CONCEITOS INTRODUTÓRIOS

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

As duas perspectivas de definição de um sistema operativo são:

- Perspectiva "**top-down**" ou do programador.
- Perspectiva "**bottom-up**" ou do construtor.

Na perspectiva "top-down",

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

Na perspectiva "bottom-up",

- O sistema operativo é visto como o programa que gere o sistema computacional, fazendo a atribuição controlada e ordeira dos seus diferentes recursos aos programas que por eles competem.
- O objectivo é, portanto, conseguir-se a rentabilização máxima do sistema computacional, garantindo-se uma utilização tão eficiente quanto possível dos recursos existentes.

2- O que são chamadas ao sistema? Dê exemplos válidos para o Unix (recorde que o Linux não é mais do que uma implementação específica do Unix). Explique qual é a sua importância no estabelecimento de um interface de programação de aplicações (API).

As chamadas ao sistema são a interface entre o sistema e os programas do utilizador. Se um programa estiver a ser corrido em modo de utilizador e precisar de um serviço do sistema (exemplo: ler dados de um ficheiro) terá que executar uma instrução trap ou uma instrução de chamada ao sistema para transferir o controlo.

Alguns exemplos de chamadas ao sistema em Unix são:

- open
- close
- link
- fork
- kill

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?

Versatilidade, eficiência/desempenho, produtividade.

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

A multiprogramação consiste na coexistência num dado sistema computacional de diferentes processos que partilham o mesmoprocessador. Assim fica garantido um melhor aproveitamento doprocessador, uma vez que se reduz substancialmente tempos mortos que ocorrem durante, e não só, as transacções de informação com dispositivos de imput/output. Os processos acedem ao processador em tempos diferentes (multiplexagem no tempo).

No multiprocessamento existem à disposição dos diferentes processos vários processadores, ou seja a carga computacional é distribuída pelos diversos processadores.

Num ambiente de multiprocessamento pode não existir multiprogramação se tivermos um processo diferente para cada processador.

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

Num sistema de multiutilizador de uso geral, pode-se falar de multiprogramação uma vez que apenas existe um processador que tem que ser atribuído aos diversos utilizadores. O tempo de processamento deve ser equivalente para todos os utilizadores. E cada um dos utilizadores pode ainda estar a trabalhar com vários processos que concorrem entre eles pela posse do processador, quando este é atribuído ao utilizador em causa.

6- Os sistemas de operação de tipo *batch* são característicos dos anos 50 e 60, quando o custo dos sistemas computacionais era muito elevado e era necessário rentabilizar a todo o custo o *hardware*. A partir daí, com a redução progressiva de custos, os sistemas tornaram-se interactivos, visando criar um ambiente de interacção com o utilizador o mais confortável e eficiente possível. Será que hoje em dia ainda se justificam sistemas deste tipo? Em que circunstâncias?

Ainda hoje se justifica a utilização de sistemas de operação tipo batch, quando se quer fazer processamento em série de grandes cargas computacionais. Com estes sistemas pode-se fazer o agendamento de várias tarefas (Job's) que vão sendo executadas em sequência. Desta forma o sistema computacional é fortemente rentabilizado. Esta rentabilização pode ser feita, por exemplo, quando um computador de rede está sem utilizador ligado aproveitando-se o processador para efectuar cálculos. Num ambiente de rede (local ou internet) pode-se aproveitar várias máquinas tendo em vista o aumento da capacidade de processamento.

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

Ambos os sistemas tiram partido das facilidade de ligação entre sistemas computacionais (ao nível do hardware), sendo de salientar, no entanto, que um sistema distribuído pode ser formado apenas por uma única máquina com multiprocessadores.

Os sistemas distribuídos usam essa interligação para estabelecer um ambiente integrado de interacção com o(s) utilizador(es) que encara(m) o sistema computacional paralelo como uma entidade única. Desta forma os sistemas distribuídos levam ao o aumento da velocidade de processamento por incorporação de novos processadores ou sistemas computacionais, permitem a paralelização de aplicações e permitem ainda a implementação de mecanismos de tolerância a falhas

Por outro lado os sistemas de rede permitem serviços de transferência de ficheiros (ftp), partilha de sistemas de ficheiros remotos, criando a ilusão de que são locais (*NFS*), partilha de dispositivos remotos (impressoras) e ainda mais alguns exemplos como telnet, email e acesso a internet.

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

VER ANTERIOR

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

Os palmtops são dispositivos portáteis, como tal são leves, compactos e com baixo consumo de energia. Estes factores limitam a memória (memória de massa é quase inexistente) e o set de instruções do processador é bastante reduzido.

Os sistemas de operação destes dispositivos devem garantir a compatibilidade com as aplicações diversas aplicações (folha de texto, calculo, etc). Por exemplo um pdf feito num desktop pode ser lido num PDA.

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

FILOSOFIA

Gestão do Processador

1- A modelação do ambiente de multiprogramação através da activação e desactivação de um conjunto de processadores virtuais, cada um deles associado a um processo particular, supõe que dois factos essenciais relativos ao comportamento dos processos sejam garantidos. Quais são eles?

Os dois factos que têm que ser garantidos são:

- A execução dos processos não é afectada pelo instante, ou local do código, onde ocorre a comutação.
- Não são impostas quaisquer restrições relativamente aos tempos de execução, totais ou parciais, dos processos.

2- Qual é a importância da tabela de controlo de processos (*PCT*) na operacionalização de um ambiente de multiprogramação? Que tipos de campos devem existir em cada entrada da tabela?

