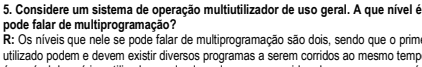
Capitulo I - ***Conceitos Introdutórios***

1.Descreva as duas perspectivas de definição de um sistema de operação. Mostre claramente em que circunstâncias cada uma delas é relevante.
As duas perspectivas de definição de um sistema de operação são o *Top-down e Botton-up*. **A Perspectiva 'top-down' (ou do programador):** 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 especificidades próprias; detecção de situações de erro e estabelecimento de uma resposta adequada. ***Perspectiva 'bottom-up' (ou do construtor):*** 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.
2. O que são *chamadas ao sistema?* Dê exemplos válidos para o Unix (recorde que o Linux não é mais do que uma implementação especifica do Unix). Explique qual é a sua importância no estabelecimento de um *interface de programação de aplicações* (API).
R: As chamadas ao sistema, são comandos dados ao sistema em que o sistema operativo executa uma série de operações, existem de certa forma para simplificar a vida aos utilizadores, são exemplos, disso o CD, o LL, etc. Desta forma permite estabelecer um interface, na interligação entre diversos programas de forma a que um programa possa utilizar funcionalidades implementadas por outro, permitindo por exemplo, que um processo controle outro.
3. Os sistemas de operação actuais apresentam um ambiente de interacção com o utilizador de características eminentemente gráficas. Contudo, quase todos eles fornecem em alternativa um ambiente de interacção baseado em linhas de comandos. Qual será a razão principal deste facto?
R: A razão principal deste facto, é porque, a utilização do modo de comandos permite uma rentabilização no acesso ao hardware que o ambiente gráfico não proporciona e permite tb uma comunicação com o hardware e a execução de primitivas que o ambiente gráfico não permite. Este modo de utilização exige do utilizador conhecimentos mais profundos sobre o sistema computacional. Além disso a interacção utilizador com sistema é mt mais rápida que com a interacção gráfica.
4. Distinga multiprocessamento de multiprogramação. Será possível conceber-se multiprocessamento sem multiprogramação? Em que circunstâncias?
R.: Na multiprogramação temos o processamento no tempo (multiplexagem no tempo), daí que vários programas são processados por um único processador, ou seja , temos processamento paralelo. 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 procurando eliminar os tempos mortos que ocorrem durante transacções de informação com dispositivos de I/O . 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 processador lhe está inteiramente dedicado.
Paralelismo - é a capacidade apresentada por um sistema computacional de poder executar em simultâneo dois ou mais programas; o que exige que o sistema computacional seja formado por mais do que um processador (um por cada programa em execução simultânea). Quando tal acontece, diz-se que o sistema operativo associado tem características de ***multiprocessamento. Concorrência*** - 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. Quando tal acontece, diz-se que o sistema operativo associado tem características de ***multiprogramação***.



Num monoprocessador (sistema computacional com um único processador), os programas A e B estão a ser executados em concorrência.

5. Considere um sistema de operação multiutilizador de uso geral. A que nível é que nele se pode falar de multiprogramação?
R: Os níveis que nele se pode falar de multiprogramação são dois, sendo que o primeiro em cada utilizado podem e devem existir diversos programas a serem corridos ao mesmo tempo; o segundo é ao nível dos vários utilizados, podendo cada um ser considerado um programa a nível abstracto, havendo a necessidade do sistema operativo comutar de uma forma genérica entre cada utilizador e particularmente cada aplicações de cada utilizador.
Exemplo:

1	2	3	4

Utilizador A Utilizador B

6. Os sistemas de operação de tipo *batch* são característicos dos anos 50 e 60, quando o custo dos sistemas computacionais era muito elevado e era necessário rentabilizar a todo o custo o *hardware*. A partir daí, com a redução progressiva de custos, os sistemas tornaram-se interactivos, visando criar um ambiente de interacção com o utilizador o mais confortável e eficiente possível. Será que hoje em dia ainda se justificam sistemas deste tipo? Em que circunstâncias?
Em ultima análise sim, porque do ponto de vista computacional os sistemas operativos do tipo batch são os mais eficientes na medida que não são mais que programas executados sequencialmente. Desta forma, o sistema vocacionado para o cálculo maciço de informação em que a interacção es com o utilizador são praticamente nulas, podem ser úteis, devido à sua eficácia a nível de interacção com o hardware.
7. Quais são as semelhanças e as diferenças principais entre um sistema de operação de rede e um sistema distribuído?
R: Sistema operativo de rede: *Objectivo:* tirar partido das facilidades actuais de interligação de sistemas computacionais (ao nível de hardware) para estabelecer um conjunto de serviços comuns a toda uma comunidade. Serviços: transferência de ficheiros entre sistemas computacionais (ftp); partilha de sistemas de ficheiros remotos, criando a ilusão de que são locais (NFS); partilha de dispositivos remotos (impressoras, etc.); acesso a sistemas computacionais remotos (telnet, remote login); correio electrónico (E-mail); acesso à Internet. **Sistema operativo distribuído:** • *Objectivo:* tirar partido das facilidades actuais de construção de sistemas computacionais com processadores múltiplos, ou de interligação de sistemas computacionais distintos, para estabelecer um ambiente integrado muito mais eficiente de interacção com o(s) utilizador(es). • *Metodologia:* garantir uma independência tão completa quanto possível do(s) processador(es), ou sistema(s) computaciona(l-is), onde os programas são executados tendo em vista; o balanceamento estático e/ou dinâmico de carga; o aumento da velocidade de processamento por incorporação de novos processadores ou sistemas computacionais; a paralelização de aplicações; a implementação de mecanismos de tolerância a falhas.
Características comuns aos sistemas de operação actuais: • apresentam um ambiente gráfico de interacção com o utilizador; • são sistemas interactivos multiutilizadores e multitarefa: - permitem a interacção em simultâneo com múltiplos utilizadores; - permitem que cada utilizador tenha em simultâneo múltiplos programas em execução; • implementam uma organização de memória virtual; • são sistemas de operação de rede: - permitem o acesso de um modo quase indistinto a sistemas de ficheiros e a dispositivos de entrada / saída locais e remotos; - incluem aplicações que permitem, entre outras coisas, o *login* em máquinas remotas, o correio electrónico e a navegação na Internet; • têm uma colecção muito grande de *device drivers* para potenciar a interligação de tipos muito variados de dispositivos de entrada / saída; • permitem em muitos casos a ligação dinâmica de dispositivos (*plug and play*).

8. Os sistemas de operação de uso geral actuais são tipicamente *sistemas de operação de rede.* Faça a sua caracterização.
R: Devido ao advento das comunicações actuais, a interligação, a partilha de informação enquanto bem necessita e os utilizadores cada vez mais, procurar sistema em que as suas aplicações são de envio e recepção de informação, isto é, não o processamento puro, mas sim a troca de informação a que o nível de sistema operativo que se reflecte num *sistema operativo de rede* que tenha por um lado como *objectivo* tirar partido das facilidades actuais de interligação de sistemas computacionais (ao nível de hardware) para estabelecer um conjunto de serviços comuns a toda uma comunidade. E por outro lado, que disponibiliza serviços tais como: transferência de ficheiros entre sistemas computacionais (ftp); partilha de sistemas de ficheiros remotos, criando a ilusão de que são locais (NFS); partilha de dispositivos remotos (impressoras, etc.); acesso a sistemas computacionais remotos (telnet, remote login); correio electrónico (E-mail); acesso à Internet.
9. Os sistemas de operação dos *palmtops* ou *personal digital assistants* (PDA) têm características particulares face ao tipo de situações em que são usados. Descreva-as.
R: Os *palmtops* ou *personal digital assistants* (PDA), são dispositivos com características físicas e de vocação particular, são acima de tudo limitados em ferramentas (devido a limitação de tamanho, etc.) que limitam o processamento; por outro lado estão vocacionados para apresentarem um ambiente user-friendly que permite uma simples interacção utilizador S.O. Boa facilidade de comunicação com exterior . Podemos pois considerar que se trata de um sistema operativo com característica únicas vocacionadas para a recepção de dados e apresentação dos mesmos através de um ambiente gráfico.

