







1 - INTRODUÇÃO

- 1- Para que serve um SO? Que recursos gere?
- 2- Existem 2 interfaces. Quais?
- 3- Qual a missão do SO?
- 4- Quais as desvantagens de uma alternativa ao SO?
- 5- Quais os critérios de qualidade do SO? Descreva-os.
- 6- O que é um monitor de controlo, o que o constitui e qual a sua limitação principal?
- 7- Diga o que entende por tratamento em lotes (batch).
- 8- Em que consiste a multiprogramação?
- 9- O que entende por tempo partilhado e quais as consequências deste?
- 10- O que é a memória virtual e qual a propriedade que a impulsiona?
- 11- Distinga tempo virtual de tempo real. Distinga igualmente os 2 tipos de tempo real.
- 12- O que é um sistema aberto?
- 13- Descreva a organização típica dum SO.
- 14- Aplicações em modo utilizador não podem interferir com dados/execução do SO. Porquê?
- 15- Como se "separa" o modo núcleo do modo utilizador?
- 16- Descreva estrutura monolítica e os seus problemas.
- 17- Descreva sistema em camadas e a sua desvantagem principal.
- 18- O que permitiu o micro-núcleo? O que o constitui?



RESPOSTAS:

- 1- Um sistema operativo  serve para gerir recursos e fornecer uma interface de acesso. Gere recursos do tipo lógico permitindo abstracção dos recursos físicos.
- 2- As duas interfaces são: interface operacional (comandos) e biblioteca de funções  do SO.
- 3- A missão do SO é criar uma máquina virtual sobre a física que ofereça recursos lógicos básicos necessários ao desenvolvimento de aplicações, podendo assim ser independente do hardware onde se executa.
- 4- O esforço de programação poderia ser muito grande, um conjunto significativo de funções seria repetido, cada aplicação poderia otimizar o seu desempenho mas globalmente a máquina ficaria subaproveitada, não seria possível existirem políticas globais de segurança e tolerância a falhas.
- 5- Os critérios de qualidade são:
- *Desempenho*: gestão eficiente dos recursos físicos que suportam os recursos lógicos
 - *Segurança*: isolamento dos utilizadores, permitir partilha segura dos recursos lógicos
 - *Fiabilidade e Disponibilidade*: detectar um conjunto de falhas, tolerar um conjunto de erros
 - *Interface de Programação Completa e Simples*: facilitar a concepção das aplicações, a sua manutenção e portabilidade
 - *Interface de Operação e Gestão dos Recursos Lógicos fácil de usar*
- 6- O monitor de controlo permite ao utilizador carregar programas em memória, editá-los, etc, obter resultados dos programas. *(falta limitação principal)*
- 7- Como os periféricos mecânicos eram muito lentos, decidiu-se separar as entradas e saídas do processamento, assim as E/S e os programas podem ser executados em paralelo.
- 8- O mecanismo de interrupções permite multiplexar o processador entre várias actividades concorrentes; a multiprogramação permite então a execução concorrente de vários programas e nomeadamente permite otimizar a utilização do processador.
- 9- O tempo partilhado  cria a ilusão que o computador está permanentemente disponível para o utilizador. Como consequência do tempo partilhado teve de existir uma revisão dos algoritmos de escalonamento, definição de mecanismos de segurança, aparecimento de sistemas de ficheiros, hierarquia de memória.
- 10- A memória virtual elimina a restrição física imposta pelo tamanho da memória física permitindo um grau de multiprogramação muito superior e é impulsionada pela propriedade da localidade.
- 11- O tempo virtual é o tempo de execução dos programas que não têm relação com o tempo cronológico exterior ao computador; o tempo real tenta garantir que o computador produz uma resposta a um acontecimento externo num intervalo de tempo determinado (caso contrário o sistema não cumpre a sua especificação - falha).
Os sistemas de tempo real  são de 2 tipos:
- *Tempo Real Relaxado ou Soft Real Time* : sistema onde se admite que certas respostas a acontecimentos externos


podem não ser dadas exactamente nos intervalos de tempo especificados

- *Tempo Real Estrito ou Hard Real Time* : sistema onde o não cumprimento de um requisito temporal corresponde a uma falha, obrigando a um escalonamento de processos que torna estes sistemas totalmente incompatíveis com o funcionamento em tempo virtual interactivo

12- Um sistema aberto é um sistema portátil e interoperável, de interface normalizada e com génese no sistema UNIX (resumo: sistema aberto = LINUX p.e.).

13- Um sistema operativo organizasse em: Kernel  (núcleo dividido em módulos), Biblioteca das funções de sistema (system calls  usadas para aplicações), Processos Sistema.

14- Porque essas aplicações podem suprimir/alterar registos, situação crítica a nível de segurança.

