Sugestões de resposta às questões de revisão da componente teórica

Sistemas Operativos 2016/17

V2.0

Atenção: Estas sugestões de resposta foram feitas por mim (enquanto aluno) no contexto de estudo e revisão e por esse mesmo motivo não substituem de todo uma pesquisa mais aprofundada seja à bibliografia sugerida, apontamentos das aulas etc.
Respostas que tenham um NS à frente (são poucas mas há) são respostas que não foram respondidas com 100% de certeza. Todas as outras, tentaram ser, dentro do possível mas claro, poderão existir pequenos enganos. Uma vez mais, trata-se de um resumo breve e não pode ser tomado como informação aprofundada do conteúdo em questão referido pelas perguntas.

As questões são retiradas dos seguintes 3 pdfs: conceitos fundamentais de sistemas operativos, escalonamento - conceitos e algoritmos e gestão de memória e teoria virtual das teóricas de SO de 2016/17 e algumas pesquisas feitas na net assim como do livro fundamentos de sistemas operativos dos autores José Alves Marques e Paulo Guedes (que recomendo bastante!).

Quem quiser usar como ferramenta de suporte ao estudo está à vontade.

Bom estudo!
Miguel Ruivo

Índice

FUNDAMENTOS DE SISTEMAS OPERATIVOS

1. Afinal o que é um sistema operativo?

Um sistema operativo engloba um conjunto de programas e serve de intermediário entre o equipamento (máquina) e o utilizador. Tudo o que é gestão da máquina é realizada pelo SO, desde gestão dos recursos de hardware à implementação de políticas de otimização. É também o SO que disponibiliza ao utilizador (humano) os recursos para uso da máquina.

2. Que pontos de vista diferentes existem quanto ao que é ou deixa de ser um sistema operativo?

Podemos dizer que estamos perante um SO quando temos um sistema que é capaz de gerir dinamicamente os recursos do mesmo servindo de elo de ligação entre o hardware e o software proporcionando ao utilizador da máquina uma experiência o mais intuitiva possível.

3. Que entidades interagem com o SO?

As interações com o SO são feitas por parte do utilizador (requisita serviços ao alto nível), pelas aplicações (requisita serviços e recursos via API) e pelo hardware (através da interface de hardware, notifica acontecimentos e receção de E/S assim como envio de dados e instruções para os dispositivos).

4. Em que condições um programa (uma sequência de instruções qualquer) pode aceder ao hardware?

A maneira como os programas (processos) têm acesso ao hardware são através das chamadas ao sistema. Estas chamadas têm como objetivo requisitar recursos ou serviços ao SO através da API (funções de sistema).

5. O utilizador ser humano interage diretamente com o SO? De que forma?

O utilizador apercebe-se da existência do sistema de uma maneira indireta, de certo modo, trata-se de uma entidade que faz com que a máquina funcione, ou seja, através de alguma interface (explorer, bash, etc.) esta interface é considerada externa ao sistema em si e tem como objetivo facilitar a utilização do SO mesmo por utilizadores inexperientes.

6. Como é que o hardware interage com o SO?

O hardware interage com o sistema apenas e não diretamente com o utilizador ou programas. O sistema é visto como a única entidade "gestora" responsável pela máquina. O SO é responsável por configurar adequadamente o hardware quando é feito o arranque do sistema, controlar o hardware validando as operações efetuadas sobre ele e efetuando correções de erros assim como receber notificações por parte do hardware e tratar das mesmas (geralmente através dos device drivers).

7. Quais os objetivos de um SO moderno?

O objetivo fulcral de um SO é conseguir garantir um ambiente e ligação utilizador -> programa -> hardware de forma mais transparente e retro compatível possível. Numa situação ideal os programadores teriam a capacidade de desenvolver as suas aplicações preocupando-se apenas com o seu código fonte pois saberiam que qualquer que fosse o SO iria ser capaz de criar o elo de ligação entre o programa e hadware da máquina. É quase assim, mas ainda hoje é necessário ter em conta que diferentes sistemas têm diferentes API logo poderá ser necessário recompilar para cada sistema. Do ponto de vista de utilizador o objetivo é criar o ambiente mais limpo e user-friendly ao mesmo para que com o menor dos conhecimentos possa realizar cada vez mais segundo as suas necessidades.

8. O que é um API?

O API ou Application Programming Interface, de uma forma simples e óbvia, são as funçoes que cada Sistema tem e que os programadores poderão tirar partido quando estão a desenvolver as suas aplicações.

9. Como é que programas-utilizador utilizam os recursos do Sistema operativo?

Os programas utilizam recursos do sistema operativo através das chamadas ao sitema como já foi visto, e o utilizador através de uma interface que lhe dá acesso aos serviços e ferramentas necessárias para a execução das suas tarefas.

10. O que é o isolamento de processos?

O isolamento do processo consiste em cada processo ter uma visão da máquina como sendo um todo para si mesma, isto é, como se de uma máquina virtual se tratasse (para cada processo) tendo cada um deles o seu respetivo espaço.

11. O que é isso de abstrair o hardware? Dê exemplos.

A abstração do hardware dá-se na camada entre o hardware (físico) e o software do computador. Tem como objetivo fazer o que o nome indica, abstrair. Ou seja, é a designação que se dá ao fato de um programa feito ter a capacidade de se "ligar" e tirar partido do hardware sem que tal se aperceba ou tenha sido configurado para o fazer de forma direta. Isto acontece porque

o SO faz o elo de ligação com o recurso à API do mesmo. Como exemplo, pode-se desenvolver uma aplicação para um sistema e tal aplicação correr sem qualquer problema noutro sem ter sido novamente programada. Isso deve-se a uma boa abstração de hardware.

12. O que impede um programa ou utilizador de tomar controlo de uma máquina?

Uma vez que os programas são geridos pelo SO e o utilizador só pode ter acesso aquilo que, de certo modo, o SO "deixa", acaba por se manter a integridade do mesmo. É sabido como exemplo que um processo (programa) tem o seu espaço de endereçamento disponível, ainda que este veja o sistema como um todo disponível só para ele, se por mero acaso tentar aceder a partes que não lhe são permitidas é gerada uma exceção e por norma o programa termina.

13. Que tipos de sistema existem?

- Existem os sistemas com base no tempo da máquina: De tempo virtual ou tempo real. Nos primeiros o tempo real não é importante para o computador, no segundo já é e objectivo é garantir que as tarefas sejam cumpridas em intervalos de tempo menores ou pre-determinados.
- Com base no numero de processadores e natureza da sua utilização: Sistemas paralelos ou distribuídos.

INTERRUPÇÕES

1. Como é que os programas-utilizador encontram rotinas do sistema operativo?

Através de chamadas ao sistema.

