1. Em que consiste a multiprogramação?

A multiprogramação consiste na coexistência num dado sistema computacional de diferentes processos partilhando o mesmo processador.
As razões que conduziram à sua introdução radicam-se na tentativa de garantir um melhor aproveitamento do recurso processador, procurando eliminar os tempos mortos que ocorrem durante as transacções de informação com os dispositivos de entrada/saída. Deste modo o tempo de processamento de um grupo de tarefas ”jobs” em sistemas tipo ‘batch’ pode ser minimizado.

2. Processos independentes vs processos cooperantes

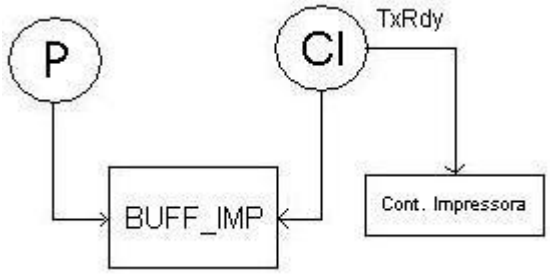
Num ambiente multiprogramado, os processos que coexistem podem ter comportamentos diversos em termos de interacção:
•**processos independentes** - são processos que são criados, têm o seu ‘tempo de vida' e terminam sem interactuarem de um modo explícito; a interacção que ocorre é implícita e tem origem na sua competição pelos recursos do sistema computacional; exemplos de processos deste tipo são os processos lançados pelos diferentes utilizadores num ambiente interactivo e/ou os processos que resultam da execução de 'jobs' num ambiente 'batch';
•**processos cooperantes** - são processos que, de um modo explícito, partilham informação ou comunicam entre si; a partilha exige a existência de um espaço de endereçagem comum aos processos intervenientes, enquanto que a comunicação pode ser feita tanto através da partilha de um espaço de endereçagem, como da existência de um canal de comunicação que ligue os processos.

3. Programa vs Processo

De um modo geral, um programa pode ser definido como uma sequência de instruções que descrevem a realização de uma determinada tarefa por um computador. Contudo, para que essa tarefa seja de facto realizada, o programa correspondente tem que ser executado.
A instanciação (ou execução) de um programa designa-se por processo e representa a realização dessa tarefa.
Tratando-se da representação de uma actividade que está a decorrer, o processo caracteriza-se em cada instante por:
• O código do programa respectivo e o valor actual de todas as variáveis associadas (espaço de endereçagem);
• O valor actual de todos os registos internos do processador;
• Os dados que estão a ser transferidos para os dispositivos de entrada-saída;
• O seu estado de execução.

4. 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.

A figura abaixo descreve de uma maneira esquemática a situação proposta.



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 da chamada ao sistema
- void escreve_linha (char *linha)
que transfere para o buffer de comunicação BUFF_INP a linha em questão.
O processo CI é um processo de sistema activado pela interrupção TxRdy do registo de transmissão do controlador da impressora. Quando é acordado, este processo procura ler um carácter, pertencente a uma

linha armazenada em BUFF_IMP, e escrevê-lo no registo de transmissão da imperssora. Para isso socorre-se das duas chamadas ao sistema seguintes:

- char le_caracter ()
- void transm_caracter (char car)

Em termos gerais, a interacção estre dois processos pode ser descrita da seguinte forma: Sempre que tiver um alinha para imprimir, o processo P invoca o procedimento *escreve_linha* que transfere para o buffer de comunicação BUFF_IMP 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 carácter para o registo de transmissão do controlador da impressora, procedimento *transm_caracter*, despontando assim o mecanismo de interrupções que vai acordar sucessivamente o processo CI, até que toda a informação tenha sido efectivamente transferida.
Por sua vez o processo CI, quando é acordado pela interrupção, invoca a função *le_caracter* para recolher um carácter previamente armazenado 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 armazenamento disponível no buffer.

A titulo elucidativo, apresenta-se a seguir uma implementação

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

Os sistemas multiprogramados de utilizador único são característicos dos computadores pessoais e de muitos sistemas de tempo real. Neles, é suposto existir um único dispositivo de entrada/saída standard (terminal) através do qual se realiza a comunicação com o sistema. O Windows da Microsoft é talvez o exemplo mais divulgado hoje em dia, deste tipo de sistemas na área dos computadores pessoais.
Os sistemas multiprogramados multi-utilizador visam, como foi referido atrás, a partilha em simultâneo do sistema computacional por um leque mais ou menos alargado de utentes, cada um deles comunicando com o sistema através do seu próprio dispositivo de entrada/saída standard (seja ele um terminal local, ou um terminal remoto). O Unix ATT constitui um exemplo popular deste tipo de sistemas.
A necessidade de criar a ilusão aos diferentes utilizadores do sistema computacional que o partilham em simultâneo ou ao longo do tempo, de dedicação plena, exige a introdução de mecanismos especiais de autentificação e protecção que se tornam evidentes, quer a nível da gestão de processos, quer da gestão das memórias principais e de massa.
Desde logo, o acesso ao sistema exige a identificação do utente (“login”) para que ele possa ser redireccionado 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 da memória principal é absolutamente crítica, porque se torna imperioso 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.

6. Os sistemas de operação de uso geral actuais são tipicamente sistemas de operação de rede. Faça a sua caracterização.

Os sistemas de rede desenvolveram-se na década de 90 a partir da vital necessidade de comunicação entre sistemas. Algumas das principais características são:

- Login remotos: poder trabalhar num sistema computacional fisicamente distinto (sem abandonar o nosso próprio sistema computacional);
- Partilha de ficheiros remotos: ser capaz de aceder transparentemente à memória de massa de um dado sistema computacional;
- Impressão remota: possibilidade de utilizar qualquer dispositivo ligado à rede (ex. impressora);
- Correio electrónico: possibilidade de estabelecer comunicação (mensagens) entre utilizadores da rede.

{ Permite não só a partilha com os diferentes utilizadores mas também a comunicação entre o sistema e o utilizador, como por exemplo através de mensagens de erro. }

7. Distinga deadlock de adiamento indefinido. Qual é a diferença entre politicas de prevenção de deadlock no sentido estrito e no sentido lato. Dê um exemplo de cada uma delas.

•***'deadlock'***-quando dois ou mais processos ficam a aguardar eternamente a sua entrada nas respectivas regiões críticas, esperando acontecimentos, que se pode demonstrar, que nunca irão acontecer; o resultado é por isso um bloqueio das operações;
•***adiamento indefinido***-quando um ou mais processos competem pelo acesso às respectivas regiões críticas e, devido a uma conjunção de circunstâncias em que surgem continuamente processos novos (de prioridade mais elevada, por exemplo) que o(s) ultrapassam nesse desígnio, o seu acesso é sucessivamente adiado; está-se por isso perante um impedimento real à sua continuação.

A politica de prevenção de deadlock no sentido estrito tem como finalidade a negação de uma das 4 condições que originam deadlock para deste modo eliminá-lo:
- Negação do principio da exclusão mútua
- Negação da condição de espera
- Negação da condição de libertação
- Negação da espera circular

Sendo a prevenção no sentido estrito muito radical, recorre-se ás politicas de prevenção no sentido lato que se baseiam numa politica de monitorização de atribuição de recursos, tendo como base o número de recursos que cada um pode vir a precisar e os recursos que se encontram disponíveis).
No caso do algoritmo de Djikteq trata-se de uma politica de prevenção de deadlock no sentido lato.

??? Tentar dar exemplo em relação à solução do problema do jantar dos filósofos. ????

8. O que é uma região critica? Porque é que é importante garantir o acesso com exclusão mútua a uma região critica? Que condicionamentos devem ser satisfeitos pelas primitivas de acesso?

É a região de código usada para manipulação de dados entre processos que comunicam entre si.
Como é uma zona partilhada por vários processos, tem que se garantir exclusão mútua, de modo a que quando um determinado processo esteja a ler a informação, esta não possa ser alterada por outro processo. Estas regiões tornam-se essenciais quando existem processos que pretendem aceder a dados partilhados.

Quando dois ou mais processos competem pelo uso de um recurso do sistema, que não pode ser partilhado, existe a necessidade de usar mecanismos para aplicar a exclusão mútua. Um recurso nestas circunstâncias designa-se por recurso crítico e a parte do programa que o usa define-se como sendo a região crítica.

Qualquer solução capaz de providenciar a exclusão mútua tem de respeitar os seguintes requisitos (algoritmo):
• Garantia efectiva de imposição da exclusão mútua - só a um processo de cada vez deve ser permitido o acesso à região crítica, de entre todos os processos que têm regiões críticas para o mesmo recurso;
• Um processo que pare a execução fora da região crítica, deve-o fazer sem interferir nos outros processos;
• Não pode ser adiada indefinidamente a possibilidade de acesso à região crítica a qualquer processo que o requeira;
• Quando nenhum processo está na região crítica, qualquer processo que requeira o acesso à região crítica não pode ser impedir de o fazer;
• Independência da velocidade de execução relativa dos processos intervenientes, ou do seu número – a solução não deve presumir o que quer que seja acerca destes factores;
• O tempo de permanência de um processo na região crítica é necessariamente finito.

Um algoritmo distribuído é caracterizado pelas seguintes propriedades:
1. Todos os nós têm uma quantidade de informação igual, em termos médios;

2. Cada nó tem apenas uma informação parcelar de todo o sistema e tem de tomar decisões baseada nessa informação;
3. Todos os nós carregam uma responsabilidade igual na decisão final;
4. Todos os nós despendem um esforço igual, em termos médios, na tomada da decisão final;
5. A falha de um nó, não deve em princípio, provocar um colapso do sistema global;
6. Não existe um relógio comum, com o qual se possa regular a temporização dos acontecimentos.

9. Defina recurso.

Genericamente, um recurso é algo que um processo precisa para a sua execução. Os recursos tanto podem ser dispositivos físicos do sistema computacional (processador(es), 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 operativo (tabela de controlo de processos, 'buffers' de comunicações, etc.) ou entre processos de uma mesma aplicação.
Uma propriedade essencial dos recursos é o carácter da apropriação que os processos deles fazem. Nestes termos, os recursos podem ser divididos em:
•**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 caso do processador, ou das regiões de memória principal onde o espaço de endereçagem de um processo está alojado;
•**recursos 'non-preemptable'**- caso contrário; são, por exemplo, o caso da impressora, ou de uma estrutura de dados partilhada que exige exclusão mútua para a sua manipulação.