10. O sistema de operação Linux resulta do trabalho cooperativo de muita gente, localizada em muitas partes do mundo, que comunica entre si usando a Internet. Mostre porque é que este facto é relevante para a arquitectura interna do sistema.
R: O sistema de operação Linux assenta numa - *arquitectura em camadas* - trata-se de uma decomposição hierárquica do kernel, em que cada nova módulo é construído sobre os módulos anteriores e usa apenas as operações que estes lhe fornecem; este mecanismo de decomposição, embora conceptualmente muito seguro, exige uma cuidadosa definição das funcionalidades de cada camada e cria, se o número de camadas for muito elevado, um *overhead* de comunicações que não é desprezável;

Capitulo II - ***Gestão do Processador***

1. A modelação do ambiente de multiprogramação através da activação e desactivação de um conjunto de processadores virtuais, cada um deles associado a um processo particular, supõe que dois factos essenciais relativos ao comportamento dos processos sejam garantidos. Quais são eles?
R: A multiprogramação, ao criar uma imagem de aparente simultaneidade na execução de diferentes programas pelo mesmo processador, torna muito complexa a percepção das diferentes actividades que estão em curso. Esta imagem pode, porém, ser simplificada se, em vez de procurar seguir-se o percurso do processador nas suas constantes comutações entre processos, se supuser a existência de um conjunto de processadores virtuais, um por cada processo que concorrentemente coexiste, e se admitir que os processos associados são executados em paralelo. Para que tal modelo seja viável, é preciso garantir que: a execução dos processos não é afectada pelo instante, ou local no código, onde ocorre a comutação, não são impostas quaisquer restrições relativamente aos tempos de execução, totais ou parciais, dos processos.
2.Qual é a importância da tabela de controlo de processos (*PCT*) na operacionalização de um ambiente de multiprogramação? Que tipos de campos devem existir em cada entrada da tabela?
R: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.
Os tipos de campos devem existir em cada entrada da tabela *PCT* são: *identificadores* - do processo, do pai do processo e do utilizador a que o processo pertence; *caracterização do espaço de endereçagem* - sua localização (em memória principal ou na área de 'swapping'), de acordo com o tipo de organização de memória estabelecido; *contexto do processador* - valores de todos os registos internos do processador no momento em que se deu a comutação do processo; *contexto de I/0* - informação sobre os canais de comunicação e 'buffers' associados; *informação de estado e de 'scheduling'* - estado de execução do processo (de acordo com o diagrama de estados) e outra informação associada.
3. O que é o *scheduling* do processador? Que critérios devem ser satisfeitos pelos algoritmos que o põem em prática? Quais são os mais importantes num sistema multiutilizador de uso geral, num sistema de tipo *batch* e num sistema de tempo real?
R: O *scheduling* do processador é o sistema que permite a organização da comutação entre processos, isto é, permite calendarizar de forma eficaz e racional a atribuição dos recursos físicos aos diversos processos num ambiente de multiprogramação.
Desta forma a atribuição desses recursos deverá seguir determinados objectivos principais a serem satisfeitos numa política de 'scheduling' do processador que são os seguintes: Justiça - direito de cada processo obter uma parcela razoável de tempo de processador; Throughput - maximizar o número de processos terminados por unidade de tempo; Tempo de resposta - minimizar o tempo de resposta de utilizadores interactivos; Tempo de turn around - minimizar o tempo gasto pelo processador em tarefas relacionadas com a implementação da política escolhida; Eficiência - minimização do tempo associado com a selecção do próximo processo a que vai ser atribuído o processador e com a comutação propriamente dita.
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

4. Descreva o diagrama de estados do *scheduling* do processador em três níveis. Qual é o papel desempenhado por cada nível? Num sistema de tipo *batch* multiprogramado fará sentido a existência de três níveis de *scheduling*?
O papel desempenhado por cada nível será:
1.Scheduling de baixo nível (Gestão do Processador): Ao longo da sua existência, um processo vai encontrar-se em situações distintas, designadas por ***estados***, as mais importantes das quais são as seguintes: • ***run*** - quando detém a posse do processador e está, portanto, em execução; • ***ready-to-run*** - quando aguarda a atribuição do processador para entrar em execução; • ***blocked*** - quando está impedido de continuar, até que um acontecimento externo ocorra (acesso a um recurso, completamento de uma operação de entrada- saída, etc.). As transições entre estados resultam normalmente de uma intervenção externa, mas podem nalguns casos ser despoletadas pelo próprio processo. A parte do sistema operativo que lida com estas transições, chama-se '***scheduler*** *(do processador)* e constitui parte integrante do seu núcleo central ('***kernel***'), responsável por lidar com as interrupções e agendar a atribuição do processador e dos outros recursos do sistema computacional.
dispatch - um dos processos da fila de espera dos 'processos prontos a serem executados' é seleccionado pelo '*scheduler*' para execução; *timer-run-out* - o 'scheduler' verificou que o processo em execução esgotou o intervalo de tempo de processador que lhe tinha sido atribuído; *sleep* - o processo decidiu que está impedido de prosseguir, ficando a aguardar a ocorrência de um acontecimento externo; *wake up* - o acontecimento externo que o processo aguardava, ocorreu.

2. Scheduling de médio nível (Gestão da Memória Principal): costuma-se criar em memória de massa uma extensão à memória principal, comummente designada por ***área de 'swapping'***, que funciona como área de armazenamento secundária. Liberta-se, assim, espaço em memória principal para a execução de processos que, de outro modo, não poderiam existir e potencia-se, portanto, um aumento da taxa de utilização do processador.
Surgem então dois novos estados: • ***suspended-ready*** - quando o espaço de endereçagem do processo é transferido para a ***área de 'swapping'*** (*swapped out*), ficando suspensa a continuação da sua execução, enquanto tal situação se mantiver; • ***suspended-block*** - quando o espaço de endereçagem do processo é transferido para a memória principal (*swapped in*), retomando eventualmente a sua execução.
suspend - suspensão de um processo, por transferência do seu espaço de endereçagem para a área de 'swapping';
resume - eventual retoma da execução de um processo, por transferência do seu espaço de endereçagem para a memória principal;
event completion - o acontecimento externo que o processo aguardava, ocorreu.

3. *Scheduling de alto nível (Gestão do ambiente de Multiprogramação):*
activate - lançamento do processo em execução; o processo pode ser colocado na *fila de espera dos processos prontos a serem executados*, se houver espaço em memória principal para carregar o seu espaço de endereçamento, ou *suspenso*, em caso contrário;
exit - terminação normal do processo;
abort - terminação anormal do processo, provocada pela ocorrência de um erro fatal ou por acção de um processo com autoridade necessária para isso.

Num sistema de tipo *batch* multiprogramado, não fará sentido a existência de três níveis de *scheduling,* visto que, nele não existe multiutilizador, isto é, o batch tende a bloquear os processos e a correr outros para optimiza-los.