2. O que é o núcleo? Em que difere do modo de utilizador?

O núcleo (ex: kernel) constitui a camada mais baixa a seguir ao hardware de uma máquina. A diferença do núcleo para o utilizador é que o núcleo deverá ter a prioridade máxima de acesso quando executa certas rotinas servindo ele de "intermediário" entre as camadas superiores e o hardware (de uma forma simplificada falando).

3. É possível implementar um Linux ou Windows moderno num 8086? Porquê?

Um Linux moderno ou Windows (só se não forem sistemas de arquitetura de 32 ou 64 bits que só por aí já não seria possível uma vez que o 8086 é de 16 bits), não poderão ser implementados num 8086 pois a maioria dos sistemas atuais necessitam de uma quantidade de RAM que o 8086 não consegue endereçar (máximo que consegue endereçar são 20 bits (1 MB de RAM)).

4. O que é uma trap e para que serve?

Uma trap é um tipo de interrupção pedida pelo próprio sistema, por exemplo nos x86 o INT 21H que provoca o inicio da rotina 21H (função), é análogo a um CALL com a diferença de que não é preciso conhecer o endereço para onde se está a saltar e sim apenas o numero da interrupção pois o endereço de destino está na tabela de interrupções.

5. Como é que o sistema deteta que um programa está a tentar usar um endereço de memória inválido?

Um SO deteta que um programa está a tentar aceder a uma posição inválida de memória (entenda-se que por inválida será ou uma posição que não existe ou não lhe "pertence"), gerando uma excepção. As excepções são análogas às interrupções e às traps contudo não são desencadeadas a pedido do programa mas sim pelo próprio processador.

6. Como é que o controlador de disco rígido (por exemplo) avisa o sistema de que ocorreu um erro?

Através de uma interrupção um periférico pode avisar o SO de ocorreu um erro. O controlador do dispositivo ativa uma linha de IRQ, o processador interrompe o curso normal de execução e executa uma rotina para o serviço de interrupções, ou seja, vai à tabela de interrupções "ver" qual foi e desencadeia a passagem ao estado executável dos processos bloqueados à espera de conclusão de E/S. Esta rotina deve residir no núcleo.

7. O que é uma função de sistema?

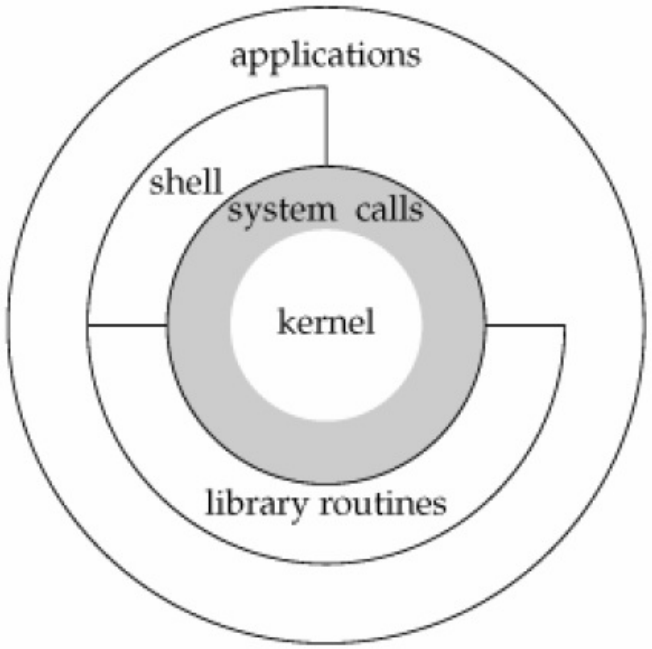
São as chamadas ao sistema que no fundo são a única forma de um processo requisitar serviços do sistema. Isto passa por passar a execução do código dentro do processo para o código do sistema.

8. Qual a diferença entre função sistema e função biblioteca?

As primeiras são facultadas pelo kernel (núcleo) enquanto que as segundas são facultadas pela biblioteca que as detém. Uma imagem ajuda a perceber as diferenças.

10. Dê exemplo de uma função biblioteca.

printf(), scanf() são duas funções biblioteca (stdio.h) a que todos devem estar habituados.



11.Dê exemplo de uma função sistema.

open() por exemplo é uma função de sistema enquanto que fopen() é uma função de biblioteca (stdio.h).

12. O que é a fase de linkagem e para que serve (o que faz)?

De uma forma simples, o linker é o que "pega" em objetos gerados pelo compilador e os junta todos criando o executável. É na fase de linkagem que isso é realizado.

13. O que era o int 21h do MS-DOS? Era importante? Porquê?

Era uma chamada ao sistema do MS-DOS só que o problema é que no MS-DOS tinha pouca ou nenhuma protecção e era facilmente modificada. Isto devia-se a não haver restrições por privilégios.

EVOLUÇÃO, TIPO, ESTRUTURA E ARQUITECTURA DO SO, CARACTERÍSTICAS E VANTAGENS DE CADA ARQUITECTURA

1. O que é o núcleo? Como pode ser identificado facilmente?

Constituem o núcleo do SO o conjunto de funcionalidades (ou camadas) do sistema onde estão as funções mais importantes e mais privilegiadas. O

nuleo deve ser sempre executado no modo mais privilegiado disponível da máquina. Estas funções agrupam-se deste modo pois existe necessidade de executar funções que fazem gestão de hardware, proteção quando a interferências do resto do sistema e execução prioritária (em resposta a acontecimentos internos ao sistema ou gerados pelo hardware).

2. O núcleo corresponde ao sistema operativo? Justifique e dê exemplos.

Sim. O núcleo é, por assim dizer, a camada mais importante do Sistema operativo. Como exemplo pode-se dizer que o núcleo é responsável pela gestão de processos e de memória da máquina.

3. Quais as arquiteturas de SO que conhece? Descreva-as.

Dois tipos de arquiteturas conhecidas de SO são:

1. **Sistemas Monolíticos:** Vantagem de ser ligeiramente mais rápidos (dependendo da aplicação) mas não são estruturados (conhecidos como Big Mess) e têm dificuldade em isolar as responsabilidades do sistema (ex: Unix).

2. **Sistemas em camadas:** Como o nome indicia as responsabilidades são divididas em camadas desde a mais baixa para a mais alta (gestão de processos, gestão de memória, comunicação E/S de dispositivos, sistema de ficheiros, interface do sistema (comandos e funções de sistema e por ultimo aplicações). É mais fácil de interpretar esta arquitetura, contudo, nem sempre se pode dizer que é 100% a mais correta pois é difícil consolidar como devem estar estruturadas as camadas.
3. **Máquina virtual:** Uma máquina virtual tenta simular uma máquina real como se uma cópia da mesma se tratasse.

