TODAS AS RESPOSTAS DADAS DEVERÃO SER CONFIRMADAS E SE HOUVER ALGUM UPDATE É FAVOR ACTUALIZAR E INCREMENTAR A VERSÃO DESTE .WORD

E CLARO, RE-DISTRIBUIR.

**exameA**

**1.**

**a)** Uma possível resposta é o facto de o SO gerir recursos, isto é, ele é responsável por gerir a memória virtual, CPU, também processos (acho) e qualquer tipo de dispositivos. Outra das vertentes é estender a máquina, isto é, fazer com que se possa comunicar com a máquina com mais facilidade, isto em termos de programação, pois existem inúmeras coisas que teríamos que saber e controlar, por exemplo para simplesmente ligar o cabo da impressora ao PC.

Fonte: <http://wiki.answers.com/Q/What_is_the_purpose_of_an_operating_system>

**b)**

**i)** Um processo é um programa em execução, isto é, é o programa, contando com o counter, variáveis, registos e ponteiro da pilha do mesmo.

**ii)** Tabela de Processos é uma tabela, que é da responsabilidade do kernel (núcleo) e esta tabela contem o process ID e o estado de todos os processos.

**iii)** Multiprogramação é a capacidade de um SO poder correr dois ou mais processos.

**Fonte:** <http://pt.wikipedia.org/wiki/Multiprocessamento> (iii) - o resto foi do .word SO Caderno de 2001

**2.**

**a)** A comunicação entre processos tem várias fases. Possivelmente a resposta era só dizer quais são, mas tentarei resumir todos.

**Condições de concorrência** – Basicamente o que isto nos diz é que os processos em alguns SO partilham o seu espaço onde podem ler e escrever, e por isso existe algo, chamado directório spool, que é basicamente um directório onde eles colocam essa informação, por exemplo, peço à impressora para imprimir algo, um processo coloca esse pedido no directório spool, e depois o processo da minha impressora irá ler periodicamente o spool para ver se existe alguma instrução para ele, neste caso existe por isso ele imprime, ao fazer isso remove o pedido do processo, para não repetir o pedido.

**Secções críticas** – Exclusão múltipla. Isto é, impedir que processos acedam a dados partilhados ao mesmo tempo, ou seja, evitar condições de concorrência, na qual no caso acima, 2 impressoras, suponhamos, acedessem ao mesmo tempo ao directório spool e imprimissem o ficheiro. Toda esta actividade de aceder a locais de memória partilhada chama-se uma secção crítica, e para a evitar existem 4 passos:

1. Não podem existir 2 processos ao mesmo tempo numa secção crítica.
2. Não podem existir pressupostos acerca da velocidade ou número de CPU’s.
3. Nenhum processo que se encontre fora da sua secção crítica deve bloquear outros processos.
4. Nenhum processo deve esperar infinitamente para entrar na sua secção crítica.

**Trocas de mensagens** – Os processos podem trocar mensagens entre si e tratar essa informação de forma adequada.

**Aspectos dos sistemas de trocas de mensagens** – Basicamente é como em redes, numa rede se uma máquina manda mensagens para outra esta pode-se perder, por isso existe algo chamado acknowledgement que é um tipo especial de mensagens, caso o remetente não receba uma acknowledgement este pode enviar de novo a mensagem.

**Escalonamento de processos** – Toda esta parte é da responsabilidade do escalonador. Este tem o objectivo de decidir que processo vai ser executado e durante quanto tempo. O escalonamento de processos também contem uma etiqueta:

1. JUSTIÇA – assegurar que cada processo obtém uma fatia justa de tempo de CPU.
2. EFICIÊNCIA – Manter o CPU ocupado 100% do tempo.
3. TEMPO DE RESPOSTA – Minimizar o tempo de resposta a utilizador interactivos.
4. TURNAROUND – Minimizar o tempo que os utilizador batch esperam pelo output. ( Batch - ??? )
5. THROUGHPUT – Maximizar o número de tarefas executadas por hora.

**Nota:** Preemptive scheduling – Estratégia de escalonamento que permite que processos logicamente executados possam ser suspensos.

**Algoritmo Round-Robin** – A cada processo é atribuído um intervalo temporal, chamado Quantum, no qual pode se executar. Caso o processo esteja em execução quando termina o seu quantum, o CPU simplesmente é atribuído ao próximo processo. O Quantum não pode ser nem alto, nem baixo, se for muito baixo, há troca demasiada rápida entre processos e isso afecta a eficiência, caso seja demasiado grande, há processos que sofrem com isso.

**Escalonamento por prioridade** – Neste caso os processos têm prioridades, sendo assim os com maior prioridade são dados tempo de CPU. Para evitar que os mais baixos se prejudiquem em demasia a cada iteração do relógio, o escalonador baixa a prioridade de alguns processos.