5. Os estados *READY-TO-RUN* e *BLOCKED*, entre outros, têm associadas filas de espera de processos que se encontram nesses estados. Conceptualmente, porém, existe apenas uma fila de espera associada ao estado *READY-TO-RUN*, mas filas de espera múltiplas associadas ao estado *BLOCKED.* Em princípio, uma por cada dispositivo ou recurso. Porque é que é assim?
R: Porque o CPU é o único recurso que a fila de espera associado ao estado READY-TO-RUN espera ser atribuída, competindo entre si para esse fim. No que toca a fila de espera associado ao estado BLOCKED, o ponto de vista é outro, visto que, eles estão bloqueados à espera de um recurso "genérico", (como por exemplo: I/O, memória, ect.) logo para cada recurso existe uma fila de espera.
6. Indique quais são as funções principais desempenhadas pelo *kernel* de um sistema de operação. Neste sentido, explique porque é que a sua operação pode ser considerada como um *serviço de excepções.*
R: Uma das funções principais desempenhadas pelo *kernel* de um sistema de operação é o atendimento de excepções, ora podemos considerar que o kernel funciona como um ambiente de processamento uniforme em que as requisições externos por parte dele (excepções) ele responde com rotinas próprias que permitem o tratamento de erros e ocorrências. Desta forma toda a operação do kernel pode ser, visto como, atendimento a excepção desde uma linha de comando a uma comutação de processamento.
Assim, para que o sistema operativo, ao nível do ***kernel***, funcione no modo privilegiado, com total acesso a toda a funcionalidade do processador, as *chamadas ao sistema* associadas, quando não despoletadas pelo próprio 'hardware', são implementadas a partir de instruções ***trap***. Cria-se, portanto, um ambiente operacional uniforme, em que todo o processamento pode ser encarado como o *serviço de excepções*. Nesta perspectiva, a comutação de processos pode ser visualizada globalmente como uma vulgar rotina de serviço à excepção, apresentando, porém, uma característica peculiar que a distingue de todas as outras: *normalmente, a instrução que vai ser executada, após o serviço da excepção, é diferente daquela cujo endereço foi salvaguardado ao dar-se início ao processamento da excepção.*
7. O que é uma *comutação de contexto*? Descreva detalhadamente as operações mais importantes que são realizadas quando há uma comutação de contexto.
R: Uma *comutação de contexto,* consiste em alterar tudo o que tem a ver com o processo em causa (contexto do CPU e I/O) e alterar por outra previamente agendada. As operações mais importantes que são realizadas quando há uma comutação de contexto são as seguintes: - Salvaguarda da caracterização do espaço de endereçamento actual e dos contextos do processador e de I/O na entrada da PCT referente ao processo; - Actualização do estado do processo e de outra informação associada na entrada da PCT referente ao processo; - Selecção do próximo processo para execução da fila de espera READY-TO-RUN, usando um algoritmo específico de *scheduling*; - Colocação no estado RUN do processo agendado para execução e modificação de informação associada por actualização da entrada da PCT referente ao processo; - Restauro da caracterização do espaço de endereçamento actual e dos contextos do processador e de I/O por transferência da entrada da PCT referente ao processo.
8. Classifique os critérios devem ser satisfeitos pelos algoritmos de *scheduling* segundo as perspectivas sistémica e comportamental, e respectivas subclasses. Justifique devidamente as suas opções.
R: Os critérios a serem satisfeitos pelos algoritmos de *scheduling* do processador podem ser enquadrados segundo duas perspectivas: • *perspectiva sistémica:* -critérios orientados para o utilizador ou à rentabilização da CPU, isto é, ao longo do tempo o scheduling dá mais importância ao utilizador ou à necessidade de manter a CPU ocupada ; - *critérios orientados para o sistema:* estão relacionados com o uso eficiente dos recursos do sistema de operação; • *perspectiva comportamental:* - critérios orientados para o desempenho: são quantitativos e passíveis, portanto, de serem medidos; - outro tipo de critérios: são qualitativos e difíceis de serem medidos de uma maneira directa.
9. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma delas.
R: As prioridades podem ser de dois tipos: • ***prioridades estáticas*** - quando o método de definição é determinístico independe da forma de como o sistema evolui ; • ***prioridades dinâmicas*** - quando o método de definição depende da história passada de execução do processo;
Do ponto de vista da, 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. Enquanto que na *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. 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. 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. Quando o sistema computacional tem uma carga muito grande de processos I/O-intensivos, os processos CPU-intensivos correm o risco de adiamento indefinido, se a disciplina de 'scheduling' anterior for aplicada sem qualquer correcção. Um meio de alterar esta situação é incorporar no cálculo da prioridade o tempo que o processo aguarda a atribuição do processador no estado READY-TO-RUN. Técnicas com estas características constituem aquilo que muitas vezes se designa por aging dos processos.
10. Num sistema de operação multiutilizador de uso geral, há razões diversas que conduzem ao estabelecimento de diferentes classes de processos com direitos de acesso ao processador diferenciados. Explique porquê.
R: Devido a necessidade de certos processos serem atendidos num tempo T definido. Por exemplo, se for preciso responder a um sistema áudio ele terá que ser feita em tempo real sob pena de o atraso significar um erro que inviabiliza a necessidade no futuro dessa time-slot, e é isto a nível dos utilizadores, porque existe a questão do sistema operativo que terá de ter privilégios excepcionais com vista ao atendimento de excepção e gestão de kernel.
11. Entre as políticas de *scheduling preemptive* e *non-preemptive*, ou uma combinação das duas, qual delas escolheria para um sistema de tempo real? Justifique claramente as razões da sua opção.
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.

12. Foi referido nas aulas que os sistemas de operação de tipo *batch* usam principalmente uma política de *scheduling non-preemptive*. Será, porém, uma política pura, ou admite excepções?

R: 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).

13. Justifique claramente se a disciplina de *scheduling* usada em Linux para a classe *SCHED_OTHER* é uma política de prioridade estática ou dinâmica?

R: A disciplina de *scheduling* usada em Linux para a classe *SCHED_OTHER* é uma política de prioridade estática, na medida em que o Linux utiliza um algoritmo baseado em *créditos* no estabelecimento da sua prioridade, ora cada vez que o processador é atribuído a um processo a sua prioridade é decrementada, sendo desta forma toda a *scheduling* delimitado logo estático.

14. O que é o *aging* dos processos? Dê exemplos de duas disciplinas de *scheduling* com esta característica, mostrando como ela é implementada em cada caso.

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». O aging é algo de mt útil pois evita o starvation . ex .sistemas I/O intensivos que pela necessidade de resposta iriam ganhar sucessivamente a prioridade no acesso ao processador.

15. Distinga *threads* de processos. Assumindo que pretende desenvolver uma aplicação concorrente usando um dos paradigmas, descreva o modo como cada um afecta o desenho da arquitectura dos programas associados.

O conceito de *processo* corporiza as propriedades seguintes: • *pertença de recursos* - um espaço de endereçamento próprio e um conjunto de canais de comunicação com os dispositivos de entrada / saída; • *fio de execução* (*thread*) - um *program counter* que sinaliza a localização da instrução que deve ser executada a seguir, um conjunto de *registos internos do processador* que contêm os valores actuais das variáveis em processamento e um *stack* que armazena a história de execução (um *frame* por cada rotina invocada e que ainda não retornou).

