Conceitos Introdutórios----------

1. Descreva as duas perspectivas de definição de um sistema de operação. Mostre claramente em que circunstâncias cada uma delas é relevante.

Um sistema de operação (SO) pode ser definido segundo duas perspectivas:

- perspectiva "Top-down" (ou do programador). Nesta perspectiva o SO tenta "mascarar" de uma maneira uniforme o hardware (HW), disponibilizando assim um interface entre as aplicações e o HW ao programador, que o liberta dos detalhes do HW. Cria-se assim um modelo funcional do sistema computacional (SC), designado por máquina virtual, que é mais simples de compreender e programar. O interface entre os programas e o HW é feita pelo SO através das chamadas ao sistema (System Calls), o que possibilita a portabilidade de aplicações entre sistemas computacionais estruturalmente diferentes.
- perspectiva "Bottom-up" (ou do construtor). Nesta perspectiva o SO tenta gerir, disponibilizar de uma maneira ordeira, justa e eficiente os varios recursos do SC, aos programas que por eles competem.

Circunstância em k cada uma delas é relevante:

-Top-Down - organização da memória de massa em sistemas de ficheiros, estabelecimento do ambiente base de interacção com o utilizador;

-Bottom-Up – Device-Drivers;

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).

Chamadas ao sistema é um mecanismo que permite aos programas que correm em user-mode pedirem a execução de certa função, que é restrita aos programas em supervisor-mode (por razões de segurança e estabilidade). Exemplos Unix: open, close, fork, kill, read, wait. A importância das chamadas ao sistema no estabelecimento de um interface de programação de aplicações (API) são:

- as chamadas ao sistema providenciam os blocos fundamentais à construção de programas;
- visto as chamadas ao sistema correrem em supervisor-mode, elas estão livres de bugs;
- as chamadas ao sistema são fornecidas pelo SO, o que possibilita a portabilidade de aplicações entre sistemas estruturalmente distintos.

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?

O ambiente gráfico é baseado na comunicação entre o utilizador e uma janela, desenhada no ecrã (monitor) cujo elemento de interacção é o rato. Os SO actuais além do ambiente gráfico tb fornecem um ambiente de interacção baseado em linha de comados que possibilita a execução de comandos mais complexos e estruturados, através de uma meta-linguagem de programação, o que é complicado de fazzer com um ambiente grafico, daí a razão dos SO actuais disponibilizarem tb a linha de comandos.

4. Distinga multiprocessamento de multiprogramação. Será possível conceber-se multiprocessamento sem multiprogramação? Em que circunstâncias?

Em multiprogramação temos a ilusão de que o sistema computacional pode executar em simultâneo mais programas que o nº de processadores. Ao contrário temos multiprocessamento em que executamos em simultâneo 2 ou mais programas, mas um por processador. É possível conceber multiprocessamento sem multiprogramação já que em malhas planares um processador pode estar inteiramente dedicado a um processo que lhe é atribuído.

5. Considere um sistema de operação multiutilizador de uso geral. A que níveis é que nele se pode falar de multiprogramação?

Visto termos vários utilizadores do sistema computacional, teremos utilizadores a requererem os mesmos recursos, logo, para que haja a ideia em cada utilizador que um recurso do sistema computacional lhe é inteiramente dedicado, temos que ter multiplexagem no tempo dos vários recursos pelos vários utilizadores que o requisitam. Logo teremos multiprogramação.

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?

A maior desvantagem dos sistemas bach é o não aproveitamento do CPU durante as fases de leitura/escrita em dispositivos de I/O (o que deu origem ao conceito de multiprogramação). Deste modo, hoje em dia só se justifica a utilização deste tipode SO na execução de programas CPU intensive, lidando pouco com os dispositivos I/O. Um exemplo disto é um programa de cálculo executado num supercomputador com um só CPU.

7. Quais são as semelhanças e as diferenças principais entre um sistema de operação de rede e um sistema distribuído?

Um sistema de operação de rede (SOR) é formado por múltiplos sistemas computacionais (SC por ex: computador) ligados entre si por um canal de comunicação. Canal esse que pode servir para a partilha entre os vários SC de recursos comuns a toda a comunidade. No SOR cada utilizador liga-se através de um terminal a um SC, que está interligado, por um canal de comunicação a vários outros SC.

Num sistema de operação distribuído (SOD) o utilizador liga-se ao sistema computacional através de um terminal, encarando o SC paralelo (sistemas computacionais de multi-processadores interligados entre si formando um ambiente integrado de interacção com o utilizador, como uma entidade única). Sendo assim, num SOD cada utilizador monopoliza os recurso, sendo o SO a gerir os recursos e a atribuir a cada processador uma parte de uma tarefa ou uma tarefa completa tentando maximizar a eficiência do recurso. Num SOR, ao contrário, há partilha de recursos pelos vários utilizadores (sistemas computacionais), sendo atribuído a cada SC o recurso durante um time-slot. Logo, nos SOR há partilha de recursos computacionais tendo em vista um menor nº de componentes de HW.

Nos SOD temos partilha de dados para processamento entre os vários processadores e/ou SC interligados.

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

Um SOR é formado por múltiplos SC ligados entre si por um canal de comunicação, SC estes que partilham recursos, tais como a partilha de dispositivos remotos (impressoras, etc); acesso à Internet e a SC remotos; partilha de ficheiros de login possibilitando o acesso à conta específica a partir de qualquer SC do grupo; troca de ficheiros, etc. Cada utilizador para se ligar ao sistema usa o terminal de um dado SC (computador) a partir do qual tem acesso a vários recursos partilhados e ao ficheiro de login, o que permite a ligação do utilizador à rede a partir de qualquer SC do grupo.

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.

Os palmtops ou PDA devido ao seu pequeno tamanho e peso e ao uso de bateria, têm pouca memória, CPU mais lentos do que os PCs e tem um display pequeno. O SO que vai gerir estes recursos tem de gerir eficientemente a memória e não pode utilizar em demazia o processador pois se utilizar as aplicações terão pouco tempo de CPU. Os displays pequenos levam o SO a ter de condensar a informação.

10. O sistema de operação Linux resulta do trabalho cooperativo de muita gente, localizada em muitas partes do mundo, que comunicam entre si usando a Internet. Mostre porque é que este facto é relevante para a arquitectura interna do sistema.

O SO Linux na sua arquitectura interna actual é implementado usando uma abordagem modular, abordagem em que o Kernel (núcleo) é concebido numa perspectiva top-down em que as diferentes funcionalidades são identificadas e isoladas umas das outras através de especificação de interfaces de comunicação, que torna possível o teste de cada módulo em separado e possibilita a introdução de novos módulos e funcionalidades com o mínimos de riscos. O SO Linux usa uma abordagem modular dinâmica, em que os módulos são carregados dinamicamente quando necessários. Logo o trabalho da implementação e concepção e teste de um módulo pode ser entregue a uma pessoa que não precisa de conhecer o resto dos módulos para o conceber. Logo esta abordagem de implementação e concepção é ideal para um sistema operativo como o Linux que é resultado do trabalho cooperativo de muita gente localizada em muitas partes do mundo, que comunicam entre si usando a internet.

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?

É necessário garantir que a execução dos processos não é afectada pelo instante, ou local no código, onde ocorre a comutação; e que 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 tipo de campos devem existir em cada entrada da tabela?