15- O modo utilizador separa-se do modo kernel por meio de uma barreira de protecção que só é quebrada aquando invocada uma interrupção de software (trap ). Em seguida um exemplo de como isso pode acontecer:

fwrite <- requer chamada ao sistema

1. coloca registo de código ref "fwrite"

2. interrupção software (trap)

a) comutação para modo núcleo

b) invocando handler do núcleo (agulhagem)

> olha para o registo com código "fwrite"

> invoca rotina do núcleo correspondente ao "fwrite"


- invoca E/O (reservada ao modo núcleo)

c) handler retorna

3. retorna ao programa utilizador

16- A estrutura monolítica é constituída apenas por um único sistema, é internamente organizada por módulos, tem estruturas de dados globais. O problema surge quando é necessário dar suporte a novos periféricos (fez posteriormente surgir os gestores de dispositivos ou device drivers).

17- No sistema em camadas cada camada usa os serviços da camada precedente, é fácil modificar o código de uma camada, possui mecanismos de protecção o que implica uma maior segurança e robustez do sistema. A desvantagem deste sistema é de ter um mau desempenho devido às várias comutações entre as várias camadas.

18- O micro-núcleo  tem reduzidas dimensões e permitiu possuir apenas o essencial do SO: gestão de fluxos de execução (threads), gestão dos espaços de endereçamento, comunicação entre processos, gestão das interrupções.

2 - PROCESSOS

1- O que entende por pseudoconcorrência?

2- Defina programa e processo.

3- Quais as semelhanças entre um processo e um processador?

4- Diga quais as propriedades que definem o objecto "processo".

5- Quais os comandos para criar, eliminar processos pais e filhos?

6- O que entende por modelo de segurança?

7- Como é constituído o controlo dos Direitos de Acesso?

8- Como se identifica um processo em UNIX? Como se relacionam estes?

9- A execução de um processo via fork() apenas permite lançar um processo com o mesmo código. Quais os problemas resultantes disto?

10- Como se dá a autenticação?

11- Em quantos (e quais) categorias tem a protecção no acesso aos recursos?

12- Conceptualmente comprime-se a linha da matriz de ACL associada a um recurso em apenas 3 entradas. Quais os motivos? Desvantagem?

13- O que são: SETUID, BIT SETUID, UID e GID (real e effective)?

14- O que são threads (tarefas)?

15- Porque usar tarefas em vez de processos?

16- Como é a programação num ambiente multitarefa?

17- Quais as alternativas de implementação de tarefas?

18- O que são pseudotarefas?

19- Compara as tarefas modo utilizador e modo núcleo quanto a:

- capacidade de utilização em diferentes SOs
- velocidade de criação e comutação (compare também com processos)
- tirar partido da execução paralela em multiprocessadores
- aproveitamento do CPU quando uma tarefa bloqueia

- 20- O que são rotinas assíncronas? Qual o seu nome alternativo?
- 21- Como tratá-las na programação sequencial?
- 22- Poder-se-ia lançar uma tarefa por acontecimento. Qual a desvantagem?
- 23- Diga como se efectua o tratamento dos signals.
- 24- A chamada de sistema kill tem um nome enganador. Porquê?
- 25- Diga o que fazem as funções associadas aos signals: unsigned alarm, pause e unsigned sleep.
- 26- Quais as versões iniciais dos signals?
- 27- O que são processos, threads, fibers e jobs em windows?

RESPOSTAS:

- 1- Consiste na implementação de sistemas multiprogramados sobre um computador com um único processador.
- 2- Um programa é um ficheiro executável sem actividade, e um processo é um objecto do sistema operativo que suporta a execução dos programas.
- 3- Um processo e um processador ambos têm: espaço de endereçamento virtual (conjunto de posições de memória acessíveis, código, dados, pilha, dimensão variável), reportório de instruções (do processador executáveis em modo utilizador, funções do sistema operativo), contexto de execução/estado interno (valor dos registos do processador, toda a informação necessária para retomar a execução de um processo, memorizado quando o processo é retirado de execução).
- 4- As propriedades do objecto processo são: identificador, programa, espaço de endereçamento, prioridade, processo pai, canais de entrada saída, ficheiros, quotas de utilização de recursos, contexto de segurança.
- 5- Os comandos são:
 - para criar: `IdProcesso = CriaProc (Código, Prioridade,...)`
 - para eliminar: `EliminarProc (IdProcesso)`
 - para bloquear: `Estado = EsperarTerminacao (Idprocesso)`
- 6- Um processo em execução tem de estar associado a um Utilizador (entidade que pode ser responsabilizada pelos seus actos), os utilizadores são representados no sistema por o UID e para facilitar a partilha o utilizador pode pertencer a um grupo de utilizadores (GID).
- 7- O controlo dos direitos de acesso é basicamente constituído por uma autorização, que é uma operação que valida os direitos do utilizador sobre um recurso antes deste poder executar uma operação sobre ele. A autorização baseia-se conceptualmente numa Matriz de Direitos de Acesso. Para um dado objecto, a coluna da matriz define a Lista de Direitos de Acesso (ACL) e para um dado utilizador a linha respectiva define todos os seus direitos normalmente designados por Capacidade.
- 8- A identificação de um processo é feita a partir de um inteiro designado por PID, e alguns identificadores são pré-atribuídos: processo 0 é o swapper (gestão de memória) e o processo 1 init é o de inicialização do sistema. Os processos relacionam-se de forma hierárquica: o processo herda todo o ambiente do processo pai, o processo sabe quem é o processo que descende designado por processo pai, quando o processo pai termina os subprocessos continuam a executar-se ou seja são adoptados pelo processo de inicialização (pid=1). Os processos têm também prioridades variáveis.
- 9- A execução de um processo via `fork()` apenas permite lançar um processo com o mesmo código. Quais os problemas resultantes disto?
- 10- Um processo tem associados 2 identificadores que são atribuídos quando o utilizador efectua o login (se autentica) perante o sistema (UID e GID).
- 11- Para um recurso a protecção é definida em 3 categorias:
 - Dono (owner): utilizador que normalmente criou o recurso
 - Grupo (group): conjunto de utilizadores com afinidades de trabalho que justificam direitos semelhantes
 - Restantes utilizadores (world)
- 12- Tal assim acontece devido ao facto do unix já ter sido criado à muito tempo, tanto que os novos SOs são bem mais flexíveis.
- 13- O SETUID permite alterar dinamicamente o utilizador; o Bit SetUID faz com que o processo assuma a identidade do dono do ficheiro durante a execução do programa; o Real UID e GID são os originais do processo enquanto que os Effective UID e GID são usados para verificar permissões de acesso e que pode ter sido modificado pelo SETUID.
- 14- As threads são mecanismos simples para criar fluxos de execução independentes, partilhando um contexto comum.