Estas propriedades, embora surjam reunidas num *processo*, podem ser tratadas separadamente pelo sistema de operação. Quando tal acontece, os *processos* dedicam-se a

agrupar um conjunto de recursos e os *threads*, também conhecidos por *light weight processes*, constituem entidades executáveis independentes dentro do contexto de um mesmo processo. *Multithreading* representa então a situação em que é possível criar-se *threads* múltiplos de execução no contexto de um processo. Se quisermos recorrer a um dos paradigmas , por exemplo as threads significa que necessitamos apenas de um espaço de endereçamento evitando desta forma a comutação de contexto I/O, criar novo espaço de endereçamento tornando desta forma tudo mais simples.

16. Indique, justificadamente, em que situações um ambiente *multithreaded* pode ser vantajoso.

R: ***Vantagens de um ambiente multithreaded:*** • As vantagens de um ambiente multithread são evidentes, na medida em que precisamos de menos contexto I/O menos espaço de endereçamento...

maior simplicidade na decomposição da solução e maior modularidade naimplementação - programas que envolvem múltiplas actividades e atendimento a múltiplas solicitações são mais fáceis de conceber e mais fáceis de implementar numa perspectiva concorrencial do que numa perspectiva puramente sequencial; • *melhor gestão de recursos do sistema computacional* - havendo uma partilha do espaço de endereçamento e do contexto de *I/O* entre os *threads* em que uma aplicação é dividida, torna-se mais simples gerir a ocupação da memória principal e o acesso aos dispositivos de entrada / saída de uma maneira eficaz; • *eficiência e velocidade de execução* - uma decomposição da solução em *threads* por oposição a processos, ao envolver menos recursos por parte do sistema de operação, possibilita que operações como a sua criação e destruição e a mudança de contexto, se tornem menos pesadas e, portanto, mais eficientes; além disso, em multiprocessadores simétricos torna-se possível calendarizar para execução em paralelo múltiplos *threads* da mesma aplicação, aumentando assim a sua velocidade de execução.

17. Que tipo de alternativas pode o sistema de operação fornecer à implementação de um ambiente *multithreaded*? Em que condições é que num multiprocessador simétrico os diferentes *threads* de uma mesma aplicação podem ser executados em paralelo?

R: Na utilização de diversos processos, a implementação dos threads possui uma vantagem devido a simplicidade de implementação e não há alternativas a este nível, mas sim, em outras circunstâncias, paga-se sempre algo, aqui é simplificada e velocidade pelo uso de um bloqueio estragar tudo. Se o multiprocessador partilhar memoria e I/O cada thread pode correr num processador distinto, denotar a vantagem desta implementação principalmente a nível das threads de gestão de *scheduling* do sistema operativo na medida que o bloqueio de uma não necessita de mudar o contexto (bloqueio de todo o sistema multiprocessador). Num multiprocessadores simétricos torna-se possível calendarizar para execução em paralelo múltiplos *threads* da mesma aplicação, aumentando assim a sua velocidade de execução.

18. Explique como é que os *threads* são implementados em Linux.

R: O Linux lida com a questão da implementação de *threads* de um modo muito artificioso. À parte da *chamada ao sistema fork* que cria um novo processo a partir dum já existente por cópia integral do seu contexto alargado (espaço de endereçamento, contexto de *I/O* e contexto do processador), existe uma outra, *clone*, que cria um novo processo a partir de um já existente por cópia apenas do seu contexto restrito (contexto do processador), partilhando o espaço de endereçamento e o contexto de *I/O* e iniciando a sua execução pela invocação de uma função que é passada como parâmetro.

Assim, não há distinção efectiva entre processos e *threads*, que o Linux designa indiferentemente de *tasks*, e eles são tratados pelo *kernel* da mesma maneira.

19. O principal problema da implementação de *threads*, a partir de uma biblioteca que fornece primitivas para a sua criação, gestão e *scheduling* no nível utilizador, é que quando um *thread* particular executa uma *chamada ao sistema* bloqueante, todo o processo é bloqueado, mesmo que existam *threads* que estão prontos a serem executados. Será que este problema não pode ser minimizado?

R. Para se minimizar este problema seria necessário criar subtabelas na TCP que permitiam ao *scheduling do kernel* comutar entre as *threads* salvaguardando apenas o contexto do processador. Mas, para tal seriam necessárias alterações ao nível da definição de prioridades e de *scheduling* de forma a optimizar a utilização da principal vantagem das threads a nível de gestão do processador a velocidade de comutação devido a comutação de contexto ser limitada.

20. Num ambiente *multithreaded* em que os *threads* são implementados no nível utilizador, vai haver um par de *stacks* (sistema / utilizador) por *thread*, ou um único par comum a todo o processo? E se os *threads* forem implementados ao nível do *kernel*? Justifique.

R: Sendo o stack pertencente ao espaço de endereçamento apenas um par deverão existir, mas poderá ser ou não vantajoso a nível do utilizador haver mais do que essa stack, ou a thread em causa lança uma nova stack dentro desse espaço comum (o que é complicada) ou o sistema em si possibilita a criação de stack em cada thread. Na minha opinião, uma stack única é o mais racional, a thread não é um processo, possui os seus pontos fortes e fracos e deve ser utilizada dessa forma. A nível do kernel, a situação é diferente, visto que, não é lançada pelo utilizador, logo poderá ser implementada outra filosofia atendendo aos privilégios que possui.

Capitulo III - *Comunicação entre Processos*

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

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

Em cada categoria o problema da exclusão mútua coloca-se da seguinte forma:

• *processos independentes* que competem por acesso a um recurso comum do sistema computacional; • é da responsabilidade do *SO* garantir que a atribuição do recurso seja feita de uma forma controlada para que não haja perda de informação; • isto impõe, em geral, que só um processo de cada vez pode ter acesso ao recurso (*exclusão mútua*). • Processos cooperantes que partilham informação ou comunicam entre si; • é da responsabilidade dos processos envolvidos garantir que o acesso à região partilhada seja feito de uma forma controlada para que não haja perda de informação; • isto impõe, em geral, que só um processo de cada vez pode ter acesso à região partilhada (*exclusão mútua*); • o canal de comunicação é tipicamente um recurso do sistema computacional, logo o acesso a ele está enquadrado na *competição* por acesso a um recurso comum.

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

Quando dois recursos necessitam de um recurso em causa eles competem pela posse resultando no bloqueio de um e a continuação de um deles. São exemplo disso o jantar dos filósofos quando dois deles competem pelo mesmo garfo ou quando por exemplo duas renas a tentar competir por escrever o seu ID de forma a ???? por exemplo.

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

Uma região critica é código que permite o acesso a região partilhada. É preciso estabelecer condições para aceder a esta região critica, são estas: • garantia efectiva de imposição de exclusão mútua - o acesso à região crítica associada a um mesmo recurso, ou região partilhada, só pode ser permitido a um processo de cada vez, de entre todos os processos que competem pelo acesso; • independência da velocidade de execução relativa dos processos intervenientes, ou do seu número - nada deve ser presumido acerca destes factores; • um processo fora da região crítica não pode impedir outro de lá entrar; • não pode ser adiada indefinidamente a possibilidade de acesso à região crítica a qualquer processo que o requeira; • o tempo de permanência de um processo na região crítica é necessariamente finito.