É na TCP que está a informação necessária para o sheduler fazer a gestão do processador. Os vários campos da PCT são:

- descrição do espaço de endereçamento: sua localização (em memória principal ou na área de 'swapping'), de acordo com o tipo de organização de memória estabelecido;
- situação de I/O: informação sobre os canais de comunicação e 'buffers' associados;
- situação do processador : valores de todos os registos internos do processador no momento em que se deu a comutação do processo. Salvaguarda dos registos internos;
- Identificadores: do processo, do pai do processo e do utilizador a que o processo pertence;
- Informação de estado e de 'scheduling': Estado de execução do processo e outros dados.

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?

O scheduling do processador é o agendamento das transições de estado dos vários processos tentando atribuir de uma maneira justa e eficiente o processador a cada processo. Os critérios que devem ser satisfeitos pelo algoritmo de sheduling do processador são: justiça (todo o processo tem o direito a uma fracção do tempo de processador num intervalo de tempo considerado como referência; eficiência (tentar manter o processador o maior tempo possível na execução de processos dos utilizadores); throughput (maximizar o nº de processo terminados num determinado tempo); tempo de turnaround (minimizar o tempo de espera pelo completamento de um job nos sistemas batch); tempo de resposta (nos sistemas interactivos deve minimizar-se o tempo que o sistema demora a responder a um pedido de um processo interactivo); previsibilidade (o tempo de execução de um processo deve ser relativamente constante e independente da sobrecarga pontual a que o SC está sujeito); deadlines (deve procurar garantir-se o máximo de cumprimento das metas temporais impostas pelos processos em execução). Critérios mais importantes para sistemas: multiutilizador (tempo de resposta, justiça, previsibilidade); batch (tempo de turnaround, deadlines, thoughput); tempo real (tempo de resposta, deadlines, previsibilidade).

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?

Há 3 níveis no diagrama de estados do scheduling do processador: scheduling de baixo nível (gestão do processador), médio nível (gestão da memória principal) e alto nível (gestão do ambiente de multiprogramação).

No scheduling de baixo nível um processo pode estar num de três estados run, ready-to-run e blocked.

No scheduling de médio nível temos dois estados (suspended ready e suspended blocked). Este sheduling pode ser necessário, pois visto o tamanho da memória principal ser finito o nº de processos que existem correntemente tem o crescimento limitado, logo usa-se na memória de massa uma extensão à memória principal para armazenamento do espaço de endereçamento, libertando assim espaço na memória principal potenciando o crescimento do nº de processos correntes. Assim existem 3 sinais: resume, em que o processo é swapped in, isto é, o seu espaço de endereçamento é transferido para a memória principal, podendo então ser retomada a sua execução; suspend, em que o processo é swapped out, entando em suspensão; event completion, em que o evento que o processo estava à espera que ocorresse ocorre.

No scheduling de alto nível temos os estados de criação e terminação de um processo. Assim existem 3 sinais: activate, quando o processo é criado; exit/abort, terminando o programa de maneira natural/forçada.

Num sistema do tipo batch faz sentido a utilização dos 3 níveis: baixo nível, para atribuição de time-slot entre os vários processos; médio nível, para uma gestão eficiente da memória principal e para um maior número de processos correntes; alto nível, pois os processos não são eternos e para gerir eficientemente a multiprogramação.

Num sistema batch multiprogramado temos multiprogramação, processos a concorrerem a um mesmo recurso, que podem ser I/O intensive ou CPU-intensive, podendo usar muita ou pouca memória do SC.

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?

Porque no estado ready-to-run os processos competem todos pelos mesmo recurso, o processador, passando ao estado run quando o tem em sua posse, enquanto que no estado blocked "competem" e esperam a ocorrência de um evento despoletado por um dispositivo (recurso). Mas cada processo espera por um evento diferente, logo é lógico e mais justo e eficiente a utilização de filas de espera separadas, múltiplas, uma para cada evento/recurso.

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.

As funções desempenhadas pelo Kernel de um SO são: tratamento das interrupções e agendamento da atribuição do processador e de outros recursos do SC. Uma interrupção provoca uma excepção, logo quando um processo tenta aceder a um recurso o sistema passa a operar em modo de supervisor, o que é despoletado por uma excepção. Como é o kernel que executa estas duas funções, a operação do kernel pode ser considerada como um serviço a excepções. Quando um processo requer um recurso fá-lo através de uma system call/excepção ao scheduling que agenda essa atribuição

7. O que é uma comutação de contexto? Descreva detalhadamente as operações mais importantes que são realizadas quando há uma comutação de contexto.

Uma comutação de contexto pode ser visualizada globalmente como uma rotina de serviço a excepções, com a diferença de que a instrução da ser executada após o serviço da excepção é diferente daquela cujo endereço foi salvaguardado aquando do atendimento da excepção.

As operações mais importantes por ordem dão: salvaguarda da caracterização do espaço de endereçamento do contexto I/O e do processador na entrata da TCP correspondente ao processo; actualização do estado e outra informação associada, nomeadamente a informação de scheduling; passagem ao estado run do próximo processo (que estava no estado ready-to-run), feito pelo scheduler; restauro do contexto de I/O e do processador e da caracterização do espaço de endereçamento por transferência de informação da entrada respectiva da TCP 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.

Na perspectiva sistémica temos: critérios orientados para o utilizador (estão relacionados com o comportamento do sistema de operação na perspectiva dos utilizadores ou dos processos, para minimizar o tempo de resposta); critérios orientados para o sistema(estão relacionados com o uso eficiente dos recursos do sistema de operação).

Na perspectiva comportamental temos: critérios orientados para o desempenho(são quantitativos, logo poderm ser medidos) e outro tipo de critérios (são qualitativos e difíceis de medir de uma maneira directa).

9. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma delas.

As prioridades podem ser estáticas ou dinâmicas. Nas prioridades estáticas o método de definição é determinístico. Os processos são agrupados em classes de prioridade fixa, de acordo com a sua importância relativa. Trata-se da disciplina de atribuição mais injusta, podendo ocorrer adiamento indefinido para os processos de prioridade mais baixa. Nas prioridades dinâmicas o método de definição depende da história passada de execução do processo. Em SO interactivos é usual definirem-se classes de prioridade com carácter funcional. Os processos transitam entre elas de acordo com a ocupação da última ou últimas janelas de execução.

Um exemplo de atribuição de prioridade estática é o seguinte: na criação de um processo é-lhe atribuído um dado nível de prioridade. À medida que o processo é calendarizado para execução a sua prioridade é decrementada de

uma unidade se este esgota a janela de execução, ou incrementada de uma unidade se isto não acontece, sendo a pioridade reposta no valor inicial quando atinge o mínimo.
Um exemplo de prioridade dinâmica é associar a cada classe vários níveis, desde o nível 1 (+ prioritário) ao nível 4 (-prioritário): terminais, I/O, janela pequena, janela grande.

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.
Os sistemas multiutilizador possuem um elevado grau de interactividade. Como tal a utilização de um algoritmo de scheduling baseado em prioridades estáticas não faz muito sentido, pois pode levar a tempos de resposta muito elevados. Assim utiliza-se um sistema de prioridades dinâmicas dividindo as diferentes classes dos processos conforme a ocupação das janelas de execução e atribuir atribuir a estas diferentes classes. Assim os processos que passassem do estado Blocked para o estado Ready-to-run por acção de um dispositivo standard, atribui-se a prioridade mais elevada, uma vez que esta transição está directamente associada com o utilizador. Imediatamente abaixo encontram-se os processos que acordam por acção de um dispositivo de I/O genérico, e abaixo destes tem-se em conta o nº de janelas em execução completadas pelo processo.