Num ambiente de multiprogramação o sistema operativo tem que gerir a informação relativa aos vários processos. Esta informação é mantida na tabela de controlo de processos (PCT). Esta tabela é usada intensivamente pelo scheduler para fazer a gestão do sistema operativo. Nesta tabela devem existir os seguintes campos:

- **Identificadores** – do processo, do pai do processo e do utilizador a quem o processo pertence.
- **Caracterização do espaço de endereçamento** – sua localização (em memória principal ou na área swapping), de acordo com o tipo de organização de memória estabelecido.
- **Contexto do processador** – valores de todos os registos internos do processador no momento em que se deu a comutação do processo.
- **Contexto de I/O** – informação sobre os canais de comunicação e buffers associados.
- **Informação de estado e de scheduling** – estado de execução do processo (de acordo com o diagrama de estados) e outra informação associada.

3- O que é o *scheduling* do processador? Que critérios devem ser satisfeitos pelos algoritmos que o põem em prática? Quais são os mais importantes num sistema multiutilizador de uso geral, num sistema de tipo *batch* e num sistema de tempo real?

O scheduling de um processador é a actividade desempenhada pelo scheduller. Este programa encarrega-se de gerir o processador do sistema computacional, atribuindo o processador, (mediante uma politica previamente definida, o algoritmo de escalonamento), aos diversos processos que competem pela sua posse.

Os algoritmos de scheduling devem satisfazer os seguintes critérios.

- **Critério de justiça** – todo o processo, ao longo de um intervalo de tempo considerado de referência, deve ter à sua fracção de tempo de processador.
- Previsibilidade – o tempo de execução de um processo deve ser razoavelmente constante e independente da sobrecarga pontual a que o sistema computacional possa estar sujeito.
- **Throughput** – deve procurar-se maximizar o número de processos terminados por unidade de tempo.
- **Tempo de resposta** – deve procurar minimizar-se o tempo de resposta às solicitações feitas pelos processos interactivos.
- **Tempo de turnaround** – deve procurar minimizar-se o tempo de espera pelo complemento de um Job no caso de utilizadores de sistemas do tipo batch.
- **Deadlines** – deve procurar garantir-se o máximo de cumprimento possível das metas temporais impostas pelos processos em execução.
- Eficiência – deve procurar manter-se o processador o mais possível ocupado com execução dos processos dos utilizadores.

Num **sistema de multiutilizador** de uso geral os critérios mais importantes são:

- Previsibilidade
- Throughput
- Eficiência

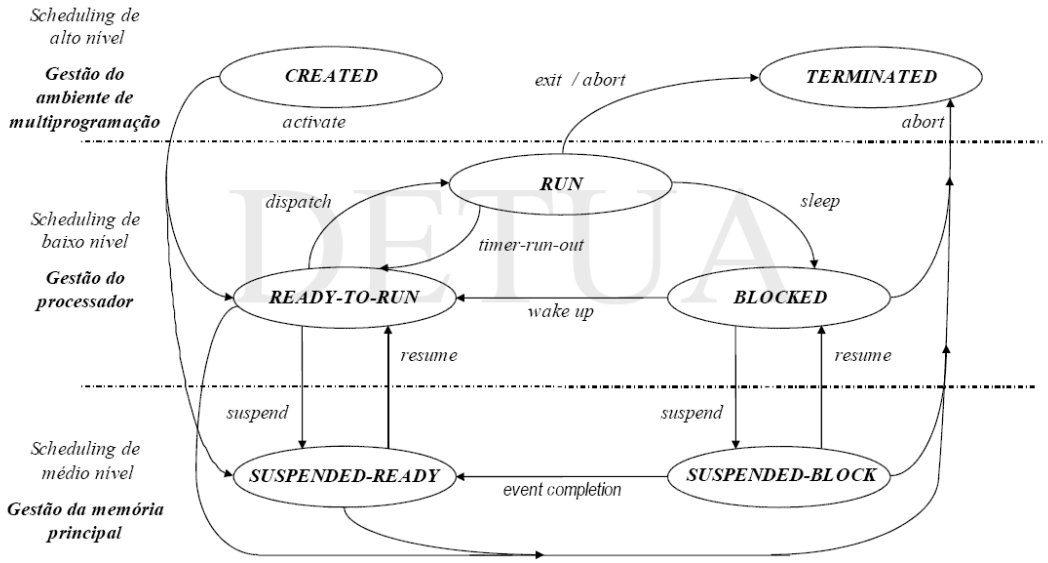
Num **sistema de tipo batch** os critérios mais importantes são:

- Previsibilidade
- Throughput
- Tempo de turnaround
- Eficiência

Num **sistema de tempo real** os critérios mais importantes são:

- Deadlines
- Tempo de resposta
- Previsibilidade

4- Descreva o diagrama de estados do *scheduling* do processador em três níveis. Qual é o papel desempenhado por cada nível? Num sistema de tipo *batch* multiprogramado fará sentido a existência de três níveis de *scheduling*?



Baixo nível: Gestão dos processos em memória principal efectuada pelo processador. Neste nível os processos podem estar em três estados diferentes: RUN (Processo a ser executado), BLOCKED (processos á espera de eventos externos (I/O)) e READY-TO-RUN (processos que estão á espera de atribuição do processador para entrar em execução).

Medio nível: Gestão dos processos entre memória principal e memória de 'swapping' efectuada pelo scheduling da memória principal. Neste nível os processos podem estar em dois estados diferentes: SUSPENDED-READY, SUSPENDED-BLOCK.

Alto Nivel : A situação descrita até agora pressupõe que os processos são eternos, existindo permanentemente enquanto o sistema computacional estiver operacional. À parte de alguns processos de sistema, porém, este não é o caso mais frequente. Os processos são em geral criados, têm um tempo de vida mais ou menos longo e terminam.

5- Os estados *READY-TO-RUN* e *BLOCKED*, entre outros, têm associadas filas de espera de processos que se encontram nesses estados. Conceptualmente, porém, existe apenas uma fila de espera associada ao estado *READY-TO-RUN*, mas filas de espera múltiplas associadas ao estado *BLOCKED*. Em princípio, uma por cada dispositivo ou recurso. Porque é que é assim?