15- Porque os processos obrigam ao isolamento (espaços de endereçamentos disjuntos) e existe dificuldade em partilhar dados. As threads são mais eficientes a nível de criação e comutação.

16- As tarefas partilham o mesmo espaço de endereçamento e portanto têm acesso às mesmas variáveis globais. A modificação e teste das variáveis globais tem de ser efectuada com precauções especiais para evitar erros de sincronização.

17- As alternativas são implementação de tarefas-núcleo e tarefas-utilizador (pseudotarefas).

18- As pseudotarefas são tarefas implementadas numa biblioteca de funções no espaço de endereçamento do utilizador.

19- Comparação:

- as threads kernel não podem ser usadas se o SO não suportar
- threads-utilizador são mais rápidas enquanto os processos são os mais lentos
- em multiprogramação, lança uma thread-kernel por processador, bem como nos processos
- thread-kernel tem maior aproveitamento, bem como os processos

20- As rotinas assíncronas são certos acontecimentos que devem ser tratados pelas suas aplicações, embora não seja possível prever a sua ocorrência. O seu nome alternativo é "eventos".

21- Num modelo multitarefa poderia dedicar-se uma tarefa à espera de um acontecimento, o que é penalizante.

22- A desvantagem é que isso seria penalizante a nível de desempenho por implica a existência de testes sistemáticos.

23- O tratamento dos signals pode ser feito: por omissão (termina o processo), ignorado (sinais com o SIGKILL não podem ser ignorados), através de uma rotina de tratamento (handler) através da chamada à função sistema signal.

24- A função kill envia um sinal ao processo. Apesar do nome, pode não matar o processo desde que o signal tenha um tratamento associado ou esteja ignorado.

25- O signal SIGALARM é enviado para o processo depois de decorrerem o número de segundos especificados, se o argumento for zero o envio é cancelado; o pause() aguarda a chegada de um signal; o unsigned sleep () faz um alarme e bloqueia-se à espera do signal.

26- As versões iniciais dos signals são System V (associação de uma rotina a um signal é apenas efectiva para uma activação, tratamento por omissão, pode gerar problemas se houver recepção sucessiva de signals) e BSD (recepção de um novo sinal é inibida durante a execução da rotina de tratamento).

27- Em windows, um processo é um contentor de recursos usados pelas tarefas, as threads reais são fluxos de execução, as fibers são pseudotarefas que não são vistas pelo núcleo, os jobs são grupos de processos que permitem gestão uniforme.

3 - SINCRONIZAÇÃO

1 - Qual o problema da exclusão mútua? É problema se for multi-tarefa?

2 - Quais as propriedades da secção crítica?

3 -

```
int trinco = ABERTO;
```

```
fechar() {  
    while (trinco == FECHADO) ;  
    trinco = FECHADO;  
}
```

```
abrir () {  
    trinco = ABERTO;  
}
```

Qual a propriedade que não é garantida?

4 -

```
int t1_quer_entrar = FALSE, t2_quer_entrar = FALSE;
```

```
t1_fechar() {  
    while (t2_quer_entrar == TRUE) ;  
    t1_quer_entrar = TRUE;  
}
```

```
t1_abrir() {  
    t1_quer_entrar = FALSE;  
}
```

/ t2 -> simetrico */*

Qual a propriedade que não é garantida?

5 -

```
int t1_quer_entrar = FALSE, t2_quer_entrar = FALSE;
```

```
int tar_prio = 1;
```

```
t1_fechar() {
```

```
    tar_prio = 2;
```

```
    t1_quer_entrar = TRUE;
```

```
    while (t2_quer_entrar && tar_prio == 2) ;
```

```
}
```

```
t1_abrir() {
```

```
    t1_quer_entrar = FALSE;
```

```
}
```

/ t2 -> simetrico */*

O que acontece neste caso?