10. O que distingue os algoritmos de Dekker e Peterson, enquanto soluções "software" para a imposição de exclusão mútua no acesso a uma região crítica?

O algoritmo de Dekker resolve o conflito de acesso entre os dois processos a uma região crítica, usando um mecanismo de alternância estrita. O algoritmo de Peterson, em contrapartida, estabelece uma propriedade baseada na ordem de chegada.

O algoritmo de Dekker usa um mecanismo de alternância para resolver o conflito resultante da situação de contenção de dois processos. Não é muito complicado demonstrar que efectivamente garante a exclusão mútua no acesso à região crítica, evita *'deadlock'* e *adiamento indefinido* e não presume o que quer que seja quanto à velocidade de execução relativa dos processos intervenientes.

O algoritmo de Peterson usa a seriação por ordem de chegada para resolver o conflito resultante da situação de contenção de dois processos. Isto é conseguido forçando cada processo a escrever a sua identificação na mesma variável. Assim, uma leitura subsequente permite por comparação determinar qual foi o último que aí escreveu. De novo, não é muito complicado demonstrar que este algoritmo garante efectivamente a exclusão mútua no acesso à região crítica, evita *'deadlock'* e *adiamento indefinido* e não presume o que quer que seja quanto à velocidade de execução relativa dos processos intervenientes. A sua generalização a N processos é quase directa, se se tiver em conta a analogia de formação de uma fila de espera. Existe, contudo, uma diferença subtil entre a disciplina implementada no algoritmo e a da formação de uma fila de espera convencional.

11. Descreva os objectivos mais importantes que devem ser satisfeitos numa politica de scheduling do processador. Quais entre eles são absolutamente indispensáveis em sistemas operativos do tipo batch?

Os objectivos mais importantes que devem ser satisfeitos numa politica de scheduling do processador são os seguintes:
•***justiça***- todo o processo, ao longo de um intervalo de tempo considerado de referência, deve ter direito à sua fracção de tempo de processador;
•***previsibilidade***- o tempo de execução de um processo deve ser razoavelmente constante e independente da sobrecarga pontual a que o sistema computacional possa estar sujeito;
•***'throughput'***- deve procurar maximizar-se o número de processos terminados por unidade de tempo;
•***tempo de resposta***- deve procurar minimizar-se o tempo de resposta às solicitações feitas pelos processos interactivos;

•***tempo de 'turnaround'***- deve procurar minimizar-se o tempo de espera pelo completamento de um 'job' no caso de utilizadores num sistema 'batch';
•***'deadlines'***- deve procurar garantir-se o máximo de cumprimento possível dos 'deadlines' impostos pelos processos em execução;
•***eficiência***- deve procurar manter-se o processador o mais possível ocupado com a execução dos processos dos utilizadores.

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

12. Distinga scheduling preemptive de scheduling nonpreemptive. Qual destas politicas é típica de sistemas operativos interactivos? E dos sistemas do tipo 'batch'? Justifique a resposta.

Numa disciplina de scheduling preemptive o sistema operativo tem a capacidade de retirar o processador ao processo que o detém, por acção determinada por 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 (a transição *timer-run-out* não existe neste caso).

Em sistemas do tipo interactivo, a politica de scheduling implementada é naturalmente do tipo preemptive, pois pretende-se estabelecer uma multiplexagem temporal da ocupação do processador pelos diferentes processos.

Em sistemas do tipo 'batch', a disciplina a usar é naturalmente do tipo nonpreemptive, porque se pretende atribuir o processador a um único processo de forma a maximizar o tempo de execução dum 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)

13. Explique porque é que, enquanto só existe uma lista ligada dos processos prontos a serem executados (associados ao estado READY), existem listas múltiplas de processos bloqueados (uma por recurso e associadas ao estado BLOCK).

Todos os processos que vão chegando são colocados no fim de uma lista onde todos os processos que estão READY são colocados. Quando chega ao topo da lista (ligada) será o próximo processo a ser executado quando o cpu estiver disponível. Caso um processo esteja a correr (estado RUN) e iniciar uma operação de I/O antes de o seu tempo atribuído acabar, o processo largará o cpu (, isto é, o processo bloqueia-se a ele próprio, estado blocked, até terminar a operação de I/O). Outro estado ocorre quando uma operação de I/O terminar. O processo transita assim do estado BLOCKED para o estado READY. É compreensível que exista uma lista ligada dos processos a serem executados (READY), pois à medida que os processos vão chegando e conforme o grau de prioridade, estes são colocados numa lista ordenada e vão sendo executados por ordem..
Se entretanto, por qualquer motivo, eles transitarem para o estado BLOCKED, estes terão de estar todos alinhados paralelamente (e não em série como no caso da lista ligada), para a qualquer momento, por acção de um fim do evento (podendo ser o fim da operação de I/O) transitarem para a lista ligada do estado READY, podendo ir ocupar posições mais privilegiadas em relação a processos que estariam anteriormente, logo listas múltiplas.

14. Num sistema operativo de utilizador único que apresente um ambiente de interacção gráfico com o utilizador, como o Windows Millenium, como é que a existência de janelas múltiplas possibilita a interacção com diferentes programas em execução?

15. O que é que distingue fundamentalmente um sistema operativo de rede de um sistema operativo distribuído?
+ info em 472,473,474,475,476,497

A definição mais comummente aceite, é a de que se considera um sistema distribuído, como sendo um sistema constituído por sistemas computacionais distintos e autónomos, e que isto é transparente para o utilizador.

A utilização de sistemas distribuídos tem algumas vantagens para as organizações:
• Partilha de recursos, quer em termos de capacidade de cálculo quer na capacidade de armazenamento da informação;
• Fiabilidade em caso de um elemento do sistema avariar ou perder a informação. Já que é possível e desejável manter *backups* da informação em vários computadores;
• Poupança de dinheiro na construção do sistema, que pode ir crescendo à medida das necessidades;
• Um meio de comunicação entre os utilizadores do sistema.

Por outro lado, a utilização de sistemas distribuídos tem algumas vantagens para os utilizadores:
• Acesso a informação remota;
• Um meio de comunicação entre os utilizadores do sistema;
• Entretenimento interactivo.

Uma vez que os sistemas distribuídos são construídos sobre uma rede de computadores, importa analisá-las e classificá-las. As redes de computadores podem ser classificadas em função da tecnologia de transmissão. Existem duas tecnologias de transmissão:
• Redes *broadcast;*
• Redes ponto a ponto.

16. Distinga multiprocessamento de multiprogramação.

Multiprocessamento – é a capacidade apresentada por um sistema computacional de poder executar em simultâneo dois ou mais programas; o que exige que osistema computacional seja formado por mais do que um processador (um por cada programa em execução simultânea).

Multiprogramação – é a ilusão criada por um sistema computacional de aparentemente poder executar em simultâneo mais programas do que o número de processadores existentes; o que exige que a atribuição do(s) processador(es) seja multiplexada no tempo entre os diferentes programas presentes. Num monoprocessador (sistema computacional com um único processador), os programas A e B estão a ser executados em concorrência.

17. Num sistema operativo de multi-utilizador de uso geral, como o Linux, que tipo de requisitos de hardware são impostos ao sistema computacional para garantir uma execução fiável, e livre de interferências de terceiros, dos programas de cada utilizador?
33,34,35,36

18. Que tipo de características deve apresentar um sistema operativo contemporâneo multi-utilizador de uso geral?

Protecção dos ficheiros

19. Em que termos é que se pode conceber a existência de facilidades de processamento 'batch' num sistema operativo multi-utilizador de uso geral?
1-14 / 1-15
Aproveitamento dos tempos mortos durante a execução de um, enquanto o processador aguarda a realização das operações de entrada-saída, para execução de um outro programa de um outro utilizador. ??? Como a comutação é tão rápida e os tempos de resposta dos dispositivos I/O são de ordens de grandeza superior á velocidade do CPU, os utilizadores tem a percepção de que tem o processador sempre disponível. ??? ??time sharing??

20. Comente a seguinte afirmação:
"Um sistema operativo multi-utilizador é necessariamente multiprogramado, mas um sistema operativo de rede, embora seja normalmente multiprogramado, não é necessariamente multi-utilizador."

(Será que é possível conceber-se multiprocessamento sem multiprogramação? Em que circunstâncias?)

21. O que são semáforos? Quais as operações que podem ser realizadas sobre os mesmos? Mostre como é que eles podem ser usados para garantir acesso a uma região critica com exclusão mútua.

184 / 3-32 / 3-33

Um semáforo é um dispositivo (solução de hardware), originalmente inventado por Dijkstra, que pode ser concebido como uma variável do tipo

```
typedef struct
        { unsigned int val;     /* valor de contagem */
          NODE *queue;          /* fila de espera dos processos bloqueados */
        } SEMAPHORE;
```

Sobre a qual é possível executar as duas operações atómicas seguintes:

sem_down - se o campo val for não nulo, o seu valor é decrementado caso contrário, o processo que a executou é bloqueado e a sua identi-ficação é colocada na fila de espera queue;

sem_up - se houver processos bloqueados na fila de espera queue, um deles é acordado (de acordo com uma qualquer disciplina previamente definida)caso contrário, o valor do campo val é incrementado.

Um semáforo só pode ser manipulado desta maneira e é precisamente para garantir isso que toda e qualquer referência a um semáforo particular é sempre feita de uma forma indirecta.

Para garantir acesso a uma região crítica a um processo de cada vez, criava-se um semáforo ”acess”. Para entrar na região critica um processo executava a operação sem_down(acess), colocando o valor de acess a “zero”, deste modo qualquer tentativa de outros processos para entrarem na região crítica vai ser negada, pois ao fazerem sem_down(acess) iriam ser bloqueados. Só quando o processo que está dentro da região critica realizar a operação sem_up(acess) é que os processos que foram bloqueados são acordados e puderam aceder à região crítica.

22. O que distingue dispositivos de tipo carácter de dispositivos tipo bloco? Descreva de uma maneira funcional como se desenvolve a comunicação entre um processo utilizador e um dispositivo tipo carácter.

