante elevado, podendo o processo por isso nunca ter acesso ao CPU (*starvation*).

No caso de prioridades dinâmicas é definida a prioridade em função da parcela de ocupação do último *quantum* atribuído, pelo que como os processos respeitantes aos utilizadores interactivos têm bastante operações de I/O será de esperar que a parcela seja efectivamente menor que o *quantum* pelo que a prioridade será elevada. Neste caso temos tempo de resposta menores e será difícil (ou mesmo impossível) entrar em *starvation*.

P.: Descreva os objectivos mais importantes que devem ser satisfeitos em geral numa política de 'scheduling' no processador. Quais de entre eles são absolutamente indispensáveis em sistemas operativos de tipo interactivo ('time-sharing'). E em sistemas operativos de tipo 'batch'?

R.: O tempo de resposta é naturalmente o objectivo mais crítico a ter em conta em sistemas interactivos. Adicionalmente, são ainda de considerar o throughput e a eficiência.

Em sistemas do tipo 'batch', deve-se particularmente ter em conta o tempo de turn-around, o 'throuput' e a eficiência

P.: Que vantagens traz na definição de prioridades entre os diferentes processos que coexistem num dado sistema computacional, uma atribuição de dinâmica sobre uma atribuição estática.

R.: Numa definição estática de prioridades é atribuído uma prioridade fixa no inicio da execução e vai-se decrementando esse valor no final de cada *quantum*. Quando chega a zero è reposto o valor inicial. Dado esta característica é de esperar que se um processo tiver uma prioridade inicial baixa então o tempo de resposta será bastante elevado, podendo o processo por isso nunca ter acesso ao CPU (*starvation*).

No caso de prioridades dinâmicas é definida a prioridade em função da parcela de ocupação do último *quantum* atribuído, pelo que como os processos respeitantes aos utilizadores interactivos têm bastante operações de I/O será de esperar que a parcela seja efectivamente menor que o *quantum* pelo que a prioridade será elevada. Neste caso temos a vantagem sobre a definição estática de prioridades de que o tempo de resposta é menor e será difícil (ou mesmo impossível), por isso, entrar em *starvation*.

P.: O que distingue uma disciplina de 'scheduling' 'preemptive' de uma 'nonpreemtive'? Qual delas será de esperar encontrar-se em sistemas operativos de tipo 'batch'? E em sistemas operativos do tipo interactivo("time-sharing")?

R.: Numa disciplina de tipo 'preemptive' o sistema operativo tem a possibilidade de retirar o processador ao processo que o detém, por acção determinada de um dispositivo externo, normalmente o relógio de tempo real (RTC); enquanto que numa disciplina do tipo nonpreemptive o processo que mantém a posse do processador conserva-o até bloquear ou terminar.

Em sistemas do tipo 'batch' a disciplina a usar é do tipo non-preemptive porque se procura atribuir o processador de forma a minimizar o tempo de execução de um grupo de tarefas. Logo, não faz sentido comutar processos no estado RUN. (A única excepção surge quando o processo ultrapassa o tempo total de processador que lhe foi atribuído).

Em sistemas operativos do tipo interactivo, a política de 'scheduling' a utilizar é a do tipo 'preemptive', pois pretende-se estabelecer uma multiplexagem temporal da ocupação do processador pelos diferentes processos.

P.: a)Explique porque se torna normalmente conveniente num sistema computacional onde coexistem diferentes processos estabelecer uma política de 'scheduling' do acesso a disco.

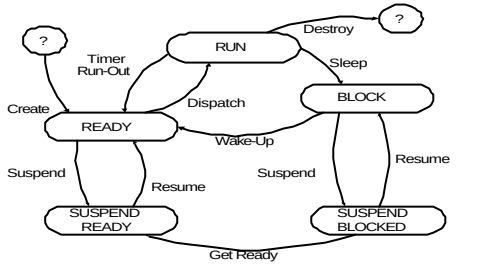
R.: A razão de ser da necessidade do estabelecimento de uma política de 'scheduling' de disco num sistema multiprogramado tem a ver com a diferença de velocidade (várias ordens de grandeza) entre o processamento de informação pelo processador e a sua transferência de e para o disco. Daqui resulta que, em média, vários processo estão bloqueados aguardando operações de I/O para o disco . Como o problema se coloca em termos de deslocamentos mecânicos das cabeças de leitura e escrita sobre a superficie do disco, torna-se absolutamente crítico ordenar os diferentes pedidos de acordo com critérios de proximidade relativa para se minimizar o tempo de resposta e a predicabilidade do sistema.

P.: b) Neste sentido, caracterize as diferenças existentes entre as políticas de 'scan', 'n-step scan' e 'circular scan' e indique qual delas acha mais vantajosa.

R.: A politica de 'scan' supõem que as cabeças de leitura e escrita se deslocam do interior para a periferia do disco e vice-versa num movimento sucessivo e repetido. Os pedidos que chegam são ordenados por distância às cabeças na direcção do deslocamento (os + próximos 1º). Esta estratégia reduz a descriminação da zona central e das extremidades, continuando no entanto a zona intermédia a ser previligiada. Nesta estratégia a previsibilidade é melhorada.
Na politica de 'n-step scan' os movimentos das cabeças são semelhantes aos da estratégia anterior, mas durante o deslocamento só são atendidos os pedidos existentes aquando do inicio desse deslocamento. Os pedidos que chegam entretanto são ordenados para serviço óptimo quando o deslocamento fôr em sentido contrário. É uma solução com bom throughput e tempo de resposta e apresenta uma previsibilidade melhor que o 'scan'. A zona intermédia continua no entanto a ser previligiada.
Na politica de 'circular scan' a cabeça desloca-se da extremidade para o centro do disco e vai servindo por ordem de proximidade os pedidos que vão chegando. Uma vez chegada ao interior retorna rapidamente à extremidade sem servir nenhum pedido, isto é, só são servidos os pedidos no deslocamento p/ o interior. Com esta estratégia deixa de haver zonas favorecidas no disco. Obtem-se uma excelente previsibilidade.
Para baixas cargas a melhor estratégia é o scan enquanto que para cargas elevadas é o circular scan

P.: a)Porque é que em muitas situações concretas se torna conveniente separar a política de 'sched uling' do processador em dois níveis? Justifique em termos do diagrama de transição de estados entre processos

R.: Em muitas situações concretas, podem coexistir num sistema multiprogramado mais processos do que aqueles que correntemente estão armazenados em memória principal. Nestas circunstâncias, o código e os dados dos restantes processo são temporariamente transferidos para uma extensão em memória de massa designada genericamente pelo nome de área de *swapping*.
Esta organização exige o estabelecimento de um diagram de estados para os processos em dois níveis. No primeiro, correspondente aos estados ditos activos, a disciplina de *scheduling* visa gerir a atribuição do processador aos diferentes processos residentes em memória principal, optimizando, portanto, a utilização do processador; no segundo, correspondente aos estados ditos passivos, a disciplina de *scheduling* visa gerir a transferência do código e dados dos diferentes processos entre as memórias principal e de massa, optimizando, portanto, a ocupação da memória principal.



P.: a) Qual é o principal inconveniente ligado com as soluções software do problema de imposição de exclusão mútua no acesso de dois processos a uma região crítica.

b) Porque é que esse problema se encontra objectivamente resolvido na maioria das soluções de hardware.