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.
Non-preemptive scheduling – quando, após a atribuição do processador a um dado processo, este o mantém na sua posse até bloquear ou terminar. A transição timer-run-out não existe neste caso. É característico dos sistemas operativos de tipo batch.
Preemptive scheduling – quando o processador pode ser retirado ao processo que o detém; tipicamente, por esgotamento do intervalo de tempo de execução que lhe foi atribuído, ou por necessidade de execução de um processo de prioridade mais elevada. É característico dos sistemas operativos de tipo interactivo.
À primeira vista utiliza-se uma política de scheduling preemptive para que um processo de prioridade máxima possa ser executado de imediato. No entanto na construção deste tipo de SO, já é sabido que os processos têm que ter um tempo de execução curto, para isso estes são executados rapidamente, bloqueando de seguida. Não tem assim muita importância utilizarem-se políticas non-preemptive. Talvez a melhor solução seja uma mistura das duas políticas. Em modo normal de funcionamento temos uma política de preemptive. No caso de ocorrer um processo com prioridade de execução muito elevada o sistema passa a funcionar em modo non-preemptive e passa a executar unicamente esse processo até que a sua prioridade baixe.

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?
Admite excepções, pois num sistema batch tenta-se minimizar o tempo de turnaround (tempo espera + execução). Para isto acontecer deve-se executar em 1º os processos, jobs que executam mais rapidamente. Logo ao adicionar-se um job ao grupo de jobs, se este job adicionado tiver um menor tempo de execução, pode-se calendarizar para execução este processo retirando o processador a um outro processo, usando-se assim uma política de scheduling preemptive.

13. Justifique claramente se a disciplina de scheduling usada em Linux para a classe SCHED_OTHER é uma política de prioridade estática ou dinâmica?
É uma disciplina de prioridade dinâmica, pois a prioridade dos processos é atribuída em créditos no instante de recreditação e esses créditos dependem dos créditos não utilizados no intervalo anterior, logo a prioridade depende da história passada do processo, logo temos uma disciplina de prioridade dinâmica.

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.
O aging dos processos tem a ver com o tempo de espera no estado Ready-to-run, sendo que quanto mais tempo espera mais aumenta a prioridade. No scheduling de prioridade estática uma vez que o processo espera numa fila, quanto mais espera mais próximo estará de ser calendarizado para execução, se o scheduler não alterar as posições na fila dos processos. No scheduling de prioridade dinâmica em que a prioridade num instante depende dos instantes anteriores e do tempo de espera no instante Ready-to-run, multiplicado por um factor que indica o quanto esse tempo de espera contribui no aumento da prioridade.

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.
Um processo é constituído pelo seu espaço de endereçamento, canais de comunicação com dispositivos I/O e fio de execução que engloba o PC, os registos internos do processador e uma stack. Se os recursos e fio de execução forem tratados em separado pelo SO temos os processos dedicados a agrupar recursos e os threads constituem as entidades executáveis independentes dentro de um mesmo processo. No caso de optarmos pelo desenvolvimento baseado em threads, vamos ter diversos fios de execução com um espaço de endereçamento e canais de comunicação comuns. O que implica uma maior rapidez de execução uma vez que a concorrência de threads não implica mudança de contexto e tendo um SC multiprocessador podemos executar cada thread num processador paralelizando assim a execução. O acesso à memória é também mais rápido visto termos um único espaço de endereçamento, logo contíguo. Há menos número de recursos, logo é mais fácil de gerir. Além disto temos uma maior modularidade e simplicidade na concepção da solução. Pois programas que envolvem multiplas actividades o atendimento a multiplas solicitações é mais fácil de conceber e implementar numa maneira concorrencial do que numa maneira só sequencial.

16. Indique justificadamente em que situações um ambiente multithreaded pode ser vantajoso.
A utilização de um ambiente multithreaded pode ser vantajoso pois existem vários fios de execução que envolvem o mesmo espaço de endereçamento e os mesmos canais de I/O . Desta forma é mais simples decompor a solução e modularizar a implementação, aumentando a eficiência no atendimento a várias solicitações; existe uma melhor gestão dos recursos do sistema computacional devido à partilha do espaço de end. e do contexto de I/O. Tudo isto resulta numa maior eficiência e rapidez de execução, visto que é mais fácil para o sistema gerir threads que processos.

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?
O SO pode fornecer "user threading" sendo os threads implementados por uma biblioteca específica ao nível do utilizador que permite a criação, gestão e scheduling dos threads sem interferência do kernel, o que implica uma elevada eficiência, mas como o kernel não vê os threads (só vê o processo a que os threads pertencem), quando um thread executa uma chamada ao sistema bloqueante, todos os threads e todo o processo é bloqueado.
O SO pode tb fornecer "kernel threading", sendo os threads implementados directamente ao nível do kernel, que providencia as operações de criação, gestão e scheduling de threads. Esta implementação é menos eficiente, mas ao contrário da anterior a execução de uma chamada ao sistema bloqueante por um dado thread do processo não bloqueia todo o processo, tornando esta implementação passível a execução paralela num multiprocessador.

18. Explique como é que os threads são implementados em Linux.
Para a criação de um novo processo usa-se a system call "fork", que copia o espaço de endereçamento, contexto de I/O e de processador. Para a criação de um novo processo só por cópia do contexto do processador existe a system call "clone", partilhando o espaço de endereçamento e o contexto de I/O . Este último processo é iniciado por invocação de uma função que é passada como parâmetro. Sendo assim não há distinção efectiva entre processos e threads, os quais o linux designa indiferentemente por tasks e eles são tratados da mesma maneira pelo kernel.

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?
Se um dado thread de um processo chama por exemplo a função scanf, a invocação desta chamada ao sistema resulta no bloqueio de todos os threads do processo. No melhor caso temos zero threads prontos a executar aquando do bloqueio, no pior caso temos todos os threads prontos a executar aquando do bloqueio. Logo se existisse uma função de gestão que agendásse a execução do scanf para outra altura em que menos threads tivessem prontos a executar, então bloquearíam menos threads que estavam prontos a executar e logo teríamos uma melhor eficiência.

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.
Para user threading deve-se usar um par de stacks (sistema/utilizador) por thread pois cada stack contém o histórico de invocações de funçõs de cada thread que ainda não retornou. Se tivéssemos só um par, se por exemplo uma thread executásse uma função e fosse calendarizada outra thread para execução nesse momento e essa thread executásse tb uma função, então a primeira thread teria que esperar que a função da segunda thread retornasse.
Para kernel threading passa-se o mesmo, mas aqui ainda temos a possibilidade de multiprocessamento de threads, logo temos de ter um par de stacks por thread.

Comunicação entre Processos----------

1.Como caracteriza os processos em termos de interacção? Mostre como em cada categoria se coloca o problema da exclusão mútua.
Em termos de interacção os processos podem ser classificados como independentes ou cooperantes.
Os processos independentes não interactuam de um modo explícito. A interacção que ocorre é implícita e tem origem na competição pelos recursos do SC. Nos processos cooperantes existe uma interacção explícita. Os processos podem partilhar informação entre si, tendo um espaço de endereçamento comum ou usando canais de comunicação.
Relativamente à exclusão mútua, no caso de processos independentes é da responsabilidade do SO garantir que a manipulação dos recursos pela parte de cada processo seja feita de uma forma controlada e não haja perda ou violação de informação. Em geral apenas um processo pode ter acesso ao recurso. Em processo cooperantes é da responsabilidade dos próprios processos que o acesso à região partilhada seja feito de uma forma controlada para que não haja perda de informação. Do mesmo modo isto implica que geralmente apenas um processo pode ter acesso à região partilhada de cada vez.