Porque faz sentido que um processo esteja bloqueado no semáforo associado ao dispositivo. Por exemplo, temos uma fila de espera para todos os processos que queiram ler caracteres do teclado, temos outra para os processos que queiram escrever mensagens no ecrã. Em suma, necessitamos de pontos de bloqueio diferentes porque os processos poderão estar bloqueados à espera de diferentes dispositivos de I/O.

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*.

O kernel é responsável pelo tratamento das interrupções e por agendar a atribuição do processador e de muitos outros recursos do sistema computacional.
Assim, para que o sistema de operação funcione no modo privilegiado (Kernel Running), com total acesso a toda a funcionalidade do processador, as chamadas ao sistema associadas, quando não despoletadas pelo próprio hardware, são implementadas a partir de instruções trap. Cria-se, portanto, um ambiente operacional uniforme, em que todo o processamento pode ser encarado como o serviço de excepções. Nesta perspectiva, a comutação de processos pode ser visualizada globalmente como uma vulgar rotina de serviço à excepção, apresentando, porém, uma característica peculiar que a distingue de todas as outras: normalmente, a instrução que vai ser executada, após o serviço da excepção, é diferente daquela cujo endereço foi salvaguardado ao dar-se início ao processamento da excepção.

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

A comutação de contexto ocorre quando o processador está a processar um determinado processo, e por algum motivo, tem que passar para outro. Quando se dá esta alteração têm que ser guardadas as informações relevante do processo anterior para que da próxima vez que este vá ser atendido o processador inicie o processamento onde tinha acabado da última vez. À execução destas tarefas chama-se comutação de contexto.
A informação relevante que é guardada na comutação de contexto é:

- PC;
- PSW;
- informação sobre os buffer de I/O;

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

Os critérios a serem satisfeitos pelos algoritmos de *scheduling* do processador segundo a **perspectiva sistémica** são :

– *critérios orientados para o utilizador* – estão relacionados com o comportamento do sistema de operação na perspectiva dos processos ou dos utilizadores;
Troughput/Tempo de Resposta/Tempo de TurnAround/Justiça

– *critérios orientados para o sistema* – estão relacionados com o uso eficiente dos recursos do sistema de operação;
Eficiência/Previsibilidade

Quanto à ***perspectiva comportamental*** *tem-se:*

– *critérios orientados para o desempenho* – são quantitativos e passíveis, portanto, de serem medidos;
Eficiência/Deadlines;

– *outro tipo de critérios* – são qualitativos e difíceis de serem medidos de uma maneira directa.

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

As disciplinas de prioridade estática possuem um método de definição de prioridade determinístico (ex.: sistemas multiutilizador –> Unix SVR4 utiliza prioridade estática para os processos utilizador), já as de prioridade dinâmica possuem um método de definição de prioridade dependente da história passada de execução do processo(ex.:sistemas de tipo batch).

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.

Num sistema de operação multiutilizador de uso geral, os processos tem direitos de acesso ao processador diferenciados, e em geral os processos de I/O intensivo têm prioridade sobre os processos de processamento intensivo. Isto é assim porque se um utilizador estiver a trabalhar em linha de comandos (por exemplo) e outro estiver a compilar um programa, o primeiro não vai estar à espera da resposta a um comando durante longos segundos, minutos até. Com o acesso diferenciado melhora-se o desempenho do processador porque enquanto um processo I/O intensivo fica à espera, outro podem processar.

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.

À primeira vista a política dominante é a preemptive já que em tempo real existem processos com mais prioridade que outros, tendo os processos com menor prioridade presentes no processador que interromper a sua execução quando existe um processo com maior prioridade em espera. Quando existem processos com igual prioridade a política aplicada pode ser non-preemptive caso os processos saibam que não podem executar por longos períodos e façam o seu trabalho e bloqueiem rapidamente. **Exemplo**: (MISTO) Sistema computacional de um carro, controlo de AC, etc, quando os travões actuam não lhes pode ser retirado o processador e os outros processos esperam.

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?

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

A disciplina de scheduling usada em linux para classe Sched_Other é uma política de prioridade dinâmica, pois esta é implementada com o recurso a créditos. O scheduller calendariza para execução o processo com mais créditos que estiver na fila de espera.

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

O aging dos processos é a técnica que define a prioridade de um processo em função do tempo de espera (no estado Ready-to-Run), quanto maior o tempo de espera maior a prioridade.

Quando o sistema computacional tem uma carga muito grande de processos *I/O* intensivos, os processos *CPU*-intensivos correm o risco de *adiamento indefinido* se a disciplina de *scheduling* dinâmica for aplicada sem qualquer correcção. Um meio de alterar a situação é incorporar no cálculo da prioridade o tempo que o processo aguarda a atribuição do processador no estado *READY-TO-RUN*. Disciplinas de *scheduling* com esta propriedade são conhecidas por promoverem o *aging dos processos*.

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

Um **processo** é constituído por vários recursos (espaço de endereçamento, bloco de controlo e contexto I/O) e pode conter fios de execução ou **threads** que são as entidades escalonadas para a execução sobre a CPU, possuindo todos os mesmos recursos e o seu próprio program counter que sinaliza qual a próxima instrução a ser executada, um conjunto de registos internos do processador e um stack com a história da execução.

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

Em situações que necessitem de:

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

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

18- Explique como é que os *threads* são implementados em Linux.
O Linux lida com a questão da implementação de threads de um modo muito artificioso. À parte da chamada ao sistema **fork** que cria um novo processo a partir dum já existente por cópia integral do seu contexto alargado (espaço de endereçamento, contexto de I/O e contexto do processador), existe uma outra, **clone**, que cria um novo processo a partir de um já existente por cópia apenas do seu contexto restrito (contexto do processador), partilhando o espaço de endereçamento e o contexto de I/O e iniciando a sua execução pela invocação de uma função que é passada como parâmetro. Assim, não há distinção efectiva entre processos e threads, que o Linux designa indiferentemente de tasks, e eles são tratados pelo kernel da mesma maneira.

