Sistemas de Operação Ano lectivo de 2004-2005

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-int*. **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; a 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 específica do Unix). Explique qual é a sua importância no estabelecimento de um *interface de programação de aplicações* (API).
R: chamadas ao sistema constituem a interface entre programas aplicativos e o sistema operacional, geralmente disponíveis como instruções em linguagem assembly, as linguagens C, C++ tambem podem substituir a linguagem assembly para efectuar chamadas directas ao sistema.

Podem ser agrupadas por serviços
-Gestão de ficheiros: create,delete,open, close file read, write
-Gestão de memória get/set system data
- Gestão de processos: end, abort, create, terminate, wait for time
Cominucação: send, receive messages

Api

Execução de programas – capacidade do sistema para carregar um programa em memória e executá-lo
Operações I/O– dado que os programas do utilizador não podem executar operações I/O directamente, o sistema operativo tem de fornecer meios por forma a fazê-lo
Manipulação dos sistema de ficheiros– capacidade dum programa para ler, escrever, criar e eliminar ficheiros
Comunicações – troca de informação entre processos em execução no mesmo computador ou em computadores diferentes ligados em rede
Comunicação implementada via *shared memory* ou *message passing*
Detecção de erros – garantia da computação correcta através da detecção de erros na CPU e hardware de memória, em dispositivos I/O, ou em programas do utilizador

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: um ambiente baseado em linha de comandos possibilista executar operações que o ambiente gráfico não o permite ou permite mas de forma mais lente.
O ambiente baseado em linhas de comando também permite à construção de comandos complexos, que num ambiente gráfico não seriam fáceis nem comodos de executar.

4. Distinga multiprocessamento de multiprogramação. Será possível conceber-se multiprocessamento sem multiprogramação? Em que circunstâncias?
R.: Na multiprogramação temos multiplexagem no tempo, daí que vários programas são processados por um único processador . Criando a ilusão que um sistema computacional de aparentemente poder executar em simultâneo mais programas do que o número de processadores existentes

(Parelelismo), mas o que temos é concorrência entre os programas o que exige que a atribuição do (s) processador (es) seja multiplexada no tempo entre os diferentes programas presentes. A multiprogramação tem como objectivo optimizar o processador ex: não fica a espera de ser introduzido valores, passa a executar outras operações.

No multiprocessamento temos multiplexagem no espaço, isto é as tarefas são distribuídas por vários processadores, um processador por cada programa, o sistema computacional pode executar em simultâneo dois ou mais programas (paralelismo).

É possível conceber-se multiprocessamento sem multiprogramação, no caso dos super computadores onde queremos que os processadores ficam unicamente atribuído a uma tarefa devido a ordem de complexidade das tarefas(cálculo maciço de informação) exigidas por essas maquinas.

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:
-1º Em cada utilizado podem existir diversos programas a serem corridos ao mesmo tempo;
-2º ao nível dos vários utilizados, podendo cada um ser considerado um programa a nível abstracto
Temos então multiplexagem no tempo do processador para cada utilizador, e em cada utilizador voltamos a ter multiplexagem no tempo do processador quanto este tem varias aplicações a decorrer em simultâneo

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?
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 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*).???????????????
???
R diferenças: sistema de rede vários sistemas operativos geram os recursos, sistemas distribuídos so um sistema operativo que gere os recursos da rede

8. Os sistemas de operação de uso geral actuais são tipicamente *sistemas de operação de rede*. Faça a sua caracterização.
R: Sistema operativo de rede: tem 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.
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.
Possui dispositivos de comunicação como: Modem ADSL, Cabos,Ethernet

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:

sistema operativo com característica vocacionadas para a recepção de dados e apresentação dos mesmos através de um ambiente gráfico.