4. **Sistemas cliente-servidor:** Também chamados de micro-kernel. Consiste em atribuir ao núcleo uma tarefa simplificada e deixando o resto para o "servidor" ou seja, para as camadas ao nível do utilizador. Pretende-se reduzir aqui as funções do sistema operacional (ex: WindowsNT).

4. Qual é o melhor? Micro-kernel ou kernel monolítico?

Bem, vai depender muito do ponto de vista e ambos têm as suas vantagens de aplicação. Por um lado, num sistema micro-kernel (ex: Hurd e Minix) é estruturado o sistema de um modo em que o kernel apenas trata de gerenciar os processos, deixando tudo o resto para o "user-space" através de servidores. No caso do sistema monolítico, como foi dito anteriormente é tudo centralizado no espaço do kernel. Funções que no micro-kernel estão descentralizadas para o user space, aqui, são tratadas no espaço do kernel ainda em que módulos distintos (ex: Linux, Windows). A discussão entre qual

é melhor será sempre um bocado subjectiva uma vez que o micro-kernel oferece mais segurança e uso mínimo do kernel apenas para aquilo que é "essencial" mas perdendo para a performance por exemplo do kernel-monolítico. Atualmente sistemas operativos (Apple e Microsoft por ex) dispõe de kernel-hibridos que tentam tirar partido da rapidez do monolítico aliada à fiabilidade e elegância de organização do micro-kernel.

5. Indique exemplos de aplicação em sistemas de máquinas virtuais.

Uma aplicação óbvia será no desenvolvimento por exemplo de sistemas operativos. O fato de estarmos a trabalhar com uma máquina virtual não consegue (nem pode) prejudicar o sistema atual onde a virtualização está a decorrer.

6. Qual a diferença entre um sistema de tempo virtual e um sistema de tempo real?

- Sistema de tempo real (STR): O sistema tem um tempo predefinido para responder e atuar, caso por algum motivo exista algum atraso na resposta então ocorre uma falha no sistema. Não tem em consideração se a velocidade de resposta é rápida ou não, mas sim o tempo que está estipulado. O STR tem que reagir, dentro do tempo estipulado ao meio em que atua.

- Sistema de tempo virtual (STV): Sistema em que o tempo real não importa ou é relevante.

7. Qual a diferença principal entre um sistema paralelo ou um distribuído?

Um sistema distribuído consiste na sub-divisão em nódulos de tarefas computacionais divididas por várias máquinas. Ou seja, é como se de uma máquina só se tratasse mas no fundo estamos a usar o poder de várias maquinas divididas atribuindo a cada uma delas uma tarefa. Útil quando é necessário muito poder de processamento podendo assim montar-se uma rede de máquinas (até utilizadores domésticos incluídos). Um sistema paralelo é focado na realização de múltiplas tarefas o mais rápido possível (multi-threading).

8. Aplicaria um sistema de tempo virtual para controlar um sistema de direção de um míssil?

Não. Num sistema de direção de um míssil pretende-se (e convém) que seja de resposta imediata a um estimulo externo. Não se obterá um resultado eficaz com um sistema de tempo virtual ou partilhado por exemplo em que a rotina que esteja a ser executada possa ter que terminar primeiro antes do estimulo externo (desviar a direção por exemplo) entre em ação. Geralmente situações destas, ou mesmo máquinas de saúde, emergência entre outras

trabalham com sistemas de tempo real pois a resposta tem que ser sempre a mais imediata possível ao estimulo externo adquirido.

PROCESSOS E PROGRAMAS

1. Qual a diferença entre um processo e um programa?

- Um programa é uma sequência de instruções armazenadas de uma forma passiva e sem qualquer concretização, é apenas uma ideia de como executar algo pretendido.
- Um processo é a entidade ativa que partilha o processador com outros processos. Correspondem a realizar instruções do programador (que estão no programa). Os escalonamentos são aplicados sempre aos processos. Os processos ainda se podem subdividir em nós (threads) que atuam como sub-processos desse processo mas que partilham o mesmo espaço de endereçamento e descritores do processo mãe.

2. Em que circunstancias um processo pode não estar associado a um programa?

Em nenhuma. O processo é sempre a entidade que corre o conjunto de instruções de um programa a ser executado.

3. Um programa pode estar a ser executado em vários processos?

Sim. A menos que o programador por algum motivo impeça a existência de várias instancias desse mesmo programa, um programa poderá ter múltiplas instancias em distintos processos independentes.

4. Um processo pode estar a executar vários programas?

Não. Um processo diz sempre respeito a um programa a ser executado.

5. Descreva as zonas típicas de memória típicas de um processo. Para que serve cada uma?

i. **Data Segment (DS):** responsável pelos dados inicializados com o executável (estáticos ou globais).

ii. **Code Segment (CS):** contém as instruções aritméticas a serem executadas pelo processador.

iii. **Stack (Pilha) (SS):** Espaço onde são colocadas as variáveis locais e endereços de retorno das chamadas. Diminui em runtime.

iv. **Heap:** Espaço reservado para alocação dinâmica de memória durante o runtime. Tem um limite.

6. Qual a diferença entre processo e thread?

Um processo constitui uma entidade que executa um conjunto de instruções de um determinado programa. Uma thread é um nó pertencente a um processo (poderá haver N threads por processo) que partilha o mesmo espaço e zonas de memória (especificadas acima) que o processo herdado. Geralmente a uma thread atribui-se uma tarefa especifica (função) do processo principal.

7. É possível um processo ter zero threads?

É possível um processo não ter nenhuma thread "a mais" além do processo principal ele mesmo. Um processo poderá ser visto como uma thread também, mas para uma resposta clara a esta questão, sim, é possível não haver threads assumindo que por thread referimos apenas aos nós criados para um processo.

8. Dê um exemplo onde seja útil ter mais que uma thread.

Num programa em que seja necessário estarem 2 tarefas (ou mais) em simultâneo a correr se estivermos limitados apenas ao processo principal e existir uma rotina de I/O por ex, um scanf, em que o processo fique bloqueado à espera do input, se existir outra tarefa (função) desse mesmo processo que fosse necessária estar a correr em pseudo-paralelismo, sem uma thread isso não será possível. Com uma thread dedicada para a função (vamos supor que imprime apenas a palavra "ola" no ecrã 100x), se surgir um scanf pelo meio esta thread irá continuar a imprimir a palavra "ola" sem ser interrompida.