2.O que é a competição por um recurso? Dê exemplos concretos de competição em, pelos menos, duas situações distintas.
A competição por um recurso ocorre sempre que dois ou mais processos querem aceder a um recurso no mesmo instante de tempo, ou quando o recurso já está na posse de um processo. Ex: 2 processos a tentar aceder a uma região de memória partilhada, 2 processos a aceder ao disco, 2 processos a aceder a um canal do comunicação.

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?
Quando se fala em acesso por parte de um processo a um recurso ou a uma região de memória partilhada, está-se na realidade a referir-se à execução por parte do processador do código de acesso correspondente. Isto constitui o conceito de região crítica de código. A um canal de comunicação ou a uma zona de memória partilhada pertencente a um conjunto de processos cooperantes é essencial garantir que o acesso a estes seja feito por um processo de cada vez. Estes recursos constituem uma região crítica de dados.
O código de acesso à região crítica de dados deve então ter como propriedades:
- garantir a exclusão mútua no acesso à região crítica de dados, evitando assim condições de corrida que podem levar à perda de informação;
- garantir a independência da velocidade de execução relativa dos processos intervenientes ou do seu número;
- um processo fora da região crítica não pode impedir outro de a aceder;
- não pode ser adiado indefinidamente o acesso à região crítica;
- 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.
Quando dois ou mais processos que partilham informação entre si executam código sem regras bem definidas quanto à execução do código da região crítica, nomeadamente exclusão mútua, temos uma condição de corrida, isto é, quem acede primeiro ao recurso é uma incógnita, pois o scheduler é quem o decide. Até pode ocorrer que um estivesse a aceder e tivesse de parar por imposição do scheduler para que o outro processo executásse o código de acesso. Ex: quando dois processos têm uma variavel (um inteiro por ex) comum armazenada na região partilhada de dados, se um processo cada processo pretende incrementar de uma unidade essa variavel, se o primeiro processo ler a variavel, incrementa-a e guarda-a com o novo valor, se antes de guardar o novo valor o 2º processo for ler a variavel para tb a incrementar, vai incrementar o valor antigo e não o valor alterado pelo 1º processo, logo a informação torna-se inconsistente. Houve então condição de corrida.

5.Distinga deadlock de adiamento indefinido. Tomando como exemplo o problema dos produtores /consumidores, descreva uma situação de cada tipo.
O deadlock corresponde à situação em que quando dois ou mais processos ficam a aguardar eternamente o acesso às regiões críticas respectivas, esperando acontecimentos que nunca irão acontecer; o resultado é, por isso, o bloqueio das operações.
Enquanto que o adiamento indefinido corresponde à situação em que quando um ou mais processos competem pelo acesso a uma região crítica e são sempre ultrapassados por outros processos que querem aceder à mesma região crítica, sendo assim adiado sucessivamente o acesso dos primeiros, resultando num impedimento à continuação destes.
Neste problema temos N produtores que adicionam elementos a um FIFO e M consumidores que retiram elementos. Consideremos relativamente ao problema produtor/consumidor que n_pos_vazias é zero e que o scheduler atribui sempre o acesso à região crítica a um determinado produtore. O produtor vai permanecer sempre no ciclo "do while". Como não é dada a oportunidade a um consumidor de retirar um elemento da fila obtém-se o bloqueio das operações e por conseguinte um deadlock. Uma vez que o acesso dos vários produtores à fila não tem prioridade podemos pensar no caso limite de apenas um produtor adicionar elementos à fila, inibindo indefinidamente todos os outros. Os outros podutores ficam assim em adiamento indefinido. Para os consumidores é análogo.

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?
As soluções ditas software tentam resolver o problema da exclusão mútua a uma região crítica baseando-se no uso do conjunto de instruções básico do processador. Ou seja, as instruções de transferência de dados de e para a memória são do tipo standard (eitura 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. Os multiprocessadores partilham a memória.
As soluções ditas hardware tentam resolver o problema baseando-se no uso de instruções especiais do processador para garantir a algum nível a leitura e subsequente escrita de uma mesma posição de memória. São muitas vezes suportadas pelo próprio SO e podem mesmo estar integradas na linguagem de programação utilizada.

7.Dijkstra propôs um algoritmo para estender a solução de Decker a N processos. Em que consiste este algoritmo? Será que ele cumpre todas as propriedades desejáveis? Porquê?
Este algoritmo consiste em atribuir mais dois estados ao processo que quer entrar na RC, interessado e decidido, antes da entrada. Atribui-se prioridade ao processo interessado em aceder á RC se o processo actualmente com prioridade não quer aceder à RC. Podemos ter adiamento indefinido, pois antes de se testar se o processo com prioridade quer aceder à região crítica for agendado outro processo para execução e este testar e o processo com prioridade não quiser aceder, este processo passa à frente do outro, podendo isto acontecer sucessivamente, com este e outros processos que passam à frente do outro, levando o processo a uma espera sucessiva no acesso à RC.

8.O que é que distingue a solução de Decker 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).

A diferença entre os dois algoritmos está na maneira como lidam com a contenção dos processos. Se dois processos querem entrar na RC, um tem de recuar e dar lugar ao outro.
No algoritmo de Decker resolve-se este problema por meio de prioridade de acesso, sendo o processo sem prioridade que recua. No algoritmo de Peterson, resolve-se o problema por meio de uma fila de espera, em que o 1º a chegar é o 1º a entrar na RC. ficando o outro à espera.
O algoritmo de Decker não é possível de generalizar a N processos, pois tendo N processos o que continuaria seria o com mais prioridade, mas dos que recuaram qual seria o próximo a ter prioridade sobre os outros? Não se sabe. Podemos ter assim um processo que não quer aceder à RC a não deixar os outros aceder, pois a prioridade pode ser de um processo que não queira aceder à RC.
O algoritmo de Peterson é passível de generalizar a N processos. Pois só processos que querem aceder à RC se colocam na fila, sendo o 1º a aceder à RC o 1º que chegar. Temos assim uma atribuição de prioridades só aos processos que querem aceder, não sendo a prioridade dada às cegas, como no algoritmo de Decker.

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?
É atendido 1º o que está na última posição. Os processos vão-se colocando na 1ª posição da fila e vão caminhando do início até à última posição da fila.

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.
Busy-Waiting é um problema comum que surge quando se aplicam soluções de software ou se utilizam flags de locking, implementadas a partir de instruções do tipo read-modify-write. Os processos intervenientes podem aguardar a entrada na região crítica no estado activo. Esta espera designa-se por busy-waiting.
Em sistemas com só um processador este facto é indesejável pois conduz a uma perda de eficiência pois a atribuição de um processador a um processo que pretende entrar na região crítica faz com que o intervalo de tempo do processador se esgote sem que qualquer trabalho útil se tenha realizado no caso de não conseguir o acesso. Podem tb ocorrer situações de deadlock se se pratica uma política de scheduling preemptive, pois um processo de prioridade mais alta pode interromper a execução do processo em espera, podendo a flag de ocupado já estar activa qd o processo é interrompido e sendo assim ninguém entra.
O problema de busy-waiting em sistemas computacionais multiprocessador não é tão grave, pois a eficiência global perdida é baixa. Também não é grave se a execução de código das regiões críticas tem uma curta duração pois a eficiência perdida se se bloqueasse o processo, pois teríamos de fazer uma mudança de contexto, para calendarizar outro processo para execução nesse processador.

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.
Flag-de-locking é uma variável que indica a ocupação, posse, ou não, de um recurso partilhado, por outro processo sendo a leitura e sucessiva escrita da variável uma operação atómica (usando a instrução tas do processador). Fica assim impedido o acesso de outro processo a esta região, ficando este em espera mas no seu estado activo (busy waiting) até que o processo que efectuou o bloqueio da região o desfaça.