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

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

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*?

Comunicação entre Processos

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

Em termos de interacção existem:

processos independentes – quando são criados, têm o seu 'tempo de vida' e terminam sem interactuarem de um modo explícito; a interacção que ocorre é implícita e tem origem na sua *competição* pelos recursos do sistema computacional; trata-se tipicamente dos processos lançados pelos diferentes utilizadores num ambiente interactivo e/ou dos processos que resultam do processamento de *jobs* num ambiente de tipo *batch*;

processos cooperantes – quando partilham informação ou comunicam entre si de um modo explícito; a *partilha* exige um espaço de endereçamento comum, enquanto que a *comunicação* pode ser feita tanto através da partilha de um espaço de endereçamento, como da existência de um canal de comunicação que interliga os processos intervenientes**; (é da responsabilidade dos processos envolvidos garantir que o acesso à região partilhada seja feito de uma forma controlada para que não haja perda de informação)**

O problema de exclusão mútua nos processos independentes coloca-se no acesso aos recursos do sistema. Quanto aos processos cooperante para alem da exclusão mutua no acesso aos recursos do sistema existem também o problema da exclusão mutua no acesso à região de dados partilhada pelos vários processos cooperantes.

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

Quando vários processos desejam aceder a um recurso vai existir uma disputa entre eles para ver qual deles acede ao recurso. Este procedimento denomina-se competição por um recurso.

- dois processos querem aceder ao disco;
- dois processos querem imprimir na impressora;
- dois processos cooperantes querem aceder ao canal de 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?

Tornando a linguagem precisa, quando se fala em acesso por parte de um processo a um recurso, ou a uma região partilhada, está na realidade a referir-se a execução de um **processo** por parte do processador do código de acesso correspondente. Este código, porque tem que ser executado de modo a evitar condições de corrida que conduzem à perda de informação, designa-se habitualmente de região crítica.

Propriedades desejáveis que a solução geral do problema deve assumir:

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

4- Não havendo exclusão mútua no acesso a uma região crítica, corre-se o risco de inconsistência de informação devido à existência de *condições de corrida* na manipulação dos dados internos a um recurso, ou a uma região partilhada. O que são condições de

corrida? Exemplifique a sua ocorrência numa situação simples em que coexistem dois processos que cooperam entre si.

À situação em que vários processos acedem e manipulam dados partilhados duma forma "simultânea", deixando os dados num estado de possível inconsistência chama-se condições de corrida. Por exemplo, a interrupção dum processo durante a actualização duma estrutura de dados pode deixa-la num estado de inconsistência.

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

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

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

- ***soluções software*** – são soluções que, quer sejam implementadas num monoprocessador, quer num multiprocessador com memória partilhada, supõem o recurso em última instância ao *conjunto de instruções básico* do processador; ou seja, as instruções de transferência de dados de e para a memória são de tipo *standard*: leitura e escrita de um valor; a única suposição adicional diz respeito ao caso do multiprocessador, em que a tentativa de acesso simultâneo a uma mesma posição de memória por parte de diferentes processadores é necessariamente serializada por intervenção de um árbitro;
- ***soluções hardware*** – são soluções que supõem o recurso a instruções especiais do processador para garantir, a algum nível, a atomicidade na leitura e subsequente escrita de uma mesma posição de memória; são muitas vezes suportadas pelo próprio sistema de operação e podem mesmo estar integradas na linguagem de programação utilizada;

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

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

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

O *algoritmo de* ***Peterson*** usa a seriação por ordem de chegada para resolver o conflito resultante da situação de contenção de dois processos. Isto é conseguido forçando cada processo a escrever a sua identificação na mesma variável. Assim, uma leitura subsequente permite por comparação determinar qual foi o último que aí escreveu. De novo, não é muito complicado demonstrar que este algoritmo garante efectivamente a exclusão mútua no acesso à região crítica, evita *deadlock* e *adiamento indefinido* e não presume o que quer que seja quanto à velocidade de execução relativa dos processos intervenientes.

A generalização a N processos do algoritmo de Dekker não é fácil. De facto, não se conhece qualquer algoritmo que o faça, respeitando todas as propriedades desejáveis para a solução.

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?

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.

Um problema comum às soluções *software* e ao uso de *flags de locking*, implementadas a partir de instruções de tipo *read-modify-write*, é que os processos intervenientes aguardam entrada na região crítica no estado activo – *busy waiting*.

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

- *perda a eficiência* – a atribuição do processador a um processo que pretende acesso a uma região crítica, associada a um recurso ou uma região partilhada em que um segundo processo se encontra na altura no seu interior, faz com que o intervalo de tempo de atribuição do processador se esgote sem que qualquer trabalho útil tenha sido realizado;

- *constrangimentos no estabelecimento do algoritmo de scheduling* – numa política *preemptive* de *scheduling* onde os processos que competem por um mesmo recurso, ou partilham uma mesma região de dados, têm prioridades diferentes, existe o risco de *deadlock* se for possível ao processo de mais alta prioridade interromper a execução do outro;

Quando a execução do código das regiões críticas tem uma duração relativamente curta, a alternativa de bloquear o processo enquanto aguarda uma oportunidade de acesso à região crítica, exige uma *mudança de contexto* para calendarizar outro processo para execução nesse processador e pode tornar-se, por isso, menos eficiente.

É neste contexto que as *flags de locking* se tornam importantes, sendo também referidas pelo nome de *spinlocks* (o processo *gira* em torno da variável enquanto aguarda o *locking*).

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.

