Conceitos Introdutórios

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

R: As duas perspectivas são: top-down (do programador) ou bottom-up (do construtor). A primeira remete para uma máquina virtual de interface com o hardware que origina um ambiente uniforme de programação mais simples de compreender e programar sem que o utilizador tenha um conhecimento preciso dos detalhes do hardware. Este ambiente possibilita a portabilidade de aplicações entre sistemas estruturalmente distintos (sistemas computacionais). Na segunda o sistema de operação é visto como o programa que gere o sistema computacional atribuindo e controlando os diferentes recursos aos programas que por eles competem. O objectivo é assim a rentabilização máxima do sistema computacional. Desta forma, a primeira perspectiva é mais relevante para o utilizador menos familiarizado com técnicas de programação, menos interessado na rapidez do sistema e mais interessado com a facilidade de utilização do sistema de operação. O segundo é indicado para o programador mais astuto, com interesse na rapidez do sistema e acesso a recursos apenas disponíveis em níveis mais "baixos" de programação.

R:

TOP-DOWN

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

BOTTOM-UP

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

R: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).**

R: 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) é:

- 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?

R: Gráfico -> baseia-se na construção de janelas sobre o écran do monitor vídeo que funcionam como centros de comunicação entre o utilizador e aplicações específicas; elementos gráficos variados, designados de ícones, são aí desenhados e podem ser manipulados usando o rato;
Linguagem de comandos (shell) -> baseia-se em linhas de texto introduzidas pelo teclado que formam os comandos a serem executados pelo ambiente; tipicamente, existe uma metalinguagem de programação que possibilita uma abordagem mais estruturada à construção de comandos complexos.

É extremamente necessário que um sistema operativo contemple uma opção de interacção por linha de comandos (shell) pois este tipo de interacção permite a realização de tarefas mais complexas, de uma forma mais estruturada e eficiente. A opção gráfica apresenta-se mais simples e intuitiva ao utilizador comum. Contudo, o utilizador experiente e com conhecimentos mais avançados do sistema computacional necessita de uma linha de comandos para tarefas mais complexas (necessário conhecer a metalinguagem de programação a utilizar).

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

R: Um sistema operativo tem características de multiprocessamento quando tem a capacidade de poder executar em simultâneo dois ou mais programas, o que exige que o sistema computacional seja formado por mais do que um processador (um por cada programa em execução simultânea). Esta capacidade é conhecida por paralelismo. Um sistema operativo tem características de multiprocessamento quando cria a ilusão de aparentemente poder executar em simultâneo mais programas do que o número de processadores existentes, o que exige que a atribuição do(s) processador(es) é multiplexada no tempo entre os diferentes programas existentes. Esta ilusão criada pelo sistema computacional é conhecida por concorrência. Pode existir multiprocessamento sem multiprogramação se o número de processadores for igual ao número de processos, já que neste caso, não vai existir concorrência entre processos por um processador.

- **5. Um tipo de multiprocessamento particular, o chamado *processamento simétrico* tornou-se muito popular com o surgimento no mercado dos processadores *dualcore*. Explique em que consiste e qual a razão desta popularidade.**

R: Um sistema computacional tem características de multiprocessamento se for formado por mais do que um CPU, permitindo-lhe assim executar mais do que um programa em simultâneo (um por cada CPU).
Processamento simétrico (SMP – Symmetric Multiprocessing) é uma arquitectura de multiprocessamento onde os dois ou mais CPUs se encontram ligados a uma única memória principal partilhada. Os mais recentes CPUs como Dual-Core e Quad-Core recorrem à arquitectura de processamento simétrico. São tão populares graças à sua maior velocidade e eficiência face a um sistema computacional monoCPU.

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

R: Num sistema de operação multiutilizador de uso geral a multiprogramação surge quando se mantêm os diferentes utilizadores, cada um ligado ao seu próprio dispositivo de entrada/saída em contacto directo e simultâneo com o sistema computacional. Usando a multiprogramação, o processador é sucessivamente atribuindo à execução dos diversos programas presentes durante intervalos de tempo muito curtos (time slots). Devido à lentidão do tempo de resposta humana, quando comparado com um computador, cria-se a ilusão nos utilizadores que o sistema lhes está inteiramente dedicado. “time-sharing” refere-se neste contexto à partilha do tempo de processador.

- **7. Embora sendo um sistema interactivo, um sistema de tempo real tem características próprias muito bem definidas. Descreva duas delas, justificando**

convenientemente a sua resposta.

R: São impostos limites máximos de tempo de resposta do sistema computacional para criar um ambiente confortável ao utilizador. Cada utilizador está em contacto directo com o sistema computacional. O CPU recorre à multiprogramação, dividindo a execução dos diferentes pedidos em time-slots, criando nos utilizadores a ilusão de que o CPU se encontra inteiramente dedicado a eles.

- **8. 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?**

R: Os sistemas operativos do tipo batch maximizam o processamento e minimizam a intervenção humana. É necessário para processamento pesado em termos computacionais (CPU intensive programs).

Pode ser agendada execução em série de vários processos (jobs), rentabilizando ao máximo o sistema computacional mas é pouco vantajoso na utilização de dispositivos I/O, dado que não aproveitam o CPU durante as fases de leitura/escrita nos mesmos.

Exemplos: num ambiente em rede, podem ser aproveitados os tempos mortos de utilização dos computadores para aumentar a capacidade de processamento dos outros.