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

Condição de corrida é quando no acesso a zona partilhada em que a leitura para teste e posterior escrita são inconsistentes. Por exemplo:

P1 lê região crítica	P1
P2 lê região crítica	P1
P1 escreve região crítica	P2
P2 testa e escreve na região crítica	P2

Logo, P2 não escreve de acordo com o esperado

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

Deadlock é quando dois ou mais processos ficam a aguardar eternamente o acesso às regiões críticas respectivas, esperando acontecimentos que, se pode demonstrar, nunca irão acontecer; o resultado é, por isso, o bloqueio das operações; enquanto que o adiamento indefinido é quando um ou mais processos competem pelo acesso a uma região crítica e, devido a uma conjunção de circunstâncias em que surgem continuamente processos novos que o(s) ultrapassam nesse desígnio, o acesso é sucessivamente adiado; está-se, por isso, perante um impedimento real à continuação dele(s). No caso do problema produtores/consumidores, se os produtores estiverem sempre com acesso à região crítica teremos starvation perante os consumidores. (um ou mais)

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

soluções software - são soluções que, quer sejam implementadas num monoprocessador, quer num multiprocessador com memória partilhada, supõem o recurso em última instância ao *conjunto de instruções básico* do processador; ou seja, as instruções de transferência de dados de e para a memória são de tipo *standard*: leitura e escrita de um valor; a única suposição adicional diz respeito ao caso do multiprocessador, em que a tentativa de acesso simultâneo a uma mesma posição de memória por parte de diferentes processadores é necessariamente serializada por intervenção de um árbitro;

soluções hardware - são soluções que supõem o recurso a instruções especiais do processador para garantir, a algum nível, a atomicidade na leitura e subsequente escrita de uma mesma posição de memória; são muitas vezes suportadas pelo próprio sistema de operação e podem mesmo estar integradas na linguagem de programação utilizada.

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

A solução que Dijkstra propôs baseia-se na alternância proposta por Dekker é de facto uma generalização em que os pressupostos são: a definição de quem entra primeiro é aleatório, o problema é que ao generalizarmos para N, a escolha sucessiva vai sendo aleatória e o facto do processo estar muito tempo à espera não ajuda a ser atendido mais rapidamente, logo adiamento indefinido.

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

O que os distingue é que 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. Enquanto que o de Dekker resolve o conflito de acesso entre dois processos competidores usando um mecanismo de alternância estrita.

O algoritmo de Peterson é passível de uma extensão em N porque os processos em causa há mais tempo estarão mais perto do acesso logo não há adiamento indefinido, e no caso de Dekker cada processo ao ser eliminado recomeça a escada, o que pode provocar adiamento indefinido.

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

A diferença é que a fila em vez de saber quando alguém se despacha sabe quando alguém chega:

1				
2	1			
	2	1		

5	4	3	2	1

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

O busy waiting (pooling) acontece quando um processo está continuamente a testar uma condição até que ela seja satisfeita. 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 a um recurso ou uma região partilhada em que um segundo processo se encontra na altura no seu interior, faz com que o intervalo de tempo de atribuição do processador se esgote sem que qualquer trabalho útil tenha sido realizado; • *constrangimentos no estabelecimento do algoritmo de scheduling* - numa política *preemptive* 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 interromper a execução do outro.

Mas em sistemas multiprocessador o busy waiting é muito eficaz na medida em que cada processador é dedicado e o busy waiting é vantajoso relativamente a um bloqueio

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

Se garantirem a atomicidade de uma operação do tipo test-and-set (tas) estamos a resolver o problema na medida em que este tipo de operação usam variáveis chamados flags de locking que após serem testados com sucesso, mudam o seu estado, qualquer processo que a seguir tentar entrar nessa região entrará em busy waiting ate que o processo que fez o locking faça unlock da flag de locking.

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

Um *semáforo* é um dispositivo de sincronização, 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 executou a operação é bloqueado e a sua identificação é colocada na fila de espera queue;

sem_up - se houver processos bloqueados na fila de espera queue, um deles 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.

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

Monitores são paradigmas implementadas ao nível da linguagem de programação que permitem a actuação em variáveis de controlo de modo simples capazes de serem sincronizados através de duas operações:

wait - o thread que invoca a operação é bloqueado na variável argumento e é colocado fora do monitor para possibilitar que aguarda acesso, possa prosseguir;

signal - se houver threads bloqueados na variável de condição deles é acordado; caso contrário, nada acontece.

As duas operações implementam soluções de sincronização e por outro lado podem ser usados para entra/saída de regiões criticas visto que quando um thread entra no monitor é feita em exclusão mútua, logo o acesso ao monitor impõe as condições de acesso a uma região partilhada. No que toca à sincronização temos duas operações fundamentais: wait e signal

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

Existem três tipos de monitores que se distinguem pela saída do monitor do thread após o sinal:

monitor de Hoare - o thread que invoca a operação de signal é colocado fora do monitor para que o thread acordado possa prosseguir; é muito geral, mas sua implementação exige a existência de um stack, onde são colocados os threads postos fora do monitor por invocação de signal;

monitor de Brinch Hansen - o thread que invoca a operação de signal liberta imediatamente o monitor (signal é a última instrução executada); é simples de implementar, mas pode tornar-se bastante restritivo porque só há possibilidade de execução de um signal em cada invocação de uma primitiva de acesso;

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

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

• vantagens :- suporte ao nível do sistema de operação - porque a sua implementação é feita pelo kernel, as operações sobre semáforos estão directamente disponíveis ao programador de aplicações, constituindo-se como uma biblioteca de chamadas ao sistema que podem ser usadas em qualquer linguagem de programação; - universalidade - são construções de muito baixo nível e podem, portanto, devido à sua versatilidade, ser usadas no desenho de qualquer tipo de soluções;

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

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

O paradigma da passagem de mensagens não envolve partilha de espaço de endereçamento e a sua aplicação, de um modo mais ou menos uniforme, é igualmente válida tanto em ambientes monoprocessador, como em ambientes multiprocessador, ou de processamento distribuído. Apenas se implementa um canal de comunicação de forma que processos cooperantes troquem informações entre si. Perante a possibilidade de troca de informações sem acesso a essa região está resolvida a questão, pois a troca de mensagens, e por inerência de informação, está resolvido, sendo desnecessário a utilização da região partilhada.

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

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

18. Que tipo de mecanismos de sincronização podem ser introduzidos nas primitivas de comunicação por passagem de mensagens? Caracterize os protótipos das funções em cada caso (suponha que são escritas em Linguagem C).
Podemos utilizar um sistema do tipo utilizando no jantar dos filósofos em que usamos duas funções uma bloqueante e outra desbloqueante com vista a sincronização.
O grau de sincronização existente pode ser classificado em dois níveis:
• sincronização não bloqueante - quando a sincronização é da responsabilidade dos processos intervenientes; a operação de envio envia a mensagem e regressa sem qualquer informação sobre se a mensagem foi efectivamente recebida; a operação de recepção, por seu lado, regressa independentemente de ter sido ou não recebida uma
mensagem;
/* operação de envio */
void msg_send_nb (unsigned int destid, MESSAGE msg);
/* operação de recepção */
void msg_receive_nb (unsigned int srcid, MESSAGE *msg, BOOLEAN *msg_arrival);

sincronização bloqueante - quando as operações de envio e de recepção contêm em si mesmas elementos de sincronização; a operação de envio envia a mensagem e bloqueia até que esta seja efectivamente recebida; a operação de recepção, por seu lado, só regressa quando uma mensagem tiver sido recebida;
/* operação de envio */
void msg_send (unsigned int destid, MESSAGE msg);
/* operação de recepção */
void msg_receive (unsigned int srcid, MESSAGE *msg);