**Múltiplas Filas** – Neste caso, existem classes de prioridades, e conforme a classe são dados intervalos temporais. A mais elevada é dada 1, a seguinte 2, a próxima 4, etc… Quando uma classe esgota o seu intervalo temporal decresce na prioridade, ou seja, é movido para a classe abaixo.

Trabalho mais curto primeiro – Basicamente o que esta estratégia nos diz é que:

|  |  |  |  |
| --- | --- | --- | --- |
| 8m | 4m | 4m | 4m |
| A | B | C | D |

Se tivermos uma tabela deste género, o tempo TURNAROUND para o processo A é de 8 mins, B é de 8+4 mins, C 8+4+4 mins e D 8+4+4+4 mins. Isto dará uma média de 14 mins. Esta estratégia o que faz é re-organizar a ordem dos processos ficando desta forma:

|  |  |  |  |
| --- | --- | --- | --- |
| 4m | 4m | 4m | 8m |
| B | C | D | A |

Agora é B 4 mins, C 4+4 mins, D4+4+4 mins e A 4+4+4+8 mins. Parecendo que não agora a média é de 11 mins.

**Escalonamento Garantido** – Esta estratégia é basicamente relacionada com performance. Ela avalia o número de utilizadores ligados, por exemplo, nós recebemos 1/nº\_utilizadores da capacidade do CPU. Mas torna-se um pouco mais complexo, porque o sistema precisa de saber não só quantos utilizadores estão ligados mas quanto é que cada um já gastou de tempo de CPU e à quanto tempo está ligado.

Politica VS Mecanismo – Como o escalonador actua por conta própria, ele não quer saber quais dos processos filhos de um processo são os mais, ou menos, importantes, mas como isso é algo crucial para a eficiência de um SO a solução é a separação entre o mecanismo de escalonamento e a politica de escalonamento. Isto significa que o algoritmo de escalonamento, possa ser parametrizado pelos processos utilizadores.

**Escalonamento a dois níveis** – Devido aos processos swapped out, o tempo de troca de processos entre estes e os que se encontram em memória principal é elevado, logo uma das soluções para este problema é usar esta estratégia. Basicamente existem 2 escalonadores, um para processos da memória principal e outro para processos swapped out, que troca processos que estejam À muito tempo swapped out por outros que estejam à muito tempo na memória virtual. Critérios a ter em conta para o escalonador da memória de disco:

1. Quanto tempo passou desde que um processo foi transferido.
2. Que quantidade de tempo de CPU é que o processo teve recentemente.
3. De que tamanho é o processo.
4. Qual é a prioridade do processo.

**Nota:** Swapped out – Processos que por falta de memória, são mantidos em disco.

**3. (TODO ESTE EXERCÍCIO PRECISA DE CONFIRMAÇÃO!!!!)**

1. Execução em modo utilizador – Este tipo de execução é onde é utilizado o espaço de memória do utilizador, ou seja as threads aqui criadas, o kernel não sabe nada sobre elas.
2. Execução em modo Kernel – Este tipo de execução é feito no espaço de memória do núcleo.
3. Pronto a executar – É um processo que está pronto a ser executado, esperando que o escalonador lhe dê tempo de CPU.
4. Bloqueado – Processo que está à espera que o input esteja disponível.
5. Pronto, em memória secundária – Se por memória secundária é a memória do disco, então é um processo que foi Swapped Out e está à espera de entrar para a memória principal para ser executado.
6. Bloqueado, em memória secundária – Se por memória secundária é a memória do disco, então é um processo que está à espera que o input esteja disponível, para ser swapped in.
7. Criação – Se for em UNIX é usado o comando fork() caso contrário, em Windows, é usado o comando CreateProcess.
8. Extinção – Processo que tem estado como exit e está à espera de ser terminado pelo processo pai.

**4.**

**a)** Segmentos são no fundo processos ou “buracos” entre processos, logo o tamanho deles supostamente deverá variar, visto que há processos maiores e menores.

**b)** Tudo o que sei é que existe vários algoritmos de paginação e é isso que vou passar a explicar. **FONTE:** <http://en.wikipedia.org/wiki/Page_fault>

**Algoritmo de Substituição –** Quando ocorre uma page fault, isto é, quando um processo tenta aceder a uma página que não está na memória física, é necessário colocar essa página na memória física, mas para isso acontecer tem que se remover uma página para dar lugar a essa.

**Algoritmo óptimo –** Quando ocorre uma page fault, existe um conjunto de páginas em memória e uma destas vai ser referenciada na próxima instrução que será executada e as restas páginas só serão referenciadas x instruções mais tarde.