São construções que garantem a leitura e escrita de uma variável de forma atómica (sem interrupção). Ao utilizar este tipos de flags, o primeiro processo a tentar entrar na região critica, isto é, o primeiro a encontrar a flag de locking irá ser o primeiro a entrar na região critica, evitando que dois processos possam entrar em simultâneo nesta.

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

Um *semáforo* é um dispositivo de sincronização, originalmente inventado por Dijkstra, que pode ser concebido como uma variável do tipo:

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

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

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

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

Para um processo aceder a uma região critica é necessário fazer um down do semáforo de acesso (estando este inicialmente a 1), deste modo nenhum outro processo poderá entrar na região critica pois caso faça outro down do semáforo de acesso este irá bloquear o processo. Deste modo é garantida a sincronização dos processos entre si pois quando o processo que está na região critica sair irá fazer um up do semáforo acesso permitindo a um processo que esteja bloqueado, executar.

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

Um *monitor* é um dispositivo de sincronização, 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.

Quando as estruturas de dados são implementadas com monitores, a linguagem de programação garante que a execução de uma primitiva do monitor é feita em regime de exclusão mútua. Assim, o compilador, ao compilar um monitor, gera o código necessário para impor esta situação. Um thread entra no monitor por invocação de uma das suas primitivas, o que constitui a única forma de acesso à estrutura de dados interna. Como a execução das primitivas decorre em regime de exclusão mútua, quando um outro thread está no seu interior, o thread é bloqueado à entrada, aguardando a sua vez. A sincronização entre threads é gerida pelas variáveis de condição. Existem duas operações que podem ser executadas sobre uma variável de condição wait – o thread que invoca a operação é bloqueado na variável de condição argumento 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 argumento, um deles é acordado; caso contrário, nada acontece.

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

Para impedir a coexistência de dois *threads* dentro do *monitor*, é necessária uma regra que estipule como a contenção decorrente do *signal* é resolvida

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

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

monitor de Lampson / Redel*l* – 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 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?

Conceptualmente, o principal problema com o uso dos *semáforos* é que eles servem simultaneamente para garantir o acesso com exclusão mútua a uma região crítica e para sincronizar os processos intervenientes. Assim, e porque se trata de primitivas de muito baixo nível, a sua aplicação é feita segundo uma perspectiva bottom-up (os processos são bloqueados antes de entrarem na região crítica, se as condições à sua continuação não estiverem reunidas) e não top-down (os processos entram na região crítica e bloqueiam, se as condições à sua continuação não estiverem reunidas).

A primeira abordagem torna-se logicamente confusa e muito sujeita a erros, sobretudo em interacções de alguma complexidade, porque as primitivas de sincronização podem estar dispersas por todo o programa. Uma solução é introduzir ao nível da própria linguagem de programação uma construção [concorrente] que trate separadamente o acesso com exclusão mútua a uma dada região de código e a sincronização dos processos.

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?

Uma forma alternativa de comunicação entre processos é através da *troca de mensagens*. Trata-se de um mecanismo absolutamente geral. Não exigindo partilha do espaço de endereçamento, a sua aplicação, de um modo mais ou menos uniforme, é igualmente válida tanto em ambientes monoprocessador, como em ambientes multiprocessador, ou de processamento distribuído.

O princípio em que se baseia é muito simples:

- sempre que um processo PR, dito *remetente*, pretende comunicar com um processo PD,

dito *destinatário*, envia-lhe uma mensagem através de um canal de comunicação estabelecido entre ambos (*operação de envio*);
- o processo PD, para receber a mensagem, tem tão só que aceder ao canal de comunicação e aguardar a sua chegada (*operação de recepção*).
A mensagem apenas está na posse de um processo de cada vez logo o problema do acesso com exclusão mútua a uma região crítica está implicitamente resolvido.

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?

Uma forma alternativa de comunicação entre processos é através da *troca de mensagens*. Trata-se de um mecanismo absolutamente geral. Não exigindo partilha do espaço de endereçamento, a sua aplicação, de um modo mais ou menos uniforme, é igualmente válida tanto em ambientes monoprocessador, como em ambientes multiprocessador, ou de processamento distribuído. O princípio em que se baseia é muito simples:
• sempre que um processo PR, dito *remetente*, pretende comunicar com um processo PD, dito *destinatário*, envia-lhe uma mensagem através de um canal de comunicação estabelecido entre ambos (*operação de envio*);
• o processo PD, para receber a mensagem, tem tão só que aceder ao canal de comunicação e aguardar a sua chegada (*operação de recepção*).
A mensagem apenas está na posse de um processo de cada vez logo o problema do acesso com exclusão mútua a uma região crítica está implicitamente resolvido.

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.

18- Que tipo de mecanismos de sincronização podem ser introduzidos nas primitivas de comunicação por passagem de mensagens? Caracterize os protótipos das funções em cada caso (suponha que são escritas em Linguagem C).
A *troca de mensagens* só conduzirá, porém, a uma comunicação fiável entre os processos *remetente* e *destinatário*, se for garantida alguma forma de sincronização entre eles.
O grau de sincronização existente pode ser classificado em dois níveis
- ***sincronização não bloqueante*** - quando a sincronização é da responsabilidade dos processos intervenientes; a *operação de envio* envia a mensagem e regressa sem qualquer informação sobre se a mensagem foi efectivamente recebida; a *operação de recepção*, por seu lado, regressa independentemente de ter sido ou não recebida uma mensagem;

```
/* operação de envio */
void msg_send_nb (unsigned int destid, MESSAGE msg);
/* operação de recepção */
void msg_receive_nb (unsigned int srcid, MESSAGE *msg,
BOOLEAN *msg_arrival);
```

- ***sincronização bloqueante*** - quando as operações de envio e de recepção contêm em si mesmas elementos de sincronização; a *operação de envio* envia a mensagem e bloqueia até que esta seja efectivamente recebida; a *operação de recepção*, por seu lado, só regressa quando uma mensagem tiver sido recebida;

```
/* operação de envio */
void msg_send (unsigned int destid, MESSAGE msg);
/* operação de recepção */
void msg_receive (unsigned int srcid, MESSAGE *msg);
```

19- O que são sinais? O *standard* Posix estabelece que o significado de dois deles é mantido em aberto para que o programador de aplicações lhes possa atribuir um papel específico. Mostre como poderia garantir o acesso a uma região crítica com exclusão mútua por parte de dois processos usando um deles.