Recursos pequenos(processador e memoria
Rentabilizar a interface como o utilizador e não com o hardware boa comunicação com o utilizador
Requesitos de gestão de energia
Multiprogramação espesifica na parte de comunicação
Faramentas limitadas dervido a limitação do tamanho

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 é constituído por – *arquitectura em camadas* – trata-se de uma decomposição hierárquica do kernel, em que cada novo 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 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 *Tabela de Controlo de Processos* (*PCT*), é usada intensivamente pelo *scheduler* para fazer a gestão do processador e de outros recursos do sistema computacional e Guarda a informação dos processos, de modo a que exista continuidade na execução do processo quando lhe for novamente atribuido o processador.
Os campos da *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;
deadlines – deve procurar garantir-se o máximo de cumprimento possível das metas temporais impostas pelos processos em execução;
Eficiência – minimização do tempo associado com a selecção do próximo processo a que vai ser atribuído o processador e com a comutação propriamente dita.

sistema multiutilizador: Justiça, Eficiência, Tempo de resposta , Throughput
num sistema de tipo *batch*: Throughput turn-around
sistema de tempo real:Eficiência, *deadlines*

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*?
R: O diagrama de estados do *scheduling* do processador em três níveis é:

O papel desempenhado por cada nível será:
1.Scheduling de baixo nível (Gestão do Processador): estados

• ***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.
• ***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.

?? não existe o nível médio memoria
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 recuar de um recuar "genérico", (como por exemplo: I/O, memória, ect.) logo para cada recuo 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*.

??????????????????????
Fuçoes: tratamento das interrupções, agendar a atribuição do processador e de outros recursos do sistema computacional.

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::
• *perspectiva sistémica:*
-critérios orientados para o utilizador: estão relacionados com o comportamento do sistema de operação na perspectiva dos processos ou dos utilizadores;
- *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.

?????????????????? falta colocar os critérios ex: justiça …..??????????????????????????????????

9. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma delas.
R: • *prioridades estáticas* – quando o método de definição é determinístico;
• *prioridades dinâmicas* – quando o método de definição depende da história passada de execução do processo;
prioridade estática:
1 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.

2Na 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 quando esgota a janela de execução atribuída, e incrementada de uma unidade se bloqueia antes de esgotar a janela, sendo reposta no valor inicial quando atinge o valor mínimo deste modo é corigido o critério de injustiça no método anterior. ex: sistemas multiutilizador. O Unix *SVR4* usa uma disciplina de *scheduling* deste tipo para os processos utilizador.

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.
?????????????????? falta exemplo -……..??????????????????????????????????

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 mesmo utilizado 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.

R: devido a processos ter uma certa prioridade em ser executados em relação aos outros, para bem do bom funcionamento do sistema ex: questão do sistema operativo que terá de ter privilégios excepcionais com vista ao atendimento de excepção e gestão de kernel, aplicações de tempo real como áudio e vídeo.
?? completar ????????????????????????

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:
'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);
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.

R:

Num sistema de tempo real a politica de schedulling deverá ser mista. O objectivo principal de um sistema de tempo real e conseguir atender certos elementos externos respeitando o deadline por eles imposto. Logo para estes processo e imperativo que a estratégia seja non-preemptive. No entanto se outro processo estiver em execuçao e este precisar de ser executado é necessário que a estratégia para o outro processo seja preemptive. Assim a solução ideal é associar prioridades estáticas aos diferentes processos e defenir que para os processos de maior prioridade a politica de shedulling e non-preemptive e para os processos com menor prioridade a politica de schedulling e preenptive.(???)

??
Depende foi o que o prof disse mas n me lembro de mais nada

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).
?????????????????????? pelos vistos admite execções mas n pré.. so ,uda quando bloquei ou termina ????????????????????????????????

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.
??????????????????????????verificar as decrementação?????????????????????????????????

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».

R: *Aging C*onsiste no aumento da prioridade dos processos que estão à espera de ser executado à medida que o tempo progride. Deste modo é possível evitar o adiamento indefinido, como os processos com menor prioridade vão aumentado de prioridade vão acabar por ter maior prioridade do que os processos prioritários e ser finalmente atendidos
??????????????????????????????????????falta exemplos prioridade dinamica ????????????????????????????
????????????????????????????????

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.

??????????????????n me convese falta muita coisa ????????????????????????????