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 que pode ser definido da seguinte maneira:

```
typedef struct
{ unsigned int val;
  NODE *Queue;
}SEMAPHORE;
```

Sobre o qual é possível executar duas operações:
- down, se o campo val=0 o processo que executou a operação é bloqueado e o seu pid é colocado na fila de espera Queue, caso contrário val--.
- up, se o campo val=0, val++. Caso contrário um dos processos da fila de espera é acordado de acordo com regras bem definidas.

```
void down(unsigned int sem_id)
{ int_disable();
  if (sem[sem_id].val= =0)
       sleep(getpid(),sem_id);
  else
       sem[sem_id].val- - ;
  int_enable();
}
```

```
void up(unsigned int sem_id)
{int_disable();
  if(sem[sem_id].val= =0)
       sem[sem_id].val++;
  else
       wakeup(sem_id);
  int_enable();
}
```

O campo val indica se há alguém na região crítica (val=0). Se sim, o processo que executou o down é bloqueado. O up sinaliza a saída da RC.

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.
Um monitor é um dispositivo de sincronização que pode ser concebido como um módulo especial, suportado pela linguagem de programação e constituído por uma estrutura de dados interna, código de inicialização e primitivas de acesso. Um monitor garante que a execução de uma primitiva nele contida é feita em exclusão mútua, sendo isto responsabilidade do compilador. Um thread entra num monitor invocando uma das suas primitivas. Caso outro thread invoque essa primitiva, é bloqueado à entrada esperando a sua vez (operações wait e signal).

14. Que tipos de monitores existem? Distinga-os.
Existem 3 tipos de monitores:
Hoore – o thread que invoca a operação de signal é colocado fora do monitor para que o thread acordado possa prosseguir;
Brinch Hansen - o thread que invoca a operação de signal liberta imediatamente o monitor;
Lampson / Redell – o thread que invoca a operação de signal continua a sua execução, o thread acordado mantém-se fora do monitor e compete pelo acesso a ele. Pode tb causar adiamento indefinido.

15. Que vantagens e inconvenientes as soluções baseadas em monitores trazem sobre soluções baseadas em semáforos?
Vantagens dos monitores: permitem separar o acesso à região crítica e a sincronização entre processos, perspectiva bottom-up(bloqueiam antes de entrar); exclusão mútua não é assegurada pelo programador, mas sim pelo compilador; mais fácil.
Desvantagens dos monitores: não permite máxima eficiência; pode consumir mais recursos (memória da stack) que os semáforos; pode levar a situações de adiamento indefinido sem o programador ter culpa; a linguagem de programação tem de suportar os monitores.

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?
Baseia-se no uso de um canal de comunicações entre dois ou mais processos, sobre o qual eles podem comunicar e transferir informação através das primitivas send e receive.
O problema de acesso com exclusão mútua a uma RC coloca-se quando um processo quer executar o código de acesso ao canal de comunicações e existem vários processos a tentar aceder ao mesmo recurso. O que pode acontecer por exemplo no receive de uma mensagem e no send de uma mensagem em que os processos tentam aceder ao canal para ler ou escrever uma mensagem respectivamente. Este problema está implicitamente resolvido pelo SO, pois é ele quem resolve o problema de acesso com exclusão mútua a um recurso do sistema computacional, que neste caso é o canal de comunicações.

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.

```
//inicialização da info partilhada e envio
  msg_receive(msg_src_id,&message);     //entrada na RC
  manipulação_da_info_partilhada( );
  msg_send_nb(msg_dest_id, message);   //saída da RC
```

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).
A troca de mensagens entre processos é possível através de duas operações, send e receive, que quanto ao grau de sincronização podem ser classificadas em:
- bloqueantes: em que na operação de envio de uma msgenvia-se a msg e espera-se que ela seja recebida. Na operação de recepção bloqueia-se se não há msg na fila, até ser recebida a msg.
void msg_send(unsigned int msg_dest_id, MESSAGE msg)
void msg_receive(unsigned int msg_src_id, MESSAGE *msg, boolean *msg_arrival)
- não bloqueantes: na operação de envio envia-se a msg e regressa-se sem qq informação se a msg foi recebida ou não. Na operação de recepção caso haja msg lê-se a msg, caso não haja continua-se a execução sem msg lida.
void msg_send_nb(unsigned int msg_dest_id, MESSAGE msg)
void msg_receive_nb(unsigned int msg_src_id, 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 constituem uma interrupção produzida no contexto de um processo onde lhe é comunicada a ocorrência de um acontecimento especial. Pode ser produzido pelo kernel (devido a eventos relacionados com o software ou hardware), pelo próprio processo ou pelo utilizador. O processo pode optar por ignorá-lo, bloqueá-lo ou executar uma acção associada. Por exemplo quando 2 processos pretendem aceder à região crítica, o que acede lança um sinal que bloqueia o outro, e quando sai lança um sinal que o desbloqueia.

20. O que é um PIPE?
Um Pipe é um canal de comunicação half-duplex elementar que é estabelecido entre 2 ou mais processos. Só pode ser utilizado entre processos que estão relacionados entre si, ou seja, a comunicação só é feita entre pai e filho ou entre irmãos. A sua principal utilização é na composição de comandos mais complexos a partir de uma organização em cadeia de comandos mais simples em que o stdout de um comando é redireccionado para a entrada de um pipe e o stdin do comando seguinte é redireccionado para a saída do mesmo pipe.
Na comunicação full-duplex é comum usarem-se 2 pipes. Um pipe redirecciona o stdout do P1 para o stdin do P2. O outro redirecciona o stdout do P2 para o stdin do P1. Podia-se utilizar só um pipe, mas seria mais ineficiente e complexo.

21. Como classifica os recursos em termos do tipo de apropriação que os processos fazem deles? Neste sentido, como classificaria o canal de comunicações, uma impressora e a memória de massa?
Os recursos podem ser classificados, no que diz respeito ao tipo de apropriação que os processos fazem deles, de duas maneiras:
Recursos Preemptable – em que a retirada do acesso a um processo em qq altura não leva a consequências indesejáveis (memória de massa se quisermos escrever em endereços distintos).
Recursos non-Preemptable – são recursos que não podem ser retirados aos processo que os detêm (canal de comunicações, impressora, memória de massa se 2 processos quiserem escrever no mm endereço).

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.
A ocorrência de deadlock implica a ocorrência de um dos seguintes acontecimentos:
- condição de exclusão mútua (um único processo detém o recurso);
- condição de espera com retenção;
- condição de não libertação (só o próprio processo pode libertar o recurso);
- condição de espera circular.
Para não haver deadlock no sentido estrito temos que garantir uma das seguintes condições:
- não há exclusão mútua no acesso a um recurso (ex: os processos acedem qd querem não havendo assim deadlock, mas condições de corrida, o que tb não é muito bom);
- não há espera com retenção, um processo não deve manter em sua posse todos os recursos anteriormente solicitados ao requerer um novo recurso, do qual pode ter que esperar.Deve-se pedir todos os recursos necessários de uma só vez (ex:um processo está a escrever um bloco n outro processo vai ler o bloco n-1 e n. Ao requisitar o bloco n-1 é-lhe dado permissão mas ao requisitar o n não, pois o outro processo está a escrever nele, logo este processo deve libertar o bloco n-1 para o caso de outro processo precisar de o requisitar, tenta-se ler mais tarde os blocos n e n-1);
- há libertação de recursos, o processo deve libertar todos os recursos na sua posse qd não obtém todos os recursos que necessita (ex: se um processo quer ler mais do que um bloco, requisita logo todos os blocos que quer ler
- não há espera circular, não se deixa formar uma cadeia circular de processos e recursos em que um processo requer um recurso detido pelo processo que se apresenta a seguir na cadeia (ex: divide-se igualmente o nº de blocos disponíveis, todos os blocos acedem sempre primeiro ao bloco de menor número excepto o que detem o 1º e o último bloco, este terá de aceder primeiro ao bloco de maior nº para não haver espera circular);

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 é uma política de prevenção de deadlock no sentido lato, em que a atribuiçao de um novo recurso a um dado processo só é feita se daí não resultar uma situação insegura, ou seja, só é atribuído o recurso quando existe pelo menos uma sucessão de atribuições que conduza à terminação de todos os processos que coexistem.
Exemplo: um processo deseja escrever em 1 bloco dos n disponíveis, mas esse recurso (1 bloco) só lhe é atribuído se o nº de blocos já lhe atribuídos for menor que o nº total de blocos já atribuídos aos outros processos.