Um *sinal* constitui uma interrupção produzida no contexto de um processo onde lhe é comunicada a ocorrência de um acontecimento especial. Pode ser despoletado pelo *kernel*, em resultado de situações de erro ao nível hardware ou software, pelo próprio processo, por outro processo, ou pelo utilizador através do dispositivo *standard* de entrada / saída. Tal como o processador no tratamento de excepções, o processo assume uma de três atitudes possíveis relativamente a um sinal:

- ***ignorá-lo*** – não fazer nada face à sua ocorrência;
- ***bloqueá-lo*** – impedir que interrompa o processo durante intervalos de processamento bem definidos;
- ***executar uma acção associada*** – pode ser a acção estabelecida por defeito quando o processo é criado (conduz habitualmente à sua terminação ou suspensão de execução), ou uma acção específica que é introduzida (*registada*) pelo próprio processo em *runtime*.

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

Temos duas soluções, pode-se usar 1 ou 2 pipes. É de salientar que um pipe se assemelha a um canal em que se coloca informação de um lado e se lê no outro.

Se usarmos 1 pipe o processo P1 redirecciona o stdout para a entrada do pipe e o processo P2 redirecciona o stdin para a saída do pipe. Desta forma temos comunicação de P1 para P2, para P2 comunicar com P1, o processo P2 redirecciona o sdtou para a entrada do pipe e o processo P1 redirecciona o stdin para a saída do pipe. Como é obvio o acesso ao pipe tem de ser feito em alternância.

Temos outra solução, que envolve dois pipes, assim temos o pipe1 e o pipe2. O pocesso P1 tem de redireccionar o stdin para a saída do pipe2 e o stdout para a entrada do pipe1. Com efeito, o processo P2 redirecciona o stdin para a saída do pipe1 e o stdou para a entrada do pipe2.

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?

Genericamente, um *recurso* é algo que um processo precisa para a sua execução. Os recursos tanto podem ser *componentes físicos do sistema computacional* (processadores, regiões de memória principal ou de memória de massa, dispositivos concretos de entrada / saída, etc), como *estruturas de dados comuns* definidas ao nível do sistema de operação (tabela de controlo de processos, canais de comunicação, etc), ou entre processos de uma mesma aplicação. Uma propriedade essencial dos recursos é o tipo de apropriação que os processos fazem deles. Nestes termos, os recursos dividem-se em

• *recursos preemptable* – quando podem ser retirados aos processos que os detêm, sem que daí resulte qualquer consequência irreparável à boa execução dos processos; são, por exemplo, em ambientes multiprogramados, o processador, ou as regiões de memória principal onde o espaço de endereçamento de um processo está alojado;

• *recursos non-preemptable* – em caso contrário; são, por exemplo, a impressora, ou uma estrutura de dados partilhada que exige exclusão mútua para a sua manipulação.

O canal de comunicações, uma impressora e a memória de massa são do tipo *recursos non-preemptable*

22- Distinga as diferentes políticas de *prevenção de deadlock no sentido estrito*. Dê um exemplo ilustrativo de cada uma delas numa situação em que um grupo de processos usa um conjunto de blocos de disco para armazenamento temporário de informação.

As políticas de *prevenção de deadlock no* ***sentido estrito***, embora seguras, são muito restritivas, pouco eficientes e difíceis de aplicar em situações muito gerais. Resumindo, tem-se que:

• ***negação da condição de exclusão mútua*** – só pode ser aplicada a recursos passíveis de partilha em simultâneo;
• ***negação da condição de espera com retenção*** – exige o conhecimento prévio de todos os recursos que vão ser necessários, considera sempre o pior caso possível (uso de todos os recursos em simultâneo);
• ***imposição da condição de não libertação*** – ao supor a libertação de todos os recursos anteriores quando o próximo não puder ser atribuído, atrasa a execução do processo de modo muitas vezes substancial;
• ***negação da condição de espera circular*** – desaproveita recursos eventualmente disponíveis que poderiam ser usados na continuação do processo.

23- Em que consiste o algoritmo dos banqueiros de Dijkstra? Dê um exemplo da sua aplicação na situação descrita na questão anterior

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?

Políticas de prevenção de deadlock no sentido lato é uma alternativa menos restritiva do que a *prevenção de deadlock no sentido estrito* é não negar *a priori* qualquer das condições necessárias à ocorrência de *deadlock*, mas monitorar continuamente o estado interno do sistema de modo a garantir que a sua evolução se faz apenas entre estados ditos *seguros*.

Define-se neste contexto *estado seguro* como uma qualquer distribuição dos recursos do sistema, livres ou atribuídos aos processos que coexistem, que possibilita a terminação de todos eles. Por oposição, um estado é *inseguro* se não for possível fazer-se uma tal afirmação sobre ele.

Convém notar o seguinte

• *é necessário o conhecimento completo de todos os recursos do sistema e cada processo tem que indicar à cabeça a lista de todos os recursos que vai precisar* – só assim se pode caracterizar um estado seguro;

• *um estado inseguro não é sinónimo de deadlock* – vai, contudo, considerar-se sempre o pior caso possível para garantir a sua não ocorrência.