R.: a) O principal inconveniente tem a ver com a necessidade de eles competirem pelo acesso à região no estado activo, busy waiting. O que é pouco eficiente porque, quando o processador é atribuido ao processo em espera, nenhum trabalho util é realizado.

b) A principal caracteristica das soluções de hardware é pro vidienciarem mecanismos que possibilitam estender o âmbito das operações atómicas. Concretamente, a comutação de processos pode ser impedida por inibição das interrupções, ou os processos em espera podem ser bloqueados através de dispositivos de tipo semáforo ou monitor.

P.: Explique em que condições se torna necessário garantir a exclusão mútua entre processos.

R.: A exclusão mútua deve-se verificar quando existem processos que pretendem aceder a dados partilhados. Quando um processo está a aceder a dados partilhados (por vários processos) diz-se que está a entrar numa região crítica. Neste caso todos os processos que pretenderem entrar nessa região crítica têm de ser impedidos de lá entrarem, pois se o processo q está a aceder à região fôr interrompido as consequências são imprevisiveis.

P.: O que é o *deadlock*? E um adiamento indefenido? Destinga as duas situações. Descreva sucintamente as condições necessárias de ocorrência de 'deadlock'.

R.: .: 'Dead lock' é uma situação em que um processo se encontra bloqueado, aguradando a ocorrência de um acontecimento que NUNCA vai surgir. Adiamento indefinido é uma situação em que um processo pode em principio terminar, mas o acesso a um recurso de que necessita é-lhe sucessivamente negado por processos de prioridade mais alta.
As condições necessárias para a ocorrência de *deadlock* são:
Condição de Exclusão Mútua - os processos obtêm controlo exclusivo sobre os recursos que requerem.
Espera com Retenção ('wait for condition) - os processos retêm os recursos já a eles alocados enquanto esperam por recursos adicionais
Condição da Não Libertação ('no preemption condition') - os recursos não podem ser removidos dos processos que os retêm até que os recursos sejam utilizados até ao fim
Condição de Círculo Vicioso ou Espera Circular - existe um encadeamento circular de processos em que cada processo segura um ou mais recursos que por sua vez são requisitados pelo seguinte processo no círculo que por sua vez retém outros recursos sendo estes requisitados por outro processo, etc.

P.: Distinga entre políticas de prevenção de 'deadlock' no sentido estrito e no sentido lato. Em qual delas se situa a aplicação para a alocação de recursos do algoritmo dos banqueiros de Djikstra?

R.: A política de prevenção de *deadlock* no sentido estrito (*prevention*) tem como finalidade a negação de uma das quatro condições que originam *deadlock* para deste modo poder eliminá-lo:
-Negação do princípio de exclusão mútua, garantindo que um recurso pode ser acedido por um ou mais processos ao mesmo tempo. Mas isto não se pode fazer, já que se não se garantir a exclusão mútua não se pode proteger uma área de dados ou um programa que será partilhado por um ou mais processos.
-Negação da condição de espera. Se um processo verifica que não consegue a atribuição do próximo recurso, liberta-os todos e recomeça mais tarde ou então faz-se simplesmente com que o processo só possua um único recurso de cada vez.
-Negação da condição de libertação. Um processo liberta um recurso quando se verifica a requisição deste último. Pode trazer problemas caso o processo esteja a meio da utilização pretendida para o recurso (p.exp. impressão).
-Negação da espera circular. Alterando o tipo de estrutura, em vez de ser circular, hierarquiza-se a estrutura.
Sendo a prevenção no sentido estrito muito radical, recorre-se às políticas de prevenção no sentido lato que se baseiam numa politica de monitorização da atribuição de recursos, tendo como base o número de recursos possuídos por cada utilizador (o máximo de recursos que cada um pode vir a precisar e os recursos que se encontram disponíveis). No caso do algoritmo dos Banqueiros de Djikstra trata-se de uma política de prevenção de *deadlock* no sentido lato.

P.: O que é uma região crítica? Em que consiste o princípio da exclusão mútua no acesso de diferentes processos a uma região crítica?

R.: Região crítica é uma região de código ou de programa que não deve ser partilhada por vários processos ao mesmo tempo. Por exemplo, numa área de dados onde se faz a leitura e escrita de informação. Neste caso se um processo estiver a ler a informação a informação que lá está não pode ser alterada ao mesmo tempo por um outro processo.
O princípio da exclusão mútua consiste no impedimento de acesso de qualquer processo a uma região crítica quando essa estiver a ser acedida por outro valor falso.

P.: Os algoritmos de Dekker e Peterson são duas soluções so ftware do problema de exclusão mútua no acesso de dois processos a uma região crítica. Qual é a diferença fundamental entre eles? Indique alternativamente duas soluções hardware para o mesmo problema. Explique o seu princípio de funcionamento.

O algoritmo de Dekkers resolve o conflito de acesso entre os dois processos competidores usando um mecanismo de alternância estrita. O algorítmo de Peterson estabelece uma prioridade baseada na ordem de chegado isto é, acede à região critica o processo que fôr + rápido a fazer o acesso.

Uma solução de *hardware* para o problema de exclusão mútua de dois processos a uma região crítica é a instrução *test and set* , que tem o seguinte funcionamento:
- ler uma variável
- guardar o seu valor numa área segura
- colocar a variável com um determinado valor

Estes passos devem ser executados sem interrupção. A variável possui um valor Verdade se algum processo estiver na região crítica, caso contrário possui um valor Falso.
Outra solução de hardware é o "disable" dos "interrupts". O disable dos interrupts evita a comutação de processos, pelo que as regiões críticas são executadas sem interrupções.

P.: Considere uma situação em que num dado sistema computacional coexistem m processos partilhando n recursos idênticos. A alocação e libertação dos diferentes recursos efectua-se sempre um de cada vez. Por outro lado, nenhum processo precisa nunca mais de n recursos e a necessidade global máxima de recursos é sempre inferior a n+m. Mostre porque nesta situação nunca ocorre 'deadlock'.

R.: Dado que cada processo precisa pelo menos de um recurso e como a necessidade global de recursos é sempre inferior a m+n, isso significa que o número total de recursos acima do primeiro que os diferentes processos necessita, é no máximo de n-1.
Sejam I os processos que necessitam de mais di que um recurso, tem-se assim que a sua necessidade global de recursos é I+n+1. Ora como os (m-I) processos que nessecitam apenas de um recurso terminam sempre, os restantes I têm inevitavelmente à sua disposição m+n+1 (>I+n-1) recursos. E, portanto, também terminarão sempre.

P.: Diga o que entende por monitor e indique as vantagens da sua utilização para resolver a exclusão mútuta.

R.: O monitor é um módulo que contem as rotinas e a inf ormação necessária à manipulação de 1 (ou vários) recurso(s) do sistema potencialmente partilhado por vários processos. O monitor constitui uma solução de alto nível para o problema da exclusão mutua entre processos, dado que além de permitir a entrada de 1 processo de cada vez, a informação que contem é apenas vista pelos processos que lhe têm acesso e é invisivel aques que aguardam a entrada.
Uma das vantagnes dos monitores é o uso de "condition variables" que simplificam o sincronismo entre periféricos. Os programas usando monitores são também + simples de escrever. Como a informação dentro do monitor está bem protegida o sistema torna-se seguro e fiável.

P.: Defina semáforo. Explique como é que os semáforos permitem garantir a exclusão mútuta.