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?
Estado seguro é um estado em que a distribuição de recursos livres ou atribuídos do sistema leva sempre a que haja uma sucessão de estados seguintes que permitem a terminação de todos os processos que coexistem. O princípio subjacente é o conhecimento de todos os recursos do sistema e cada processo tem de indicar à cabeça a lista de todos os recursos que vai precisar. Só assim se pode caracterizar um estado seguro.

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 são pouco eficientes e difíceis de aplicar em situações muito gerais e tb são restritivas. As políticas de prevenção de deadlock no sentido lato não são menos restritivas e ineficientes, mas são mais versáteis. Ambas as prevencções pode provocar adiamento indefinido.
Resumindo, tem-se que
• impedimento de lançamento de um novo processo – a avaliação de estado seguro é feita a priori, o que tem como consequência uma política extremamente restritiva que leva a condição de pior caso possível ao limite;
• impedimento de atribuição de um novo recurso a um dado processo – a avaliação de estado seguro é adiada para o momento de atribuição de um novo recurso, o que possibilita uma gestão mais eficiente dos recursos disponíveis e, portanto, um aumento de throughput do sistema.
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 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.

Gestão da memória----------

1.Descreva os diferentes níveis em que se estrutura a memória de um sistema computacional, caracterizando-os em termos de capacidade, tempo de acesso e custo. Explique, face a isso, que funções são atribuídas a cada nível.

Memória cache vai de KBs a poucos MBs, é rápida cara e volátil. Tem como função conter a cópia das posições de memória (instruções e operandos) mais frequentemente utilizados pelo processador num passado próximo.
Memória principal vai de centenas de MBs a alguns GBs, tem velocidade e preços médios e é volátil. Tem como função o armazenamento do espaço de endereçamento dos vários processos. Para garantir uma elevada taxa de utilização do processador este nº de espaços de endereçamento deve ser elevado.
Memória de massa tem capacidade de centenas de GBs, é lenta, barata e não volátil. Tem como função fornecer um sistema de ficheiros onde são salvaguardados mais ou menos permantemente os dados e código. Tb pode ser usada como área de swapping, extensão da memória principal para que haja um maior nº de processos a coexistir num determinado instante.

2.Qual é o princípio que está subjacente à organização hieràrquica de memória? Dê razões que mostrem porque é que a aplicação de tal princípio faz sentido.

O princípio subjacente à organização hierárquica de memória é que quanto mais afastado se está do processador menos vezes uma instrução ou um operando é referenciado. As razões que mostram que a aplicação deste princípio faz sentido é que as instruções ou operandos que são referenciados em sucessão têm uma grande probabilidade de estarem em endereços de memória contíguos.

3.Assumindo que o papel desempenhado pela gestão de memória num ambiente de multiprogramação se centra, sobretudo, no controlo da transferência de dados entre a memória principal e a memória de massa, indique quais são as actividades principais que têm de ser considerados.

- transferência para a área de swapping do total ou parte do espaço de endereçamento de um processo quando o espaço em memória principal não é suficiente para conter todos os processos que coexistem.
- salvaguarda de um registo que indique as posições livres e ocupadas na memória principal.
- alocação de porções de memória principal para os processos que dela vão precisar, ou libertação qd não necessária.

4Porque é que a imagem binária do espaço de endereçamento de um processo é necessariamente relocatável num ambiente de multiprogramação.

Uma vez que a situação de termos em memória principal processos bloqueados e na área de swapping processos prontos a executar, leva a uma menor eficiência na utilização do processador e a um desperdício do espaço de memória principal. Faz sentido, se a memória principal não tiver espaço livre, transferir os processos bloqueados em memória principal para a área de swapping e transferir os processos prontos a executar na área de swapping para a memória principal. Por outro lado um processo pode precisar de muita memória para armazenar as suas variáveis e esgotar o seu espaço de endereçamento.Neste caso é necessário alocar mais memória e associá-la ao processo que a requisita.

5.Distinga linkagem estática de linkagem dinâmica. Qual é a mais exigente? Justifique a sua resposta.

Na linkagem estática todos os ficheiros objecto e bibliotecas do sistema são juntas num ficheiro único. Temos assim que todas as referências existentes no programa existem no espaço de endereçamento desde o início.
Na linkagem dinâmica as referências a funções de bibliotecas do sistema são substituídas por stubs, que determina a localização da função de sistema em memória principal ou a carrega em memória principal. Sendo em seguida a referência ao stub substituída pela referência da função, ou seja, em linkagem dinâmica não temos todas as referências satisfeitas ao início, elas vão-se satisfazendo ao longo do decorrer do programa, quando necessário.
A mais exigente é a linkagem estática, pois temos um maior espaço de endereçamento é quando há mudança de contexto e o processo passa ao estado blocked ou suspended-blocked em que vai haver transferência do espaço de endereçamento, como o espaço é grande e a memória de massa é lenta, demora-se um pouco.

6.Assuma que um conjunto de processos cooperam entre si partilhando dados residentes numa região de memória, comum aos diferentes espaços de endereçamento. Responda justificadamente às questões seguintes:
- em que região do espaço de endereçamento dos processos vai ser definida a área partilhada?
-será que o endereço lógico do início da área partilhada é necessariamente o mesmo em todos os processos?
- que tipo de estruturas de dados em linguagem C tem que ser usada para possibilitar o acesso às diferentes áreas da região partilhada?

- Uma vez que se trata de memória partilhada, temos que assumir que os dados partilhados podem tomar diferente tamanho ao longo da execução do programa. Como tal esses dados encontram-se na zona de definição dinâmica, região global;
- Não, pois o espaço ocupado pelo código e pela zona de definição estática pode ser diferente, o que resulta num endereço lógico inicial da zona de definição dinâmica diferente para os diversos processos. O tamanho das duas primeiras zonas (código e defenição estática) não é fixo, é definido pelo loader.
- Ponteiros. Na execução dinâmica o conteúdo do ponteiro é o endereço físico da variável para o qual aponta. Este endereço, e não o endereço lógico, é válido para todos os processos. Deste modo, o acesso às variaveis partilhadas deve ser feito usando ponteiros.

7. Distinga relativamente a um processo espaço de endereçamento lógico de espaço de endereçamento físico. Que problemas têm que ser resolvidos para garantir que a gestão de memória num ambiente de multiprogramação é eficiente e segura?

Endereçamento físico refere-se à localização na memória principal do espaço de endereçamento do processo.
O endereçamento lógico refere-se à imagem binária do espaço de endereçamento do processo e é um endereçamento relativo relativo a um dado endereço, offset. Problemas que têm de ser resolvidos:
- conversão em runtime de endereçamento lógico para endereçamento físico;
- garantir que o endereçamento físico obtido na conversão anterior está dentro da gama de endereços do espaço de endereçamento do processo.