25- Que tipos de custos estão envolvidos na implementação das políticas de *prevenção de deadlock nos sentidos estrito e lato*? Descreva-os com detalhe.

As políticas de *prevenção de deadlock no* ***sentido estrito***, embora seguras, são muito restritivas, pouco eficientes e difíceis de aplicar em situações muito gerais. Resumindo, tem-se que:
• ***negação da condição de exclusão mútua*** – só pode ser aplicada a recursos passíveis de partilha em simultâneo;
• ***negação da condição de espera com retenção*** – exige o conhecimento prévio de todos os recursos que vão ser necessários, considera sempre o pior caso possível (uso de todos os recursos em simultâneo);
• ***imposição da condição de não libertação*** – ao supor a libertação de todos os recursos anteriores quando o próximo não puder ser atribuído, atrasa a execução do processo de modo muitas vezes substancial;
• ***negação da condição de espera circular*** – desaproveita recursos eventualmente disponíveis que poderiam ser usados na continuação do processo.
Uma alternativa (*prevenção de deadlock no* ***sentido lato***) menos restritiva do que a *prevenção de deadlock no sentido estrito* é não negar *a priori* qualquer das condições necessárias à ocorrência de *deadlock*, mas monitorar continuamente o estado interno do sistema de modo a garantir que a sua evolução se faz apenas entre estados ditos *seguros*. Define-se-se neste contexto *estado seguro* como uma qualquer distribuição dos recursos do sistema, livres ou atribuídos aos processos que coexistem, que possibilita a terminação de todos eles. Por oposição, um estado é *inseguro* se não for possível fazer-se uma tal afirmação sobre ele. Políticas com esta característica designam-se de *políticas de prevenção de deadlock no sentido lato*.

Gestão de memória

1. **O que distingue fundamentalmente uma organização de memória virtual de uma organização de memória real? Qual é o papel desempenhado pela área de 'swapping' em cada uma delas?**

Numa memória real o espaço de endereçamento de um processo em execução é necessário estar totalmente em memória principal, numa memória virtual não é necessário conter todo o espaço de endereçamento do processo em execução na memória principal. Por este motivo para uma memória principal com o mesmo tamanho é possivel colocar mais processos em memória principal do que no caso de uma organização real, ou seja, na memória virtual só se têm na memória principal apenas as partes necessárias á execução do processo e não a sua totalidade, enquanto que todo o espaço de endereçagem é mantido na memória secundária (área de swapping), as partes não presentes na memória principal são carregadas á medida que forem necessárias á execução do processo.
No caso da memória virtual não existe limitação do espaço de endereçamento fisico do processo, no caso real existe limitação do tamanho do espaço de endereçamento devido à memória principal ser menor que a de massa.
Para implementar uma memória virtual é necessário os seguintes requesitos:
Suporte de hardware para a gestão da memória (MMU);
Software para movimentação de partes do processo entre a memória principal e a memória secundária.
A vantagem do uso de memória real em relação a uma virtual é quando temos sistemas de tempo real e e é necessário que a execução de um determinado processo seja rápido, se tivermos todo o processo em memória principal é mais rápido, pois não é necessário estar a aceder á área de swapping caso ocorra uma page fault. Um exemplo é o de monitorização da temperatura de uma caldeira que necessita de uma resposta rápida por parte do sistema.
A área de swapping no caso da memória real serve como arrecadação e existe uma correspondência de 1 para 1 do espaço de endereçamento do processo.
A área de swapping no caso da memória virtual serve para manter todo o espaço de endereçamento de um processo, esse espaço pode estar dividido em três tipos diferentes: paginada, segmentada e paginada segmentada.

2. **Qual é a diferença entre uma organização de memória virtual segmentada de uma paginada? Quais são as vantagens e inconvenientes de cada uma delas?**

Numa organização de memória virtual paginada, a dimensão do bloco base de armazenamento em memória principal é fixa, enquanto que numa organização de memória virtual segmentada a dimensão dos vários blocos é variavel.
Na paginada, a partição do espaço de endereçamento reflecte apenas aspectos estruturais da própria memória física e resulta de uma divisão automática em blocos de comprimento fixo realizada na linkagem.
Na segmentada, a partição do espaço está directamente associada à organização do programa (código, stack, variáveis, etc...) estabelecida pelo programador, cada segmento corresponde a um módulo distinto.
A Segmentada tem a vantagem de cada segmento poder crescer ou diminuir separadamente, facilita a partilha de procedimentos e dados entre processos. A desvantagem é que tal como as partiçoes de mem real dinamicas sofre de fragmentaçao externa.
A virtual tem a vantagem de não ser necessario colocar todo o bloco de código do processo em memória principal, basta a página correspondente ao código que está a ser necessário no momento. A desvantagem é que tal como as partiçoes de tamanho fixo sofre de fragm interna.

3. **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.**

Não é necessário colocar todo o espaço de endereçamento do processo em memória principal, porque só são colocadas as partes estritamente necessárias de cada processo (aumenta a taxa de utilização do processador).
É necessário existir hardware que faça a tranferencia dos endereços virtuais para os endereços físicos (MMU) e software para movimentação de partes do processo entre a memória principal e secundária.
Podem estar mais processos em memória principal do que no caso de memória real porque ao contrário da real em que existe uma relação de 1 para 1 no espaço de endereçamento do processo neste caso essa relação é de muitos para 1, como só são colocados os pedaços necessários à execução do processo é possivel colocar lá mais processos, dai se tira que a ocupação em memória principal é mais bem aproveitada.
Esta vantagem é fundamentada pelo príncipio da localidade de referencia ,ou seja, as instruções de um processo e respectivos dados tendem a concentrar-se no tempo e no espaço.