EVOLUÇÃO DOS SISTEMAS OPERATIVOS E DO ESCALONAMENTO

1. Inicialmente não fazia sentido falar em SO porque? (1st gen)

Inicialmente não havia sequer SO porque não havia a mínima necessidade de gestão da máquina como a que há nos dias de hoje. Os computadores antigos (pegando na máquina de alan turing como referência usada na segunda guerra mundial) eram criados com um propósito e quando muito eram ajustados certos parâmetros manualmente. Não dispunham sequer de dispositivo de monitorização (controlo de parâmetros). As máquinas não dispõem de softwares porque são criadas (os circuitos) para um fim em particular. Mais tarde surgiram máquinas que executavam software, mas eram bastante simples e limitadas, o monitor das mesmas servia apenas para

gerir carga de instruções. Nem sequer fazia sentido o utilizador estar "ao pé da máquina" pois não havia grande coisa que pudesse fazer. Eram lineares.

2. Descreva as principais forças e motivações que fizeram o SO evoluir.

Com o surgimento dos transístores (que substituiriam as válvulas) a criação de máquinas tornou-se mais simples e deu um grande salto (senão dos maiores ainda hoje) a nível tecnológico. Permitiu a construção mais eficiente de componentes e menos falíveis além de fazer assim surgir os circuitos integrados. Surgiram novos dispositivos também eles mais autónomos (de armazenamento de massa por ex, que iriam substituir as fitas magnéticas por serem mais rápidos e permitirem acesso aleatório). Estávamos perante o surgimento da multiprogramação simples: dispositivos de memória rápida e acesso aleatório, boa gestão de E/S por parte do SO, partilha de memória entre outros mecanismos do SO que permitiam partilhar e gerir recursos entre os programas em execução da máquina. A evolução tecnológica e o aumento da expectativa dos utilizadores também foram fortes motivações de uma forma geral para a evolução dos SO e para a passagem do monoprocessamento para a multiprogramação.

3. Explique como é que a forma como os utilizadores interagem com a máquina afeta o tipo de escalonamento?

Uma forma simples de entender como é que 2 formas de escalonamento podem afetar ou ser mais adequadas para cada tipo utilização são dando os seguintes exemplos:

1. Utilizador liga a máquina e não pretende fazer nada com ela a não ser processar algo de uma forma linear (algoritmo por ex.) que trata de I/O recebendo uma informação, processando e devolvendo o resultado – escalonamento multiprogramado aqui poderá servir. Há o risco de um dos processos entrar em starvation (nunca ser atendido) mas, assumindo que a carga do CPU é baixa e os processos a decorrer também o são, à partida poderá ser um escalonamento eficaz e não afetará o utilizador uma vez que o tempo também não é critério aqui.
2. Utilizador liga a máquina e pretende executar 3 programas distintos para tarefas distintas, o tempo de execução e conclusão são fatores relevantes e o utilizador irá estar a interagir com a máquina devido aos programas em questão que corre. – Escalonamento preeemptivo baseado em intervalo de tempo (quantum) poderá ser uma boa solução uma vez que garante que todos os processos são atendidos evitando que algum deles se apodere da máquina bloqueando o sistema e atrasando todos os outros processos que estão a decorrer.

4. Qual a diferença entre programação simples e preemptiva?

Respondido "sem querer" na alínea anterior. Em suma uma multiprogramação simples aproveita melhor o tempo de E/S de um

programa para executar outro assim como tira aproveitamento da CPU, mas não há garantia de que um dos programas monopolize a máquina. O SO torna-se complexo pois precisa de partilhar e gerir os recursos entre programas.

b. Na multiprogramação preemptiva, existe uma forçagem de cedência da vez da CPU aos processos que deverá ser partilhado por todos (mediante critérios algorítmicos que se verão mais à frente).

BASIS FOR COMPARISON	PREEMPTIVE SCHEDULING	NON PREEMPTIVE SCHEDULING
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.
#tablepress-131 from cache techd_middle		

5. Porque razão a monoprogramação desperdiça o tempo da CPU?

O monoprocessamento só executa um programa de cada vez, existe um desperdício enorme da CPU pois haverá alturas em que o CPU poderá estar parado à espera de algo (I/O por ex) desse programa (processo) e que poderia estar a tratar de outro processo rentabilizando o tempo.

6. **Porque razão e em que circunstancias os periféricos poderão desperdiçar tempo da CPU?**

   Numa situação em que a CPU esteja a aguardar um I/O por parte de um periférico e numa situação de monoprocessamento, o tempo será desperdiçado pois o periférico está a atuar e a CPU à espera parada.

7. **Como podem as operações de I/O ser usadas para otimizar a performance global da máquina?**

   Com surgimento de dispositivos mais autónomos é possível que a CPU possa estar a executar programas enquanto os periféricos de (E/S) trabalham, isto vai levar posteriormente a que se possa aproveitar os tempos em que o programa está parado nas operações de I/O para executar as instruções de outros programas. Estamos aqui num cenário de multiprogramação simples.

EVOLUÇÃO DOS SISTEMAS OPERATIVOS E ESCALONAMENTO (parte 2)

1. **De que forma as operações I/O são usadas para o escalonamento?**

   Sabe-se que existem vários tipos de operações I/O que consome tempo (por ex. leitura e acesso a uma informação em disco), por isso mesmo, alguns tipos de escalonamento reordenam a informação que estão a ler do disco (como exemplo) de modo a ser mais linear e evitar tantos desperdícios de tempo.

2. **Que tipo de requisitos quanto a periféricos existem quando se fala em multiprogramação?**
3. **Qual a principal característica da primeira geração de computadores e qual a sua influencia nos SO da atualidade?**

   A primeira geração consistia em processamento em série. Os computadores desta altura eram máquinas enormes (ainda com válvulas) e sem qualquer linguagem de programação (ex. alan turing). Não existe conceito de SO e desperdiçava-se imenso tempo no escalonamento (era feito em unidades de tempo fixas 30 min por ex), a estabilidade era fraca e eram pouco flexíveis.

4. Qual a principal diferença da segunda geração de computadores para a terceira?

A grande diferença (muito grande!) da 3ª geração para a 2ª foi sem duvida o surgimento de nova tecnologia, entenda-se, circuitos integrados. Com o surgimento dos mesmos existiu uma redução de preços e aumento de toda a capacidade geral de processamento e memoria e consegue-se perceber que em relação à 2nd gen, aumentou-se o numero de operações de E/S e a necessidade de interação direta entre o utilizador e o programa. Um problema notório aqui é que se continua a perder ainda muito tempo entre E/S de dados devido a existir mais interação com o utilizador. Chegam a existir pedidos prontos para serem atendidos enquanto o processador está inativo durante uma operação de E/S. Surge aqui então como solução o conceito de **pseudo-paralelismo** (executar um programa enquanto o outro está à espera). A multiprogramação é então, a grande novidade que surge com a 3ª geração com a possibilidade de serem também sistemas multiprogramados de tempo partilhado (divisão de tempo entre processos a executar) – escalonamento e despacho tratam da mux. da cpu.