R.: Um *semáforo* é uma variável protegida que só pode ser acedida e alterada pelas operações P e V (como exemplo) havendo ainda uma operação de inicialização. Representamos a operação P sobre o semáforo S por P(S) e do mesmo modo a operação V sobre S por V(S). As operações P e V são indivisíveis. P(S) pode ser implementado da seguinte maneira:

if S>0
then S=S-1
else *espera em S*

e a operação V(S):

if *mais um processo espera em S*
then *deixa prosseguir um dos processos*
else S=S+1

A exclusão mútua no semáforo S é assegurada com as operações P(S) e V(S). Se vários processos tentarem um P(S) (operação para entrar na região crítica) simultaneamente, apenas a um será dada a autorização para prosseguir. Os outros processos terão de esperar, mas a implementação de P e V devem garantir que os processos não sofram de adiamento indefinido.

P.: O que entende por processo? O que é o *Process Control Block*? Que tipo de informação contém?

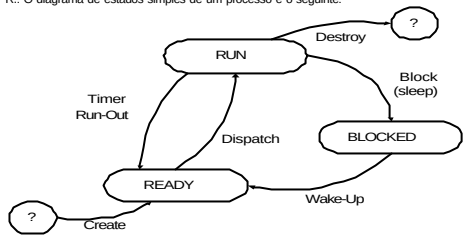
R.: Existem várias definições de *processo* tais como as que se seguem:
-um programa em execução
-uma actividade assíncrona
-a entidade com o qual o processador está ocupado
Contudo a mais utilizada é do *programa em execução*. Um programa pode ser definido como uma seq. de instruções que descrevem a realização de uma determinada tarefa por 1 PC. P/ q essa tarefa seja de facto realizada o programa correspondente tem q ser executado. A execução de um programa designa-se por processo.
O PCB (*Process Control Block*) é uma estrutura de dados usada intensivamente pelo "scheduler" para fazer a gestão do processador e de outros recursos do sistema computacional. Contem toda a informação importante que caracteriza o processo, incluindo:
-o estado actual do progresso
-a identificação do processo
-um ponteiro para o *pai* do processo (isto é, para o processo que criou este)
-um ponteiro para o *filho* deste processo (isto é, o processo criado por este processo)
-a prioridade do processo
-ponteiros para a localização da memória do processo
-ponteiros para os recursos alocados
-uma área de salvaguarda para os registos
-o processador sobre o qual ele corre (isto num sistema multiprocessador).

P.: Qual é a difereça que existe entre a sincronização bloqueante remota e bloqueante *rendez-vous* na comunicação entre processos por passagem de mensagens.

R.: Na sincronização bloqueante remota, o remetente, após o envio da mensagem, bloqueia aguardando confirmação da sua recepção por parte do destinatário. Na sincronização bloqueante *rendez-vous*, a mensagem só é transferidam após sincronização prévia entre o remetente e o destinatário.

P.: Desenhe o diagram de estados básico de um processo, indicando os tipos de eventos que provocam a mudança de estado. Explique quais as modificações introduzidas no diagrama básico num sistema que utiliza *swapping*.

R.: O diagrama de estados simples de um processo é o seguinte:

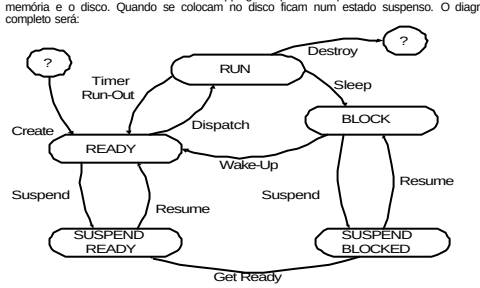


Todos os processos que vão chegando são colocados no fim duma lista onde todos os processos que estão *ready* são colocados. Quando chega ao topo da lista será o próximo processo a ser executado quando o CPU estiver disponível. Chegando a altura acontece uma transição do estado *ready* para o estado *run*. Chama-se à disponibilização do CPU para o primeiro processo da lista *dispatching* e é efectuado por uma entidade do sistema chamado *dispatcher*.

Para prevenir a monopolização do sistema por parte de um processo, seja ele acidentalmente ou maliciosamente, o sistema operativo coloca um *hardware interrupting clock* (ou *interval timer*) para permitir ao utilizador correr durante um intervalo de tempo específico. Se o processo não «largar» o CPU voluntariamente antes de o tempo acabar, o *interrupting clock* gera um *interrupt* de modo que o sistema operativo fique outra vez com o controlo. O sistema operativo obriga depois o processo a passar do estado *run* para o *ready* através de um *timerrunout* e faz o *dispatch* (*ready* para *run*) do processo que se segue na lista. No estado de RUN se o processo precisar de aceder a algum recurso que não está imediatamente disponível bloqueia-se (sleep). Fica então no estado BLOCKED. Quando o recurso fica disponível o processo é deslocado(wakeup) e fica pronto para uma nova execução(ready). Voltará ao CPU quando fôr a sua vez(dispatch).

Note-se que a única transição de estado provocada pelo processo utilizador é *sleep*, as outras todas são provocadas por entidades externas ao processo, além disso note também que se podem considerar mais duas transições de estado que acontecem quando o processo terminar a sua execução sendo por isso retirado para fora do CPU (*destroy*) e quando se acrescenta um novo processo (*create*) sendo este colocado no estado *ready*.

Em sistemas com «swapping» os processos podem ser comutados entre a memória e o disco. Quando se colocam no disco ficam num estado suspenso. O diagram completo será:



P.: Em que consiste o princípio de exclusão mútua no acesso de diferentes processos a uma região crítica? Indique sucintamente que pressupostos devem ser tomados em conta na implementação de primitivas de exclusão mútua.

R.: O princípio de exclusão mútua no acesso de diferentes processos a uma região crítica consiste em se garantir que, de cada vez, o acesso à região em causa é efectuado por um e um só processo.
A implementação das primitivas de acesso deve cumprir os seguintes pressupostos:
- a exclusão mútua no acesso deve ser efectivamente garantida;
- nada deve ser assumido relativa/ à velocidade relativa dos diferentes processos que competem pelo acesso;
- qnd os processos estão fora da região crítica, não podem impedir os outros de lá entrarem;
nenhum processo, em nenhuma circunstância, deve ser colocado numa posição de espera indefinida.
- o tempo de permanencia de um processo na região crítica é necessariamente finito

P.: Distinga concorrência e paralelismo e explique qual destes dois conceitos é fundamental na implementação de um sistema multiprogramado.

R.: Diz-se que há paralelismo num sistema computacional quando mais do que um processo é excutado simultaneamente, o que implica a existência de mais do que um processador; diz-se que há concorrência, por outro lado, quando a atribuição do(s) processador(es) a diferentes processos é multiplexado no tempo, criando a ilusão de paralelismo na excução de um grupo de processos. Um sistema multiprogramado é um sistema onde coexistem diferentes processos excutados em concorrência.

P.: Em que consiste a multiprogramação ?

R.: A multiprogramação consiste na coexistência num dado sistema computacional de diferentes processos em memória central que são executados em time-sharing, isto é de forma concorrente por 1 unico processador. Ela garante um melhor aproveitamento do recurso 'processador', procurando assim eliminar os tempo mortos que ocorrem durante as transacções de informação com os dispositivos de entrada/saída. Sendo assim o tempo de processamento de um grupo de tarefas em sistemas de tipo 'batch' pode ser minimizado.