19. O que são sinais? O *standard* Posix estabelece que o significado de dois deles é mantido em aberto para que o programador de aplicações lhes possa atribuir um papel específico. Mostre como poderia garantir o acesso a uma região crítica com exclusão mútua por parte de dois processos usando um deles.
***Sugestão* - Veja no manual *on-line* como se pode usar neste contexto a chamada ao sistema *wait*.**
Sinais são mecanismos que o sistema põe à disposição do kernel, de qualquer processo ou utilizador que após a sua activação despoleta uma rotina de serviço (embora pode haver vários tipos de resposta a sinais) que terá de ignorar até resposta imediata, chamadas ao sistema.
Um sinal constitui uma interrupção produzida no contexto de um processo onde lhe é comunicada a ocorrência de um acontecimento especial. Pode ser despoletado pelo kernel, em resultado de situações de erro ao nível hardware ou software, pelo próprio processo, por outro processo, ou pelo utilizador através do dispositivo standard de entrada / saída.
Tal como o processador no tratamento de excepções, o processo assume uma de três atitudes possíveis relativamente a um sinal
• ignorá-lo - não fazer nada face à sua ocorrência;
• bloqueá-lo - impedir que interrompa o processo durante intervalos de processamento bem definidos;
• executar uma acção associada - pode ser a acção estabelecida por defeito quando o processo é criado (conduz habitualmente à sua terminação ou suspensão de execução), ou uma acção específica que é introduzida (registada) pelo próprio processo em runtime.
Uma forma é de enviar cada sinal de entrada na região critica e saída, seria qualquer programa que tente aceder posta em wait.

20. Mostre como poderia criar um canal de comunicação bidireccional (*full-duplex*) entre dois processos parentes usando a chamada ao sistema *pipe*.
21. Como classifica os recursos em termos do tipo de apropriação que os processos fazem deles? Neste sentido, como classificaria o canal de comunicações, uma impressora e a memória de massa?
Um recurso é algo que um processo precisa para a sua execução. Os recursos tanto podem ser componentes físicos do sistema computacional (processadores, regiões de memória principal ou de memória de massa, dispositivos concretos de entrada / saída, etc.), como estruturas de dados comuns definidas ao nível do sistema de operação (tabela de controlo de processos, canais de comunicação, etc.), ou entre processos de uma mesma aplicação.
Os recursos dividem-se 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 processador, ou as regiões de memória principal onde o espaço de endereçamento de um processo está alojado; • recursos non-preemptable - em caso contrário; são, por exemplo, a impressora, ou uma estrutura de dados partilhada que exige exclusão mútua para a sua manipulação.

22. Distinga as diferentes políticas de *prevenção de deadlock no sentido estrito*. Dê um exemplo ilustrativo de cada uma delas numa situação em que um grupo de processos usa um conjunto de blocos de disco para armazenamento temporário de informação.
As políticas de prevenção de deadlock no sentido estrito, embora seguras, são muito restritivas, pouco eficientes e difíceis de aplicar em situações muito gerais.
Resumindo, tem-se que :• negação da condição de exclusão mútua - só pode ser aplicada a recursos passíveis de partilha em simultâneo; • negação da condição de espera com retenção - exige o conhecimento prévio de todos os recursos que vão ser necessários, considera sempre o pior caso possível (uso de todos os recursos em simultâneo); • imposição da condição de não libertação - ao supor a libertação de todos os recursos anteriores quando o próximo não puder ser atribuído, atrasa a execução do processo de modo muitas vezes substancial; • negação da condição de espera circular - desaproveita recursos eventualmente disponíveis que poderiam ser usados na continuação do processo.

23. Em que consiste o algoritmo dos banqueiros de Dijkstra? Dê um exemplo da sua aplicação na situação descrita na questão anterior.
O algoritmo dos banqueiros de Dijkstra aplica-se à detecção de estados inseguros, ora a apropriação de um estado seguro necessário apenas se verifica que não há risco de deadlock. No caso anterior há que considerar o problema circular de deadlock na apropriação. Exemplo:

• cada filósofo, quando pretende os garfos, continua a pegar primeiro no garfo da esquerda e, só depois, no da direita, mas agora mesmo que o garfo da esquerda esteja sobre a mesa, o filósofo nem sempre pode pegar nele;
• só o poderá fazer se pelo menos um dos outros filósofos estiver a meditar; caso contrário, a atribuição origina uma situação insegura;
• não se trata, contudo, de uma solução geral (o adiamento indefinido não está completamente resolvido).

24. As políticas de *prevenção de deadlock no sentido lato* baseiam-se na transição do sistema entre estados ditos *seguros*. O que é um estado seguro? Qual é o princípio que está subjacente a esta definição?
Define-se neste contexto estado seguro como uma qualquer distribuição dos recursos do sistema, livres ou atribuídos aos processos que coexistem, que possibilita a terminação de todos eles. Por oposição, um estado é inseguro se não for possível fazer-se uma tal afirmação sobre ele.

25. Que tipos de custos estão envolvidos na implementação das políticas de *prevenção de deadlock nos sentidos estrito e lato*? Descreva-os com detalhe.
As políticas de prevenção de deadlock no sentido estrito, embora seguras, são muito restritivas, pouco eficientes e difíceis de aplicar em situações muito gerais, enquanto que as políticas de prevenção de deadlock no sentido lato, embora igualmente seguras, não são menos restritivas e ineficientes do que as políticas de prevenção de deadlock no sentido estrito. São, porém, mais versáteis e, por isso, mais fáceis de aplicar em situações gerais.
Mas convém notar o seguinte: • é necessário o conhecimento completo de todos os recursos do sistema e cada processo tem que indicar à cabeça a lista de todos os recursos que vai precisar - só assim se pode caracterizar um estado seguro;
• um estado inseguro não é sinónimo de deadlock - vai, contudo, considerar-se sempre o pior caso possível para garantir a sua não ocorrência.
Pelo que os custos são maiores no sentido lato porque precisa-se de hardware e maior capacidade de processamento enquanto que no sentido restrito os custos são a nível de software.