Dispositivos de Blocos: estes dispositivos armazenam informação em blocos de tamanho fixo, cada um numa localização bem definida. A principal propriedade destes dispositivos é a de permitir a leitura/escrita de cada bloco independentemente de todos os outros. Os discos e as tapes são dispositivos deste tipo.

Dispositivos de Caracteres: estes dispositivos processam a informação em grupos de caracteres, sem ter em atenção alguma estrutura de tipo bloco. Estes dispositivos não são endereçáveis e não permitem operações de pesquisa. As impressoras, os terminais de vídeo e os teclados são dispositivos deste tipo.

???? de uma maneira funcional???

23. Em que consiste o mecanismo de spooling? Mostre como é que ele pode ser utilizado no caso da impressora.

Podem evitar-se sempre situação de deadlock? Como? Qual é o seu inconveniente?

321

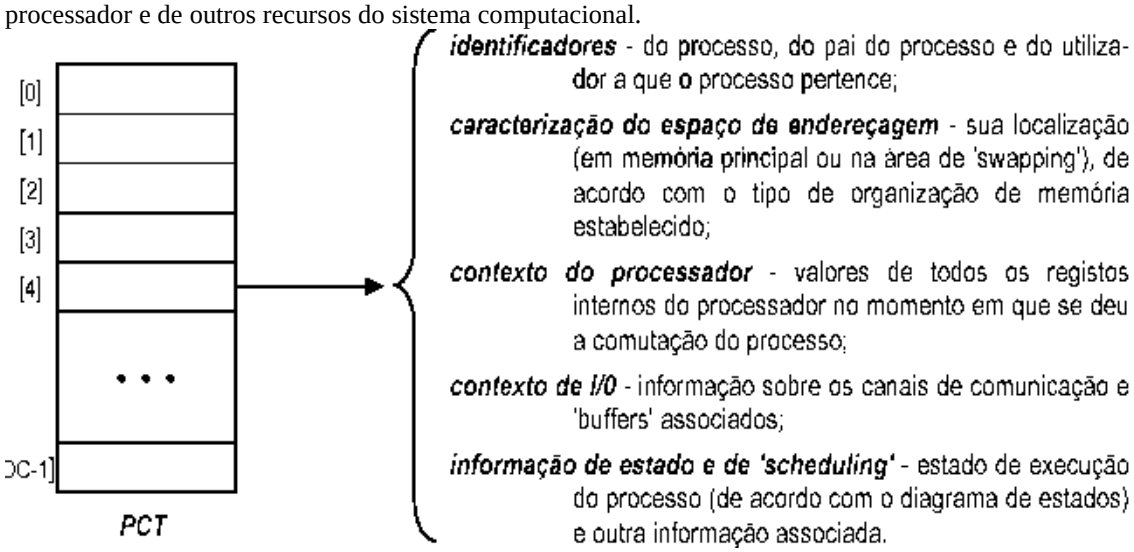
O mecanismo de spooling é um mecanismo que guarda num buffer a informação que está a ser transmitida para um dispositivo que não consegue receber dados “interleaved”.

No caso da impressora, se vários utilizadores estiverem a enviar trabalhos para imprimir, como a impressora não os consegue imprimir por partes é necessário que o sistema operativo guarde num buffer, os trabalhos enviados e que sempre que um deles termine seja colocado outro a imprimir.

??????????????

24. A tabela de controlo de processos é um elemento essencial na construção de um ambiente multiprogramado. Explique porque é que, mesmo numa organização de memória virtual, ela deve estar permanentemente residente em memória principal. Que tipo de informações é ai normalmente armazenada?

A implementação de um ambiente multiprogramado implica a existência de uma quantidade variada de informação acerca de cada processo. Esta informação é mantida numa tabela, designada pelo nome de Tabela de Controlo de Processos(PCT), que é usada intensivamente pelo scheduler para fazer a gestão do processador e de outros recursos do sistema computacional.

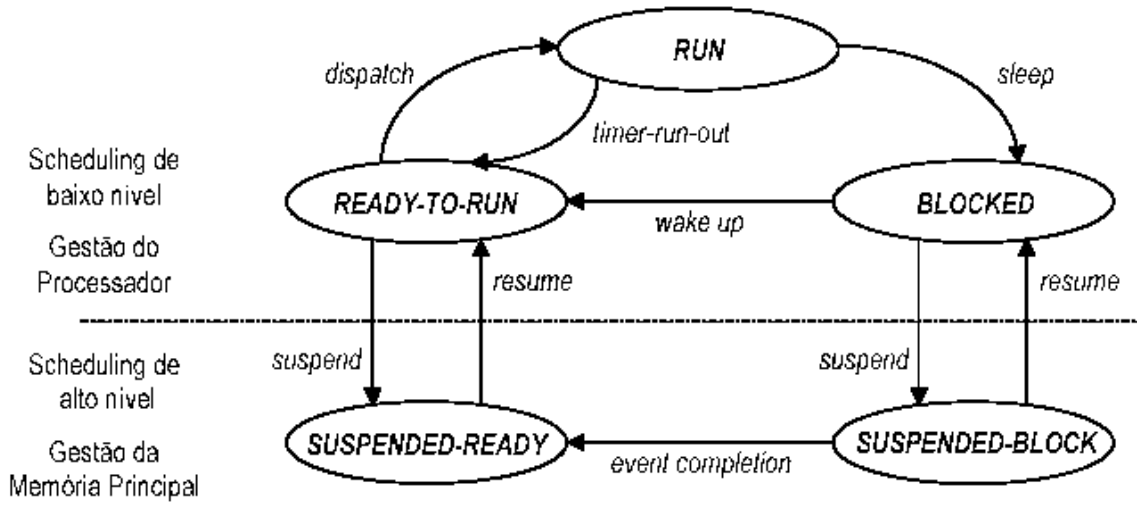


?? Porque é que é precisa???

Porque como guarda a informação dos diversos processos em uso e o sistema alterna constantemente entre os diversos processos, é necessário que esteja disponível na memória principal por uma questão de rapidez de execução.

25. Distinga ‘scheduling’ do processador de baixo nível de ‘scheduling’ do processador de alto nível. Descreva o diagrama de estados correspondente a cada um deles.

2-9



26. Porque é que sobre a abstracção da memória de massa, concebida como um ‘array’ de blocos para armazenamento de dados, se torna fundamental introduzir o conceito de sistemas de ficheiros? Quais são os elementos essenciais definidores da sua arquitectura?

?

Ao dividir a memória de massa em ficheiros e estruturas de directórios torna-se possível produzir funções "simples" de acesso a ficheiros, e que tornam "invisível" a memória de massa.

27. Explique a construção de 'lock-flags' a partir de instruções do processador de 'test-and-set' ou 'read-modify-write'. Porque é que em sistemas computacionais monoprocessador a implementação de 'lock-flags' como solução do acesso com exclusão mútua a uma região critica é um mecanismo pouco eficiente? Haverá situações em que seja justificada a sua utilização?

Hoje em dia, os processadores têm no seu 'conjunto de instruções' uma instrução especial, designada normalmente por ***test-and-set***(***tas***), que permite em sucessão, e de uma maneira atómica portanto, a leitura de uma posição de memória, a afectação do registo de 'status' em função do seu conteúdo e a escrita na mesma posição de um valor diferente de zero. Assumindo que, quando a região crítica não está a ser acedida, a posição de memória reservada para a variável *ocupado* tem o valor zero, tem-se então

```
void entrada_em_RC (void)
{
        lea   ad_reg, end_ocupado
  loop: tas   (ad_reg)              <==>  while (assert (&ocupado));
        bnz   loop
}
```

Um problema comum às soluções 'software' e ao uso de *variáveis de 'locking'*, implementadas a partir de instruções de tipo *read-modify-write*, é que os processos intervenientes aguardam entrada na região crítica no estado activo -***busy waiting***.
Este facto é naturalmente indesejável em sistemas computacionais monoprocessador, já que conduz a:
•***perda a eficiência***-a atribuição do processador a um processo que pretende acesso a uma região crítica, associada com um recurso ou uma região partilhada em que um segundo processo se encontra na altura no interior da região respectiva, faz com que o intervalo de tempo de atribuição do processador se esgote sem que qualquer trabalho útil tenha sido realizado;
•***contrangimentos no estabelecimento do algoritmo de 'scheduling'*** -numa política *'preemptive'* de *'scheduling'*, 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 interrompera execução do outro.
Assim, torna-se conveniente procurar soluções em que um processo bloqueie quando é impedido de entrar na região crítica respectiva.

28. O que são chamadas ao sistema (system call ou monitor call)? Qual é a sua importância na construção de um interface com o programador de aplicações.

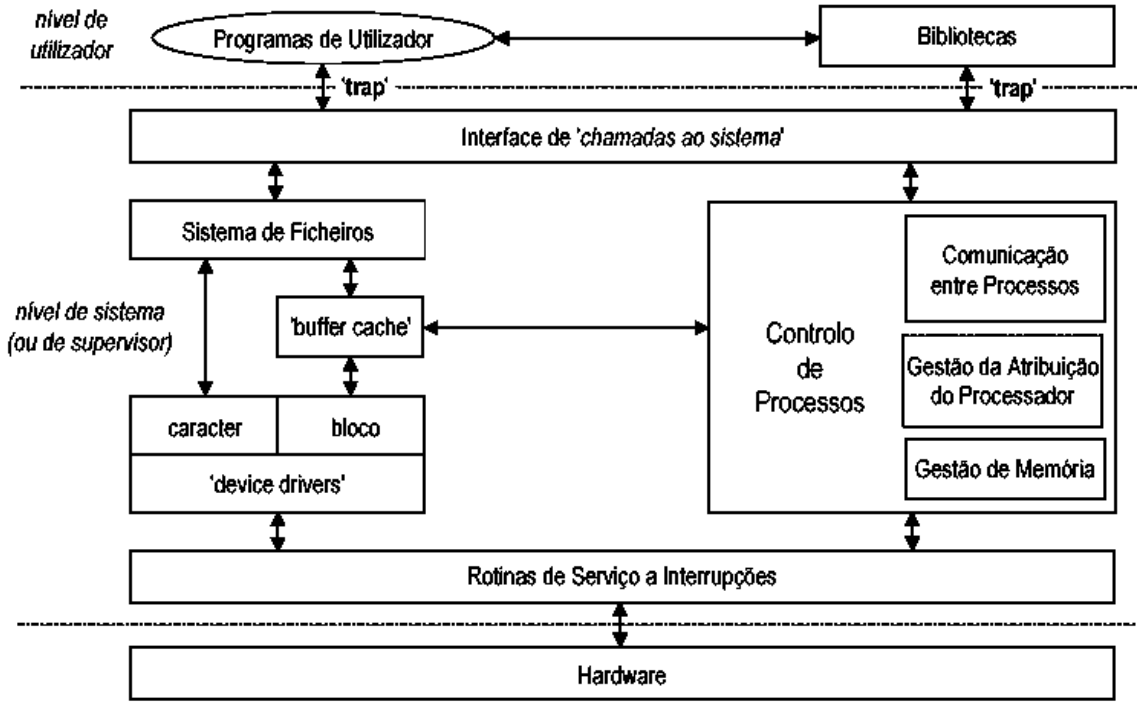
20, 39, 53, 54, 55
É uma função que faz o pedido do utilizador ao sistema operativo para efectuar uma operação de I/O.
São interrupções geradas por software.
Quando utilizador necessita de executar uma operação de I/O, gera uma system call, que vai "invocar" o sistema operativo, o sistema operativo vai verificar quem gerou a interrupção e qual a operação pertendida, se os dados estiverem correctos (operação válida e autorização para a efectuar), o sistema operativo realiza-a e no final devolve o controlo novamente para o utilizador.