P.: O que é que distingue um sistema multiprogramado de um único utilizador de um sistema multiprogramado multi-utilizador? Dê exemplos de cada um deles.

R.: Os sistemas multiprogramados de utilizador único são caracteristicos dos computadores pessoais e de muitos sistemas de tempo real. Neles é suposto existir um único dispositivo de entrada/saída (terminal) através do qual se realiza a comunicação com o sistema. O Windows é talvez um exemplo destes.
Os sistemas multiprogramados multi-utilizador visam, como foi referido atrás, a partilha em simultâneo do sistema computacional por vários utilizadores, cada um deles comunicando com o

sistema através do seu próprio dispositivo de entrada/saída (seja ele um terminal local ou remoto). Este sistema permite a execução de tarefas de vários utilizadores em simultâneo, de forma concorrente ou //. Temos como exemplo o UNIX.

A necessidade de criar a ilusão aos diferentes utilizadores do sistema computacional que o partilham em simultaneo ou ao longo do tempo, de dedicação plena, exige a introdução de mecanismos especiais de autenticação e protecção que se tornam evidentes, quer a nivel de gestão de processos, quer da gestão das memórias principal e de massa.

Desde logo, o acesso ao sistema exige a identificação (user name (login) para que ele possa ser redirecionado para o respectivo directório de trabalho. A inviolabilidade da informação contida nos ficheiros aí existentes é mantida através de um esquema de protecções mais ou menos elaborado que estabelece uma hierarquia dos utentes com permissões de acesso. Por outro lado, os processos têm agora que ser identificados com uma relação de pertença clara, para que os utilizadores individuais tenham pleno controlo sobre os seus próprios processos e o sistema operativo possa discriminar o acesso aos diferentes recursos do sistema. Finalmente, a protecção de memória principal é absolutamente crítica, porque se torna impiedoso garantir que nenhum processo de um utilizador particular possa corromper de uma forma acidental a área de código ou de dados associada a qualquer outro processo.

P.: O que distingue a multiprogramação do multiprocessamento?
R.: Na multiprogramação temos o processamento no tempo (multiplexagem no tempo), daí que vários programas são processados por um único processador. No multiprocessamento temos processamento no espaço, isto é, as diversas tarefas são distribuídas por vários processadores.

Sistema computacional multiprogramado é um sistema computacional onde a um mesmo processador são atribuídos processos diferentes, ou seja, o processador executa diferentes programas de uma forma concorrente.

A multiprogramação tem como função, em geral, optimizar a gestão do processador. Nos sistemas de tipo 'batch', ela manifesta-se na minimização do tempo de 'turn-around', impondo o bloqueio dos processos durante as operações de I/O. Nos sistemas de tipo interactivo, ela constitui o elemento essencial da sua operacionalidade, dado que se procura criar em cada utilizador ligado ao sistema a ilusão de que o processados lhe está inteiramente dedicado.

P.: O que é um sistema computacional multiprogramado? Explique como é que esta característica se manifesta em sistemas operativos de tipo «batch» e em sistemas operativos de tipo interactivo ('time-sharing').
R.: Sistema computacional multiprogramado é um sistema computacional onde a um mesmo processador são atribuídos processos diferentes, uo seja, o processador executa diferentes programas de uma forma concorrente.

A multiprogramação temo como função, em geral, optimizar a gestão do processador. Nos sistemas de tipo 'batch', ela manifesta-se na minimizaçãodo tempo de 'turn-around', impondo o bloqueio dos processos durante as operações de I/O. Nos sistemas de tipo interactivo, ela constitui o elemento essencial da sua operacionalidade, dado que se procura criar em cada utilizador ligado ao sistema a ilusão de que o processados lhe está inteiramente dedicado.

P.: Que diferença fundamental existe entre uma organização de memória real e um organização de memória virtual. Qual das duas escolheria no caso de um sistema de tempo real?
R.: Uma organização de memoria real pressupõe o armazenamento da totalidade do código de dados na memória principal. Num sistema de memória real os endereços referenciados por um programa em execução correspondem a endereços físicos na memória principal da máquina, pelo que existe uma correspondência biunívoca entre endereços lógicos e físicos. Este é o modo + simples de distribuição de memória pois só 1 processo de cada vez possui a memoria principal. Verifica-se portanto um desperdicio de recursos e não se podem executar programas > que o tamanho da memoria.

Num sistema de memória virtual o espaço dos endereços lógicos gerados por um programa é, ou poderá ser, maior que as dimensões físicas da memória principal do sistema, havendo por isso uma desassociação dos endereços referenciados por um processo que está a correr dos endereços disponíveis na memória principal. É construida em memoria de massa uma imagem do processo e em cada instante carrega-se para a memoria principal apenas uma uma parte deste. Dado a não correspondência biunívoca endereço lógico/físico, os sistemas de memória virtual necessitam de um mecanismo chamado *Dynamic Adress Translation* (DAT) que é efectuado em tempo real de execução dos programas e que traduz cada endereço lógico no correspondente endereço físico mediante o uso de tabelas residentes em memória principal, em memória cache, ou ainda em memória associativa.
Esta tradução conduz a um inevitável *overhead* pelo que para um sistema de tempo real escolheria uma organização de memória real, pois como um sistema de tempo real terá que ser muito rápido é necessário que todo o processo se encontre em memória principal para que a sua execução esteja dentro do tempo de resposta estabelecido.

P.: Explique o funcionamento de um sistema de memória virtual. Qual é a importância que organizações de memória virtual têm em sistemas de tipo interactivo ('time-sharing')?
R.: Em sistemas de memória virtual um processo é fraccionado em blocos,de tamanho fixo ou variável, pelo compilador sendo em cada bloco os endereços contíguos. Para se executar um programa, os blocos em que este se divide vão sendo carregados em memória real. Os sistemas que usam blocos de tamanho fixo são designados por sistemas paginados, enquanto que os sistemas que usam blocos de tamanhos variáveis são designados por sistemas segmentados.
Para descodificar o endereço virtual o processador tem q, com o nº de pagina, aceder à tabela de mapeamento onde consulta a flag R. Se esta estiver activa lê o endereço base e coloca-o no registo base para gerar o endereço físico. De cada vez q gera 1 endereço este é descodificado pelo MMU. Se se quiser aceder à memoria que não foi carregada em memória principal (flag R não activa) tem que ser gerado um interrupt *memory page fault*, porque o CPU não pode aceder directamente á memoria de massa.
A medida que os processos vão acabando vão sendo abertos buracos na memória e enquanto que nos sistemas paginados o seu uso é trivial (uma página velha é simplesmente substituída por outra) nos sistemas segmentados têm que recorrer a várias estratégias de substituição dos segmentos:

-*Best-Fit* - um processo que chega é colocado na memória principal no buraco onde melhor cabe e onde, por isso, deixará a menor quantidade de espaço não utilizado.
-*First-Fit* - um processo que chega à memória principal é colocado no primeiro buraco suficientemente grande que encontra
-*Worst-Fit* - neste caso coloca-se o processo na memória principal no buraco maior possível, sendo assim, mesmo quando o processo for lá colocado eventualmente ainda restará espaço suficiente para albergar outros processos

Se ao fim de algum tempo o somatório dos buracos de memória livres for maior ou igual que ao necessitado pelo processo, não havendo contudo nenhum buraco suficientemente grande para albergar o processo, será efectuado uma *garbage-collection* ou *compactação* que consiste em mover todos as regiões de memória ocupadas para um dos extremos da memória principal. A protecção da área de memoria de memoria neste tipo organização é conseguida mediante o uso de dois registos fronteira que contêm os endereços limite da zona de memoria principal contigua atribuida ao processo. Só os endereços gerados dentro deste limite são considerados válidos, todos os outros dão origem a uma interrupção do tipo memory fault. Cada referência é testada de modo a validar ou não a sua restrição aos limites contidos nos registos fronteira.