6 -

```
int tar_prio = 1;
```

```
t1_fechar() {
```

```
    tar_prio = 2;
```

```
    while (tar_prio == 2) ;
```

```
}
```

```
t1_abrir() {
```

```
}
```

/ t2 -> simetrico */*

Qual a propriedade que não é garantida?

7 -

```
int t1_quer_entrar = FALSE;
```

```
int t2_quer_entrar = FALSE;
```

```
int tar_prio = 1;
```

```
t1_fechar() {
```

```
    t1_quer_entrar = TRUE;
```

```
    tar_prio = 2;
```

```
    while (t2_quer_entrar &&
```

```
    tar_prio == 2);
```

```
}
```

```
t1_abrir() {
```

```
    t1_quer_entrar = FALSE;
```

```
}
```

```
t2_fechar() {
```

```
    t2_quer_entrar = TRUE;
```

```
    tar_prio = 1;
```

```
    while (t1_quer_entrar &&
```

```
    tar_prio == 1);
```

```
}
```

```
t2_abrir() {
```

```
    t2_quer_entrar = FALSE;
```

```
}
```

Porque motivo é garantida a ausência de "starvation"?

8 - Relativamente às soluções algorítmicas de Peterson e Barkery, que podemos concluir?

9 - Que conclusões podemos tirar sobre as soluções com suporte do hardware?

10 - Quais as soluções com suporte do SO?

12 - Quais as duas primitivas de sincronização?

13 - É possível obter-se exclusão mútua com trincos(locks)?

14 - Qual o diagrama de estados dos processos/tarefas?

15 - Explique o processo como o trinco bloqueia a tarefa? E para desbloquear?

16 - Quais as limitações dos trincos?

17 - Quais as primitivas dos semáforos?

18 - Quais as variantes dos semáforos?

19 - É possível exclusão mútua com semáforos? E se a pilha estiver completa?

20 - O que é cooperação entre processos?

21 - Estude bem os casos seguintes: produtor-consumidor, leitores-escritores e jantar de filósofos

22 - Qual o objectivo e as limitações dos mecanismos directos de sincronização?

RESPOSTAS:

1 - Sem trincos, se o ambiente for multitarefa podem-se gerar problemas a nível de tarefas e dos resultados delas implícitos.

- 2 - Exclusão mútua e progresso(liveness) -- ausência de interbloqueagem (deadlock) e ausência de minguia (starvation).
 - 3 - Não garante Exclusão Mútua.
 - 4 - Problema de DeadLock.
 - 5 - É garantida exclusão mútua, não há interbloqueagem nem starvation.
 - 6 - Garante exclusão mútua mas só funciona quando há concorrência (bloqueia no while).
 - 7 - Algoritmo de Peterson.
 - 8 - São complexas e provocam latência, só contemplam espera activa.
 - 9 - Facilitam o problema da resolução de sincronizações por meio de funções abrir() e fechar(), as quais usam instruções especiais oferecidas pelos processadores, nomeadamente a inibição das interrupções. Contudo não podem ser utilizados directamente por programas em modo utilizador, e só contemplam espera activa.
 - 10 - As soluções são a utilização de software trap (interrupção SW), comutação para modo núcleo, estruturas de dados e código de sincronização pertencente ao núcleo, usa o suporte do hardware.
 - 12 - Trincos e Semáforos.
 - 13 - O trinco lógico serve exclusivamente para implementar exclusão mútua.
 - 14 - Um Executável, ao ser seleccionado pelo Despacho, é colocado em Execução.
- Depois disso pode-lhe acontecer uma de duas coisas: ou é retirado pelo Despacho e volta a estar Executável ou então é Bloqueado num Trinco Lógico, fica Bloqueado, e depois é Desbloqueado, voltando a estar Executável de novo.
- 15 - O trinco retira a tarefa de execução, salvaguarda o seu contexto, marca o seu estado como bloqueada, coloca a estrutura de dados que descreve a tarefa na fila de espera associada ao trinco. No caso contrário, caso existam tarefas bloqueadas, o trinco marca o estado da tarefa como "executável", e retira a estrutura de dados que descreve a tarefa da fila de espera associada ao trinco.
 - 16 - Os trincos são limitados nomeadamente quanto a bloquear tarefas se a pilha já estiver cheia.
 - 17 - As primitivas dos semáforos são: s=criar_semaforo(num_unidades), esperar(s), assinalar (s).
 - 18 - Há 4 tipos: Genérico - assinalar() liberta um processo qualquer da fila; FIFO - assinalar() liberta o processo que se bloqueou há mais tempo; Semáforo com Prioridades: o processo especifica em esperar() a prioridade, assinalar() liberta os processos por ordem de prioridades; Semáforo com Unidades - as primitivas esperar() e assinalar() permitem especificar o número de unidades a esperar ou assinalar.
 - 19 - Devido ao facto do semáforo ser mais genérico que o trinco lógico, pode ser usado para garantir exclusão mútua mas é mais ineficiente que o trinco lógico por isso o programador tem de garantir o uso simétrico do assinalar() e do esperar().
 - 20 - Na cooperação entre tarefas estas competem por recursos e indicam umas às outras a existência/ausência de recursos e a ocorrência de acontecimentos.
 - 22 - O objectivo é suspender temporariamente a execução de subprocessos. As limitações são: a sincronização directa implica o conhecimento do identificador do processo sobre o qual se pretende actual; não se pode dar aos programas dos utilizadores a possibilidade de interferirem com outros utilizadores; a restrição habitual é apenas permitir o uso de sincronização directa entre processos do mesmo utilizador.