**Algoritmo NRU (Not Recently Used) –** Para que os SO obtem estatísticas úteis acerca da paginação, contêm 2 bits associados a cada página; M e R. R de referenciada (página lida ou escrita) e M de modificada. Sempre que os bits são postos a 1, só voltarão a 0 se o SO assim o fizer. No fundo, este algoritmo remove uma página ao acaso do grupo de ordem mais baixa não vazio.

**Algoritmo FIFO –** Tal como o nome First In First Out, neste algoritmo a primeira página a entrar é a primeira página a sair.

**Algoritmo second chance –** É um update ao algoritmo FIFO, na qual páginas demasiado referenciadas não são removidas. Isto vê-se pelo bit R do algoritmo mais antigo, se for 0 quer dizer que não foi referenciada, 1 foi referenciada.

**Algoritmo Relógio –** Basicamente este algoritmo é como se fosse o ponteiro de um relógio que aponta para a última página. Sempre que ocorre uma page fault o bit R dessa página é verificado, se 0 ela é despejada, caso contrário o ponteiro move-se para a próxima página e o bit R passa a 0.

**Algoritmo LRU (Least Recently Used) –** Este algoritmo tenta prever que páginas serão futuramente referenciadas, baseando-se na observação das páginas mais utilizadas pelas últimas instruções.

**exameB**

**1.**

Como gestor de recursos creio que o que o SO tenta gerir são o CPU, espaço de memória e threads. As entidades em concorrência penso que sejam os processos. Os objectivos do SO como gestor de recursos são fornecer de forma adequada e controlada a alocação do processador, das memórias e de dispositivos por todos os programas que competem por recursos.

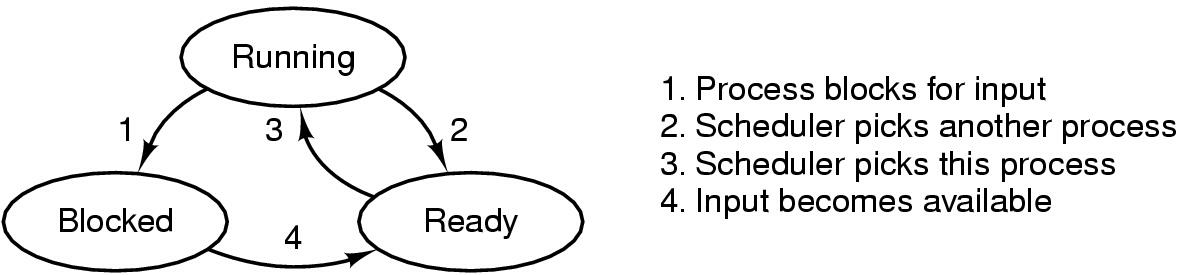
**2.**

**Multiprogramming –** É a alocação da memória do computador para mais do que uma aplicação.

**Multiprocessing –** É a utilização de um ou mais CPUs num sistema.

**Escalonador do SO –** Parte do SO que faz a decisão de que processo irá utilizar os recursos do computador e durante quanto tempo.

**3.**



**5.**

1- Detecção e recuperação. Deixar deadlocks ocorrer, detectá-los e actuar.

2- Desvio Dinamico através da cuidadosa alocação de recursos.

3- Prevenção, negando 1 das 4 condições necessárias para criar 1 deadlock.

**exameC**

**1.**

**a)** Memória, CPU e I/O.

**b)** A repartição é feita pelo escalonador?

**c)** Escalonador?

**2.** Respondido anteriormente.

**3.** …?

**4.**

**First-Fit:** Faz parte da gestão de memória com listas ligadas e o que nos diz é que este é um algoritmo no qual o gestor pesquisa toda a lista de segmentos até encontrar um espaço suficientemente grande.

**Best-Fit:** Procura toda a lista em busca do espaço mais adequado ao pedido, mas em vez de dividir um espaço que pode vir a ser utilizado mais tarde, este algoritmo tenta encontrar um “buraco” com o tamanho mais aproximado possível.

(não percebo a pergunta, supostamente eles dizem que é 48 kb separado por blocos de 8, ou seja 8, 8, 8, 8, etc… Se fazem pedidos de ocupar 16kb, ocupam-se 2x 8kb, não existe qualquer ordem, porque se os espaços de memória têm todos os mesmo tamanho a lista que teria de ser percorrida seria a lista dos pedidos, e não da memória disponível.)

**exameD**

**1.** Gerir recursos e estender a máquina. Agora… Módulos funcionais?

**2.** **Pseudo-paralelismo –** Troca rápida, entre programas, feita pela CPU.

**3.** Guardar a informação é necessário pois, no caso do texto, se um texto está a ser escrito e perde a vez para um outro processo, é necessário guardar a informação de onde parou, para continuar a escrita no ponto correcto. (incompleta)