Nos sistemas paginados há uma melhor utilização da memória principal.

Em sistemas do tipo *time-sharing* nos quais os processos comutam quando ficam bloqueados ou quando ocorre o fim do tempo que lhes foi atribuído, a organização de memória virtual é importante, pois são sistemas que possibilitam a utilização simultânea por vários utilizadores, logo é necessário fazer uma dissociação completa do espaço em relação ao sistema de memória real, possibilitando assim a partição de dados em 'pedaço' a colocar de cada vez na memória principal, o que não seria possível numa organização de memória real.

P.: Descreva a organização de um sistema hierarquizado de memória. Justifique a importância dos diferentes níveis num sistema multiprogramado.
R.: Um sistema hierarquizado de memória tem três níveis: a memória 'cache', a memória principal e a memória de massa. As diferenças entre niveis estão relacionadas com a capacidade de armazenamento e o tempo de acesso (baixos no caso da 'cache' e elevados no caso da memória de mass). O processador só tem capacidade de acesso às memórias 'cache' e de massa.

Num sistema multiprogramado, vários programas estão a ser executados em concorrência, o que implica a ter a sua carga total ou parcial na memória principal. Para que a capacidade deste tipo de memória não constitua um impedimento ao número máximo total de processos que coexistem, convém muitas vezes deslocar alguns deles totalmente ou parcialmente temporáriamente para a memória de massa. Tem-se assim a possibilidade de executar em concorrência um número de programas que ultrapasse o espaço de armazenamento (capacidade) da memória principal. Por outro lado como a memória 'cache' é a mais rápida esta deve ser utilizada para armazenar partes de código ou estruturas de dados mais frequentemente referenciadas.

P.: Descreva uma organização de memória real com partições de tamanho variável. Indique como nela podem ser resolvidos os problemas de carga de processos e de protecção de áreas de memória pertencentes a diferentes processos.
R.: Numa organização de memória real compartições variáveis, a dimensão de cada partição é exactamente aquela necessária ao armazenamento contíguo do código e estruturas de dados próprios de cada processo. Faz-se assim uma optimização do espaço de memória. Porém à medida que os processos existentes vão sendo terminados e outros vão sendo criados, a memória vai-se fragmentando e pode ocorrer uma situação em q, embora a memória total disponível seja superior à necessária para carregar um novo programa, não exista nenhuma zona de endereços contíguos com a dimensão suficiente, o q exige a implementação de mecanismos de compactação ('garbage collection').
De um modo geral existem, no entanto, existem sempre um ou mais blocos (de endereços contiguos) disponíveis onde o código e as estruturas de dados associados a um novo processo podem ser carregados. A selecção de um entre de eles faz-se geralmente com um dos seguintes três critérios:
- first-fit - escolhe-se o primeiro bloco com dimensão suficiente
- best-fit - escolhe-se o bloco de dimensão mais pequena
- worst-fit - escolhe-se o bloco de dimensão maior.

A protecção da área de memória pertencente a cada processo é habitualmente feita através de dois registos fronteira que contêm os endereços limite da zona de memória principal contígua atribuída ao processo. Só os endereços gerados dentro deste intervalo são considerados válidos, todos os outros dão origem a uma interrupção do tipo 'memory-fault'.

P.: a) Descreva uma organização de memória real com partições de tamanho fixo. Indique como nela podem ser resolvidos os problemas de carga de processos e de protecção de áreas de memórias pertencentes a diferentes processos.
P.: b) Num sistema deste tipo as diversas partições podem ser todas do mesmo tamanho ou estarem agrupadas em conjuntos de tamanho diferentes. Comente sobre as vantagens e inconvenientes de cada uma destas opções.
R.: a) Num sistema com partições fixas a memória principal (real) é dividida em partições de tamanho fixo. Cada partição pode segurar uma única tarefa. Aqui o CPU poderá então comutar rapidamente entre os utilizadores para criar a ilusão de simultaneidade. As tarefas são traduzidas com assemblladores e compiladores absolutos para que corram numa partição específica. Se a tarefa estiver pronta a correr e a sua partição estiver ocupada, então a tarefa terá que esperar mesmo que outras partições estejam disponíveis.

O carregamento pode ser efectuado por compiladores relocáveis, assembladores e *loaders* que são utilizados para produzirem programas relocáveis, que assim poderão correr em qualquer partição disponível que seja suficientemente grande, neutralizando assim a desvantagem de as tarefas (programas) terem que correr sempre na sua respectiva partição.

A protecção é implementada utilizando vários registos de limite. Com dois registos pode-se assim guardar os limites inferior e superior da partição, ou então o limite inferior (ou superior) e o tamanho da região que pode ser endereçada.
R..b) No caso das partições serem variáveis as tarefas poderão ocupar o espaço que quiserem (isto é, de que necessitam), não havendo por isso limites fixos nas partições. Neste caso as tarefas simplesmente não podem exceder a memória total disponível, além disso a partição terá então sempre o mesmo tamanho da tarefa que contém.

Nas partições fixas existe a desvantagem de o tamanho das tarefas a executar estar limitado ao tamanho das partições fixas, podendo também haver desperdício se as tarefas necessitarem de menos espaço que aquele disponibilizado por cada partição. Nas partições variáveis e por isso de tamanho diferente o desperdício só se começa a verificar quando as tarefas depois de terminarem deixarem âcunas na memória principal (real). Estas lacunas poderão depois ser utilizadas por outras tarefas, só que as lacunas poderão, eventualmente, exceder em tamanho o espaço das lacunas. Contudo pode-se sempre ocorrer a tecnicas de compactação no caso das partições variáveis tal como o *garbage-collection* para evitar o desperdício de memória neste caso.

P.: Há quem afirme que, em última análise, a estratégia 'first-fit' de carga de processos em memória não é mais do que uma estratégia de carga puramente aleatória. Comente.
R.: Esta afirmação é falsa porque, numa estratégia puramente aleatória, a probabilidade de escolha de um qualquer dos blocos disponíveis e de dimensão suficiente é igual para todos os blocos, só que numa estratégia 'first-fit' é sempre escolhido o primeiro bloco de dimensão suficiente. Contudo ela pode ser considerada verdadeira sob o ponto de vista de que não é possivel distinguir entre si os estados de fragmentação da memória resultantes de cada uma das duas situações.

P.: Qual é o papel desempenhado pela memória de massa numa organização de memória virtual?
R.: Numa organização de memória virtual, o espaço de endereçagem do processo (código+dados+stack) está totalmente dissociado da memória principal do sistema computacional. Durante a sua execução, só a parte deste espaço está em cada momento aí residente. O papel da memória de massa é, portanto, manter uma imagem actualizada do espaço completo do processo, visando a transferência de blocos específicos de e para a memória principal. Constitui aquilo que se designa por *swapping*.