Por exemplo, para a mesma memória principal, no caso em da real se tivermos dois processos cuja soma do espaço de endereçamento seja superior ao espaço físico da memória, sempre que quisermos efectuar a troca de um para o outro é necessário aceder á área de swapping, no caso da memória virtual já é possivel conter os pedaços estritamente necessários de cada processo carregados na memória principal não sendo necessário aceder á área de swapping para fazer a comutação de processos (só é necessário aceder á área de swapping quando ocorre uma page fault ou o conteudo da página em memória principal ter sido modificado e ser necessário salvaguarda-lo).
Concluindo, um sistema de memória virtual permite a multiprogramação de forma mais eficaz e liberta o programador da limitação da memória física.

Gestão de dispositivos de entrada-saída

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

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

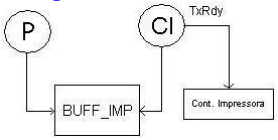
2. **Que papel desempenham os 'device-drivers' no estabelecimento de um interface uniforme de comunicação com os dispositivos de entrada-saída?**

Os device drives (controlador) permitem criar uma interface abstracta com os dispositivos de I/O e a construção de software independente do dispositivo, que implementam operações comuns a todos os dispositivos de I/O. Obtem-se assim uma plataforma uniforme para o software dos utilizadores. A função de um device driver consiste pois, em aceitar pedidos genéricos do software, por exemplo ler/escrever um bloco, e providenciar a sua execução pelo dispositivo.
Os device drives servem para independentemente do dispositivo de entrada/saida ele possa ser acedido através de funções pré-estabelecidas. Os device drives tiram o peso ao programador de como funciona o dispositivo e possa usar funções já conhecidas.

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

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

Dispositivos de Caracteres: estes dispositivos processam a informação em grupos de caracteres, sem ter em atenção alguma estrutura de tipo bloco. Estes dispositivos não são endereçáveis e não permitem operações de pesquisa. As impressoras, os terminais de vídeo e os teclados são dispositivos deste tipo.
Os de caracter transmitem informação com mensagens com o tamanho fixo de um byte e ainformação não pode ser pesquisada, os dispositivos de bloco transmitem a informação com mensagens com tamanho de um bloco com vários bytes e é possivel aceder a um bloco independentemente dos outros, ou seja, premite a pesquisa indexada.
A figura abaixo descreve de uma maneira esquemática a situação proposta.



O processo P é o processo utilizador que em momentos particulares da sua execução pretende escrever na impressora linhas de texto. Para isso, socorre-se da chamada ao sistema

- void escreve_linha (char *linha)

que transfere para o buffer de comunicação BUFF_INP a linha em questão.
O processo CI é um processo de sistema activado pela interrupção TxRdy do registo de transmissão do controlador da impressora. Quando é acordado, este processo procura ler um carácter, pertencente a uma linha armazenada em BUFF_IMP, e escrevê-lo no registo de transmissão da imperssora. Para isso socorre-se das duas chamadas ao sistema seguintes:

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

Em termos gerais, a interacção estre dois processos pode ser descrita da seguinte forma: Sempre que tiver um alinha para imprimir, o processo P invoca o procedimento *escreve_linha* que transfere para o buffer de

comunicação BUFF_IMP a linha em questão. O seu regresso é imediato, desde que haja espaço de armazenamento suficiente, caso contrário, o processo bloqueia. Na situação particular em que o buffer de comunicação esteja inicialmente vazio, o procedimento *escreve_linha* realiza a acção suplementar de transferir o primeiro carácter para o registo de transmissão do controlador da impressora, procedimento *transm_caracter*, despontando assim o mecanismo de interrupções que vai acordar sucessivamente o processo CI, até que toda a informação tenha sido efectivamente transferida.
Por sua vez o processo CI, quando é acordado pela interrupção, invoca a função *le_caracter* para recolher um carácter previamente armazenado em BUFF_IMP e , caso exista algum escreve-o no registo de transmissão do controlador da impressora, procedimento *transm_caracter*. Na situação particular em que o buffer de comunicação esteja inicialmente cheio a função *le_caracter* realiza a acção suplementar de tentar acordar o processo P eventualmente bloqueado a aguardar espaço de armazenamento disponível no buffer.

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

Utilização da DMA.
Politicas de limpeza de páginas em memória principal
O uso de dois buffers, um para páginas modificadas e delas não modificadas.

Gestão da memória de massa

1. **Porque é que sobre a abstracção da memória de massa, concebida como um 'array' de blocos para armazenamento de dados, se torna fundamental introduzir o conceito de sistema de ficheiros? Quais são os elementos essenciais que definem a sua arquitectura?**

O sistema de ficheiros permite abstrair o utilizador da organização da memória de massa, fornecendo-lhe apenas algumas funções de acesso aos ficheiros. (leitura, escrita, criação, eliminação, truncar, procura)
Um sistema de ficheiros é composto por duas partes distintas: ficheiros e directórios (um directório é um ficheiro com atributos especiais).
Os ficheiros contém os dados, os programas e toda a informação que se pertende armazenar.
Os directórios servem para estruturar a organização dos ficheiros.
Um ficheiro é uma forma de guardar dados de uma forma organizada.
Um ficheiro é caracterizado pelos seguintes atributos:

- Nome: Nome simbólico para o utilizador reconhecer o ficheiro
- Tipo: Se o ficheiro é do tipo objecto, executável, código fonte, batch, texto, arquivos, etc.
- Localização: É um ponteiro para o dispositivo onde o ficheiro se encontra e a localização dentro desse dispositivo.
- Tamanho: O tamanho do ficheiro (em bytes, palavras ou blocos), e possivelmente o tamanho máximo permitido.
- Protecção: Informação de acesso ao ficheiro por parte dos diversos utilizadores.
- Tempo, data e dono: Data de criação, modificação e utilização.

2. **Em que consiste o problema da consistência da informação num sistema de ficheiros? Que razões levam à sua ocorrência? Como é que ele pode ser minimizado?**

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

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

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