5. O que era um "lote" no processamento em lote?

Um lote no processamento em lote era o nome dado ao conjunto de tarefas que iriam ser processadas após o lote anterior estar concluído. (linearmente).

6. Quais as tarefas do sistema operativo no processamento em lote?

O monitor (gestor) o que fazia era gerir as entradas e saídas, carregar os programas e executar os programas.

7. Explique as diferenças entre o processamento online, offline e spooling.

- **Processamento online** – O CPU encarregava-se de tratar de manipular a E/S o que gerava operações muito lentas.

- **Processamento Offline** – Nesta fase surgiu um dispositivo externo que era responsável por tratar da manipulação da E/S e depois enviava o mesmo para o CPU através de uma fita magnética. Permitiu melhorar a operação e o tempo de processamento, mas ainda assim não era muito eficaz pois manter o sincronismo entre o CPU e o dispositivo nem sempre era certo.

- **Spooling** – Utilização dos dispositivos de acesso aleatório (discos) com a entrada dos mesmos torna-se possível ter os periféricos ligados ao computador central e consegue manter-se o paralelismo entre operações de E/S de processamento. Com o spooling vai ser possível a multi-programação. Os tempos de processamento melhoram pois também se sabe que o disco é mais rápido que as fitas magnéticas.

8. **Quais foram os fatores mais importantes que nos levaram ao surgimento de multiprogramação?**

   Um dos fatores (senão o principal) que levou ao surgimento da multiprogramação foi devido ao grande crescimento (após surgimento dos transístores) tecnológico. Começou-se a aperceber de que havia muito tempo em que o CPU estava parado à espera de operações E/S entre outros e tinha que se arranjar alguma solução para evitar o desperdício. Os computadores pessoais estavam a começar a ter algum papel aqui e era necessário arranjar uma solução para aumento de eficiência da CPU, a multiprogramação, que é usada nos dias de hoje, foi a solução arranjada.

GESTÃO DO PROCESSADOR - ESCALONAMENTO

1. **Qual a diferença entre a multiprogramação simples e a preemptiva?**

   1. **Simples:** Não forçam o processo a sair do processador, só sai quando assim o desejar (fim do processo p ex). Depende muito pouco do algoritmo de escalonamento.
   2. **Preemptivos:** forçam o processo a sair com base no critério definido (depende muito do algoritmo de escalonamento) por ex, fim de quantum, surge um processo mais prioritário ou mais curto.

2. **Que tipos de preempção podem haver no escalonamento preemptivo?**

   i. Quantum (cada processo tem uma quantidade fixa de quantum associada, quando termina é feito o despacho).

   ii. Prioridade (consoante a prioridade do processo, pode ser substituído ou não)

   iii. Tempo de execução (prevê-se que um processo tenha uma execução mais curta é beneficiado em relação aos mais longos).

3. **Como é que um computador só com um processador/núcleo consegue executar vários programas em simultâneo?**

   Devido ao conceito de pseudo-paralelismo. De um ponto de visto macroscópico, a CPU parece estar a executar vários programas em simultâneo, mas o que está a fazer (na multiprogramação) é a trocar muito rapidamente (em muito curtos espaços de tempo imperceptíveis) entre processos distintos rentabilizando ao máximo o processamento disponível, dando a sensação de que está a correr vários programas em simultâneo quando na verdade só está em execução um de cada vez. Contudo, a sensação gerada dá-nos a impressão de que tudo corre em paralelo.

4. Se um programa entrar em ciclo infinito "encalha" a máquina toda? Dê a sua resposta do ponto de vista da multiprogramação simples e depois da preemptiva.

Resposta rápida: Simples, sim. Preemptiva, não.

1. Na simples não existe um mecanismo que force ao despacho do processo, logo independentemente de não haver mais nenhum processo ou existirem N à espera de ser "tratados" se o que está a utilizar a CPU decidir "encalhar" ou ficar a aguardar algo que bloqueie indeterminadamente pode levar todos os outros processos a starvation fazendo com que nunca sejam atendidos até à conclusão desse processo.
2. Na preemptiva com base em quantum (por ex. q=2), cada processo tem q(2) de quantum para a CPU. Esgotando esse quantum volta para o fim da "fila de espera" e o processo seguinte é atendido. Deste modo garante-se que um processo não impede os outros de reagirem ou terem direito ao seu "bocado" de CPU.

5. O que é o pseudo-paralelismo?

Ver resposta à pergunta 3.

6. Quais são as vantagens da multiprogramação preemptiva em relação à simples?

Evita a situação de starvation (já vista noutras respostas) e garante que todos os processos são "atendidos" com base numa preempção. Assegura também que os processos prioritários possam reagir rapidamente o que é indispensável em sistemas de tempo-real e é aconselhável em sistemas de multiutilizador. Aumenta de maneira geral a estabilidade do sistema, mas, pode como consequência negativa, gerar mais comutação (certamente que irá gerar) o que levará a menor desempenho global em algumas situações, além de tornar o SO mais complexo.

7. Num cenário real poderá um sistema preemptivo ter menos performance global em relação a um sistema multiprogramado, mas não preemptivo? Explique.

Sim. É possível. Convém lembrar que as trocas geram maior overhead do SO(comutação ocorre mais vezes) e gastam tempo, e esse tempo é diretamente proporcional ao numero de processos que estão em lista de espera para execução. Assumindo que temos um sistema preemptivo e N processos a executar com tempo partilhado, ao fim de algum tempo, se não houver bloqueios por parte do multiprogramado simples podemos concluir que o tempo total de execução para todos os processos poderá ser inferior no simples em relação ao preemptivo. Pode acontecer, contudo, estamos sempre sujeitos às vulnerabilidades de um sistema multiprogramação não preemptivo já faladas.

8. Qual destes é mais estável e porquê: monoprocessamento, multiprogramação simples, multiprogramação preemptiva?

No monoprocessamento já é sabido que os tempos de E/S não são aproveitados logo enquanto um processo estiver à espera de "algo" não será feito nada e o resultado será ineficiência a nível de tempos totais de execução.

b. Na multiprogramação simples (tempo não partilhado) este problema é solucionado aproveitando os tempos de E/S de um programa para executar outro, mais eficaz e estável que a situação do monoprocessamento, contudo não previne a monopolização da CPU.

c. Resposta é a multiprogramação preemptiva. Conjuga a multiprogramação garantido que todos os processos são atendidos e uma vez que estamos a falar de estabilidade do sistema, aqui, ainda que se possa em algumas situações perder para tempo de execução em relação ao multiprogramação simples, ganha-se sempre na estabilidade do sistema.

9. Que tipo de prioridades conhece? Para que serve cada tipo? Dê exemplos de aplicação.

- **Prioridades Fixas:** com a prioridade fixa, o escalonamento assegura-se de que o processador executa sempre o processo com a prioridade mais alta.