29. O que distingue os níveis supervisor e utilizador no funcionamento dos processadores actuais? Em que termos a sua existência é importante para a implementação de um sistema de operação?

Os processadores actuais têm basicamente dois níveis de funcionamento:
•***nível supervisor***-todo o conjunto de instruções do processador (*instruction set*) pode ser executado; trata-se de um modo de funcionamento privilegiado, reservado para o sistema operativo;
•***nível utilizador***-só uma parte do conjunto de instruções do processador pode ser executada; estão excluídas as instruções de entrada-saída e quase todas as instruções que permitem modificar o conteúdo dos registos da unidade de controlo; constitui o modo normal de funcionamento.

A passagem do *nível supervisor* para o *nível utilizador* efectua-se por alteração de um bit do '*program status word*'. Por razões de segurança, contudo, não há instruções que possibilitem directamente a passagem inversa. Ela só é realizada quando o processador processa uma ***excepção***. Uma ***excepção*** é, no fundo, algo que interrompe a normal execução de instruções.

??? È importante para que existam operações que não estejam disponíveis ao utilizador por uma questão de segurança. ???



30. Descreva justificadamente os diversos papéis desempenhados pela memória de massa num sistema computacional. Dê razões que expliquem porque é que ela deve ser constituída por mais do que uma unidade de disco magnético.

31. Em que consiste o algoritmo dos banqueiros de Dijkstra? Dê um exemplo da sua aplicação. Indique as suas limitações principais. Será que ele impede sempre a ocorrência de 'deadlock'?

3-22

32. Qual é a função de um device driver?

Os device drivers permitem criar uma interface abstracta com os dispositivos de I/O e a construção de software independente do dispositivo, que implementam operações comuns a todos os dispositivos de I/O. Obtém-se assim uma plataforma uniforme para o software dos utilizadores.

A função de um device driver consiste pois, em aceitar pedidos genéricos do software, por exemplo ler/escrever um bloco, e providenciar a sua execução pelo dispositivo.

33. Porque é que a ocorrência de busy waiting é nociva nas soluções de acesso a uma região crítica com exclusão mútua?

3-31

Este facto é naturalmente indesejável em sistemas computacionais monoprocessador, já que conduz a:

•***perda a eficiência***-a atribuição do processador a um processo que pretende acesso a uma região crítica, associada com um recurso ou uma região partilhada em que um segundo processo se encontra na altura no

interior da região respectiva, faz com que o intervalo de tempo de atribuição do processador se esgote sem que qualquer trabalho útil tenha sido realizado;
•***contrangimentos no estabelecimento do algoritmo de 'scheduling'*** -numa política *'preemptive'* de *'scheduling'*, 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 interrompera execução do outro.

34. Indique quais são as condições necessárias à ocorrência de deadlock, como fazer a sua prevenção e como sair de uma situação de deadlock.

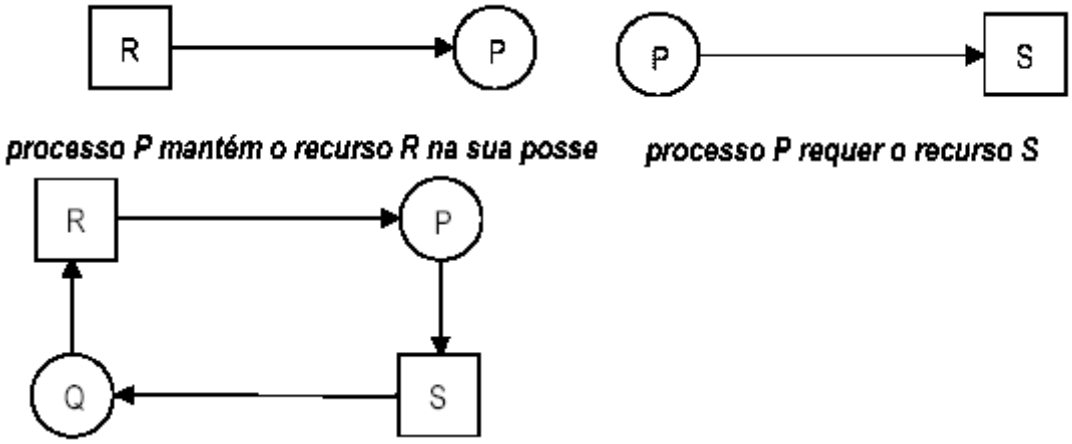
Pode demonstrar-se que, sempre que ocorre deadlock, há quatro condições que ocorrem necessariamente. São elas:
• **Condição de exclusão mútua** – cada recurso existente foi atribuído a um e um só processo, ou está livre;
• **Condição de espera com retenção** – cada processo, ao requerer um novo recurso, 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 ciclo vicioso (ou espera circular)** – 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.

Para prevenir uma situação de deadlock, basta evitar a ocorrência de uma das quatro condições necessárias à ocorrência de deadlocks:
• Regra geral é impraticável negar a **exclusão mútua**, excepção feita a usar técnicas de spooling;
• Uma forma de negar a **condição de espera com retenção,** é a de assegurar que um processo consegue todos os recursos que necessita num dado instante e bloquear o processo até que todos os recursos estejam disponíveis;
• Uma forma de negar a **condição de não libertação**, é a de exigir que um processo liberte os recursos entretanto adquiridos aquando da não obtenção de um recurso. E depois obtê-los todos de uma vez. Em alternativa o recurso requerido pode ser retirado a outro processo;
• Uma forma de negar a **condição de ciclo vicioso**, consiste em definir uma ordem linear aos recursos. Se um processo adquiriu recursos de tipo R apenas pode requerer recursos de ordem superior a R.

Uma outra forma de prevenir a ocorrência de deadlocks, consiste em verificar de forma dinâmica, se a alocação de um recurso não leva a uma situação potencial de ocorrência de deadlocks:
• Não iniciar um processo se as suas necessidades podem levar a uma situação de deadlock;
• Não atribuir um novo recurso a um processo, se esta atribuição levar a uma situação de deadlock (**estado seguro / não seguro**).
Esta política tem no entanto um conjunto de desvantagens:
• É necessário saber antecipadamente o total de recursos necessários a cada processo;
• Os processos devem ser independentes, ou seja a ordem de execução não deve depender de sincronização;
• Deve existir um número fixo de recursos para alocar;
• Nenhum processo pode terminar mantendo recursos em sua posse.

Para recuperar de uma situação de deadlock:
• O processo mais comum, consiste em abortar todos os processos em situação de ‘deadlock’;
• Recolocar todos os processos em situação de ‘deadlock’ a um estado anterior e relançar os processos. Isto não impede que a situação de ‘deadlock’ não volte a acontecer;
• Abortar sucessivamente processos em situação de ‘deadlock’ até que a situação desapareça, usando para o efeito um critério de custo mínimo;
• Retirar recursos aos processos em situação de ‘deadlock’ até que a situação desapareça, usando para o efeito um critério de custo mínimo.

Numa situação de 'deadlock', só os recursos 'non-preemptable' são relevantes. Os restantes podem sempre ser retirados, se tal for necessário, ao(s) processo(s) que o(s) detêm e atribuídos a outros para garantir o prosseguimento a execução destes últimos.
Assim, usando este tipo de classificação torna-se possível desenvolver uma notação esquemática que representa graficamente situações de ‘deadlock’.



Situação típica de 'deadlock' (a mais simples possível)

35. O que é o working set de um processo. Explique como é que ele pode ser usado na implementação de uma política eficaz de substituição de blocos.

2003-11-26

O *working set* de um processo, no momento virtual t é dado por W(t,.) e define o conjunto de páginas que foram referenciadas nas últimas unidades de tempo virtual.
Considere-se tempo virtual como o tempo de execução do processo medido em número de instruções executadas.

A estratégia working set pode ser aplicada da seguinte forma:
• Monitorizar o working set de cada processo;
• Remover periodicamente do conjunto residente de páginas do processo, as páginas que não estão no seu working set;
• Um processo só pode estar em execução se o seu conjunto residente de páginas inclui o seu working set, ou por outras palavras, se o seu working set estiver carregado na memória principal.

Esta estratégia é interessante, porque aceita o princípio da localidade de referência
e explora-a de forma a atingir uma gestão de memória que minimize os page faults.
Infelizmente, existem alguns problemas na sua implementação:
1. O passado nem sempre prediz o futuro. A dimensão do conjunto residente de páginas muda ao longo do tempo de execução do processo;
2. É impraticável uma monitorização efectiva do working set de cada processo.
Isso implicaria assinalar o tempo virtual do processo em cada página referenciada e manter as páginas do processo ordenadas cronologicamente;
3. O valor ideal de Δ é desconhecido.
Mas apesar disso, alguns algoritmos de substituição de páginas tentam aplicar esta
estratégia, como é o caso do Page Fault Frequency e o Working Set Clock.