MEMORIA VIRTUALiem memoria virtual é constituido em disco uma imagem do processo. A informação esta dividida em blocos. Se esses blocos têm tamanho fixo então o **sistema é paginado**, e os blocos designam-se por paginas. Caso contrario se os blocos tem tamanho variavel designam-se por segmentos, e os **sistemas** q os usam chamam-se **segmentados**.
SISTEMA PAGINADO-> o tamango dos blocos é fixo pelo que a mem atribuida a um dado processo pode ser constituida por varios blocos de informacao. É mantido um registo, para cada processo, que indica quais os blocos que se encontram em mem real. Os endereços virtuais destes blocos contêm o número do bloco em questão e o deslocamento a partir do inicio deste bloco. Para além disso o CPU matém um regfisto base onde está o endereço do inicio da tabela de paginação. A traducaso do endereço virtual em endereço real é feita do segfuinte modo: ao endereço base é somado o número do bloco. Do endereço resultante fica-se a saber a entrad da tabela de blocos do processo correspondente aquele bloco. Esta entrada dá o endereço em memoria principal do bloco em questão. Se a estre enderço somarmos o delocamento termos o end em moria real. Para tornar mais rapida esta tradução a tabela de paginação pode ser armazeanda em mem cache ou associativa. Para haver proteccao das pagoinas de memoria de diferentes processos é necessario acrescentar masi info à tabela de paginação.Há entaõ um registo com o numero total de paginas quer o processo tem. Se o numero da ppag somado com o endereço base der um valor superior ao jumeor xda pags há memory fault.
SISTEMA SEGMENTADO-> os blocos são feitos tão grandes como necessário, conforme o tamanho do processo e a informação associada. Este mapeamento é mais natural pois cada segmento corresponde a uma entidade conceptrual, lógica e não fisica como no sistemas paginado. A proteccao da info e a sua partilha é assim facilitada. Existe um certo paralelismo entre estes dois tipos de organização de mem viirtual e os sistemas de partições fixas e variaveis em sistemas de mamoria real.
Descreva a organização de um sistema hierarquizado de memoria .Justifque a importância dos diferentes níveis num sistema multiprogramado.R: Um sist hierarquizado de mem supõe 3 niveis: memoria cache, principal, e de massa. A dif entre cada nível esta relacionada com a capacidade e de tempo de acesso, que são mínimos na cache e máximos na memoria principal. O processador apenas tem acesso directo à cache e à memoria principal. Num sistema multiprogramado diferentes programas estão, em principio, a ser concurrentemente executados, o que impõe a sua carga total ou parcial em mem principal. Para que a capacidade deste tipo de mem não constitua um impedimento ao numero máximo total de processos que coexistam, torna-se muitas vezes necessário deslocar temporariamente alguns daqueles, total ou parcialmente, para a memoria de massa. Abre-se assim a possibilidade de execucao de um numero de programas cujas necessidades em armazenamento ultrapassa a capacidade da mem principal. A mem cache sendo de acesso mais rápido que a mem principal deve assim ser utilizada para armazenar partes de código ou estruturas de dados mais frequentemente utilizadas.
Distinga uma organizacao de mem real de uma virtual. Qual a importância que organizacoes de mem virtual têm em sistemas do tipo interactivo?R: A mem real é a mem do system que pode ser directamente referenciada pelo CPU (mem cache e principal). Uma das diferencias entre mem real e mem virtual é que em mem virtual é construída em disco uma imagem do processo. Assim, em cada instante carrega-se apenas uma parte do processo. Deste modo o conceito de mem virtual perrmite que o espaço de endereçamento de um processo seja muito superior ao espaço físico da mem(mem principal). No entanto, como processo precisa de ser executado em mem física a traducao de endereços virtuais para endereços reais deve ser rápida e eficiente.
Descreva uma organizacao de mem real com particoes de tamanho variável. Indique como nela podem ser resolvidos os problemas de carga de processos e da proteccao de areas de mem pertencentes a diferentes processos. R: Numa organização deste tipo, a dimensão de cada partição é exactamente aquela necessária ao armazenamento contíguo do código e estruturas de dados próprios de cada processo. Faz-se portanto uma optimização do espaço de memória. Porem à medida que os processos existentes vão sendo terminados e outros processos vão sendo criados, a memoria vai-se fragmentando e pode ocorrer uma situação em que, embora a memoria total disponível seja superior à necessária não existe nenhuma zona de endereços contíguos com dimensão suficiente para carregar um novo programa, o que exige a implementação de um mecanismo de compactação ´Garbage Collection´. De um modo geral, no entanto, existem sempre um ou mais blocos de endereços contíguos disponíveis onde o código e as estruturas de dados associados a um novo processo podem ser carregados. A selecção de um deles faz-se usualmente de acordo com um dos 3 critérios: **First-Fit** - escolhe-se o primeiro bloco com dimensão suficiente **Best - Fit** - escolhe-se o bloco com dimensão mais pequena **Worst- Fit** - escolhe-se op bloco com maior dimensão. A protecção da área de memoria pertencente a cada processo e feita através da implementação de dois registos fronteira que contêm os endereços limite da zona de memoria principal contigua atribuída ao processo. Só os endereços gerados dentro desse intervalo são considerados válidos, todos os outros dão origem a uma interrupção de tipo ´memory fault´.
O que é que caracteriza o conceito de WORKING-SET como uma politica de gestão do espaço de memoria fisica atribuída a um processo. R: Este conceito está relacionado com o conceito de localidade que nos sugere que os processo tendem a referenciar áreas de armazenamento segundo padrões bem localizados e uniformes quer no tempo quer no espaço físico da mem. Aproveitando-se desta propriedade inerente aos processos as politicas de WS defendem que as paginas de mem referenciadas pelo processo devem ser mantidas em memorias primarias para optimizar a execução do mesmo.
Que tipo de vantagens se procura retirar da implementação de uma politica de spoling da impressora? Esta politca procura aumentar o throughput do sistema computacional, medindo a transferência de daos para ela, que é um periférico lento, através do recurso intercalar a um periférico rápido de armazanamento temporário(disco). **Qual a sobrecarga que uam tal politica impõe sobre os outros recursos do sistemas?** Impõe sobretudo a nessidade de aumento de capacidade de mem de armazanamento de massa. No entanto dado que tem de existir um novo processo (spooler) encarregue da transferência de dados entre a mem de massa e a impressora, tem tb implicações em termos da mem prinicapl.
Principio da optimidade - Para se obter performance optima deve-se substituir o bloco que não vai ser usado durante mais mais no futuro. Esta estrateggia nao pode ser usada ja que nao se pode prever o futuro.Na pratica pretende-se uma estrategia que se aproxime desdte principio.
FIFO- Permite substituir a pagina qure esat à mais tempo em meoria. É pouco eficente porque a pag. Substituida pode ser uma que estava asder constantemente usada. Exist uma anomalia nesta estrategia: aumentar o fifo atribuido a um processo não leva necessariamente à reducao de page fault(antes pelo contrario).
LRU(LAST RECENT USED)- baseia-se no principio da localidade, segundo o qual um processo passa periodos de tempo razoaveis, a executar zonas restritas de codigo. É um aboa aproximacao do principio de optimidade. No entanto obriga ao armazanemaneto de informaçao temporal relativa às execucoes das diversas pags. Pelo que apresenta um grande overhead.
Gestão de Memória de Massa Embora seja possível uma única cabeça de leitura e escrita, usa-se normalmente 2 cabeças separadas para se fazer a verificação dos dados que se escreve, logo após a escrita. O armazenamento dos dados não é feito de forma continua, para não ficarem perdidos devido à ocorrência de defeitos no material do disco. Antes pelo contrário, a informação é armazenada de forma fragmentada. Os discos encontram-se divididos em blocos (sectores) de 512 a 1024 bytes, cada sector tem o seguinte aspecto. Por questões de simplicidade, cada pista tem o mesmo nº de sectores. Para se aceder à informação indica-se o nº da pista e o nº do sector. Num flopy disc há contacto físico entre a superfície do disco e a cabeça de leitura, pois não há variações de entreferro. Devido a isso estes discos são lentos e têm um tempo de vida limitado. (desgaste por contacto). Se o disco é rígido o entreferro, disco-cabeça de leitura é constante. Assim evita-se o contacto, permitindo velocidades de rotação elevadas.
O acesso ao disco é feito indicando o nº do cilindro, nº da superfície e o nº do sector.
Tempo de Latência tempo necessário para se atingir o sector desejado na pista em que a cabeça se encontra (está limitada pela velocidade de rotação).
Tempo de Pesquisa tempo necessário para que a cabeça atinja o cilindro desejado. Como o tempo de pesquisa é bastante superior ao tempo de latência, o uso eficiente do disco rígido obriga à optimização das operações de pesquisa.
Características desejáveis numa estratégia de gestão do disco
- Troughput (nº de pedidos por unidade de tempo)
- Tempo médio de resposta
- Variância dos tempos de resposta (prevesibilidade)