- **Prioridades Dinâmicas:** esta prioridade tenta solucionar o problema de starvation que pode acontecer. Supondo o exemplo dado nas aulas da fila do supermercado, se o processo for o próximo na fila de espera e estiver constantemente a entrar outro com prioridade mais alta pode correr o risco de nunca ser atendido. Por isso mesmo, o escalonamento atribui prioridades aos processos e modifica-as conforme os seus comportamentos (que recursos está a usar, a idade do processo e quanta atenção já teve por parte do processador).

10. Existem vários tipos de escalonadores? Quais/para quê?

Vamos ter em conta os 3 escalonadores:

1. **Longo prazo:** responsável por selecionar os processos que estão na memória secundária e levar para a memoria primaria (ligado à multiprogramação).

2. **Médio prazo:** Responsável sobretudo pelos swaps (in e out) dos processos que estão na memória primária colocando na secundária.
3. **Curto prazo:** responsável pelo despacho. Comutação dos processos, ou seja, guardar o estado atual do processo que está em execução na CPU no descritor do mesmo, retira-lo de execução e colocar o novo processo que entrará em execução retomando-o no ponto em que ficou. (ex: interrupção do relógio (fim de quantum), chamadas ao sistema, envio/recepção de sinais).

GESTÃO DO PROCESSADOR – ESCALONAMENTO (parte 2)

1. Descreva o conceito de "estado de um processo" e indique a sua utilidade.

O estado descreve a prontidão do processo a cada instante, ou seja, qual a situação do processo em questão.

2. Quais os estados que conhece e o respetivo significado?

Estados possíveis para um processo:

- **Novo:** Foi admitido a execução mas ainda não foi executado (falta recursos).
- **Execução:** O processo está a ser utilizado pela CPU.
- **Executável:** O processo está pronto para ser executado pela CPU mas em standby.
- **Bloqueado:** O processo está parado à espera de "algo" (I/O talvez?)
- **Terminado:** Já terminado mas a informação resultada da sua terminação ainda não foi aproveitada.
- **Suspenso:** O processo está suspenso, é parecido com bloqueado, mas diferente pois o processo não está à espera de nada.

3. **Que transições de estado podem ocorrer e em que circunstancias?**

   Suspenso **<->** Bloqueado

   b. Novo **->** Executável **<->** Suspenso **<-**Em execução **<-** Terminado

   c. Executável **<->** Em execução

4. **Um processo pode passar diretamente de bloqueado para em execução? Em que circunstancias? Porquê?**

   Não. Primeiro tem que passar sempre ao estado de executável, ou seja, tem que estar disponível para ser executado novamente e só aí então pode ser colocado pelo despacho em execução novamente.

5. **Qual a diferença de suspenso para bloqueado?**

   Esta situação pode ocorrer a pedido do próprio processo ou a pedido de outro processo. A diferença é que não está à espera de nenhum acontecimento enquanto que no estado bloqueado está.

6. **O que é o escalonamento por níveis e para que serve (?) – NS**

   O escalonamento por níveis seleciona o processo que tem mais prioridade e coloca-o na memoria principal deixando o outro na memoria secundaria (disco).

CONCEITOS USADOS EM ESCALONAMENTO

1. **O que entende por conceito de justiça no escalonamento?**

   Numa situação ideal e na ausência de informação em contrário todos os processos deverão ter o mesmo direito a usar os recursos do sistema (incluindo o processador). Deve-se evitar ao máxmo situações de starvation.

2. **Num contexto de escalonamento diga situações em que a justiça é desejável e outras em que não é.**

   Em sistemas de tempo-real a justiça pode ser aplicada, mas não é desejável. Não se justifica que um sistema não responda logo a um estimulo externo (ou interno) de um processo altamente prioritário devido ao escalonamento de

justiça estar a ser aplicado mesmo que já exista um outro processo quase em situação de starvation.

3. O que entende pelo conceito de starvation?

Starvation é o nome que se dá a uma situação em que um processo é sempre colocado para trás na fila de espera de execução porque surge sempre um mais prioritário fazendo com que este nunca seja atendido. Pode também acontecer numa situação de multiprogramação simples em que um dos processos bloqueia nunca passando a vez a outro. O escalonamentos de tempo partilhado procuram evitar esta situação.

4. Descreva o que entende por processo CPU bound e I/O bound? Explique.

i. **CPU bound**: Quando o processo necessita meramente do CPU de forma intensiva ficando raramente no estado bloqueado. (cálculos por exemplo)

ii. **I/O bound:** Necessidade do processo de informação externa de E/S (disco por ex.) ficando bastantes vezes bloqueado à espera.

5. Um processo pode ser simulatenamente CPU bound e I/O bound? Explique.

Sim pode. Um processo que tenha na sua composição do programa que executa inúmeras operações aritméticas com recurso a leitura de ficheiros (por exemplo), é simultaneamente CPU bound e I/O bound. Um processo que tenha como objetivo desfragmentar um disco é um bom exemplo para esta situação.

6. O que entende por equilíbrio de recursos?

Como o nome indica um equilíbrio de recursos significa que um escalonamento deve manter (tentar) todos os recursos do sistema igualmente ocupados para evitar picos de utilização intensa que depois poderão levar a esperas.

7. Qual a relação entre gestão do processador e a gestão do resto da máquina ?

Quando um escalonamento é aplicado há que ter em consideração quais as funções que a máquina vai ter. Se vai ser usada de forma interactiva, para processamento não interactivo, se vai ser usada em ambiente de controlo ou até se existiram processos/threads mais importantes que outros. Com base nos critérios em que a máquina vai ser usada o escalonamento deve ser adaptado nas suas características.

INDICADORES DE DESEMPENHO

Como as perguntas deste capitulo são abertas esta é alguma informação a reter:

Os indicadores de desempenho têm como objetivo avaliar a eficiência que os algoritmos de escalonamento têm para lidar com os processos.

Consegue-se, portanto, comparar diferentes algoritmos de escalonamento.

Os indicadores para medir o desempenho são:

- **Utilização da CPU**
  - Refere-se à utilização da CPU. Não é um bom indicador uma vez que idealmente obtém-se 100% e pode significar uma sobrecarga da CPU como se pode obter inferior a 100% e significar que o algoritmo está otimizado.

- **Throughput**
  - Quantidade de processos concluídos por unidade de tempo. É um bom indicador uma vez que pode mesurar a quantidade de processos que, dado um determinado de tempo, conseguiram ser concluídos. Atenção, contudo, que pode ser enganador caso a carga e/ou duração dos processos seja intensa/longa. Interessa maximizar.