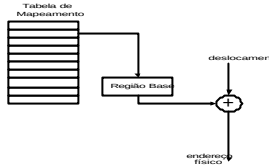
P.: O que é que se procura garantir neste tipo de organizações com a política de substituição de blocos?
R.: O grande objectivo da política de substituição de blocos é conseguir implementar uma estratégia de substituição que seja tão próxima quanto possivel do principio da optimalidade. Ou seja, após a ocorrência de uma tentativa de acesso a um vloco não residente ('block fault'), o bloco a substituir deve ser aquele que , de entre todos os actualmente residentes, será referenciado mais tarde.

P.: Ao considerar-se a implementação de uma organização de memória virtual num sistema computacional, pode-se eventualmente optar entre sistemas de segmentação e de paginação puros. Explique as diferenças entre eles e indique que tipos de organização de memória real lhes são correspondentes.
R.: Num sistema de memoria virtual 1 dos factores a ter em conta na sua implementação, é o tamanho da área de memoria destinada a mapeamento. Assim opta-se geralmente por um mapeamento por blocos de informação. Estes blocos podem contudo ter tamanho fixo ou variável dando origem aos 2 sistemas de mapeamento citados: paginação e segmentação respectiva/. Num sistema paginado o tamanho dos blocos é fixo pelo que a memoria atribuida a um dado processo pode ser constituida por vários blocos em nº suficiente para conter a informação. Num sistema de segmentação os blocos são feitos tão grandes quanto necessário, conforme o tamanho do processo e informação associada. O mapeamento por segmentação é + natural pois a cada segmento corresponde uma entidade conceptual logica, e não física como no sistema paginado. A protecção de informação e a sua partilha é assim facilitada. Existe um certo paralelismo entre estes 2 tipos de organização de memoria virtual e os sistemas de partições fixas e variáveis em sistemas de mem real.

P.: O que é que caracteriza o conceito de "working set" como uma politica de gestão do espaço de memória física atribuída a um processo.
R.:O conceito de WS refere-se ao conjunto de páginas que devem ser carregadas em memoria principal para que o processo seja executado de forma eficiente. Este conceito está relacionado com o conceito de localidade que refere q os sistemas tendem a referenciar áreas de armazenamento segundo padrões bem localizados e uniformes, quer no tempo, quer no espaço fisico de memoria. Aproveitando-se desta propriedade inerente aos processos, as politicas de "working set" defendem q as páginas de memória referenciadas pelo processo devem, dentro do processo, ser mantidas em memória primária para otpimizar a execução do mesmo.

P.: Descreva uma organização de memória virtual paginada. Indique como nela se efectua a tradução do endereço virtual para um endereço físico e se resolve o problema da protecção de áreas de memória pertencentes a diferentes processos.
R.: Em memória virtual a informação está dividida em blocos. Numa organização de memória virtual paginada essa informação está armazenada em blocos de tamanho fixo. É mantido 1 registo, para cada processo que indica quais os blocos que se encontram em memória real. Os endereços virtuais destes blocos contêm o nº do bloco em questão e o deslocamento a partir do inicio deste bloco. Para além disso, o CPU mantém um registo base onde está o endereço do inicio da tabela de paginação. A tradução do endereço virtual em endereço real é feita assim: ao endereço base é somado o nº do bloco. Do endereço resultante fica-se a conhecer a entrada da tabela de blocos do processo correspondente aquele bloco. Esta entrada dá o endereço em memória principal do bloco em questão. Se a este endereço somarmos o deslocamento temos o endereço em memória. Para tornar + rápida a tradução dos endereços, a tabela de paginação pode ser armazenada em memória cache ou em memória associativa.

Para haver + protecção das páginas de memoria dos diferentes processos é necessario acrescentar + informação à tabela de paginação. Há então 1 registo com o nº total de paginas que o processo tem. Se o nº da página somado com o endereço base der 1 valor superior ao nº total de paginas ocorre "memory fault"



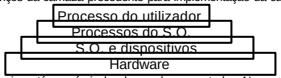
P.: Em que consiste o princípio da optimalidade na substituição de páginas em memória principal. Descreva as estratégias de substituição mais conhecidas e indique qual delas se aproxima mais do principio da optimalidade.
R.: c) O principio de optimalidade manifesta-se no tempo, já que o processo que foi referenciado mais recentemente será o referenciado num futuro próximo, e no espaço, já que o processo que foi referenciado indica-nos que os processos que estão mais perto desse são os que têm mais probabilidades de serem referenciados num futuro mais próximo.

O principio da optimalidade é usado quando a memória fica cheia e queremos carregar mais blocos da memória de massa, tendo o sistema operativo que decidir qual dos blocos residentes em memória irá ser retirado, seguindo os seguintes critérios:
R.: 1º aquele que não necessite de ser utilizado;
2º se todos necessitarem de ser utilizados retira-se aquele que venha a ser utilizado mais tarde.

Quando não se sabe qual é que vai ser utilizado mais tarde retira-se aquele que foi usado menos vezes assumindo que a probabilidade de acesso é distribuída, logo a sua probabilidade de ser usado será menor (gera menos *page-faults*). Isto não é viável porque pode ter sido menos utilizada anteriormente podendo ainda a ser mais utilizada no futuro tendo que ser depois novamente carregada.
As estratégias de substituição de páginas em memória principal são:
- FIFO - quando uma página preciso de ser substituída escolhes-se aquela que foi armazenada há mais tempo.
- LRU - *Least Recently Used* - Escolhe-se a página que foi usada há mais tempo. Necessita de uma implementação em cada página para se saber o tempo.
- LFU - *Least Frequently Used* - Escolhe-se a página menos frequentemente utilizada, mas pode-se escolher a página errada se se escolher a página recentemente colocada.
- NUR - *Not Used Recently* - Escolhe a página que não foi usada mais recentemente.

A estratégia LFU é aquela que se aproxima mais da optimalidade.

P.: Caracterize em poucas palavras a função exercida pelo sistema operativo num sistema computacional. Considere as duas perspectivas: 'top-down' e 'bottom-up'.
R.: O sistema operativo é um programa que gere o acesso aos vários recursos da maquina, por parte dos utilizadores, de forma eficiente e segura. É o programa que dá "vida" à maquina possibilitando a criação de um ambiente de interacção com o(s) utilizador(es) para a realização de trabalho util. A sua concepção é normalmente abordada pela decomposição do sistema em camadas funcionais, as quais implementam uma máquina virtual. Cada camada ou máquina virtual utiliza os serviços da camada precedente para implementação da camada seguinte:



Na base da hierarquia está o próprio hardware do computador. Na camada seguinte está o sistema operativo constituído pelo núcleo (funções mais importantes) e por funções, além dos dispositivos do sistema (p.exp., driver de impressora). No nível acima estão os processos gerados pelo S.O. e ultimo nível as aplicações do utilizador.

TOP-DOWN

Numa perspectiva *top-down*, o sistema operativo faculta um interface para uma máquina virtual que é conceptualmente mais simples, ou seja, cria uma máquina virtual de fácil utilização. Num sistema multiprogramado, por exemplo, cada utilizador trabalha diante de um terminal como se houvesse um CPU para cada um, ou seja, uma máquina para cada um. Portanto, neste ponto de vista consideram-se os programas de aplicação ao nível das aplicações do sistema, uma vez que para aceder a ficheiros estes programas têm que uma interacção directa com o sistema operativo. Esta é a perspectiva do utilizador do sistema.