Processo A actividade é a execução dum programa incluindo a gestão de todos os recursos necessários, tem um espaço de endereçamento próprio e um conjunto de canais de comunicação com os dispositivos de entrada / saída;;
Thread A actividade é uma linha d execução independentes no âmbito dum determinado processo, tem 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

Cada processo terá pelo menos um thread. Uma linha execução q vai percorrendo o código correspondente ao programa até atingir o final. O conceito d thread vem separar estas duas componentes permite assim q a 1 mesmo conjunto d recursos gerido por 1 processo possam corresponder diversas linhas execução + ou menos independentes.
Uma decomposição da solução em *threads* por oposição a processos, ao envolver menos recursos por parte do SO, possibilita q operações como a sua criação, destruição e a mudança d contexto se tornem menos pesadas e + eficientes; em multiprocessadores simétricos torna-se possível calendarizar para execução em paralelo múltiplos *threads* da mesma aplicação, aumentando a sua velocidade execução. Havendo uma partilha do espaço endereçamento e do contexto de *I/O* entre os *threads* em q uma aplicação é dividida, torna-se + simples gerir a ocupação da memória principal e o acesso aos dispositivos entrada / saída duma maneira eficaz. Cada thread está associado à execução duma função ou proc q implementa uma actividade específica. A comunicação entre múltiplos threads é feita pelo acesso à estrutura dados interna q é global.

16. Indique, justificadamente, em que situações um ambiente *multithreaded* pode ser vantajoso.
R: ***Vantagens de um ambiente multithreaded:***
maior simplicidade na decomposição da solução e maior modularidade na implementação – 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:
-user threads – os threads são implementados por uma biblioteca específica ao nível utilizador que fornece apoio à criação, gestão e scheduling dethreads sem interferência do kernel; isto significa que a sua implementação é muito eficiente, mas, como o kernel vê apenas o processo a que eles pertencem, 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;

-kernel threads - os threads são implementados directamente ao nível do kernel que providencia as operações de criação, gestão e scheduling de threads; a sua implementação é menos eficiente do que no caso anterior, mas o bloqueio de um thread particular não afecta a calendarização para execução dos restantes e torna-se possível a sua execução paralela num
multiprocessador.

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* comutarentre 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 e 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, mas numa a que depende da separação??????????????????????????????????

<u>Capitulo III</u> – *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: • <u>*processos independentes*</u> – 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*;
• <u>*processos cooperantes*</u> – quando partilham informação ou comunicam entre si de um modo explícito;
- a *partilha* exige um espaço de endereçamento comum,
- 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:
• <u>*processos independentes*</u> 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*).
• <u>Processos cooperantes</u> 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 processos necessitam do mesmo recurso ao mesmo tempo e competem pela sua posse resultando no bloqueio de um e a continuação do outro. Exemplo uma impressora quando 2 programas distintos tentem imprimir 1 documento acesso a uma região de memoria partilhada que o seu acesso impõe a necessidade d mecanismos d sincronização q garantam q o recurso é utilizado d forma consistente pelos processos envolvidos.

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.

propriedades:
• 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.

que conduzem, ou podem conduzir, a inconsistência de informação.
Por exemplo: duvias no exemplo ??????????????????????????????????
P1 lê região crítica P1
P2 lê região crítica P1
P1 escreve região crítica P2
P2 testa e escreve na região critica 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;
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 critica teremos adiamento indefinido 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 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:

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 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.

Servem para garantir atomicidade (a sua execução não pode ser interrompida) na leitura e escrita sucessivas duma variável. Os processadores tem a instrução test-and-set (tas) q permite duma maneira atómica a leitura duma posição memória, este tipo d operação usam variáveis flags d locking q após serem testados com sucesso, mudam o seu estado, qualquer processo q a seguir tentar entrar nessa região entrará em busy waiting até q o processo q fez o locking faça unlock da flag d 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.

Mecanismo sincronização sem busy waiting; tem associada uma fila processos bloqueados; apenas suporta duas operações atómicas

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.

É 1 dispositivo sincronização processos; constituído por uma estrutura dados interna, por código inicicalização e por 1 conjunto primitivas acesso agrupados numa espécie de modulo. Processos invocam os procedimentos contidos no monitor mas não acedem directamente às suas estruturas dados; Apenas 1 processo pode estar activo no monitor; Se já existe algum processo activo no monitor o processo é suspenso até q o outro processo abandone o monitor; exclusão mútua garantida pelo gerador d código(complidador) e não pelo programador, assim, o compilador ao compilar 1 monitor, gera código necessário para impor esta situação.