- **Tempo de conclusão (turnaround)**
  - Corresponde ao tempo total que um processo levou a concluir desde que foi iniciado. Este indicador poderá variar bastante com a carga e a duração dos processos. Em sistemas interativos o indicador tempo de resposta torna-se mais interessante uma vez que um processo longo poderá ser tolerado desde que a interatividade não degrade.

- **Prazos (deadlines)**
  - Existem certos processos que têm deadlines a cumprir. Isso significa que se um processo não for cumprido dentro da seu deadline, acaba por ser "prejudicado". Interessa maximizar este valor.

- **Tempo de resposta (latência)**

- Um indicador talvez dos mais interessantes relativamente à interatividade com o utilizador. Refere-se a tentar ter o máximo de processos com a menor latência (interessante!). Logicamente só servirá como indicador em sistemas multiprogramados.

\ Razão de Penalização

- Tempo de espera (rácio) que o processo obteve em relação ao que teria caso estivesse a executar sozinho. Indica a penalização sofrida devido à existência de outros processos. O valor ideal é de 1.0.

\ Tempo de espera

- Tempo total que um processo esteve em espera no estado executável à espera da CPU para continuar a sua tarefa. Idealmente os processos nunca teriam que esperar uns pelos outros.

Em suma, pretende-se maximizar o throughput, minimizar a latência e maximizar o equilíbrio da utilização dos periféricos mantendo tudo ocupado. Deve haver sempre justiça (todos os processos/users serem atendidos).

Nem sempre é fácil pois os dispositivos E/S podem ficar inactivos devido à dificuldade em prever o tipo de processos (CPU bound ou I/O) e a otimização pode causar sem querer starvation de processos.

ALGORITMOS DE ESCALONAMENTO

Uma vez mais, existem poucas questões neste capitulo uma vez que se tratam de algoritmos de escalonamento é importante saber identifica-los e compreender as situações em que cada um é/deve ser aplicado.

\ Não preemptivos

- FCFS (First Come First Served)
  - ✓ Os processos são atendidos pela ordem de chegada.
  - ✓ Processos muito curtos poderão ficar à espera.
  - ✓ Não serve sistemas multiprogramados de tempo partilhado.
  - ✓ Favorece processos CPU Bound.
  - ✓ Só é adequado quando estamos perante processos não interativos, mais ou menos todo da mesma duração e o tempo de conclusão não é importante.

- SPN (Short Process Next)

- ✓ Funcionamento semelhante ao FCFS mas com a diferença que o próximo processo a ser executado é o que tem menor tempo estimado de execução.
- ✓ Objectivo: Diminuir penalização sofrida pelos processos curtos.
- ✓ Pode fazer com que processos longos sofram de starvation.
- ✓ Com muitos processos pequenos atinge-se o melhor turnaround.
- ✓ Problema: Como saber qual o tempo que os processos vão demorar?

- o HRRN
  - ✓ Elimina possibilidade dos processos mais longos sofrerem starvation (como acontece no SRT e SPN).
  - ✓ Tem-se em consideração o tempo que o processo esteve à espera do processador e o tempo de execução esperado onde.
  - ✓ Calculo do RR = (w+s) / s onde w é o tempo à espera e o s o tempo execução esperado. Pretende-se o valor mais baixo.

\ **Preemptivos**

- o Round Robin
  - ✓ Versão preemptiva do FCFS com base num intervalo de tempo (quantum).
  - ✓ Quando findo o quantum o processo volta para o fim da lista dos executáveis.
  - ✓ Corresponde ao algoritmo mais simples através do qual se consegue obter o tempo partilhado em sistemas multiprogramados.
  - ✓ Ter em atenção o tamanho do quantum. Se for demasiado pequeno haverá perda de eficiência devido a muita comutação, se for demasiado grande poderá perder-se interatividade. Idealmente o quantum deverá ser ligeiramente superior a uma interação típica.
  - ✓ Algoritmo justo.
  - ✓ Evita starvation.
  - ✓ Turnaround e tempo espera (valores médios) melhores que o FCFS quando estamos com processos de duração variável.

- o SRT (Short Remaining Time)
  - ✓ Versão preemptiva do SPN. O próximo a executar é aquele cujo o tempo esperado é menor (de processamento) só que sendo preemptivo se um processo estava bloqueado passar a executável e tiver um tempo restante de processamento menor que o atualmente em execução, ocorrera preempção, no entanto, poderá levar a starvation dos processos longos.
  - ✓ Menos interatividade que o RR mas menos overhead também.

ALGORITMOS DE FEEDBACK

- Existem ainda os algoritmos de feedback que tomam como consideração a atividade recente da maquina para perceber o que se irá passar num futuro próximo. Nem sempre é eficaz.
- Uma implementação típica do algoritmo de feedback faz com que a prioridade do processo vá sendo diminuída ao longo da idade do processo, ou seja, se um processo for curto nunca fica pouco prioritário. Se for longo vai perdendo prioridade ao longo do tempo para não influenciar os mais curtos.
- Têm quantum variável para recuperar um processo que esteja a ficar penalizado e vice-versa.
- Menos provável de causar starvation mudando a duração dos quantuns do que mudar as prioridades.

1. **O que é o escalonamento por níveis (multi-lista) e para que serve?**

   Cada conjunto de processos corresponde a uma lista de processos a executar. Tem-se, portanto, uma lista de processos a executar dos quais vão sendo promovidos e despromovidos (indo para o fim da lista).

2. **O que entende pela questão de escalabilidade no escalonamento. Indique uma forma de lidar com essa questão.**

   Uma solução em engenharia é escalável (adaptável a qualquer dimensão do problema que vai tratar) se o custo (tempo, espaço ocupado etc.) for proporcional à dimensão do problema.

   b. Para impedir que um algoritmo de escalonamento se torne não escalável deve-se impedir que existam muitos processos em analise de cada vez, logo, a solução passa por agrupar os executáveis em conjuntos.

   c. Vai-se pegando nos conjuntos e promovendo/despromovendo fazendo com que sejam analisados apenas N processos de cada vez.

ALGORITMOS DE ESCALONAMENTO UNIX, LINUX E WINDOWSNT

1. **Explique como funciona o escalonamento no Unix (genericamente), no Linux e no WindowsNT. Relacione com os conceitos dados nesta matéria.**

- **Unix:** baseado em Round Robin de tempo partilhado, quantum típico de 100 ms, prioridades vão se ajustando dinamicamente consoante o comportamento do processo.
- **Linux:** O quantum é um conjunto de notificações de relógio.
- **WindowsNT:** existem 32 prioridades fixas, a threads(processos) são colocadas em níveis pelo utilizador ou pelo sistema conforme a sua importância. As prioridades dinâmicas são usadas da forma habitual, mas tendo como base as prioridades base (fixas) da thread. Prioridades ajustadas no fim da operação I/O, no fim de uma espera de sincronização ou no fim de um quantum. Se uma thread se interromper será aumentada a sua prioridade conforme a tabela. Se a thread usar o quantum todo, a prioridade converge para a sua prioridade base.