O algoritmo Page Fault Frequency usa um bit indicativo de utilização em cada página que é colocado a 1 quando a página é acedida. Quando um processo tem um page fault o sistema operativo anota o tempo virtual desde o page fault anterior. Se o tempo decorrido entre page faults consecutivos é inferior a um tempo pré-determinado então é acrescentada mais uma página ao conjunto residente do processo. Senão, todas as páginas com o bit a 0 são desalocadas e a dimensão do conjunto residente do processo é reduzida. Ao mesmo tempo todos os bits de utilização de todas as páginas do processo são colocados a 0.
O algoritmo pode ser refinado utilizando não um tempo pré-determinado, mas dois: um tempo superior que desperta o crescimento da dimensão do conjunto residente do processo e um tempo inferior que desperta o decrescimento da dimensão do conjunto residente do processo.
Este algoritmo porta-se mal em períodos transitórios, ou seja, quando existe uma mudança para uma nova localidade de referência.

O algoritmo Working Set Clock baseia-se no algoritmo clock e portanto as páginas são colocadas numa lista circular onde existe um ponteiro que indica a página a substituir. Cada página tem um bit indicativo

de utilização, um bit indicativo de modificação e um campo onde é assinalado o tempo virtual de acesso à página.
Quando acontece um page fault, se o bit de utilização é 1, então é sinal que a página foi recentemente acedida e portanto não é escolhida para ser substituída. O bit é colocado a 0 e o ponteiro avança para a próxima página. Pelo contrário, se o bit de utilização é 0 e o tempo virtual de acesso for maior do que um tempo pré-determinado e o bit de modificação é 0 então a página não pertence ao working set e está limpa, portanto, a página é escolhida para ser substituída. Mas, se o bit de modificação é 1 então a página é agendada para ser reescrita na memória secundária. E, o ponteiro avança para a próxima página à procura de uma página limpa que possa ser imediatamente utilizada.

Se algoritmo de pesquisa der a volta completa à lista, então é porque aconteceu uma de duas situações:
1. Pelo menos uma página foi agendada para ser reescrita na memória secundária. Nesse caso o algoritmo continua à procura de uma página limpa, quanto mais não seja, até que uma das páginas agendadas para ser reescrita na memória secundária fique disponível.
2. Nenhuma página foi agendada para ser reescrita na memória secundária.
Nesse caso todas as páginas fazem parte do working set do processo e uma página limpa é escolhida para substituição. Se não existir uma página limpa, então a página corrente é seleccionada.

36. Explique e compare os algoritmos de schedduling do disco SCAN e CSCAN.

SCAN – este algoritmo é semelhante ao SSTF, mas actua numa direcção preferencial. A cabeça é movimentada sempre na mesma direcção, em função do menor movimento da cabeça da posição actual, até que atinge a última pista nessa direcção ou até não existirem pedidos nessa direcção. Depois a direcção é invertida. É por vezes chamado de algoritmo do elevador. Tem uma melhor distribuição do serviço que o SSTF e assegura que nenhum pedido fica indefenidamente à espera. O desfavorecimento das pistas interiores e exteriores do disco, típico do SSTF, é parcialmente eliminado. Mas, favorece os pedidos mais recentes.

C-SCAN (***Circular* SCAN**) – este algoritmo assegura que quando a última pista na direcção de atendimento é atingida, a cabeça é recolocada no extremo oposto do disco e o processamento dos pedidos recomeça. Assim é reduzido o tempo de espera máximo de novos pedidos. O desfavorecimento das pistas interiores e exteriores do disco é completamente eliminado. Pode ser implementado de maneira a que, os novos pedidos chegados durante o percurso de atendimento, sejam colocados numa outra fila de espera e atendidos no próximo percurso.

37. Explique detalhadamente como é implementado o método de alocação indexada de blocos, no caso da alocação de blocos de tamanho fixo.
(Diga como é implementado o método de alocação de blocos, e, para que tipo de acesso a ficheiros este método é indicado.)

38. Explique o que é um sistema operativo nas duas perspectivas base: do utilizador (top-down) e do construtor (bottom-up).

1-6 / 1-7 / 1-8

top-down

• O sistema operativo transmite ao programador uma abstracção do sistema computacional que o liberta do conhecimento preciso dos detalhes do 'hardware' subjacente;
• Ou seja, o sistema operativo fornece um modelo funcional do sistema computacional, designado pelo nome de *máquina virtual*, que é mais simples de compreender e programar;
• O interface com o 'hardware', assim criado, origina um ambiente uniforme de programação, que é operado através das *chamadas ao sistema*, e possibilita por isso a portabilidade de aplicações entre sistemas computacionais estruturalmente distintos.

Exemplos das funcionalidades criadas pelo sistema operativo:
• Estabelecimento do ambiente base de interacção com o utilizador;
• Disponibilização de facilidades para o desenvolvimento, teste e validação de programas;
• Fornecimento de mecanismos para a execução controlada de programas, sua comunicação e sincronização mútuas;

• Dissociação do espaço de endereçagem do programa das limitações impostas pelo tamanho da memória principal;
• Organização da memória de massa em sistemas de ficheiros;
• Definição de um modelo geral de acesso aos dispositivos de entrada / saída, independente das suas e especificidades próprias;
• Detecção de situações de erro e estabelecimento de uma resposta adequada.

bottom-up

Pode definir-se *sistema computacional* como sendo um sistema formado por um conjunto de recursos (processador(es), memória principal, memória de massa e diferentes tipos de controladores de dispositivos de entrada / saída) destinados ao processamento e armazenamento de informação.
Neste sentido, o sistema operativo pode ser visto como:
• o programa que gere o sistema computacional, fazendo a atribuição controlada e ordeira dos seus diferentes recursos aos programas que por eles competem;
• o objectivo é, portanto, conseguir-se a rentabilização máxima do sistema computacional, garantindo-se uma utilização tão eficiente quanto possível dos recursos existentes.

39. Que tipos de sincronização entre remetente e destinatário são possíveis no paradigma de passagem de mensagens?

3-44 / 3-45
Uma forma alternativa de comunicação entre processos é através da troca de mensagens. Trata-se de um mecanismo absolutamente geral, já que, não exigindo partilha do espaço de endereçagem, a sua aplicação, de um modo mais ou menos uniforme, é igualmente válida tanto em ambientes monoprocessador, como em ambientes multi-processador ou de processamento distribuído.

O princípio em que se baseia é muito simples:
• Sempre que um processo PR, dito remetente, pretende comunicar com um processo PD, dito destinatário, envia-lhe uma mensagem através de um canal de comunicação estabelecido entre ambos (operação de envio);
• O processo PD, para receber a mensagem, tem tão só que aceder ao canal de comunicação e aguardar a sua chegada (operação de recepção).

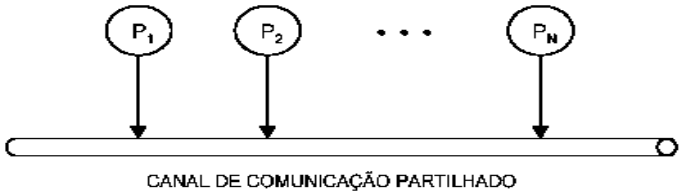
Porém, a troca de mensagens só conduzirá a uma comunicação efectiva entre os processos remetente e destinatário, se for garantida alguma forma de sincronização entre eles.
Em geral, o grau de sincronização existente pode ser classificado em dois níveis:
• Sincronização não bloqueante - quando todo o tipo de sincronização é explicitamente da responsabilidade dos processos intervenientes; a operação de envio envia a mensagem e regressa sem qualquer informação sobre se a mensagem foi efectivamente recebida; a operação de recepção, por seu lado, regressa independentemente de ter sido ou não recebida uma mensagem.

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;

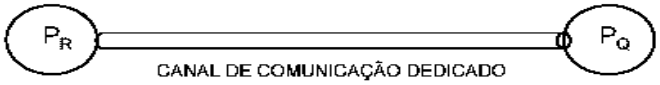
```
/* operação de envio*/
voidmsg_send (unsigned int destid, MESSAGE msg);
/* operação de recepção*/
voidmsg_receive (unsigned int srcid, MESSAGE *msg);
```

A sincronização bloqueante divide-se ainda em dois tipos:
• Rendez-vous - quando, só após uma sincronização prévia entre as operações de envio e de recepção, a transferência da mensagem tem efectivamente lugar; não exige um armazenamento intercalar da mensagem e é típica de canais de comunicação dedicados (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; pode existir ou não armazenamento intercalar da mensagem e é típica de canais de comunicação partilhados.

Para que a mensagem possa ser encaminhada entre o processo remetente e o processo destinatário, é fundamental que as operações de envio e de recepção tenham um parâmetro que faça de algum modo referência à identidade do processo interlocutor.
Se essa referência é explícita, a endereçagem diz-se directa. Caso contrário, o que é referenciado explicitamente é o canal de comunicação e a endereçagem diz-se indirecta.
Adicionalmente, o canal de comunicação pode permitir o armazenamento intercalar da mensagem, tipicamente em estruturas de dados partilhadas que implementam filas de espera de armazenamento, onde a ordem cronológica de chegada é normalmente respeitada -caixas de correio.



endereçagem directa



endereçagem indirecta

40. Como caracteriza uma organização de memória virtual? Que implicações é que uma tal organização introduz no desenho do processador? Qual a função da área de swapping?

Em alternativa a uma organização em **memória real**, podemos conceber que o espaço de endereçagem de um processo pode exceder o total de memória disponível no sistema computacional. Mas, apenas as partes necessárias à execução do processo são mantidas na memória primária, enquanto que, todo o espaço de endereçagem do processo é mantido na memória secundária. As partes não presentes na memória principal serão carregadas à medida que forem necessárias à execução do processo. A este conceito, chama-se uma organização em **memória virtual**.

Um sistema multiprogramado organizado com memória virtual, tem comparativamente a um sistema com memória real, as seguintes vantagens:
• É possível manter mais processos na memória, uma vez que, só temos as partes estritamente necessárias de cada processo. Portanto, potencia-se um aumento da taxa de utilização do processador;
• O espaço de endereçagem de um processo pode ser maior do que a memória física;
• Não há tanta necessidade de transferência de processos entre a memória primária e secundária, a não ser para as partes que tenham sido entretanto modificadas durante a execução do processo.