4 - PROCESSOS NÚCLEO

- 1 - O que é o gestor de processos?
- 2 - Qual o Diagrama de Estados dos Processos?
- 3 - Quais os 2 tipos de contextos e em que eles consistem?
- 4 - Como é que o Sistema Operativo pode ser invocado? Essas invocações são quais? Defina-as.
- 5 - Qual o processo de invocação do SO?
- 6 - Qual a função do Despacho?
- 7 - Como estão estruturadas as funções de sistema e quais as suas vantagens?
- 8 - Quais os objectivos do escalonamento?

- 9 - Quando deve a política de escalonamento ser invocada? Qual o problema daí gerado?
- 10 - Quais as 2 classes de processos do escalonamento em Tempo Partilhado? Qual destas devem ter maior prioridade no acesso ao CPU?
- 12 - Quais as políticas de Escalonamento em Sistemas de Tempo Partilhado?
- 13 - O que é o tempo de execução partilhado (time-slices ou round-robin), qual o seu objectivo, como se implementa e quais as suas desvantagens?
- 14 - O que são prioridades? De que tipo podem ser?
- 15 - Numa gestão multilista, em que situações um processo deve ser promovido? E relegado?
- 16 - O que é a preempção, qual o seu objectivo, a sua implementação e desvantagens?
- 17 - Em quantas estruturas se encontrava dividido o UNIX? Em que consistiam estas?
- 18 - Porque motivo o task_struct do linux deixou de estar separado em 2 colunas?
- 19 - Diga em que circunstâncias é trocado o modo utilizador para o modo núcleo. Ao que corresponde essa mudança?
- 20 - Como é feito o escalonamento nos vários modos e quais destes têm maior prioridade?
- 21 - Como se calculam as prioridades em Modo Utilizador?
- 22 - Existe algum problema com o algoritmo de escalonamento do Linux? Qual? Porquê?
- 23 - Porque é que o facto de um processo ser mais prioritário ser escolhido em primeiro lugar melhora a escalabilidade?
- 24 - Em que consiste o escalonamento em "real-time"?
- 25 - Em que circunstâncias ocorre o despacho?
- 26 - Quais as chamadas sistema do escalonamento?
- 27 - Como é criado um processo filho?
- 28 - Como se pode terminar um processo? E suspendê-lo?
- 29 - Como se dá a execução de um programa?
- 30 - Quais as 2 primitivas de sincronização interna?
- 31 - Como é enviado um signal? Como é este tratado?
- 32 - Como funciona o Gestor de Processos em Windows?

RESPOSTAS:

- 1 - O gestor de processos é a entidade do núcleo responsável por suportar a execução dos processos. Faz a multiplexagem do processador (despacho e escalonamento), faz a gestão das interrupções e encarrega-se das funções de sincronização.
- 2 - Resposta no tema 3 pergunta 14.
- 3 - Existe o contexto de Hardware e o contexto de Software. O Contexto de Hardware são os registos do processador e os registos da unidade de gestão de memória. O Contexto de Software é a identificação, a prioridade, o estado e outras informações de um processo.
- 4 - As actividades do sistema operativo podem ser consideradas como desencadeadas por interrupções, as quais podem ser provocadas por: Hardware (relógio ou periféricos), Software (traps, software interrupts) ou Excepções (provocadas pelo programa em execução como divisão por zero ou acesso a memória indevido).
- 5 - O processo é o seguinte:
 - 1- Interrupção (salvaguarda contexto na pilha actual)
 - 2- Gestor das Interrupções (identificação da interrupção -- vector de int.)
 - 3- Rotina de Serviço de Interrupção (serve a interrupção possivelmente alterando o estado de algum processo)
 - 4- Despacho
 - 5- Retorno da Interrupção
- 6 - O Despacho tem como função comutar o processador sempre que lhe seja indicado para o fazer. A sua funcionalidade é: copia o contexto hardware do processo em execução da pilha actual para o respectivo descritor (entrada na tabela de processos), escolhe o processo mais prioritário entre os executáveis, carrega o contexto hardware no processador, transfere o controlo para o novo processo: coloca program counter guardado no contexto do novo processo na pilha núcleo, return from interrupt (RTI) é "enganado", para o processo comutado a rotina de interrupção "demorou muito tempo a executar-se".
- 7 - Estão estruturadas em 2 entidades funcionais: a função propriamente dita (faz parte do código do sistema operativo) e a rotina de interface que é ligada com o código do utilizador e que usa instruções de interrupção por software (traps) para invocar a função no núcleo). As vantagens são: protecção (o código das funções sistema está residente no núcleo e não pode ser acedido pelos processos utilizador), a interrupção muda o estado do Processador de modo utilizador para modo núcleo, partilha das funções sistema por todos os processos e o sistema operativo pode ser modificado (novas versões) transparentemente desde que não se altere a interface.
- 8 - O objectivo é optimizar a utilização do processador (e dos restantes componentes do sistema. As políticas de escalonamento definem objectivos mais específicos:
 - Batch: produtividade (throughput - maximizar o número máximo de jobs por hora), turn around time (tempo entre a submissão do trabalho e a obtenção do resultado), utilização do processador (manter o processador com elevada