Estratégia de optimização de Pesquisa
FCFS (First Come First Served) é uma estratégia justa. No entanto resulta em padrões de busca aleatória pelo que a previsibilidade do sistema é má e depende da carga do sistema.
SST (Smallest Seek Time) é sempre servido o pedido de acesso ao bloco que esteja mais próximo da cabeça. Melhora-se o tempo de acesso mas a prevesibilidade é má já que o pedido para acesso a blocos nas extremidades e no centro do disco são descriminadas.
Scan a cabeça de leitura desloca-se da extremidade para o centro do disco, e então regressa á extremidade, repetindo sempre este movimento. Os pedidos que chegam são ordenados por distância à cabeça na direcção de deslocamento (os mais próximos em primeiro lugar). Esta estratégia reduz a discriminação da zona central e das extremidades. No entanto a zona intermédia continua a ser privilegiada. Melhora a prevesibilidade. Para baixas cargas é a melhor estratégia.
N-Step_Scan os movimentos da cabeça são semelhantes aos do scan, mas durante o deslocamento num sentido só são acedidos os pedidos existentes aquando do inicio desse deslocamento. Os pedidos que chegam entretanto, são ordenados para serviço óptimo, quando o deslocamento for em sentido contrário. É uma boa solução com bom Troughput e tempo de resposta. Apresenta uma melhor prevesibilidade que o scan. No entanto continua a haver privilégios da zona intermédia do disco. Evita adiantamento indefinido de pedidos de acesso.
Circular Scan a cabeça desloca-se do exterior para o centro do disco, servindo por ordem de proximidade os pedidos que vão chegando. Uma vez chegado ao centro, retorna rapidamente ao exterior sem servir nenhum pedido, i.é só são servidos pedidos no deslocamento para o interior. É semelhante ao scan. Com este método deixa de haver zonas favorecidas do disco. Se os pedidos, que chegam durante um deslocamento para o centro só forem servidos no próximo deslocamento para o centro, temos N-Step C-Scan. Com o C-Scan obtem-se uma excelente prevesibilidade. para elevadas cargas é a melhor estratégia.
Para baixas cargas a melhor estratégia é o Scan, enquanto que para elevadas é o C-Scan.
Organização Lógica dos Ficheiros - File System do ponto de vista do utilizador o disco está dividido em directórios organizados numa estrutura hierárquica. No entanto no disco a informação é organizada em sectores. Assim, acima do gestor básico do disco, que faz o scheduling e optimização dos acessos, é necessário um gestor de disco que funcione ao nível lógico. Os tipos de objectos são: Directórios Ficheiros (endereçamento directo). Um Link é uma entrada do directório que aponta para um objecto existente noutro ponto da estrutura hierárquica de ficheiros. Esse objecto não é efectivamente copiado no directório em que existe o link. Os tipos básicos de acesso são: Leitura, Escrita e Execução. Os utilizadores são agrupados em grupos com determinadas permissões de acesso aos objectos do file system. Dentro de cada grupo pode haver partilha de informação, pode tb existir partilha de informação entre grupos.

Capitulo V - *Gestão de dispositivos de entrada-saída*

- **Porque é que num ambiente multiprogramado é fundamental desacoplar a comunicação dos processos utilizadores com os diferentes dispositivos de entrada-saída?**

Para não ficarem em busy waiting.Devido à diferença de velocidade entre os processos e os dispositivos de entrada/saida.

- **Que papel desempenham os 'device-drivers' no estabelecimento de um interface uniforme de comunicação com os dispositivos de entrada-saída?**

Os device drives (controlador) 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. Obtem-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.

Os device drives servem para independentemente do dispositivo de entrada/saida ele possa ser acedido através de funções pré-estabelecidas. Os device drives tiram o peso ao programador de como funciona o dispositivo e possa usar funções já conhecidas.

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

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.

Os de caracter transmitem informação com mensagens com o tamanho fixo de um byte e ainformação não pode ser pesquisada, os dispositivos de bloco transmitem a informação com mensagens com tamanho de um bloco com vários bytes e é possivel aceder a um bloco independentemente dos outros, ou seja, premite a pesquisa indexada.

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.

- **Durante a operação normal de um sistema computacional, o acesso à memória de massa é quase constante. Como o seu tempo de acesso é tipicamente ordens de grandeza mais lento do que o acesso à memória principal, há o risco do desempenho global do sistema de operação ser fortemente afectado pelo 'engarrafamento' que daí resulta. Apresente sugestões de como minimizar este efeito.**

Utilização da DMA.

Politicas de limpeza de páginas em memória principal

O uso de dois buffers, um para páginas modificadas e delas não modificadas.

Capitulo VI - *Gestão da memória de massa*

- **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 sistema de ficheiros? Quais são os elementos essenciais que definem a sua arquitectura?** O sistema de ficheiros permite abstrair o utilizador da organização da memória de massa, fornecendo-lhe apenas algumas funções de acesso aos ficheiros. (leitura, escrita, criação, eliminação, truncar, procura) Um sistema de ficheiros é composto por duas partes distintas: ficheiros e directórios (um directório é um ficheiro com atributos especiais).

Os ficheiros contém os dados, os programas e toda a informação que se pretende armazenar.

Os directórios servem para estruturar a organização dos ficheiros. Um ficheiro é uma forma de guardar dados de uma forma organizada.Um ficheiro é caracterizado pelos seguintes atributos:

· Nome: Nome simbólico para o utilizador reconhecer o ficheiro

· Tipo: Se o ficheiro é do tipo objecto, executável, código fonte, batch, texto, arquivos, etc.

· Localização: É um ponteiro para o dispositivo onde o ficheiro se encontra e a localização dentro desse dispositivo.

· Tamanho: O tamanho do ficheiro (em bytes, palavras ou blocos), e possivelmente o tamanho máximo permitido.

· Protecção: Informação de acesso ao ficheiro por parte dos diversos utilizadores.

· Tempo, data e dono: Data de criação, modificação e utilização.

- **Em que consiste o problema da consistência da informação num sistema de ficheiros? Que razões levam à sua ocorrência? Como é que ele pode ser minimizado?**Tem haver com o problema resultante do facto de durante operações de criação, escrita, e outras operaçãos que alterem os ficheiros, o sistema falhe por alguma razão e a operação não seja totalmente realizada, por exemplo, pode ser sinalizado que um dado bloco de dados está a ser utilizado por um ficheiro mas como a operação foi interrompida, não foi possivel lá escrever a informação. Pode ser minizado fazendo regularmente uma verificação de consistência do sistema (scandisk), ou seja, percorrer todos os nós i e respectivos blocos de dados e verificar se estão correctamente. Uma solução utilizada também pode ser a realização de um diário, ou seja, sempre que se efectua uma operação que pode levar a uma situação de inconsistência do sistema, os vários passos dessa operação são guardados no diário e se no final forem bem executados, são sinalizados como tal, deste modo sempre que o sistema se desligue por uma razão anómala, basta ir ao diário e ver se no momento estava a ser efectuada alguma operação crítica e se foi bem realizada, se não foi finalizada o sistema realiza a operação inversa, ou seja, os passos que já tinham sido realizados são agora desfeitos.

- **Descreva justificadamente os diversos papeis 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.**Armazenamento de dados (ficheiros e directórios). Área de swapping: extensão da memória principal, de forma a aumentar a taxa de utilização do processador e a libertar o programador da limitação da memória física, através da utilização da memória virtual. Para não se perderem dados importantes quando um disco falhar, usando vários discos com cópias iguais. Quando é necessário armazenar muita informação.