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

R: Um sistema de operação de rede tem como objectivo tirar partido das actuais facilidades de interligação do sistema computacional para estabelecer um conjunto de serviços comuns a uma comunidade, tais como transferências de ficheiros, partilha de dispositivos remotos ou acesso à internet.

Um sistema de operação distribuído tem o objectivo de tirar partido das actuais facilidades de construção de sistemas computacionais multiprocessador ou de interligação de sistemas computacionais distintos para estabelecer um ambiente de interacção com o utilizador, que encara o sistema computacional paralelo como uma entidade única.

A semelhança entre o sistema de operação de rede e distribuído é que ambos envolvem, ou podem envolver, sistemas computacionais distintos, interligados entre si. As suas divergências resultam de um sistema operativo de rede estabelecer um conjunto de serviços comuns a cada sistema computacional, enquanto o sistema de operação distribuído considera os vários sistemas computacionais interligados como uma entidade única que interage com o utilizador.

R: Têm uma característica comum: tentam tirar partido da facilidade de ligação entre sistema computacional (hardware), sendo que um sistema distribuído pode ser formado apenas por uma máquina com vários CPUs.

Sistemas Distribuídos -> tentam estabelecer um ambiente integrado de interacção com os utilizadores, que encaram o sistema computacional paralelo como uma entidade única. Desta forma, permitem o aumento da capacidade de processamento por incorporação de novos CPUs ou sistema computacional, o balanceamento da carga a processar pelos diferentes CPUs, permitem a paralisação de aplicações e ainda a implementação de mecanismos de tolerância a falhas.

Sistemas de Rede -> São formados por múltiplos sistemas computacionais ligados entre si por um canal de comunicação, tentando tirar o máximo partido do mesmo. Permitem serviços de partilha de recursos e transferência de ficheiros (FTP) entre sistemas computacionais, partilhas remotas, etc… Oferecem um bom suporte de device drivers para ligação de vários dispositivos I/O onde cada utilizador recorre a um terminal para se ligar ao sistema e utilizar os recursos disponíveis.

- **10. Os sistemas de operação de uso geral actuais são tipicamente *sistemas de operação de rede*. Faça a sua caracterização.**

R: Tentam tirar partido da facilidade de ligação entre sistemas computacionais (hardware).

São formados por múltiplos sistemas computacionais ligados entre si por um canal de comunicação, tentando tirar

o máximo partido do mesmo. Permitem serviços de partilha de recursos e transferência de ficheiros (FTP) entre sistemas computacionais, partilhas remotas, etc… Oferecem um bom suporte de device drivers para ligação de vários dispositivos I/O onde cada utilizador recorre a um terminal para se ligar ao sistema e utilizar os recursos disponíveis.

- **11. A partilha de ficheiros é uma característica marcante dos sistemas de operação de rede. Procure explicar como esta facilidade pode ser usada para permitir a um qualquer utilizador o acesso à sua área de trabalho a partir de um computador genérico de uma rede local, tal como se passa nos laboratórios da universidade.**

R: Os sistemas operativos de rede têm 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. Para isso foram criados vários serviços de partilha de ficheiros tais como o NFS (partilha de sistemas de ficheiros remotos que criam a ilusão de serem locais) e o acesso a sistemas computacionais remotos (telnet, remote login).

• *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 (*browsers*).

• *Dispositivos de comunicação típicos*: modem, ADSL, cabo, Ethernet.

- **12. Os sistemas de operação dos *palmtops* ou *personal digital assistants* (PDA) têm características particulares face ao tipo de situações em que são usados. Descreva-as.**

R: Os palmtops ou 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 demasia o processador pois se utilizar as aplicações terão pouco tempo de CPU. Os displays pequenos levam o SO a ter de condensar a informação.

R: Os sistemas de operação dos palmtops ou pda tem menos device drivers que os sistemas de operação dos pcs. Mas vem com outros utilitários, como é o caso das rotinas internas que lidam com a capacidade de "touch-screen".

R: Como estes sistemas computacionais são de tamanho reduzido, têm CPUs menos eficientes, memória reduzida e recorrem a baterias, têm de possuir um sistema operativo que faça uma boa gestão de tudo isso, principalmente ao nível do processamento. Por exemplo: é necessário evitar uso excessivo do CPU aumentando assim a durabilidade da bateria.

- **13. 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 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.

- **14. Numa arquitectura de tipo *microkernel*, diferentes funcionalidades do sistema de operação são executadas em modo utilizador como processos de**

sistema, originando aplicação a um só sistema computacional de um modelo de processamento muito comum em sistemas distribuídos - o chamado *modelo cliente-servidor*. Quem são neste contexto os *clientes* e os *servidores*? Que vantagens é que esta concepção apresenta na construção do núcleo do sistema de operação?

R: Arquitectura interna:

Abordagem monolítica:

→ O kernel do sistema operativo é um programa único em que todas as funcionalidades e estruturas de dados estão directamente acessíveis;

→ É eficiente mas difícil de testar e modificar.

Abordagem modular:

→ O kernel do sistema operativo é concebido numa perspectiva top-down em que as diferentes funcionalidades são isoladas;

→ A implementação menos eficiente;

→ É possível testar ou alterar os componentes em separado e introduzir novas funcionalidades, minimizando os riscos.