BOTTOM-UP

Numa perspectiva *bottom-up* partimos do hardware da máquina e vamos subindo de nível até chegarmos às aplicações dos utilizadores. Deste modo o sistema operativo tem como função administrar as operações de um computador, ou seja, controlo todos os recursos de um sistema computacional (processador, memória principal, memória de armazenamento em massa e os dispositivos de I/O). O sistema operativo deve gerir todos estes recursos para que sejam partilhados de forma eficiente e segura por todos os utilizadores do sistema computacional. Esta é a perspectiva do gestor do sistema.

P.: Indique algumas das características que permitem considerar um dado sistema operativo como um sistema operativo de rede.
R.: Um sistema operativo pode ser considerado como um sistema operativo de rede na medida em que fazendo uso da criação de determinados programas (software) e da sua 'integração' com o hardware permite a partilha de ficheiros, transferência de ficheiros, a posse de um login remoto que permita a ligação a outra máquina sem abandonarmos o nosso sistema computacional, impressão remota bem como correio electronico. Além disso porque também permite a comunicação entre o sistema e o utilizador, como por exemplo através de mensagens de erro.

P.: Considere uma situação em que a um dado sistema computacional de tipo 'batch' chegam quase simultaneamente 5 'jobs', A, B, C, D, E, com respectivamente, tempos estimados de execução de 10, 6, 2, 4, 8 minutos e com prioridades 3, 5, 3, 2, 4 (em que 5 é a maior). Suponha ainda que todos os 'jobs' são CPU-intensivos e que o tempo de comutação de processos é desprezável. Calcule nestas condições para os seguintes algoritmos de 'scheduling' o tempo de 'turn around' do referido lote:
i) 'Scheduling de prioridade'
ii) First-come, first served (executados na ordem A, B, C, D, E);
iii) Shortest job first.
R.:

A	B	C	D	E
10	6	2	4	8 tempo de execução

O *turn-around* é p tempo médio de espera dos utilizadores 'batch' (até à exaustão).

i) *Scheduling* de prioridade:

B	E	A	C	D
6	8	10	2	4
0	6	14	24	26 tempo de espera

tempo total de espera = 70 minutos -> 70:5 = 14 minutos

ii) FIFO:

A	B	C	D	E
10	6	2	4	8
0	10	16	18	12 tempo de espera

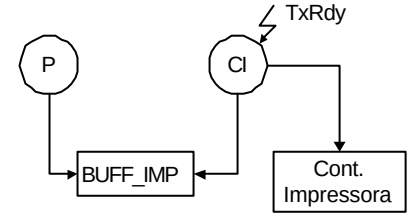
tempo total de espera= 66m logo 13m2seg

iii) SJF - maior prioridade ao que se despacha mais depressa

C	D	B	E	A
2	4	6	8	10
	2	6	12	20

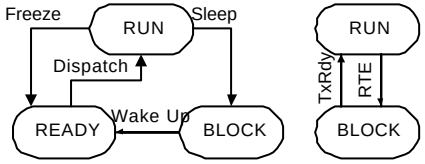
tempo total = 40m logo 40:5=8

P.: a) Uma das vantagens que a multiprogramação introduz é desacoplar completamente os processos da gestão específica dos dispositivos de entrada/saída. Explique esquematicamente como isso é conseguido para um exemplo de comunicação com uma impressora onde se pretende escrever linhas de texto. Admita que existe apenas um processo-utilizador envolvido.
P.: b) Construa o diagrama de estados e das respectivas transições associado com todos os processos envolvidos na situação anterior.
P.: c) Estenda agora o modelo anterior para o caso em que existem múltiplos processos-utilizador. Uma condição essencial a garantir agora é que a informação escrita na impressora é proveniente dos diferentes processos-utilizador intervenientes durante o seu processo de escrita.
R.: a) A figura a seguir descreve de uma maneira esquemática a situação:



O Processo P é o processo-utilizador que, em momentos particulares da sua execução pretende escrever na impressora linhas de texto. Para isso socorre-se duma 'chamada ao sistema' (void escreve_linha (char *linha)) que transfere para o 'buffer' de comunicação BUFF_IMP a linha em questão. O processo CI é um processo de sistema, acticado pela interrupção TxRdy do registo de transmissão do controlador da impressora. Quando é acordado, este processo procura ler um caracter, pertencente a uma linha armazenada em BUFF_OMP e escreve-lo no registo de transmissão da impressora. Para socorre-se de outras duas 'chamadas de sistema': char lê_caracter e void transm_caracter (char car). Temos então que sempre que tiver uma linha para imprimir, o processo P invoca o procedimento *escreve_linha* que transfere para o 'buffer' de comunicação a linha em questão. O seu regresso é imediato desde que haja espaço de armazenamento suficiente, caso contrário o processo bloqueia. Na situação particular em que o 'buffer' de comunicação esteja inicialmente vazio, o procedimento escreve_linha realiza a acção suplementar de transferir o primeiro caracter para o registo do controlador da impressora, procedimento transm_caracter, despoletando assim o mecanismo de interrupções que vai acordar sucessivamente o processo CI, até que toda a informação tenha sido transferida. Por sua vez, o processo CI, quando é acordado pela interrupção, invoca a função le_caracter para recolher um caracter previamente aramazenado em BUFF_IMP e, caso exista algum, escreve-o no registo de transmissão do controlador da impressora, procedimento transm_caracter. Na situação particular, em que o buffer de comunicação esteja inicialmente cheio , a função le_caracter realiza a acção suplementar de tentar acordar o processo P, eventualmente bloqueado a aguardar espaço de armazanamento disponivel no buffer.

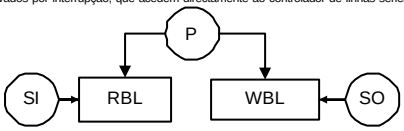
b)A figura a seguir ilustra os diagrams de estados e respectivas transições para cada um dos processo en volvidos na situação anterior:



O processo P tem um diagrama de estados convencional. As transições Sleep e WakeUp são originadas, respectivamente, pelo down e up do semáforo. Note que a transição Freeze supõe a perda do processador pelo processo P não só por esgotamento de um eventual 'time slot' que lhe tivesse sido atribuído, mas também pela entrada em acção do processo CI. O processo CI é um processo activado por interrupção. No modelo mais simples associado, só dois estados: o estado RUN, quando a interrupção TxRdy é servida, e o estado BLOCK, resultado da exaustão do processo.

c) A obtenção da impressora em regime de exclusão mútua pelos diferentes processos-utilizador exige o estabelecimento de um semáforo de acesso ao 'buffer' de comunicação. Assim, cada processo-utilizador terá agora que solicitar a impressora ao sistema operativo, antes de iniciar a impressão e libertá-la após a sua utilização. Por outro lado, para evitar acessos indevidos, torna-se vital identificar o processo a quem foi concedido o recurso, de outra forma poderia ainda acontecer que não solicitassem a impressora e procurassem de imediato escrever linhas, corrompessem a informação que está a ser ou irá ser impressa.