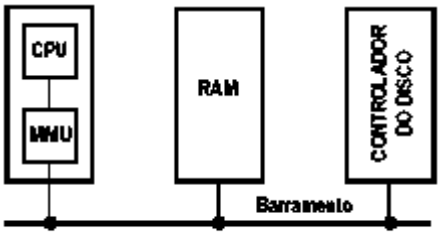
Resumindo e concluindo, um sistema com memória virtual permite a multiprogramação de forma mais eficaz e liberta o programador da limitação da memória física.

Mas apesar dos benefícios serem atractivos, será que é uma técnica viável?
Uma vez que só temos parte do processo na memória principal, não só cabem mais processos na memória, como também se diminui o tempo gasto na transferência da memória principal para a memória secundária (***swapped out***) e vice-versa (***swapped in***). No entanto, este processo tem de ser gerido de forma inteligente.

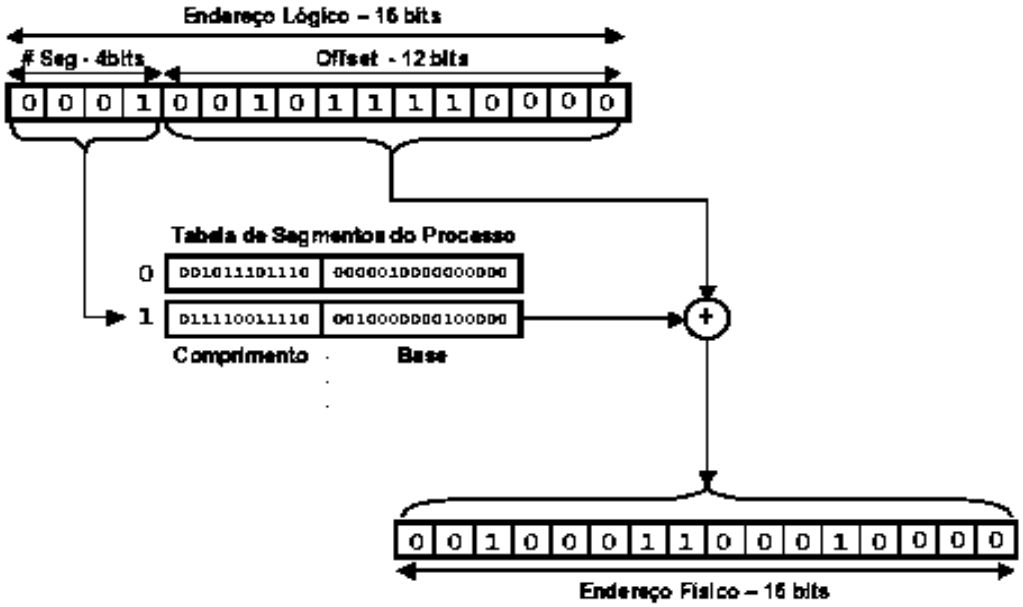
Quando uma parte do processo é carregado na memória principal (*swapped in*), outra parte do processo tem de ser salvaguardada na memória secundária, caso tenha sido modificada (*swapped out)*, ou em caso contrário a memória é simplemente reutilizada sem custos adicionais. O problema é que, se essa parte do processo tornar a ser preciso então terá de ser carregada de novo na memória. Se esta situação se repetir muitas vezes, às tantas o processador não faz mais nada do que transferir a mesmas partes de um processo entre as memórias principal e secundária, em vez de efectivamente executar instruções. A esta situação dá-se o nome de ***thrashing***.
Mas, isto não acontece devido ao **princípio da localidade de referência**, que diz que as instruções e referências a dados tendem a concentrar-se no tempo e no espaço. Experiências levadas a cabo em

sistemas multiprogramados com memória virtual, confirmam que, um processo durante pequenos períodos de tempo, só precisa de algumas partes do seu espaço de endereçagem.

Para implementar um sistema multiprogramado organizado com memória virtual, são necessários os seguintes requisitos:
• Suporte de hardware para a gestão da memória (**MMU**);
• Software para movimentação de partes do processo entre a memória principal e a memória secundária.



Sistema Computacional Simplificado



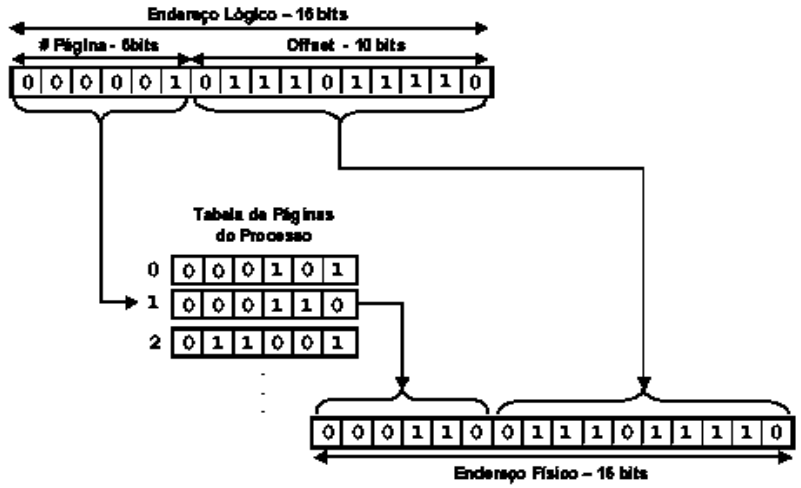
Conversão de um endereço virtual num endereço real num sistema com segmentação

41. Descreva detalhadamente uma organização de memória virtual paginada e indique uma estratégia usada na substituição das páginas da memória principal.

+ em 2003-11-24

Numa organização de memória virtual paginada, a dimensão do bloco base de armazenamento é fixo. Uma organização de memória virtual paginada recorre a uma tabela de mapeamento, ou seja, a uma tabela associada ao processo. Esta tabela faz referência ao estado de EMM, endereço de memória de massa, e ao FR, uma flag residente, que informa se o programa está em memória principal ou não, e o endereço básico de memória principal que indica o endereço onde o programa está alojado para gerar o endereço físico. As estratégicas de substituição de páginas em memória principal são:

- FIFO: Quando uma página precisa de ser substituída escolhe-se aquela que foi armazenada à mais tempo;
- LRU: Escolhe-se a página que foi usada à mais tempo. Necessita de uma implementação por cada página para se saber o tempo.
- NUR: Not Used Recently; escolhe-se a página que não foi uada mais recentemente
- 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.



Conversão de um endereço virtual num endereço real num sistema com paginação

Sumário dos algoritmos de substituição de páginas

- O algoritmo Óptimo substitui a página que vai estar mais tempo sem ser precisa, o que é difícil de prever. Pelo que, o algoritmo não é implementável.
- O algoritmo LRU é um algoritmo excelente. Mas, não pode ser implementado sem hardware especial.
- O algoritmo FIFO é fácil de aplicar. Mas, ao substituir a página mais antiga às cegas sem ter em conta a sua utilização, pode deitar fora páginas importantes.
- O algoritmo Clock (ou *Second Chance*) é um algoritmo realista e, como verifica se a página a substituir tem sido usada, é uma melhoria do algoritmo FIFO.
- O algoritmo *Page Fault Frequency* usa a estratégia do working set. Mas, porta-se mal quando existe uma mudança para uma nova localidade de referência.
- O algoritmo *Working Set Clock* combina as estratégias do *working set* e do algoritmo Clock. Tem um bom desempenho e uma implementação eficiente.

42. Como caracteriza uma organização de memória real? Que implicações é que uma tal organização introduz no desenho do processador? Qual a função da área de swapping e estabeleça o diagrama de transição de estados nesta situação.

Num sistema multiprogramado organizado com **memória real**, podemos assumir que a parte residente do sistema operativo ocupa uma parte da memória, sendo a restante distribuída pelos processos activos. A maneira mais fácil de distribuir a memória pelos múltiplos processos, consiste em dividir a memória em **partições estáticas**. Existem duas soluções: partições de tamanho igual ou partições de tamanho diferente.

43. Descreva detalhadamente uma organização real de partições variáveis e explique o que é o 'garbage collection' e como é que ele pode ser efectuado.

75 / 128 / 368

?? Consiste em ver se os processos ainda estão em execução e retira-los da memória se já terminaram ??

44. O que é o schedulling de disco? Explique porque é que ele é necessário e descreva três políticas possíveis, indicando claramente o que as distingue.

436 / 437 / 438 /439

Num ambiente multiprogramado com a competição dos vários processos pelo disco é importante definir políticas de acesso ao disco, que melhorarem o desempenho quando comparado com um acesso puramente aleatório.
Existem as seguintes estratégias de scheduling do disco:

• FIFO – atendimento dos pedidos pela ordem de chegada. É o algoritmo mais simples e é a política mais justa, mas aproxima-se muito do acesso aleatório e como tal não é boa solução;
• PRI – atendimento dos pedidos em função da prioridade dos processos. Dá prioridade a pequenos processos em detrimento de grandes processos, e portanto, aumenta a interactividade do sistema. Tem a desvantagem de que o controlo do disco sai fora da política de gestão da fila de espera do disco;
• LIFO – atendimento dos pedidos pela ordem inversa de chegada, ou seja, os mais recentes. Baseia-se na premissa de que pedidos sucessivas pertençem ao mesmo processo, e portanto, aproveita o princípio da localidade para optimizar a utilização do disco.

Estas estratégias dependem apenas dos processos. Mas, se a posição actual da cabeça é conhecida, então, esta informação pode ser usada de modo a optimizar o agendamento dos pedidos.