8. Caracterize a organização de memória designada de memória real. Quais são as consequências decorrentes deste tipo de organização?

Neste tipo de organização existe uma correspondência biunívoca entre o espaço de endereçamento lógico e o espaço de endereçamento físico de um processo. Isto tem como consequências:
- limitação do espaço de endereçamento de um processo (não é possível em nenhum caso que o espaço de endereçamento de um processo seja maior que o tamanho da memória principal;
- contiguidade do espaço de endereçamento físico (é mais simples e eficiente supor que o espaço de end. é contíguo);
- área de swapping (o seu papel é assumir de arrecadação do espaço de endereçamento de todos os processos que não possam ser carregados em memória principal por falta de espaço.

9. Descreva detalhadamente o mecanismo de tradução de um endereço lógico num endereço físico numa organização de memória real.

Algoritmo de cálculo:

```
if(end_logico>limite)
      gera_excepcao();    //erro
else
      end_fisico=end_logico+end_base;
```

- ocorre comutação de contexto
- registo base e limite são preenchidos de acordo com a entrada correspondente ao novo processo
- endereço base: endereço correspondente ao início do espaço de endereçamento físico do processo
- endereço limite: tamanho do espaço de endereçamento do processo

10. O que é que distingue a arquitectura de partições fixas da arquitectura de partições variáveis numa organização de memória real? Indique quais são as vantagens e desvantagens de cada uma delas.

Na arquitectura de partições fixas a memória principal é dividida num conjunto fixo de partições mutuamente exclusivas, mas não necessariamente iguais. Cada uma contém o espaço de endereçamento de um processo. Tem como vantagens o facto de ser simples de implementar (HW e estruturas de dados simples) e é eficiente (selecção é feita rapidamente).Tem como desvantagens a grande fragmentação interna e serve só para aplicações específicas.
Na arquitectura de partições variáveis toda a parte disponível de memória constitui à partida um bloco. Reserva-se sussessivamente regiões de tamanho suficiente para carregar o espaço de endereçamento dos processos que vão surgindo e liberta-se quando não é necessário. Para gerir a memória precisamos de 2 listas, lista das regiões ocupadas e das regiões livres. Para não existir risco de existirem regiões livres tão pequenas que não possam ser utilizadas e que tornem a pesquisa mais ineficiente, tipicamente a memória principal é dividida em blocos de tamanho fixo e a reserva é feita em entidades do tamanho desses blocos. Tem como vantagens uma menor fragmentação da memória, pois cada processo ocupa o espaço estritamente necessário. Tem como desvantagens o facto de uma pesquisa ser mais elaborada e inefeciente.
Nota: garbage collection – compactação do espaço livre agrupando as regiões num dos extremos da memória, exige a paragem de todo o processamento.

11. A organização de memória real conduz a dois tipos distintos de fragmentação da memória principal. Caracterize-os e indique a que tipo de arquitectura específica cada um está ligado.

Os 2 tipos de fragmentação da memória principal são partições fixas e partições variáveis.
Nas partições fixas a fragmentação é interna e a parte da partição interna que não contem espaço de endereçamento de processos é desperdiçada, não sendo usada para nada.
Nas partições variáveis a fragmentação é externa, as sucessivas reservas e libertações de espaço resultam em fragmentos de espaço livre tão pequenos que não podem ser utilizados. Esta fracção de memória principal desperdiçada pode em alguns casos atingir 1/3 do total de memória.

12. Entre os métodos mais comuns usados para reservar espaço em memória principal numa arquitectura de partições variáveis, destacam-se o next fit e o best fit. Compare o desempenho destes métodos em termos do grau e do tipo de fragmentação produzidos e da eficiência na reserva e libertação de espaço.

O next fit consiste em iniciar a pesquisa a partir do ponto de paragem da pesquisa anterior. É muito rápido a pesquisar e pode causar fragmentação interna pois se o espaço reservado for maior que o espaço de end do processo existe espaço desperdiçado. Mas este espaço desperdiçado faz com que na libertação do espaço reservado não existam fragmentos muito pequenos.
O best fit escolhe a região mais pequena disponível onde cabe todo o espaço de end. do processo. É mais lento a pesquisar e não há fragmentação interna, apenas externa.
Este último método resulta em mais memória desperdiçada, visto que tende a deixar fragmentos de memória livre demasiado pequenos para serem utilizados.

13. Caracterize a organização de memória designada de memória virtual. Quais são as consequências decorrentes deste tipo de organização?

Numa organização de memória virtual o espaço de endereçamento lógico e físico de um processo estão completamente dissociados. As consequências são as seguintes:
- pode ser estabelecido uma metodologia conducente à execução de processos em que o seu espaço de endereçamento pode ser maior que a memória principal;
- não contiguidade do espaço de endereçamento físico (o espaço de endereçamento de um processo dividido em blocos de tamanho fixo estão dispersos por toda a memória, procurando-se desta maneira obter uma ocupação mais eficiente do espaço disponível);
- área de swapping (é criada nesta uma imagem actualizada de todo o espaço de endereçamento dos processos que coexistem correntemente, nomeadamente da sua parte variavel).

14. 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.

Características de uma organização de memória virtual (swapping):
- não existe limitação para o espaço de endereçamento do processo (EEP) uma vez que a área de swapping funciona como extensão da memória principal, assim os processos que tenham parte do seu espaço de endereçamento na área de swapping, ou mesmo todo lá, não estão obrigatoriamente bloqueados, existindo assim um maior número de processos na memória principal;
- a parte mais inactiva de um EEP pode estar na área de swapping, deixando espaço na memória principal para outros processos.

Como o espaço de endereçamento físico de um processo não é necessariamente contíguo, não existem problemas de fragmentação.

15. O que é que distingue a arquitectura paginada da arquitectura segmentada / paginada numa organização de memória virtual? Indique quais são as vantagens e desvantagens de cada uma delas.

O que distingue uma arquitectura paginada da arquitectura segmentada/paginada é que na arquitectura seg/pag não se faz uma divisão do espaço de endereçamento lógico do processo (EEL) às cegas, pois 1º faz-se uma divisão do EEL em segmentos e depois esses segmentos são divididos em paginas.
Vantagens da arquitectura paginada:
- não conduz a fragmentação externa e a interna é desprezável (tem-se grande aproveitamento da memória principal);
- é geral, isto é, é independente do tipo dos processos que vão ser executados (nº e tamanho do seu EE);
- não exige requisitos especiais de hardware, pois a unidade de gestão de memória dos processadores actuais de uso geral já está preparado para a sua implementação.

Desvantagens da arquitectura paginada:
- acesso à memória mais longo;
- operacionalidade muito exigente, pois a sua implementação exige por parte do SO a existência de um conjunto de operações de apoio e que são complexas e que têm de ser cuidadosamente estabelecidas para que não haja muita perda de eficiência).

Vantagens da arquitectura segmentada/paginada:
- iguais às 2 primeiras vantagens da arquitectura paginada;
- gestão mais eficiente da memória no que respeita às zonas de crescimento dinâmico
- minimização do nº de paginas que têm de estar residentes em memória em cada etapa de execução do processo.

Desvantagens da arquitectura segmentada/paginada:
- iguais a todas as desvantagens da arquitectura paginada;
- exige requisitos especiais de HW.

16. Descreva detalhadamente o mecanismo de tradução de um endereço lógico num endereço físico numa organização de memória virtual paginada.

zero

17. Explique porque é que hoje em dia o sistema de operação dos computadores pessoais supõe, quase invariavelmente, uma organização de memória de tipo memória virtual.