Arquitectura em camadas:

→ Cada novo módulo é construído sobre o anterior (decomposição hierárquica);

→ Conceptualmente seguro, mas necessário cuidado na definição das várias funções de cada camada;

→ Problemas de overhead nas comunicações, no caso de muitas camadas;

Kernel minimalista (microkernel):

→ Kernel reduzido ao mínimo das funcionalidades (gestão base do CPU, memoria e comunicação);

→ Outras funcionalidades do sistema operativo são implementadas por processos de sistema (modo user) que comunicam entre si e com as aplicações, recorrendo ao mecanismo base de comunicação do kernel;

→ Funcionalidades extra podem ser carregadas dinamicamente através de módulos, ligados ao kernel já em execução.

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 envolvidos sejam garantidos. Quais são eles?**

R: Como se torna complexa a recepção das diferentes actividades que estão em curso devido a criação de uma imagem de aparente simultaneidade na execução dos diferentes processos, torna-se mais simplificado supormos a existência de processadores virtuais, um por cada processo. E necessário garantir que a execução dos processos não é afectada **pelo instante, ou local no código** onde ocorre a comutação dos processadores virtuais, de modo a não perder o código que ainda não foi executado ou valores calculados até a comutação. Também é necessário garantir que **não são impostas restrições** relativamente aos tempos **de execução totais ou parciais**, dos processos, visto não ser possível saber o tempo que demorará certo processo concorrente a executar.

- **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?**

R: É na TCP que está a informação necessária para o sheduler fazer a gestão do processador e de outros recursos do sistema computacional. Os vários campos da PCT são:

- **caracterização do espaço de endereçamento**: sua localização (em memória principal ou na área de 'swapping'), de acordo com o tipo de organização de memória estabelecido;
- **contexto de I/O**: informação sobre os canais de comunicação e 'buffers' associados;
- **contexto 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?**

R: O sheduler é a parte do sistema de operação que lida com as transições entre estados do processo (run, ready-to-run e block) e constitui parte integrante do núcleo central do processador (kernel) que é responsável pelo tratamento das interrupções e por agendar a atribuição do processador e de muitos outros recursos do sistema computacional. Os critérios que devem ser satisfeitos pelo **algoritmo de sheduling do processador** são:

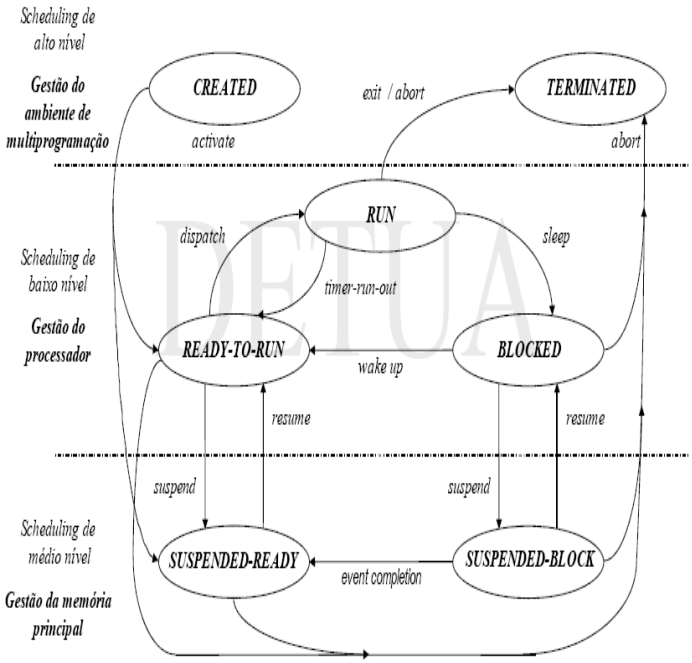
- **justiça** (todo o processo tem o direito a uma fracção do tempo de processador num intervalo de tempo considerado como referência);
- **eficiência** (tentar manter o processador o maior tempo possível na execução de processos dos utilizadores);
- **throughput** (maximizar o nº de processo terminados 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?**



R:

BAIXO -> gestão do processador, blocked (impedido de continuar ate que um acontecimento externo ocorra), ready-to-run (aguarda atribuição do processador para começar ou continuar a execução), run (detem posse do processador, logo esta a ser executado).

MEDIO -> gestão da memória principal, existe devido a necessidade de por vezes ser necessário libertar espaço na memoria principal para execução de processos que de outro modo não poderiam existir, liberta-se passando os processos para uma área secundaria de armazenamento, área de swaping. Suspended-ready (para os processos que se encontravam no estado ready-to-run), suspended-block (para os que se encontravam em blocked).

ALTO -> gestão de ambiente de multiprogramaçao, tem como base que todos os processos tem um tempo de vida. (Created, que é quando lhe é atribuído um erspaço de endereçamento e inicializaçao das estruturas de dados associadas a gerir o processo, sendo que estas operações tem de ocorrer sempre antes da execução. Terminated , que é associado com a manutenção temporária e da info relativa à historia da sua execução).

- **Num sistema de tipo *batch* multiprogramado fará sentido a existência de três níveis de *scheduling*?**

Num sistema operativo batch temos como objectivo optimizar a ocupação do processador, sendo isto conseguido pelo aproveitamento dos tempos mortos de processador, executando outro, logo **não serão necessário 3 niveis**

porque não será necessário o médio nível, pois como nos interessa utilizar o processador, não necessitamos de swap.

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?**

R: Os processos que se encontram no estado Ready-to-Run estão a aguardar atribuição do processador para começar ou continuar a execução, sendo que tem apenas uma entidade responsável pela sua atribuição, o processador, assim só necessita de uma fila de espera. Em Blocked, como os processos esperam pela ocorrência de um acontecimento externo, que podem ser vários, tais como o completamento de uma operação de entrada/saída, ou o acesso a um recurso, torna-se necessário uma fila de espera por acontecimento.