ocupação);

- Tempo partilhado: tempo de resposta (responder rapidamente aos eventos desencadeados pelos utilizadores;
- Tempo real: cumprir metas temporais (deadlines para tratamento de acontecimentos), funcionamento com desempenho previsível.

9 - Deve ser invocada sempre que um recurso do sistema é atribuído ou libertado. O problema disso é que demora tempo.

10 - As duas classes são o CPU bound e o I/O bound. O que deve ter maior prioridade de acesso é o I/O bound pois tem menor tempo de utilização do CPU.

12 - As políticas são: tempo de execução partilhado (time-slices), prioridades, preempção e modificação dinâmica das prioridades.

13 - O time-slice é um quantum de tempo que limita o tempo de execução de um processo. O seu objectivo é permitir que todos os processos executáveis tenham oportunidade de dispor do processador ciclicamente. A sua implementação consiste numa lista de processos gerida em round-robin. A sua desvantagem é que pode conduzir a tempos de resposta elevados em situações de muita carga.

14 - As prioridades permitem definir a importância de um processo no processo de escalonamento, um processo mais prioritário tem maior probabilidade de dispor do processador. A prioridade pode ser: fixa (usual em processos de tempo real), dinâmica (consoante o comportamento do processo - usual nos sistemas de tempo virtual e normalmente privilegiando os processos interactivos (I/O bound)).

15 - Deve ser promovido quando é bloqueado e é relegado quando termina o seu time-slice.

16 - A preempção é a acção de retirar o processador a um processo em execução devido à existência de outro mais prioritário. Tem como objectivo permitir que os processos mais prioritários reajam rapidamente a um dado acontecimento (reactividade aos acontecimentos externos). A sua implementação é feita via despacho que deve ser chamado na sequência de todas as acções susceptíveis de modificarem os estados dos processos. As suas vantagens devem-se à mudança frequente de contexto.

17 - O UNIX encontrava-se dividido em 2 estruturas: proc (sempre mantida em memória para suportar o escalonamento e o funcionamento dos signals) e user (só era necessária quando o processo se estivesse a executar -> transferida para disco se houvesse falta de memória).

18 - Devido ao melhor hardware que existe hoje em dia.

19 - O processo em modo utilizador executa o programa que está no seu segmento de código e muda para modo sistema sempre que uma excepção ou interrupção é desencadeada, podendo essas ser provocada pelo utilizador ou pelo hardware. A mudança de modo corresponde à mudança para modo de protecção mais privilegiado do processador, mudança do espaço de endereçamento do processo utilizador para o espaço de endereçamento do núcleo, mudança da pilha utilizador para a pilha núcleo do processo. A pilha núcleo é usada a partir do instante em que o processo muda de modo e está vazia quando o processo se executa em modo utilizador.

20 - O escalonamento é preemptivo em modo utilizador, as prioridades núcleo são sempre superiores às prioridades utilizador.

21 - $\text{Prioridade} = \text{TempoProcessador}/2 + \text{PrioridadeBase}$

22 - Sim, escalabilidade. Porque o tempo em UNIX está dividido em épocas e cada época termina quando todos os processos usaram o seu quantum.

23 - Pois são aqueles que são mais essenciais de se executarem.

24 - Possibilita definir prioridades estáticas superiores às dinâmicas e são necessárias permissões.

25 - O despacho é invocado quando o processo em execução não pode continuar (bloqueou uma chamada de sistema ou terminou) ou quando o processo retorna ao modo utilizador.

26 - As chamadas de sistema do escalonamento são: nice(int val), int getpriority(int which, int id), setpriority(int which, int id, int prio).

27 - Para se criar um processo filho reserva-se uma entrada na tabela proc (UNIX) e verifica-se que o utilizador não excedeu o número máximo e subprocessos. Atribui-se um valor ao pid, normalmente um mero incremento de um inteiro mantido pelo núcleo e copia-se a imagem do pai. Dado que a região do texto é partilhada, apenas é incrementado o contador do número de utilizadores que acedem a essa região. As restantes regiões são copiadas (algumas incrementalmente) e retornam o valor do pid do novo processo para o processo pai, zero para o processo filho (coloca os

valores apropriados nas respectivas pilhas).