2. Explique de que forma o WindowsNT evita situações de starvation.

Com o recurso às prioridades dinâmicas consegue garantir-se que um processo não sofra de starvation. Sendo as prioridades ajustadas frequentemente segundo os parâmetros definidos.

3. Explique de que forma o Linux evita situações de starvation.

O Linux utiliza como quantum um conjunto de notificações de relógio, logo, a cada época o quantum atribuído a um processo é modificado. Tem-se como base para o quantum inicial = quantum base + quantum por usar na época anterior. A prioridade é definida com base na prioridade base + quantum por usar atual – nice. Isto faz com exista um equilíbrio (dentro do possível) na prioridade dos processos evitando starvation.

4. Poder-se-á dizer que os algoritmos de escalonamento do Linux e do WindowsNT são 100% justos? E isso é bom ou mau? Explique.

Enquanto algoritmos de escalonamento, sendo de (preemptive time-sharing (Linux)), tentam ser o mais justos possíveis. O WindowsNT gere com base nas prioridades das threads (multi-level feedback escalonamento), ainda que seja uma abordagem ligeiramente diferente do Linux o resultado final pretende ser idêntico. Cada um tem os seus altos e baixos com base nos respetivos algoritmos que já foram abordados anteriormente. (Esta descrição é curta e não deverá ser tida como uma resposta completa, uma vez que são diferentes em vários aspetos e apenas pretendo que isto seja uma breve resposta).

5. Explique como são usadas as prioridades no WindowsNT.

Existem 32 tipo de prioridades sendo que as primeiras 16 são dinâmicas e as restantes fixas (real-time). As prioridades são ajustadas no fim de uma operação I/O, no fim de uma espera de sincronização ou no fim de um quantum. Se uma thread interromper voluntariamente, a sua prioridade será aumentada de acordo com a tabela. Se uma thread usar todo o seu quantum, a sua prioridade converge para a prioridade base.

6. Os quantuns no Linux têm sempre a mesma duração? Porquê?

Não. Porque dependem do conjunto de notificações de relogio. Em cada época de quantum o quantum atribuído é modificado com base no calculo mencionado na resposta 3. Isto deve-se ao fato do Linux tentar compensar os processos que têm uma grande quantidade de quantum para usar e aqueles que não necessitaram de todo o seu quantum.

GESTÃO DE MEMÓRIA

1. Quais as desvantagens do endereçamento real face ao endereçamento virtual?

1. O programa está limitado à dimensão física da memória.
2. Normalmente um programa é preparado pelo compilador para correr numa gama de endereços.
3. Multi-programação é difícil pois quase de certeza que irão existir programas a usar a mesma gama de endereços.

O endereço virtual permite solucionar todos estes problemas fazendo com que o programa tenha quase acesso "ilimitado" à memória.

2. Descreva o mecanismo de tradução do endereço de memória

Segmentado: O bloco é igual a um segmento, os segmentos podem ter tamanhos diferentes. A tradução de endereços virtuais em reais é feita pelo próprio processador com base à tabela de segmentos. Cada entrada na tabela é chamada de descritor de segmento e descreve o segmento quanto a:

1. Tamanho e localização na memória física.
2. Tipo de acesso (leitura, escrita, execução)
3. Estado (p) em memória.

b. **Paginado:** idêntico ao segmentado mas com a vantagem de que trabalha com páginas e as páginas têm sempre o mesmo tamanho. Evita fragmentos deste modo, cada entrada na tabela de páginas denomina-se de PTE e cada PTE descreve uma página quanto a:

1. Localização em memória
2. Tipo de acesso
3. Estado em memória
4. Utilização da pagina quanto a referencia

c. Em ambos os mecanismos mencionados, o programa vê o seu espaço de endereçamento como tendo a memória toda só para si mas no fundo o que acontece é que tem acesso a uma memória virtual, neste caso, o programa

vai à tabela (paginas ou segmentos) em questão e obtém o endereço real da memória. Pode acontecer que o bloco que se está a tentar aceder esteja guardado em disco e nesse caso cabe ao gestor de memória mapear novamente e trazer do disco para a memória física o bloco para que possa ser acedido pelo processo. Toda esta gestão é feita de forma transparente para o processo. Para tradução do endereço a tabela (página ou segmento) contém a base (BTS/BTL) do bloco em questão e depois é aplicado o offset de deslocação desde que não ultrapasse o limite desse bloco (LTS/LTP).

3. Elabore uma analise comparativa de memória segmentada com memória paginada salientando vantagens e desvantagens de um face ao outro.

\ Segmentada (vantagens e desvantagens)

- ✓ Adequa-se à divisão lógica dos programas, um segmento para código outro para dados etc. e não é obrigatório que haja só um segmento para cada tipo.
- ✓ Permite trabalhar eficientemente sobre uma zona de memória (segmento em que atua).
- ✳ Pode não criar um espaço de endereçamento perfeitamente linear e contínuo de uma forma transparente para o processo.
- ✳ Tem algoritmos de gestão mais complexos em relação ao das páginas.
- ✳ A transferência entre a memória principal e secundária é sempre feita em blocos (segmentos) inteiros, se forem grandes leva a custos de eficiência, as páginas têm sempre o mesmo tamanho.

\ Paginada (vantagens e desvantagens)

- ✓ Totalmente transparente para o programador e processo
- ✓ Os algoritmos de gestão são mais simples e eficientes pelo facto de todas as paginas terem o mesmo tamanho. As trocas de paginas são feitas com base no swap de paginas (tamanho típicos entre 512 bytes – 8K) sendo muito rápidas.
- ✓ Fragmentação só mesmo na ultima pagina do processo (interna).
- ✳ Não se adequa à divisão lógica dos programas como acontece na memória segmentada.
- ✳ As faltas de página representam perdas de eficiência maiores do que perda de segmentos.

\ Segmentada-Paginada (vantagens)

- ✓ Junta o melhor dos dois mundos, um endereço virtual é composto por um segmento, uma página e o respetivo deslocamento dentro da página. O

segmento irá indicar qual o descritor dentro da tabela de segmentos e o descritor contém a informação acerca da tabela de páginas a usar para restante tradução do endereço (que é feita com base na tradução da memória paginada).

- ✓ Deste modo tem-se um segmento divido em N páginas mantendo-se a lógica de divisão inicial (segmentos) à estrutura lógica dos programas e a eficiência associada às páginas.
- ✓ Mecanismos de otimização de eficiência continuam a ser usadas: registos de segmentos e TLB.