- **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: 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

R: As funções do kernel são o controlo e gestão de todas as partes integrantes do sistema computacional, desde todo o tipo de memórias, processador, recursos. O kernel sabe da existência de, por exemplo, um processo que pede atribuição do processador, ou quando uma tecla é premida no teclado através de excepções, tanto a nível externo (interrupções), execução de instrução ilegal, execução de instrução tipo trap. Nesta última situação o kernel pode ser visto como gestor de excepções, recebendo-as e gerindo qual o tratamento a dar a cada uma delas.

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

R: Ao ocorrer uma comutação entre processos, ocorre também uma comutação de contexto, visto que cada processo tem o seu contexto, desde a caracterização do espaço de endereçamento, contexto do processador, de I/O e toda a informação de estado e scheduling. De modo a não haver perda de informação durante a comutação é necessário salvaguardar esta informação na tabela de controlo de processos. Antes de ocorrer realmente a comutação todo o contexto associado ao processo que vai perder a atribuição é salvaguardada na tabela de controlo de processos, de seguida é alterado o seu estado na tabela de controlo de processos, é seleccionado o próximo processo a executar através de um algoritmo de scheduling, é colocado este processo em RUN, restaurada a sua informação da tabela de controlo de processos e depois é executado.

- **8. Classifique os critérios devem ser satisfeitos pelos algoritmos de *scheduling* segundo as perspectivas sistémica e comportamental, e respectivas subclasses.**

Justifique devidamente as suas opções.

R: Os critérios a serem satisfeitos pelos algoritmos de "scheduling" do processador podem ser enquadrados pelas perspectivas:

-> Sistémica:

- **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.

-> Comportamental:

- critérios orientados para o desempenho: são quantitativos (podem ser medidos);
- outro tipo de critérios: são qualitativos e difíceis de medir de modo directo.

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

R: 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 prioridade reposta no valor inicial quando atinge o mínimo. O sistema operativo em tempo real é um exemplo de sistema operativo que usa prioridades estáticas.

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.**

R: 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 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.**

R: 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.

Nas políticas de scheduling em sistemas operativos tempo real os processos possuem, normalmente, um tempo de

vida curto e uma execução relativamente rápida, pelo que uma política non-preemptive pura não será de todo interessante. Por outro lado, uma política de scheduling puramente preemptive executa sempre primeiro os processos com prioridade máxima, levando a que possa existir atrasos no sistema, que deixaria de ser "em tempo real".
Concluindo, a melhor solução será um sistema misto, ou seja, a política standart do sistema utilizada pode ser a preemptive, mas assim que surgir um ou mais processos com prioridade muito elevada (processos críticos do sistema) deve ser iniciada uma política non-preemptive, executando os processos críticos até ao fim, sem interrupções.

- **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: Nos sistemas batch pretende-se minimizar o tempo de turnaround, ou seja, o tempo de um processo, que inclui os tempos de espera e de execução. Para tal, é necessário abrir excepções à regra non-preemptive, tentando executar 1º os processos menos demorosos. Assim, a política non-preemptive utilizada não é pura, já que se recorre, ocasionalmente, à troca da ordem de execução dos processos (política preemptive) para diminuir o tempo de turnaround.

- **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: Na classe SCHED_OTHER o scheduling baseia-se num algoritmo baseado em créditos, sendo que entra-se em consideração o instante de recreditação i. CPUj(i) será a prioridade atribuída a j no instante i, CPUj(i-1) os créditos não utilizados no instante de recreditaçao anterior, Pbasej é a prioridade base do processo e nicej a prioridade dependente do user. Será chamado para execução o processo com maior prioridade, quando ocorre uma interrupção da real time clock, o processo perde um crédito, sendo que quando o número de créditos atinge zero o processo perde a posse de processador por esgotamento de janela. Ocorrem operações de recreditaçao sempre que na fila de espera de processos prontos a serem executados não existirem processos com créditos não nulos, entrando os bloqueados também na recreditação. Deste modo é fácil entender que 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.**

R: O aging dos processos 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, sendo por isso atendido. A disciplina de scheduling SJF (shortest job first) e SRT (shortest remaining time) tem esta característica.
R: Aging dos processo é a técnica de estimativa do valor seguinte numa série, tendo em conta a media da medida actual e a estimativa anterior. O aging é encontrado em disciplinas de scheduling para sistemas batch e para sistemas interactivos. O sheduling para sistemas batch é baseado na redução do tempo de turnaround (expera +execução) dos processos que constituem a fila de processamento, sendo que são atendidos em 1º lugar os processos com menos tempo de turnaround. Através do cálculo das estimativas dos tempos de execução, que é necessário para o cálculo da prioridade, é feito através de estimativa, logo terá como carácter o aging dos processos. Nos sistemas operativos interactivos as prioridades são estabelecidas com princípio na estimativa da fracção de ocupação da janela de execução seguinte, em termos da ocupação das janelas passadas, sendo o processador atribuído ao processo com estimativa menor, logo pelo aging.
Com aging, os processo que acederam mais recentemente ao processador serão atendidos mais tarde, o que se faz de modo a evitar o adiamento indefinido.

- **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.