P.: Uma das vantagens que a multiprogramação introduz é desacoplar completamente os processos da gestão específica dos dispositivos de entrada/saída. Explique esquematicamente como isso é conseguido para um exemplo de comunicação com uma linha séria onde se pretende ler e escrever linhas de texto.
R.: O modelo apresentado supõe que o processo utilizador, P, comunica directamente com dois *buffers*: um de leitura de linhas, RBL, e outro de escrita, WBL. A informação é depositada ou recolhida de cada um destes *buffers*, caracter a caracter, por dois processos de sistema, SI e SO, activados por interrupção, que acedem directamente ao controlador de linhas série.



O processo SI é acordado por acção da interrupção Rxdy quando o *buffer* de recepção do controlador contém um caracter, caso em que lê e o deposita a seguir no *buffer* RBL. Se existir um processo bloqueado no *buffer*, este é acordado logo que uma linha de texto esteja completa. O processo SO é acordado por acção da interrupção Txdy quando o *buffer* de transimissão do controlador está vazio, caso em que recolhe um caracter do *buffer* WBL e o escreva aí. Se o *buffer* WBL estiver na altura vazio, nenhum caracter pode ser enviado e esta situação é assinalada numa variável associada ao *buffer*. Por outro lado, se existir um processo bloqueado no *buffer*, este é acordado logo que esteja disponível espaço de armazenamento considerado suficiente.
Assim para ler uma linha de texto, o processo P invoca uma rotina de leitura que acede ao *buffer* RBL e a recolhe. Se no momento não existir uma linha completa, o processo é bloqueado. Para escrever uma linha de texto, o processo P invoca uma rotina de escrita que acede ao *buffer* WBL e aí deposita informação, iniciando ou não a transferência para o controlador, conforme o estado da variável associada. Caso o *buffer* WBL esteja cheio, o processo é bloqueado.

P.: Considere um sistema computacional onde o acesso às impressoras se efectua por um mecanismo de *spooling*. Admita duas estratégias de operação:
i) o processo gestor só começa a imprimir um ficheiro quando toda a informação se encontra disponível
ii) o processo gestor só começa a imprimir um ficheiro logo que se encontre uma impressora disponível
a) Compare estas duas estratégias e explique quais são as vantagens e inconvenientes associadas a cada uma delas.
b) Assumindo que a primeira estratégia foi a escolhida, proponha um método que permita ao processo gestor determinar se a informação de um dado ficheiro se encontra já disponível.
c) Assumindo que a segunda estratégia foi a escolhida proponha um método que permita ao processo gestor determinar se toda a informação para impressão foi já enviada.

a) Na estratégia i) assim que a informação se encontra completa o processo gestor deverá então sondar se existe alguma impressora disponível e em caso afirmativo deve proceder à impressão do ficheiro. A vantagem deste processo é que uma vez iniciada a impressão ela vai até ao fim. O processo retém o recurso, a impressora, mas está a fazer uso deste. No fim da impressão do ficheiro a impressora ficará livre. A desvantagem poderá ser o tempo de espera por uma impressora livre. Se os ficheiros a imprimir forem extensos, a impressão será demorara, podendo o número de ficheiros aumentar. Como consequência o tempo de espera poderá ser longo.
Na estratégia ii) logo que se encontre uma impressora livre é iniciada a impressão de um ficheiro mesmo que este não esteja completo. A vantagem será a de que o recurso (impressora) esteja sempre ocupada excepto na mudança de ficheiros. Se a informação chegar ao ficheiro a uma velocidade superior à da impressão há uma boa utilização. Se a informação chegar ao ficheiro não de uma forma sequênciada mas intervalada (de tempos a tempos é acrescentada informação ao ficheiro), e se esse intervalo for longo, poderá a informação existente ser impressa e o recurso ficaria retido à espera de mais informação para ser impressa, sendo está hipótese por isso uma desvantagem.

b) Um método possível seria quando o ficheiro estiver pronto, o mecanismo que gerou o ficheiro alterar a sua máscara de permissões. O mecanismo de *spooler* teria então de indagar primeiro as máscaras de permissões dos ficheiros.
Outro método seria quando o ficheiro estivesse pronto copiar para a directoria de *spooling* a referência do ficheiro, que depois de impressa seria apagada. Assim podemos fazer mais impressões.

c) Um método possível é existir um caracter que indica o fim do ficheiro. O processo gestor deverá verificar cada caracter que vai para a impressora e quando surgir o caracter de fim de ficheiro a impressão teria sido já enviada.
Outra estratégia seria a existência de um mecanismo na impressora que sinaliza-se ao processo gestor quando recebesse o caracter de fim de ficheiro.

P: Considere um sistema computacional onde a maioria dos programas que aí são executados, são executados em *batch* e exigem a consulta e a modificação simultânea de um número variável de bandas magnéticas (ao longo do tempo e diferente de programa para programa). O sistema tem disponível para esse fim 10 unidades de leitura de banda que podem estar continuamente operacionais. Admita duas estratégias de operação:
i) o sistema operativo exige que o programa lhe indique à partida o número de unidades de banda que vai necessitar e atribui-as imediatamente em bloco;
ii) o sistema operativo exige que o programa lhe indique à partida o número de unidades de banda que vai necessitar e atribui-as à medida que vão sendo efectivamente necessárias
a) Compare estas duas estratégias e explique quais são as vantagens e inconvenient4es associadas a casda uma delas.
b) Assumindo que a primeira foi a escolhida, proponha um método que minimize o tempo de *turn-around*.
c) Assumindo que a segunda foi a escolhida, proponha um método que minimize o tempo de *turn-around*.
R.: a) A estratégia i) é praticamente a negação da condição de espera para a existência de *deadlock*, enquanto que a ii) poderá ser uma utilização do algoritmo dos banqueiros de Dijkstra.
Uma desvantagem da estratégia descrita em ii) será a má gestão de recursos (unidade de leitura de banda) já que mesmo que o programa não necessite imediatamente de todas as unidades de leitura ele irá retê-las, e nenhum outro programa mais necessitado poderá utilizá-lás. Note-se também que devido à remoção de qualquer possibilidade de existência de *deadlock* (prevenção no sentido estrito) o sistema se torna muito «pesado» não sendo este método por isso utilizado em sistemas de *general purpose* mas sim em sistemas de tempo real.
A estratégia ii) é normalmente utilizado em sistemas de *general purpose*. Note que aqui apenas se pretende evitar que ocorra *deadlock* (prevenção no sentido lato) não implicando por isso a possibilidade da sua existência, daí que também é menos pesada que a i). No entanto, tem como desvantagem o fixar do número de recursos a alocar (existe um número máximo de unidades de leitura), além de fixar também o número de utilizadores, só que por vezes é impossível conhecer com antecedência as exigências em termos de recursos.
b) Para minimizar o *turn-around* (tempo de resposta a utilizadores *batch*) utiliza-se quer FIFO quer SJF (*shortest-job-first*), que é uma disciplina de *scheduling nonpreemptive*, são comumente utilizados em sistemas *batch*. No entanto, como o FIFO tem um tempo médio de espera superior, não garantindo por isso bons *turn-around*, a solução a utilizar será o SJF que minimiza o tempo médio de espera, prejudicando assim os processos mais longos.
c) Uma vez que o espaço em memória é mais limitado deve-se utilizar o SRT (*shortest-remaining-time*), assim se um processo estiver a correr poderá ser interrompido por outro que demorará muito menos tempo a acabar, melhorando assim significativamente o *turn-around*. Além disso, reduz-se assim também o tempo médio de espera, só que os mais longos serão ainda mais prejudicados.