28 - Um processo pode ser terminado pela função `exit()`, a qual fecha todos os directórios, liberta directório corrente, liberta regiões de memória, actualiza ficheiro com registo da utilização do processador, memória e I/O, envia signal `death of child` para o processo pai (por omissão ignorado) e o registo `proc/task` mantém-se no estado `zombie` (permite ao processo pai encontrar informação sobre o filho quando executa o `wait`). Outra maneira de terminar é via a função `wait()`, onde esta procura o filho `zombie`, `pid` do filho e estado do `exit` são retornados através do `wait`, liberta a estrutura `proc` do filho e se não há filho `zombie`, o pai fica bloqueado.

29 - Utilizando a função `exec` que executa um novo programa no âmbito de um processo já existente. Verifica então se o ficheiro existe e é executável, copia argumentos da chamada a `exec` da pilha do utilizador para o núcleo (pois o contexto utilizador irá ser destruído), liberta as regiões de dados e pilha ocupadas pelo processo e eventualmente a região de texto (se mais nenhum processo a estiver a usar), reserva novas regiões de memória, carrega o ficheiro de código executável, copia os argumentos da pilha do núcleo para a pilha utilizador. O processo fica no estado executável e o contexto núcleo mantém-se inalterado: identificação e ficheiros abertos.

30 - As duas primitivas são `sleep_on` (bloqueia sempre o processo) e `wake_up` (desbloqueia todos os processos).

31 - Para o envio de um signal, o sistema operativo coloca a 1 o bit correspondente ao signal, este bit encontra-se no contexto do processo a quem o signal se destina e não é guardado o número de vezes que um signal é enviado. O UNIX verifica se há signals quando o processo passa de modo núcleo para modo utilizador ou quando entra ou sai do estado bloqueado. O LINUX verifica quando o processo comuta para estado "em exec"; no descritor do processo encontra-se o enfileiramento de rotina de tratamento de cada signal; a pilha de modo utilizador é alterada para executar a função de tratamento do signal e a função de tratamento executa-se no contexto do processo que recebe o signal como se fosse uma rotina normal.

32 - O gestor funciona por meio de objectos e referências. Permite interface uniforme para acesso e partilha dos recursos do SO, centralização das funções de segurança e autorização, sistema simples de recolha automática dos objectos não necessários: gerir as referências para saber quando um objecto pode ser libertado porque ninguém o usa.

5 - GESTÃO DE MEMÓRIA

- 1 - Qual o objectivo da gestão de memória?
- 2 - O que é o espaço de endereçamento?
- 3 - Quais as hierarquias de memória?
- 4 - O que é um endereço real? E um virtual?
- 5 - Quais as limitações de um endereçamento real?
- 6 - O que é o Overlay?
- 7 - Quais os tipos de partições e fragmentação do endereçamento real?
- 8 - Como é constituído o endereço virtual e quais os seus 2 tipos de blocos?
- 9 - Em que consiste o Princípio da Localidade de Referência? Porque motivo torna a gestão em blocos eficiente?
- 10 - O que é a segmentação?
- 11 - Qual o tipo de fragmentação da memória virtual segmentada? O que permite a protecção e a partilha de memória entre processos?
- 12 - O que é a paginação?
- 13 - Qual a protecção e a partilha de memória entre processos que esta permite?
- 14 - Qual a dimensão da tabela de páginas com endereços virtuais e 32 bits e páginas de 4kBytes?
- 15 - Qual a vantagem das tabelas de páginas multi-nível?
- 16 - Qual o problema da tabela de páginas invertida?
- 17 - Em que casos ocorre o swapping e o paging?
- 18 - Em relação aos algoritmos de gestão de memória, quais os tipos de decisões que o sistema operativo tem de tomar em relação à memória principal?
- 19 - Quais os algoritmos de reserva de segmentos e qual o seu critério de escolha de blocos livres?
- 20 - No que consiste o algoritmo Buddy?
- 21 - Quais as 3 situações em que pode ocorrer a transferência de blocos?
- 22 - Quais os possíveis critérios para decidir qual o processo a transferir para disco?
- 23 - Quais os algoritmos de substituição de páginas?
- 24 - O que são working sets?
- 25 - Quais os casos que provocam transferência (swapping)?
- 26 - Quais as consequências da criação de um processo?

RESPOSTAS:

1 - O objectivo é gerir o espaço de endereçamento dos processos, assegurar que cada processo dispõe da memória que precisa, garantir que cada processo só acede à memória a que tem direito (protecção), otimizar o desempenho dos acessos.

2 - É o conjunto de posições de memória que um processo pode referenciar.

3 - As hierarquias são: Memória principal física ou primária (acesso aleatório, tempo de acesso reduzido, custo elevado e reduzida dimensão, informação volátil, RAM+caches(+registos)) e Memórias Secundárias ou de disco (acesso aleatório por blocos, tempo de acesso elevado, custo reduzido e mais abundante, informação persistente).