R: Um processo é composto por uma pertença de recursos, onde se encontra o espaço de endereçamento e conjunto de canais de comunicação com dispositivos entrada/saida, e um fio de execução (thread) onde temos um PC que sinaliza a localização da instrução que deve ser executada a seguir, um conjunto de registos internos ao processador onde temos os valores actuais das variáveis em processamento e um stack que armazena a história de execução.
Um thread terá apenas o fio de execução. Os processos embora tenham estas 2 propriedades podem ser tratadas separadamente pelo sistema operativo, sendo que os thread, também conhecidos como light weight processes, constituem entidades executáveis independentes dentro de contexto de um mesmo processo.
R: 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 Sistema Computacional multiprocessador podemos executar cada thread num processador paralelizando assim a execução. O acesso à memória é também mais rápido visto ter 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 múltiplas actividades o atendimento a múltiplas 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.**

R: As vantagens de um ambiente multithread são a sua simplicidade e modularidade da implementação (programas com múltiplas solicitações têm uma implementação mais simples seguindo uma perspectiva concorrencial do que sequencial), melhorias na gestão de recursos (existe partilha do espaço de endereçamento e do contexto de I/O entre threads) e eficiência na velocidade de execução (não é necessário mudanças de contexto nem de espaço de endereçamento e existe a possibilidade de execução de múltiplas threads em paralelo, isto é, multiprocessadores simétricos).

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

R: As alternativas fornecidas pelo sistema operativo, à implementação de um ambiente multithreaded são **User threading** (os threads são implementados por uma biblioteca específica ao nível do utilizador que permite a criação, gestão e scheduling dos mesmos sem interferência do kernel, implica elevada eficiência, mas como o kernel não vê as threads ,isto é, só vê processos, quando um thread executa uma chamada ao sistema bloqueante, todos os threads e todo o processo é bloqueado) e **Kernel threading** (os threads são implementados directamente ao nível do kernel, providenciando operações de criação, gestão e scheduling dos mesmos sendo esta implementação menos eficiente mas possui a vantagem da execução de uma chamada ao sistema bloqueante por um dado thread do processo não bloqueia todo o processo).
Podemos utilizar um ambiente baseado em múltiplos processos para executar os diferentes threads de uma mesma aplicação em paralelo. Num multiprocessador simétrico é necessário ter em atenção que os threads partilham o espaço de endereçamento e canais de comunicação com dispositivos I/O, ou seja, é sempre necessário garantir que apenas um thread de cada vez acede aos elementos partilhados entre os threads.

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

R: 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?

R: Como o kernel não consegue ver as threads (só processos), quando uma delas executa uma chamada ao sistema bloqueante, na melhor das hipóteses não existe nenhuma thread para ser executada logo não é bloqueada nenhuma. Contudo, no pior caso (todas a threads prontas a executar), todos os threads e todo o processo é bloqueado.

Para minimizar este problema podia ser implementada uma função que fizesse o scheduling das threads com chamadas ao sistema bloqueante para que fossem executadas apenas quando o número de threads prontas a executar fosse mínimo, evitando o bloqueio de várias threads e aumentando a eficiência.

Será ainda possível minimizar este problema através da implementação de kernel-threads, em que os threads são implementados directamente ao nível do kernel, que providencia as operações de criação, gestão e scheduling. A sua implementação é menos eficiente mas tem como vantagem que o bloqueio de um thread, não afecta a calendarização para execução das restantes.

- **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: Haverá **um par de stacks** (sistema/utilizador) por thread, pois cada processo necessita de ter uma stack onde se encontram os diferentes threads associados a este processo. Para o kernel os threads em nível utilizador são simplesmente um processo com apenas um thread, executando-os deste modo. No caso das threads implementados ao **nível do kernel**, existe apenas **um stack** com todos os threads do sistema, sendo que quando um thread pretende criar novo thread ou destruir um já existente, é feita uma chamada ao sistema ao kernel, que actualiza a stack.

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: Podem ser processos independentes ou cooperantes.
Os **independentes** quando são criados têm o seu tempo de vida e terminam sem entrarem em interacção com outros processos de um modo explícito, a única interacção é implícita e tem origem na sua competição pelos recursos do sistema. Este tipo de processos são tipicamente processos lançados pelos diferentes utilizadores num ambiente interactivos e/ou processos que resultam do processamento de jobs nos Batch. É da responsabilidade do **sistema operativo** garantir que a atribuição de um recurso seja feita de maneira controlada para que não haja perda de informação, sendo o responsável por **exclusão mútua.**
Os **cooperantes** partilham informação ou comunicam entre si de um modo explícito, sendo exigido para a partilha um espaço de endereçamento comum e um canal de comunicação. A responsabilidade **dos processos** garantir **exclusão mútua**.

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

R: 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 tipo de propriedades devem apresentar as primitivas de acesso com exclusão mútua a uma região crítica?**

R: 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.
<u>Propriedades das primitivas de acesso com exclusão mútua a uma região crítica:</u>
Garantia efectiva de imposição de exclusão mútua:

- de todos os processos a competir pelo acesso a uma região critica ou recurso, apenas e só um de cada vez deve conseguir faze-lo, evitando assim possíveis perdas de informação;
- Garantir 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 a qualquer processo;
- 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.**

R: Condições de corrida equivalem a quando 2 ou mais processos encontram-se a ler ou escrever numa região partilhada, sendo que o resultado final depende de que processo foi executado num certo instante.
Ex: 2 processos partilham uma variável comum, armazenada na região partilhada de dados. Considere-se que o 1º pretende incrementar a variável e o 2º pretender apenas lê-la. Se o 1º fizer a leitura da variável, a incrementar e antes de a guardar, permitir a leitura do 2º processo, a informação torna-se inconsistente. Neste caso o 2º processo

não vai ler o valor mais actualizado da variável. Isto foi uma condição de corrida.