Baseado neste conceito, existem as seguintes estratégias de scheduling do disco:
• SSTF (***Shortest Service Time First***) – Este algoritmo atende os pedidos em função do menor movimento da cabeça da posição actual, procurando minimizar o *seek time*. No entanto, isto não assegura um valor mínimo do *seek time* médio no tratamento de vários pedidos. É melhor que o FIFO, mas o padrão de atendimento tende a desfavorecer as pistas interiores e exteriores, em detrimento das pistas centrais do disco, podendo adiar indefinidamente o seu atendimento.
• SCAN – este algoritmo é semelhante ao SSTF, mas actua numa direcção preferencial. A cabeça é movimentada sempre na mesma direcção, em função do menor movimento da cabeça da posição actual, até que atinge a última pista nessa direcção ou até não existirem pedidos nessa direcção. Depois a direcção é invertida. É por vezes chamado de algoritmo do elevador. Tem uma melhor distribuição do serviço que o SSTF e assegura que nenhum pedido fica indefinidamente à espera. O desfavorecimento das pistas interiores e exteriores do disco, típico do SSTF, é parcialmente eliminado. Mas, favorece os pedidos mais recentes.
• C-SCAN (***Circular* SCAN**) – este algoritmo assegura que quando a última pista na direcção de atendimento é atingida, a cabeça é recolocada no extremo oposto do disco e o processamento dos pedidos recomeça. Assim é reduzido o tempo de espera máximo de novos pedidos. O desfavorecimento das pistas interiores e exteriores do disco é completamente eliminado. Pode ser implementado de maneira a que, os novos pedidos chegados durante o percurso de atendimento, sejam colocados numa outra fila de espera e atendidos no próximo percurso.
• N-step-SCAN – Com os algoritmos anteriores, a cabeça pode não mover-se durante um período de tempo considerável. Um conjunto de pedidos com frequência de acesso elevado a uma pista, pode monopolizar o acesso a essa pista. Para atacar esta rigidez da cabeça, a fila de espera de pedidos pode ser partida em filas de N pedidos, devidamente ordenados de maneira a optimizar o serviço de atendimento. Cada fila é processada usando o algoritmo SCAN. Os pedidos chegados durante um percurso de atendimento, são colocados ordenadamente numa nova fila de espera e podem ser atendidos no percurso de retorno.

45. Indique as principais características do modelo 'cliente-servidor'. Tome como base da sua explicação um servidor de ficheiros e quais as suas vantagens e inconvenientes.

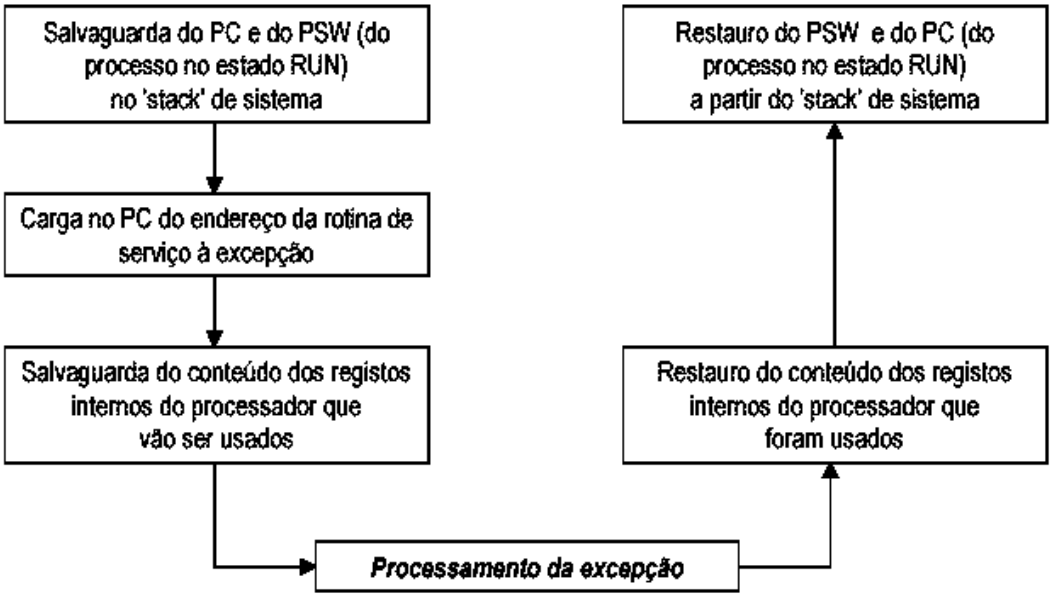
46. O que é o principio da optimalidade? Descreva detalhadamente uma política de substituição de blocos em memória principal que se aproxime deste principio.

47. Classifique claramente as redes de computadores quanto à tecnologia de comunicação usada.
Dê exemplos ilustrativos de cada caso.

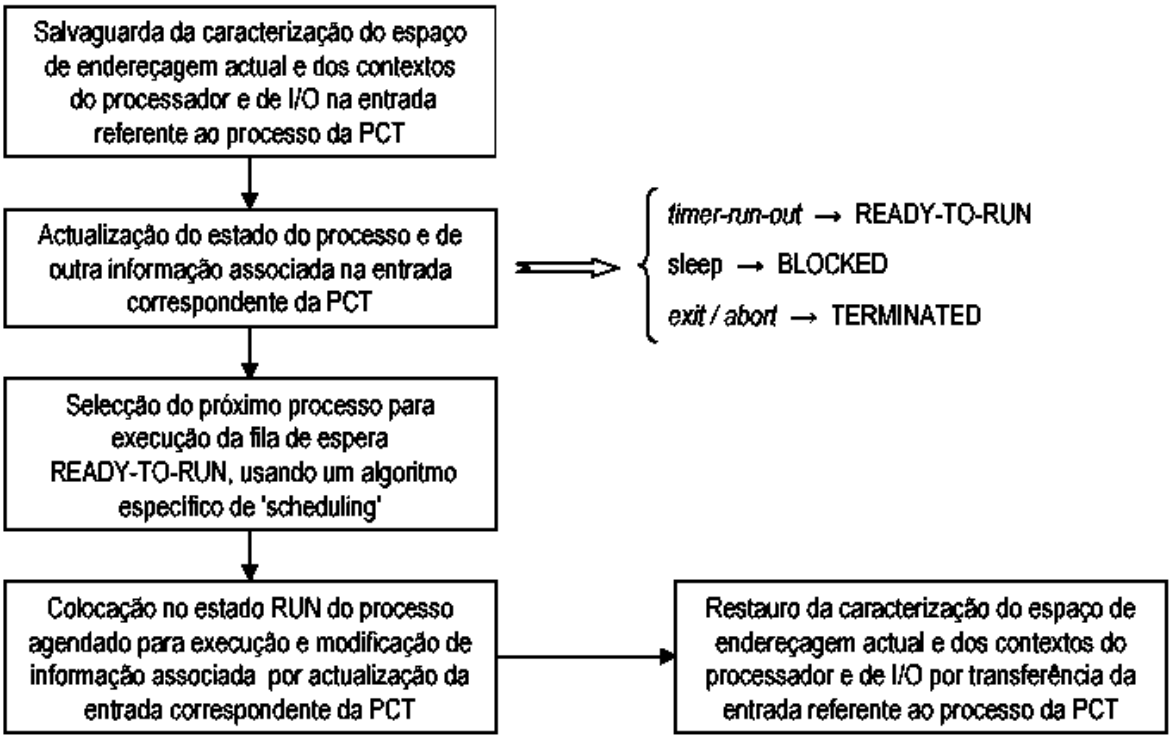
48. Indique quais os estados possíveis de um processo

49. Indique quais as operações a realizar numa comutação de processos (Processamento de uma excepção genérica).

Processamento de uma excepção genérica



Comutação propriamente dita



50. Descreva o que é prioridade estática e prioridade dinâmica.

2-30

Prioridade estática

Os processos são agrupados em classes de prioridade fixa, de acordo com a sua importância relativa. A comutação pode ocorrer sempre que seja necessário executar um processo de prioridade mais elevada. Trata-se da disciplina de atribuição mais injusta, existindo um risco claro de ocorrência de adiamento indefinido na calendarização para execução dos processos de prioridade mais baixa.
Disciplina característica dos sistemas de tempo real.

Aquando da sua criação, é atribuído a cada processo um dado nível de prioridade. Depois, à medida que o processo é calendarizado para execução, a sua prioridade é decrementada de uma unidade, sendo reposta no valor inicial quando atinge o valor mínimo. Deste modo, é colmatado o carácter injusto do método anterior.
Disciplina característica de sistemas multi-utilizador. O Unix usa uma disciplina de scheduling deste tipo

Prioridade dinâmica

Um método alternativo de privilegiar os processos interactivos consiste na definição de classes de prioridade de carácter funcional, entre as quais os processos transitam de acordo com a ocupação da última ou últimas janelas de execução.

Por exemplo:
• Nível 1 (mais prioritário): terminais - classe para que transitam os processos, após 'acordarem' no seguimento de espera por informação do dispositivo de entrada 'standard';
• Nível 2: I/O genérico-classe para que transitam os processos, após 'acordarem' no seguimento de espera por informação de um dispositivo distinto do dispositivo de entrada 'standard';
• Nível 3: janela pequena - classe para que transitam os processos quando completaram a sua última janela de execução;
• Nível 4 (menos prioritário): janela longa - classe para que transitam os processos quando completaram em sucessão um número pré-definido de janelas de execução; o objectivo é atribuir-se no futuro uma janela mais longa, mas menos vezes.

Em sistemas 'batch', um problema que se coloca é a redução do tempo de 'turn-around' (somatório dos tempos de espera e de execução) dos 'jobs' que constituem a fila de processamento.
Desde que se conheçam estimativas dos tempos de execução de todos processos na fila, é possível estabelecer-se uma ordenação para a execução dos processos que minimiza o tempo médio de 'turnaround' do grupo.
Com efeito, admita-se que a fila contém N 'jobs', cujas estimativas dos tempos de execução são, respectivamente, ten, com n = 1, 2, ... , N.
Então, o tempo médio de 'turnaround' do grupo vem dado por
tmtaround= te1+ (N-1)/N*te2+ ... + 1/N*teN, e é mínimo quando os 'jobs' estão ordenados por ordem crescente do tempo de execução.
Este método de selecção designa-se por shortest job first(SJF).

Uma abordagem semelhante pode ser usada em sistemas interactivos para se estabelecer a prioridade dos processos que competem pela posse do processador. O princípio consiste em procurar estimar-se a fracção de ocupação da janela de execução seguinte em termos da ocupação das janelas passadas, atribuindo-se o processador ao processo para o qual esta estimativa é menor.