A implementação de uma organização de memória do tipo memória virtual não impõe limites ao tamanho do espaço de endereçamento do processo (EEP) e não dá quaisquer problemas em termos de fragmentação. Embora a sua implementação em HW seja mais complexa, o tempo de alocação e libertação de segmentos de memória seja maior, no global este tipo de implementação é mais vantajoso.

18. Porque é que a área de swapping desempenha papeis diferentes nas organizações de memória real e de memória virtual?

Porque ao contrário da organização de memória real, em que temos uma correspondência biunívoca entre espaço de end. lógico e espaço de end. físico, na organização de memória virtual estes 2 estão completamente dissociados, podendo assim um processo não estar bloqueado e ter parte do seu espaço de end. na área swapping.
Enquanto na organização de memória real o espaço ou está totalmente na área swapping ou na memória principal, por causa da correspondência biunívoca. Logo na organização de memória virtual convém ter uma imagem dos EEP que coexistem correntemente, para que qd seja necessária uma parte do EEP residente na área de swapping ela seja transferida para a memória principal pelo SO.

19. Considere uma organização de memória virtual implementando uma arquitectura segmentada / paginada. Explique para que servem as tabelas de segmentação e de paginação do processo. Quantas existem de cada tipo? Descreva detalhadamente o conteúdo das entradas correspondentes.

Numa arquitectura segmentada/paginada o espaço de end. lógico do processo é dividido em segmentos, segmentos estes que são divididos em páginas às quais corresponde um frame na memória principal. Logo temos uma tabela de segmentos e uma tabela de paginas, por processo.
Conteúdo da entrada da tabela de segmentos: endereço da tabela de paginação do segmento, bits de controlo.
Conteúdo da entrada da tabela de paginas: nº do frame em memória (as frames têm tamanho fixo), nº do bloco na área de swapping (memória de massa é dividida em blocos), informação de controlo.

20. Qual é a diferença principal existente entre a divisão do espaço de endereçamento de um processo em páginas e segmentos? Porque é que uma arquitectura segmentada pura tem pouco interesse prático?
A diferença reside no facto de que a divisão em segmentos não é uma divisão às cegas, pois tem em conta a estrutura modular na implementação de um programa. Ao contrário da paginada, que é feita às cegas, só coloca no início de uma nova página zonas funcionalmente distintas do EEP.
A arquitectura segmentada pura tem pouco interesse, pois ao tratar a memória principal como um espaço contínuo, leva ao uso de técnicas de reserva de espaço usadas na arquitectura de partições variáveis, o que origina grande fragmentação externa e desperdício de espaço. Os segmentos de crescimento contínuos tb dão problemas, pois os acréscimos podem não caber na localização actual, levando à sua transferência total para outra região de memória ou se não houver espaço em memória principal é bloqueado o processo e o seu espaço de end. ou o segmento que não cabe são transferidos para a área swapping.

21. As bibliotecas de rotinas linkadas dinamicamente (DLLs) são ligadas ao espaço de endereçamento de diferentes processos em run time. Neste contexto, responda justificadamente às questões seguintes
– que tipo de código tem que ser gerado pelo compilador para que esta ligação seja possível?
– este mecanismo de ligação pode ser utilizado indiferentemente em organizações de memória real e de memória virtual?
- Tem de ser gerado código de localização na memória principal das rotinas necessárias, ou que promova a sua carga em memória principal (stubs).
- Pode, pois quer numa ou noutra organização de memória o espaço de endereçamento da rotina linkada dinamicamente terá de estar ou ser carregada na memória principal, logo podendo haver correspondência biunívoca da organização real.

22. Quer numa arquitectura paginada, quer numa arquitectura segmentada / paginada, a memória principal é vista operacionalmente como dividida em frames, onde pode ser armazenado o conteúdo de uma página de um processo. Porque é conveniente impor que o tamanho de cada frame é uma potência de dois?
Primeiro porque a memória principal tem um tamanho que é uma potência de 2, logo todos os frames têm o mesmo tamanho.
Segundo, pois sendo uma potência de 2, parte dos bits, por exemplo os mais significativos indicam-nos o nº do frame e os menos significativos a posição dentro do frame. O que simplifica a unidade de gestão de memória em termos de HW e SW.

23. Foi referido que nem todos os frames de memória principal estão disponíveis para substituição. Alguns estão locked. Incluem-se neste grupo aqueles que contêm as páginas do kernel do sistema de operação, do buffer cache do sistema de ficheiros e de um ficheiro mapeado em memória. Procure a aduzir razões que justifiquem esta decisão para cada um dos casos mencionados.
Locked significa que não estão disponíveis para substituição.
Como é lógico as páginas do kernel têm de estar locked, pois se não estivessem podiam ser substituidas (transferidas para a área swapping), logo se fossem precisas tinham de ser novamente transferidas para a memória principal e até talvez substituir uma página de outro processo, o que fazia perder algum tempo, podendo bloquear todo o sistema durante esse tempo.
Buffer cache e ficheiro mapeado em memória são ambos as técnicas usadas para uma maior rapidez de acesso, logo a sua substituição implicaria uma redução do tempo de acesso, o que vai contra o princípio com que eles foram implementados.

24. O que é o princípio da optimalidade? Qual é a sua importância no estabelecimento de algoritmos de substituição de páginas em memória principal?
O principio da optimalidade diz-nos que o frame que deve ser substituído é aquele que não vai ser mais usado, referenciado, ou se o for se-lo-á o mais tarde possível.
A sua importância é que ele é o princípio optimo de substituição, mas é não casual, pois tem de se conhecer o futuro para o usar. Logo tenta-se a partir do conhecimento passado prever o futuro para saber qual o frame que não vai ser usado ou vai ser referenciado tardiamente.

25. O que é o working set de um processo? Conceba um método realizável que permita a sua determinação.
Carrega-se a 1ª e a última página do espaço de endereçamento do processo (correspondete ao início do código e ao topo da stack). Ao longo da execução do processo serão gerados pages-fault, sendo estes carregados, mas ao longo do tempo o número de pages-fault diminui até ser zero (devido ao princípio da localidade de referência). Aí temos o chamado working set do processo.

26. Dê razões que expliquem porque é que o algoritmo do relógio, numa qualquer das suas variantes, é tão popular. Descreva uma variante do algoritmo do relógio em que são consideradas as quatro classes de frames características do algoritmo NRU.
É popular visto que as operações de fifo_in e fifo_out tornam-se num incremento de um ponteiro (módulo e nº de elementos da lista).

```
while (! frame_adequado)
{   if(Ref= =0)
        substituicao(pag_associada);
    else
        Ref=0;
        ponteiro + +     % nº elementos
}
```

27. Como distingue a estratégia demand paging da prepaging? Em que contexto é que elas são aplicadas?
Quando um processo é introduzido na fila dos processos Ready-to-run, mais tarde, em resultado de uma suspensão, é necessário decidir que páginas se vão carregar na memória principal.
Se não se carregar nenhuma página e se se esperar pelo mecanismo de page-fault para formar o working-set do processo temos demand paging.
Se se carregar a 1ª e a última página do processo da 1ª vez que passa a Ready-to-run ou as páginas residentes no momento da suspensão temos prepaging.

28. Assuma que a política de substituição de páginas numa dada organização de memória virtual é de âmbito global. Explique detalhadamente como procederia para detectar a ocorrência de thrashing e como procederia para a resolver.
Para detectar, o working set dos processos teria de que ser maior que o nº de frames unlocked disponíveis na memória principal
A solução seria ir suspendendo processos até que o problema, geração de page-fault, thrashing desapareça.