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

R: Deadlock ocorre quando 2 ou mais processos ficam a aguardar eternamente o acesso às regiões críticas respectivas, esperando acontecimento que se pode demonstrar nunca irão acontecer, o resultado é o bloqueio das operações. Adiamento indefinido ocorre quando 1 ou mais processos competem pelo acesso a uma região crítica e devido a uma conjugação de circunstâncias em que os processos novos os ultrapassam, o acesso é sucessivamente adiado, estamos assim perante um impedimento real à sua continuação. Exemplo de deadlock é quando o consumidor lê n_pos_vazias e este encontra-se com valor K (não ha info para consumir), de seguida os processos comutam e o produtor coloca informação no buffer, decrementa n_pos_vazias e vê que consumidor está bloqueado e acorda-o mas consumidor não está na lista de processos bloqueados, sendo o sinal emitido perdido, de seguida o consumidor testa o valor e como ainda tem é o antigo pensa que não há informação para consumir e bloqueia, sendo que quando o buffer encher os 2 ficam bloqueados para sempre. Adiamento indefinido ocorre quando não é feito um wake_up de consumidor a produtor quando já há posições vazias, ou de produtor para consumidor quando já há itens. Se o produtor tiver mais prioridade que consumidor, quando o buffer esta cheio ele não consegue colocar lá nada e ao haver comutação ele como tem prioridade ganha o acesso, levando ao adiamento indefinido do consumidor.

- **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?**

R: As **soluções software**, são soluções quer sejam implementadas num monoprocessador ou multiprocessador com memoria partilhada, supõem o recurso em ultima instancia ao conjunto de instruções básico do processador, ou seja, as instruções de transferência de dados de e para a memoria são do tipo standard. A única suposição adicional diz respeito no multiprocessador, em que a tentativa de acesso simultâneo a uma mesma posição de memória é necessariamente serializada por um árbitro.

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

As **soluções hardware** são soluções que supõem o recurso a instruções especiais do processador para garantir, a algum nível, a atomicidade na leitura e escrita. São muitas vezes suportadas pelo próprio sistema operativo 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?**

R: O algoritmo de Dijkstra consiste na generalização a N processos do algoritmo de Dekker. O algoritmo de Dijkstra usa um mecanismo de alternância entre os vários processos. Quando um processo quer entrar na região critica o seu estado é alterado para “interessado” e aguarda até que o estado do processo com prioridade seja “não”. Quando isto acontece o processo passa ter acesso à região critica e o seu estado é alterado para “decidido”. Na saída da região critica o estado do processo é alterado para “não” e a variável que contem o processo com prioridade também é alterada. Este algoritmo evita a ocorrência de deadlock, no entanto não evita o risco de adiamento indefinido devido à forma como o valor da variável que contêm o processo com prioridade é atribuído na função de entrada na região crítica e na forma como ela é calculada na função de saída da região crítica. No caso de existir, e sucessivamente a surgir, vários processos a quererem entrar na região critica pode acontecer que a variável que contem o processo com prioridade nunca tome certos valores.

- **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).

R: A diferença entre os dois algoritmos está na maneira como lidam com a contenção dos processos. Se dois processos querem entrar na Região Critica, um tem de recuar e dar lugar ao outro.

No algoritmo de Dekker 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 Região Critica, ficando o outro à espera.

O algoritmo de Dekker 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 Região Critica a não deixar os outros aceder, pois a prioridade pode ser de um processo que não queira aceder à Região Critica.

O algoritmo de Peterson é passível de generalizar a N processos. Pois só processos que querem aceder à Região Critica se colocam na fila, sendo o 1º a aceder à Região Critica 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 Dekker.

- **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?**

R: É atendido primeiro o que está na ultima posição. Os processos vão-se colocando na primeira posição da fila e vão caminhando do inicio até à ultima 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.**

R: 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.**

R: Flag de locking é uma variável partilhada por todos os processos, que tem como valor inicial zero, sendo que zero significa que nenhum processo encontra-se na região crítica e 1 que está algum processo lá, garantindo deste modo exclusão mútua. Quando a flag está a 0, o processo altera o valor para 1 e acede à região critica, se o processo testa a flag e está a 1, este fica à espera, no seu estado activo (busy waiting), que o valor seja alterado para 0 para poder aceder à região critica em exclusão mútua.

- **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.

R: Semáforos são basicamente variáveis do tipo estrutura que arbitram a intercomunicação entre processos. 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:
sem_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--.
sem_up -> se o campo val=0, val++. Caso contrário um dos processos da fila de espera é acordado de acordo com regras bem definidas.

```
/* operação down */
void sem_down (unsigned int semid)
{
  inibição das interrupções;
  if (sem[semid].val == 0) sleep_on_sem (getpid(), semid);
    else sem[semid].val -= 1;
  activação das interrupções;
}

/* operação up */
void sem_up (unsigned int semid)
{
   inibição das interrupções;
   if (há processos bloqueados em sem[semid])
                                  wake_up_one_on_sem (semid);
    else sem[semid].val += 1;
   activação das interrupções;
}
```

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 Região crítica. Deste modo garantimos acesso em exclusão mutua a uma região critica e sincronizando os processos entre si.