4 - Um endereço real é o endereço indicado no programa e aquele que é acedido na memória principal. O endereço virtual são endereços indicados no programa convertidos em tempo real de execução, pelo MMU(unidade de gestão de memória do processador); caso a palavra referenciada esteja em memória principal, a MMU obtém o seu endereço real e acede à memória; caso contrário, a MMU avisa o SO para este carregar a palavra em causa.

5 - As limitações são: limitações devido ao espaço de memória física da RAM (resolução por overlays), risco de correrem 2 programas um sobre o outro.

6 - O Overlay consiste na possibilidade de executar programas com dimensão superior à memória principal em sistemas com endereçamento real.

7 - As partições podem ser fixas ou variáveis. A fragmentação é interna no primeiro caso e externa no segundo.

8 - O espaço de endereçamento é constituído por bloco e deslocamento. Os seus dois tipos de blocos são: segmentos e páginas.

9 - O princípio da localidade consiste no seguinte: quando se refere a uma palavra de memória com grande probabilidade vai-se referenciar as palavras vizinhas.

10 - A segmentação consiste na divisão dos programas em segmentos lógicos que reflectem a sua estrutura funcional e na gestão de memória que suporta abstração das linguagens de programação. O segmento é a unidade de carregamento em memória (eficiência) e protecção. A dimensão dos segmentos é limitada pela arquitectura e não pode exceder a dimensão da memória principal.

11 - É fragmentação externa. A protecção dá-se a nível da verificação de limites de endereçamento intra-segmentos, verificação e limitação dos tipos de acesso ao segmento de leitura, escrita e execução, processos diferentes têm tabelas de segmentos diferentes: espaços de endereçamento disjuntos e inacessíveis a terceiros. Para a partilha de memória entre processos basta colocar nas tabelas de segmentos dos processos em questão o endereço real do segmento a partilhar; os endereços virtuais usados para aceder ao segmento partilhado podem ser diferentes nos vários processos, a protecção dum segmento partilhado é definida para cada processo através da respectiva tabela de segmentos.

12 - Na paginação, o espaço de endereçamento virtual tem dimensão superior à da memória principal. A dimensão das páginas é constante o que influencia a fragmentação interna, o número de falta de páginas, o número de entradas das tabelas de páginas e listas de páginas são mantidas pelo sistema operativo. Quando se aumenta o tamanho das páginas: pode-se aumentar/diminuir o número de falta de páginas (depende do padrão de acesso do processo), aumenta a fragmentação interna, aumenta o tempo de resolução de falta de páginas, diminui o número de entradas na tabela de páginas (aumenta offset). O tempo de resolução esse é limitado pela lentidão do disco.

13 - A nível de protecção é feita a verificação dos tipos de acesso(leitura, escrita e execução), processos diferentes têm tabelas de páginas diferentes(espaços de endereçamento disjuntos e inacessíveis a terceiros). Partilha de memória entre processos é semelhante ao usado para memória segmentada, partilha de blocos lógicos(partilha de múltiplas páginas), e não é possível partilhar menos que uma página (versus arquitectura segmentada, partilha de uma divisão lógica do programa).

14 - O tamanho de página é $2^{12} = 4Kbs$. Logo o offset seria de 12 bits e o número de páginas 20.

15 - A vantagem é a tabela apresentar-se como uma solução para o dimensionamento da tabela de páginas, sendo a ideia não manter todas as tabelas na memória e usar dois apontadores e um deslocamento.

16 - A tabela mapeia páginas físicas em páginas virtuais, sendo o seu problema a latência na tradução. Uma solução é usar TLB mais hash table.

17 - Utiliza-se quando é necessário libertar espaço na memória física e o SO copia páginas para o disco (escolhe aquelas que previsivelmente não irão ser usadas brevemente).

18 - Os tipos de decisões que o sistema operativo tem de tomar em relação à memória principal são: reserva, transferência e substituição.

19 - Os algoritmos de reserva de segmentos são: best-fit(o menor possível), worst-fit(o maior possível), first-fit(o primeiro possível), next-fit(o primeiro possível a seguir ao anterior).

20 - Quando a memória livre é dividida em blocos de dimensão b^n , b designa-se por buddy. Um dos buddies é subdividido qtas vezes for preciso para se obter um bloco da dimensão desejada. Se possível, na libertação um bloco é re combinado com o seu buddy, sendo a associação entre buddies repetida até se obter um bloco com a maior dimensão possível e consegue-se um bom equilíbrio entre o tempo de procura e a fragmentação interna e externa.

21 - Pode ocorrer on request, on demand, por prefetching.

22 - Os critérios são: estado e prioridade do processo, tempo de permanência na memória principal, dimensão do processo.

23 - Os algoritmos são: Ótimo, Least Recently Used(LRU), Not Recently Used(NRU), FIFO.

24 - É o conjunto de páginas acedidas pelo processo no intervalo de tempo de um processo.

25 - Os 4 casos são: chamada fork, chamada brk, crescimento natural do stack, quando o SO precisa de espaço para carregar em memória um processo que estava swapped-out.

26 - Embora o fork duplique os segmentos de código, dados e pilha do pai, não é feita nenhuma cópia física de memória.