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.
Impedimento d atribuição d 1 novo recurso a 1 dado processo pode resultar uma situação insegura. Quando for possível encontrar pelo menos uma sucessão d atribuição d recursos q conduza à terminação d todos os processos q coexistem.
Ex: Cada filósofo, qd pretende os garfos, continua a pegar primeiro no garfo da esquerda e só depois, no da direita, mas agora mesmo q o garfo da esquerda esteja sobre a mesa, o filósofo nem sempre pode pegar nele.

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

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.

Gestão de memória

Descreva a organização dum sistema hierarquizado de memória. Justifique os diferentes niveis num sistema multiprogramado.
1 sist hierarquizado d memória tem 3 niveis: a memória cache (vai conter uma cópia das posições d memória (instruções e operandos) + frequente/ referenciadas pelo processador no passado próximo), principal e a massa (sist ficheiros e área swapping). A diferença entre cada nivel está relacionada com a capacidade e o tempo acesso, q são mínimos no caso da 'cache' e máximos no caso da memória massa. O processador só tem capacidade acesso às memórias 'cache' e d massa. Num sistema multiprogramado, controlo da transferência dados entre a memória principal e a memória massa, visando: a manutenção dum registo sobre as partes da memória principal q estão ocupadas e aquelas q estão livres; a reserva d porções da memória principal para os processos q dela vão precisar, ou a sua libertação qd já não são necessárias; a transferência para a *área swapping* d todo o, ou parte do, espaço d endereçamento d 1 processo qd a memória principal é demasiado pequena para conter todos os processos q coexistem.

Qual é a diferença entre uma organização de memória virtual segmentada de uma paginada? Quais são as vantagens e inconvenientes de cada uma delas?
Na paginação partimos a memória principal em blocos de tamanho fixo, relativamente pequenos, designados por frames, e que, o espaço de endreçagem de um processo é por sua vez dividido em blocos do mesmo tamanho, fixo, designado por pages, sendo cada eles atribuídos a uma página de memória. A segmentação consiste em dividir o espaço de endreçagem de um processo em blocos, cujo tamanho não é necessariamente fixo, designado por segmentos. Estes podem ser de tamanhos diferentes e podem inclusivamente aumentar dinamicamente de tamanho. Na paginação não existe fragmentação externa, mas existe interna. Não se pode facilmente partilhar programas entre utilizadores, não se pode acomodar facilmente espaços de endreçagem cujo tamanho pode variar, não se pode distinguir e proteger separadamente programas e dados, o espaço de endreçagem de um processo pode exceder a memoria física e a programada não necessita de consciência de que um técnica esta a ser utilizada. Na segmentação não existe fragmentação interna, mas sim externa. De resto é tudo sim.

Indique as características principais de uma organização de memória virtual. Explique porque é que ela é vantajosa relativamente a uma organização de memória real no que respeita ao número de processos que correntemente coexistem e a uma melhor ocupação do espaço em memória principal.

Numa organização de memória virtual o espaço de endreçagem de um processo pode exceder o total de memória disponível no sistema computacional. Mas, apenas as partes necessárias na execução do processo são mantidas na memória primária, enquanto que, to espaço de endreçagem 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. Este sistema tem 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. E 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.

Descreva uma organização de memória real com partições de tamanho variável.
Numa organização de memória real compartições variáveis, a dimensão de cada partição é exactamente aquela necessária ao armazanamento contíguo do código e estruturas de dados próprios de cadsa processo. Por isso faz-se uma optimização do espaço de memória. Porém à medida que os processos existentes vão sendo terminados e outros vão sendo criados, a memória vai-se fragmentando e pode ocorrer uma situação em que, embora a memória total disponível seja superior à necessária para carregar um novo programa, não exista nenhuma zona de endereços contíguos com a dimensão suficiente, o que exige a implementação de mecanismos de compactação.

Descreva uma organização de memória real com partições de tamanho fixo.
É uma organização em que o armazenamento do código de dados é todo ele feito na memória principal. O código pode ser armazenado sob a forma de código absoluto ou código relocatável. No código absoluto cada partição só pode possuir um único processo. O processo é compilado e linkado pelo utilzador e só pode ser executado numa única partição específica. Se esta tiver ocupada terá de esperar, logo isto cria a situação de partições superlotadas e partições livres (problemas de carga de processos). costuma-se por isso utilizar código relocatável onde o código de qualquer deslocamento é feito em função de posições bem definidas. Assim is processos sendo relocatáveis podem ser executados em qualquer partição deixando de haver assim perda de tempo no armazenamento.