- **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.**

R: Um monitor é um dispositivo de sincronização, proposto de uma forma independente por Hoare e Brinch Hansen, que pode ser concebido como um módulo especial, suportado pela linguagem de programação [concorrente] e constituído por uma estrutura de dados interna, por código de inicialização e por um conjunto de primitivas de acesso. O monitor constitui uma solução de alto nível para o problema de exclusão mutua entre processos, dado que alem de permitir a entrada de um processo de cada vez, a informação que contem é apenas vista pelos processos que lhe tem acesso e é invisível aqueles que aguardam a entrada. Os programas que usam monitores são mais simples de escrever. Como a informação dentro do monitor está bem protegida o sistema torna-se seguro e fiável, e o uso de "condition variables" simplifica o sincronismo entre os processos.
Para gerir a sincronização das threads existem 2 operações a ser executadas sobre a variável de condição:

- wait – o thread que invoca a operação é bloqueado na variável de condição e é colocado fora do monitor

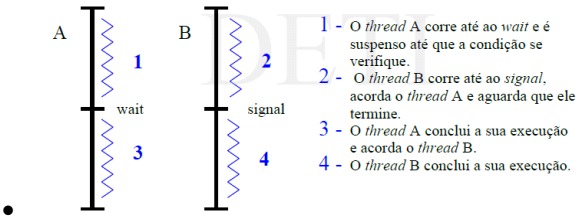
para possibilitar que um outro thread que aguarda acesso, possa prosseguir;

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

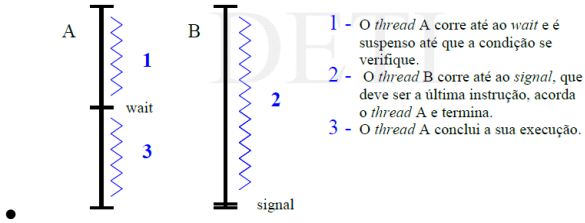
14. Que tipos de monitores existem? Distinga-os.

R: Os monitores dividem-se em 3 modelos distintos:

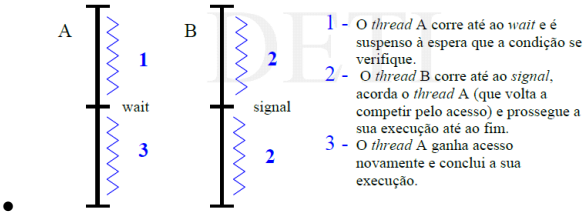
- **Hoare:** o thread que invoca a operação de signal é colocado fora do monitor para que o thread acordado possa prosseguir. É muito geral, mas a sua implementação exige a existência de um stack, onde são colocados os threads postos fora do monitor por invocação de signal.



- **Brinch e Hansen:** o thread que invoca a operação de signal liberta imediatamente o monitor (signal é a ultima instrução executada). É simples de implementar mas pode tornar-se restritivo porque so ha possibilidade de execução de um signal em cada invocação de primitiva de acesso.



- **Lampson/Redell:** o thread que invoca a operação de signal prossegue a sua execução, o thread acordado mantém-se fora do monitor e compete pelo acesso a ele. É mais 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?

R: 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?

R: Trata-se de um mecanismo geral que não exige partilha de espaço de endereçamento e que se baseia na troca de mensagens entre dois ou mais processos, através de um canal de comunicações.
O problema de acesso com exclusão mútua a uma região crítica é colocado quando 2 ou mais processo pretendem aceder ao canal, simultaneamente, para enviar/receber mensagens. Cabe ao sistema operativo resolver este problema, dado que se trata de um acesso com exclusão mútua a um recurso do sistema computacional.

- **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.**

R: //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).**

R:

Não bloqueantes:

- A sincronização é da responsabilidade dos processos intervenientes;
- A operação de envio envia-se a mensagem e regressa-se sem qualquer informação de recepção;
  - void msg_send_nb(unsigned int msg_dest_id, MESSAGE msg)
- A operação de recepção lê a mensagem, caso exista. Se não continua a execução.
  - void msg_receive_nb(unsigned int msg_src_id, MESSAGE *msg, boolean *msg_arrival)

Bloqueantes:

As operações envio/recepção contêm elementos de sincronização;
A operação de envio, envia uma mensagem e bloqueia até que seja recebida;
- void msg_send(unsigned int msg_dest_id, MESSAGE msg)

A operação de recepção bloqueia até receber uma mensagem;
- void msg_receive(unsigned int msg_src_id, MESSAGE *msg)

- Rendez-Vous:
  - Só existe transferência de mensagem após uma sincronização dos intervenientes;
  - Não existe armazenamento intercalar da mensagem;
  - Típica de canais de comunicação dedicados;
- Remota:
  - A operação de envio, envia a mensagem e bloqueia até que seja entregue;
  - Pode existir armazenamento intercalar da mensagem (caixas de correio);
  - Típica de canais partilhados;

- **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.**

R: 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. Mostre como poderia criar um canal de comunicação bidireccional (full-duplex) entre dois processos parentes usando a chamada ao sistema pipe.**

R: Um Pipe é um canal de comunicação elementar (half-duplex) que é estabelecido entre 2 ou mais processos, permitindo a sua comunicação.
No entanto, só pode ser utilizado entre processos que estão relacionados entre si (entre pai e filho ou entre irmãos) e o canal de comunicação é unidireccional.
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 standard output de um dado comando é redireccionado para a entrada de um pipe e o standard input do comando seguinte é redireccionado para a saída do mesmo pipe.
Para criarmos um full-duplex temos de criar 2 pipe half-duplex, um que fará a comunicação num sentido e o outro no outro sentido.