51. Quais as organizações possíveis da área de swapping?

A área de swapping pode ser organizada de duas maneiras:
• Área de swapping estática (figura da esquerda no acetato seguinte). Na área de swapping existe uma cópia completa do processo. Durante a sua execução, a tabela do processo é responsável por manter a informação sobre a localização das páginas virtuais na memória secundária. Quando uma página virtual é referenciada, ela tem de ser carregada da memória secundária para a memória principal (paged in). E, quando uma página de memória é libertada, e se entretanto foi modificada, então, ela é salvaguardada na memória secundária (paged out).
• Área de swapping dinâmica (figura da direita no acetato seguinte). Neste caso, na área de swapping existem apenas as páginas virtuais que não estão carregadas na memória principal. Quando uma página virtual é referenciada, ela tem de ser carregada da memória secundária para a memória principal (paged in) e o espaço por ela ocupada na memória secundária é libertado. E, quando uma página de memória é libertada, então, ela é sempre salvaguardada na memória secundária (paged out), sendo necessário alocar espaço no disco.

52. Qual é a sequência de eventos quando ocorre um page fault?

Quando acontece um *page fault* é executada a seguinte sequência de eventos:
1. O hardware envia uma interrupção à *kernel* (***trap)*** e o PC é salvaguardado na stack do sistema;

2. Uma rotina em assembler salvaguarda os registos e outra informação volátil que possa ser eventualmente destruída e invoca o sistema operativo;
3. O sistema operativo dá conta da existência do *page fault* e tenta descobrir qual a página pretendida. Informação normalmente armazenada num registo;
4. Uma vez conhecido o endereço virtual, o sistema verifica se o endereço é válido e se a protecção é consistente com o tipo de acesso. Caso contrário
aborta a situação. Em caso afirmativo o sistema procura uma página livre para substituição, senão o algoritmo escolhe uma página não livre para substituição;
5. Se a página escolhida para substituição não está limpa, ela é agendada para salvaguarda na memória secundária. Entretanto, dá-se uma mudança de contexto. O processo é suspenso e é permitido que outro processo, caso exista, execute até que a transferência da página para o disco esteja concluída;
6. A página é bloqueada para evitar que seja utilizada para outro fim;
7. Assim que a página estiver livre o sistema procura a página virtual pretendida e agenda o carregamento da página na memória principal. Entretanto, dá-se uma mudança de contexto. O processo é suspenso e é permitido que outro processo, caso exista, execute até que a transferência da
página para a memória principal esteja concluída;
8. Quando o *interrutp* do disco indicar que a página pretendida está disponível, a tabela de páginas é actualizada e a página é marcada como sendo uma página normal;
9. O sistema operativo devolve o controlo à rotina em assembler que o invocou, que recupera a informação do PC e dos registos e retorna ao espaço de endereçagem do processo, que retoma a execução como se não tivesse existido o *page fault*.

53. Partições estáticas e partições dinâmica

Num sistema multiprogramado organizado com **memória real**, podemos assumir que a parte residente do sistema operativo ocupa uma parte da memória, sendo a restante distribuída pelos processos activos.
A maneira mais fácil de distribuir a memória pelos múltiplos processos, consiste em dividir a memória em **partições estáticas**. Existem duas soluções: partições de tamanho igual ou partições de tamanho diferente.

Existem dois problemas com a utilização de uma partição estática com partições de tamanho igual:
• Um processo pode ser demasiado grande para o tamanho da participação;
• A utilização da memória é ineficiente, porque independentemente do tamanho do processo, este ocupa toda a partição. A esta situação dá-se o nome de **fragmentação interna**.
Alguns destes problemas podem ser resolvidos recorrendo a uma partição estática com partições de tamanho diferente.
Num sistema de partição estática com partições de tamanho igual a colocação de processos em memória é trivial, já que o processo pode ser colocado em qualquer partição. No caso de uma partição estática com partições de tamanho diferente existem duas possibilidades.

- Num sistema de **partição dinâmica** as partições são usadas de acordo com o tamanho dos processos. No entanto, devido à colocação e retirada da de processos da memória, vão aparecendo buracos entre as partições. A esta situação dá-se o nome de **fragmentação externa**.
Uma técnica para eliminar a fragmentação externa consiste em de tempos a tempos juntar os processos na memória de forma a eliminar os buracos. A esta técnica dá-se o nome de **compactação (*compaction*)**.

Quando a memória é atribuída aos processos de forma dinâmica, o sistema operativo tem que gerir a ocupação da memória. Existem duas formas de o fazer: um ***bitmap*** ou uma **lista ligada**.
Usando um *bitmap* a memória é dividida em unidades de alocação de memória e cria-se uma mapa com um bit para cada unidade de alocação. O bit igual a zero (0) significa que a unidade de alocação está livre e igual a um (1) significa que está ocupada. Obviamente que quanto menor for o tamanho da unidade de alocação, maior será a dimensão do *bitmap*.
Para alocar uma partição de memória com K unidades de alocação, é preciso procurar K bits iguais a 0 consecutivos no *bitmap*, e depois colocá-los a 1. Para libertar uma partição de memória é preciso colocar todos os bits das unidades de alocação constituintes da partição a 0. Um *bitmap* providencia assim uma forma simples de gerir a alocação de memória. Tem no entanto a desvantagem de ser lento.

Uma outra forma de gerir a memória consiste em manter uma lista ligada de segmentos, ou seja, grupos de unidades de alocação. Cada segmento pode corresponder a memória alocada a um processo, ou seja,

unidades de alocação de memória ocupadas, ou a buracos de memória entre dois processos, ou seja, unidades de alocação de memória livres.
Cada elemento da lista deve indicar se o segmento está ocupado (P de processo) ou se está livre (B de buraco), o índice da primeira unidade de alocação, o comprimento do segmento, ou seja o número de unidades de alocação e um ponteiro para o próximo elemento. Se a lista de segmentos for mantida ordenada por ordem do índice da primeira unidade de alocação, então é fácil actualizar a lista quando houver libertação de um segmento de memória.
Para alocar uma partição de memória com K unidades de alocação, é preciso percorrer a lista de forma a encontrar um buraco com comprimento K. Para tal, podem ser utilizados os algoritmos de colocação apresentados a seguir.

Para libertar uma partição de memória é preciso actualizar no segmento a indicação de buraco em vez de processo. E eventualmente, fundir esse segmento com o segmento que se encontra à sua direita ou à sua esquerda, ou em ambos os lados, caso eles sejam do tipo buraco.

54. Distinga segurança de protecção em termos de um sistema de operação.

Protecção (problema interno do SC):

- Desenvolvido paralelamente á multiprogramação
- Controla o acesso de programas ou mesmo de treads, processos ou utilizadores a recursos partilhados do sistema computacional -> para que não sejam destruidos ou modificados dados que mais tarde tenham informação errada (um exemplo são as regiões criticas, nas quais devem actuar apenas um processo de cada vez em regime de exclusão mútua)

Segurança (Problema devido a ambientes externos ao SC):

- Previne ou proibe acessos não autorizados ao SC e assegura que não sejam destruidos/alterados/roubados dados de proposito (virus ou roubo de tecnologia)
- Assegura a autenticação dos users do SC, para proteger a integridade da info armazenada no sistema (tanto dados como código), assim também como a protecção dos recursos a que ela pretença.

55. Indique alguns mecanismos que podem ser usados de modo a garantir a segurança quando um utilizador acede a um sistema de operação.

As medidas de segurança podem ser divididas em dois niveis: físico e humano

- Fisico -> O acesso ao sistema computacional enquanto hardware esta interdito a intrusos. (Exemplo: Computadores com informação muito importante poderá estar isolado do mundo exterior).
- Humano: Ao aceder ao SC o utilizador tem que ser autenticado com um login e uma password que lhe dão acesso somente aos recursos e processos que lhe foram atribuidos.
  Em termos de autenticação por password existem 3 tipos: Vulneráveis, encriptadas e "One-time passwords":

  - Vulneráveis: Sistema mais básico em que as passwords são facilmente descobertas, ou seja, é dificil de as manter secretas devido ao facto de alguns utilizadores usarem as mesmas passwords (exemplo: Utilizar o nome do animal de estimação como password).

  - Encriptadas: As passwords são primeiramente encriptadas por uma função f(x) que os programadores pensam ser impossivel de inverter, só depois de serem encripatadas é que são armazenadas. Quando o utilizador efectua o login com a sua password ela é encriptada com a função f(x) e comparada com a que está armazenada, se coincidirem é porque esta correcta. No entanto é possível aceder ao ficheiro das passwords encripatadas armazenadas e se um "hacker" fizer uma lista de palavras e as incriptar com a mesma função, poderá comparar com o ficheiro e descobrir a verdadeira password.

- "One-Time passwords": Um dos tipos é o utilizador e o SC conhecem um segredo que só é transmitido por meios que não permitem exposição dos mesmos e quando o utilizador fornece a password, esse segredo é usado na função f(secret, seed).

56. Mostre em que termos é que se pode proteger um sistema de ficheiros do acesso indevido. Tome como exemplo o Unix e indique quais as vantagens e inconvenientes do sistema aí implementado.

Para cada ficheiro existem no permissões diferente de acordo com o grupo a que o utilizador pertence, no linux usa-se 3 tipos de utilizadores: Owner (próprio utilizador), Group (grupo a que o utilizador pertence) e Universe (outros utilizadores).
Para cada utilizador são autorizadas as diversas operações que se poodem efectuar sobre o ficheiro. (read, write, delete, execute, Append (acrecentar informação no final do ficheiro), list (listar o nome e os atributos do ficheiro).
Nas permissões do Unix são definidos três campo de 3 bits, os campo é os grupos de utilizadores e os 3 bits são: rwx, onde r significa read acess, w siginfica write acess e x representa execution.

Vantagens: Acesso controlado aos vários ficheiros sem ser necessário criar uma premissão diferente para cada utilizador. Associa-se o utilizador a um grupo, poupando no espaço de disco que se teria de ter para criar uma lista com o nome de todos os utilizadores e respectivo acesso, e também se poupa no tempo de acesso ao ficheiro, pois não é preciso pesquisar a referida lista, processo que demoraria bastante mais tempo caso existissem muitos utilizadores.

As desvantagens: Com este sistema estamos limitados a três tipos de acessos diferentes, caso pretendesemos ter o owner com acesso total, o group com acesso de leitura e escrita e o Universe com acesso só de leitura, seria impraticavel barrar o acesso completo a um só (ou vários) utilizador(es).