Distinga uma organização de memória real de uma organização de memória virtual.
Uma organização de memória real pressupõe o armazenamento do código de dados (todo ele) na memória principal e é o processo mais simples de distribuição de memória pois só um processo de cada vez é que possui a memória, verificando-se desperdício de recurso de memória, ocupação de memória grande para um processo muito pequeno.
A organização de memória virtual é o inverso da anterior de tal forma que o código de dados não é limitado pelo espaço de memória principal. Mais que um processo de cada vez possui a memória principal. A alçocação de memória é feita a partir de partições fixas não querendo isto dizer espaço igual para todos os processos. A partição ocupada por cada processo é decidida pelo sistema operativo, este faz ideia da distribuição de processos que estão a correr, tratando-se de código absoluto (sabendo-se por isso onde acaba o sistema operativo, sabe-se onde se executa os processos). Para um sistema de tempo real escolheria uma organização de memória real, pois como um sistema de tempo real terá que ser muito rápido é necessário que todo o processo se encontre em memória principal para que a sua execução esteja dentro do tempo de resposta estabelecido.

Descreva uma organização de memória virtual paginada. Indique como nela se efectua a introdução do endereço virtual para um endereço fisico e se resolve o problema da protecção de àreas de memória pertencentes a diferentes processos.
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, FR uma flag residente que informa se o programa este 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 fisico.

Em que consiste o princípio da optimalidade na substituição de páginas em memória principal.
O principio de optimalidade manifesta-se no tempo, já que o processo que foi referenciado mais recentemente será o referenciado num futuro próximo, e no espaço, já que o processo que foi referenciado indica-nos que os processos que estão mais perto desse são os que têm mais probabilidades de serem referenciados num futuro mais próximo.

O que distingue uma organização de memória virtual paginada de uma segmentada?
Numa organização de memória virtual paginada, a dimensão do bloco base de armazenamento em memória principal é fixa, enquanto que, numa organização de memória virtual segmentada, ela é variável. No primeiro caso, a partição do espaço de endereçamento do processo reflecte apenas aspectos estruturais da própria memória física e resulta de um divisão automática em blocos de comprimento fixo realizada na *linkagem*. No segundo, a partição do espaço é lógica e está directamente associada à organização do programa estabelecida pelo programador: cada segmento corresponde a um módulo distinto, compilado separadamente.

Qual é o papel desempenhado pela memória de massa numa organização de memória virtual?
Numa organização de memória virtual, o espaço de endereçagem do processo (código+dados+stack) está totalmente dissociado da memória principal do sistema computacional. Durante a sua execução, só a parte deste espaço está em cada momento aí residente. O papel da memória de massa é, portanto, manter uma imagem actualizada do espaço completo do processo, visando a transferência de blocos específicos de e para a memória principal. Constitui aquilo que se designa por *swapping*.
O que é que se procura garantir neste tipo de organizações com a política de substituição de blocos?
O grande objectivo da política de substituição de blocos é conseguir implementar uma estratégia de substituição que seja tão próxima quanto possivel do principio da optimalidade. Ou seja, após a ocorrência de uma tentativa de acesso a um vloco não residente (‘block fault’), o bloco a substituir deve ser aquele que, de entre todos os actualmente residentes, será referenciado mais tarde.
Defina memória virtual. Explique o funcionamento de um sistema de memória virtual.

O termo *memória virtual* está normalmente associado ao facto de se poder endereçar espaço de memória maior que o disponibilizado pela memória principal de um sistema computacional, havendo por isso uma desassociação dos endereços referenciados por um processo que está a correr dos endereços disponíveis na memória principal. Os endereços referenciados por um processo que está a correr são os *endereços virtuais*. Os endereços disponíveis na memória principal são os *endereços reais*. O intervalo de endereços virtuais que um processo em execução pode referenciar é o *espaço de endereçamento virtual* do processo enquanto que o intervalo de endereços reais disponíveis num sistema computacional é o *espaço de endereçamento real* do computador.