- **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?**

R: 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 qualquer 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 processos que os detêm (**canal de comunicações**, **impressora**, **memória de massa** se dois processos quiserem escrever **no mesmo 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.**

R:
Condições necessárias à ocorrência de um deadlock:

- ***Condição de exclusão mútua*:** um recurso é detido por um único processo, ou está livre;
- ***Condição de espera com retenção*:** ao requerer um novo recurso, o processo mantêm em sua posse os recursos pedidos anteriormente;
- ***Condição de não libertação***: só o próprio processo pode libertar um recurso a ele atribuído;
- ***Condição de espera circular***: cadeia circular de processos e recursos em que cada processo pede um recurso que está na posse de outro;

Para não haver deadlock no sentido estrito temos que garantir uma das seguintes condições:

- ***não há exclusão mútua* no acesso a um recurso** (processos acedem quando querem não havendo assim deadlock, mas condições de corrida, o que também 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. Devem-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 quando não consegue obter 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.**

R: O algoritmo dos banqueiros de Dijkstra é uma política de prevenção de deadlock no sentido lato, em que a atribuição 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.
Ex: Só é atribuído um recurso a um processo se numero de recurso anteriormente a si atribuídos for menor que o numero total de recursos já atribuídos a 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?**

R: Estado seguro é um estado em que a distribuição de recursos (livres ou atribuídos) leva sempre a que haja uma sucessão de estados seguintes, que permitem a terminação, com sucesso, de todos os processos existentes. 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.**

R: As políticas de prevenção de deadlock no **sentido estrito** embora sejam **muito seguras**, são **muito restritivas**, **pouco eficientes** e **difíceis de aplicar** em situações muito gerais, por exemplo, no caso da negação da condição de exclusão mutua, só pode ser aplicada a recursos passíveis de partilha em simultâneo, na negação da condição de espera com retenção é necessário ter um conhecimento prévio de todos os recursos necessários. Na imposição da condição de não libertação, como é suposta a libertação de todos os recursos anteriores quando o próximo não puder ser atribuído, ocorrem atrasos na execução do processo. No caso da negação da espera circular são desaproveitados recursos eventualmente disponíveis que poderiam ser usados na continuação do processo.
As no **sentido lato**, embora também sejam seguras, não são menos restritivas e ineficientes do que as no sentido estrito, no entanto **são mais versáteis**, logo mais **fáceis de aplicar** em situações gerais.

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.**

R: A memória de um sistema computacional está organizada tipicamente em níveis diferentes, formando uma hierarquia:

- *Memória cache* -> **pequena** (vai de KBs a poucos MBs), é **muito 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* -> de **tamanho médio** (vai de centenas de MBs ou unidades 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* -> **grande** (tem capacidade de centenas ou milhares de GBs), é **lenta**, **barata** e **não volátil**.
  Tem como função fornecer um sistema de ficheiros onde são salvaguardados mais ou menos
  permanentemente os dados e código. Também 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 hierárquica de memória? Dê razões que mostrem porque é que a aplicação de tal princípio faz sentido.**

R: 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 que ser consideradas.**

R:

- 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 quando não necessária.

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

R: 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 espaco 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.**

R: 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 estrutura de dados em Linguagem C tem que ser usada para possibilitar o acesso às diferentes variáveis da área partilhada?**

R:

- 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 definição 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 variáveis 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?**

R: 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 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?**

R: 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 endereçamento 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 físic*o numa organização de memória real.**

R: Algoritmo de cálculo:

- 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.**

R: 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 (hardware 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 sucessivamente 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 ineficiente.

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.**

R: 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.**

R: 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 endereçamento 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 endereçamento 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?**

R: 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 variável).

- **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.**

R: Características de uma organização de memória virtual (swapping):

- não existe limitação para o espaço de endereçamento do processo 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 espaço de endereçamento do processo 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.**

R: O que distingue uma arquitectura paginada da arquitectura segmentada/paginada é que nesta não se faz uma divisão do espaço de endereçamento lógico do processo às cegas, pois primeiro faz-se uma divisão do endereçamento lógico do processo 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 espaço de endereçamento);
- 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 sistema operativo 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 hardware.

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

R:

- Extrair, do endereço lógico, o número do segmento (bits mais significativos).
- Aceder à tabela de segmentos, usando este número, p/ obter o endereço físico do início do segmento
- Comparar o deslocamento (bits menos significativos do endereço lógico) com o comprimento do segmento; se aquele for maior do que este o endereço é inválido.
- Endereço físico = endereço físico inicial do segmento + deslocamento.

- **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.**

R: 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 hardware 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 papéis diferentes nas organizações de memória real e de memória virtual?**

R: Porque ao contrário da organização de memória real, em que temos uma correspondência biunívoca entre espaço de endereçamento lógico e espaço de endereçamento 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 endereçamento 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 espaços de endereçamento do processo que coexistem correntemente, para que quando seja necessária uma parte do espaço de endereçamento do processo residente na área de swapping ela seja transferida para a memória principal pelo sistema operativo.

- **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.**

R: 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 páginas, 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 páginas: 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?**

R: 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 espaço de endereçamento do processo.
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 também 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 endereçamento 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?**

R: - 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 que é conveniente impor que o tamanho de cada frame seja uma potência de dois?**

R: 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 hardware e software.

- **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.**

R: 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?**

R: 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 óptimo 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.**

R: Carrega-se a 1ª e a última página do espaço de endereçamento do processo (correspondente 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.
R: 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?**

R: 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.**

R: 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.