** //
****

%%

SUGESTÃO DE APRESENTAÇÃO DE RESPOSTA:

%%

#1 Isto é uma questão?

@1 Isto é a resposta à questão #1.

%%%%%%%%

ALÍNEAS:

%%%%%%%%

#2 Isto é uma questão? a) isto é uma alínea desta questão?; b) isto é outra alínea desta questão?;

@2 Isto é a resposta à questão #2.

@2a Isto é a resposta à alínea a) da questão 2.

@2b Isto é a resposta à alínea b) da questão 2.

** //
****

Perguntas extraídas de testes preparados pelo Prof. Cardoso

e Cunha na edição da cadeira de 2012/13

#1 Quais os principais objectivos de um Sistema de Operação (SO)?

@1 Os principais objectivos de um SO são gerir os recursos do sistema e oferecer uma interface de acesso. O SO gere recursos do tipo lógico e assim permite abstração dos recursos físicos. Para tal o mesmo cria uma máquina virtual sobre a física que oferece os recursos básicos ao desenvolvimento de aplicações, podendo assim ser independente do hardware onde executa.

#2 Quais as diferenças entre uma chamada de subrotina, tal como é suportada por uma instrução máquina, e uma chamada ao SO, tal como é suportada por uma instrução máquina?

A grande diferença consiste no facto em que a chamada aos SO, tal como suportada por uma instrução máquina corre instruções privilegiadas a nível do hardware (o que implica o CPU mudar para modo supervisor), enquanto uma chama de subrotina, esta alteração de modo não é necessária (apenas terá de o fazer caso exista uma system call na subrotina)

@2 Uma chamada de sub-routina corre em modo utilizador mas não tem acesso a zonas criticas. Se eles precisam de aceder a zonas criticas eles chamam serviços que necessitam do kernel, ou seja chamadas de SO.
Chamadas de SO são chamadas que interrompem o programa, os conteúdos do programa são guardados até que a chamada de SO termina e o programa possa retomar

Application Programming Interface (API) is a set of functions, objects, protocols or datastructures for the support of application development for developers/programmers. It is actually a kind of function definition which specifies how to make available of a specific service of the system/OS. The API's are available from library or from Operating system itself. Whenever a programmer need a specific service from OS, he/she can use appropriate API to do that. Processes in a system are run in different modes, process run in user mode have no access to the privileged instructions. If they

> want perform any privileged instructions or need of any services they request kernal for that service through System Calls. System calls are made by way of software interrupt. This is actually a
> request for the service whereas API is a function description that can be used by the programmer for his programs, its like a tool used to obtain a specific task in his programs. A system call is

generally made by an application programme to the operating system.this instruction interrupts the current executing programme and transfers control to the interrupt routine.The contents of theexecuting programme are saved.After the interrupt routine finishes its function,control is transferred back to the executing programme.

OS @ boot (kernel mode)	Hardware	
initialize trap table		
	remember address of... syscall handler	

OS @ run (kernel mode)	Hardware	Program (user mode)
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs from kernel stack	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call
		trap into OS
	save regs to kernel stack	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs from kernel stack	
	move to user mode	
	jump to PC after trap	
		...
		return from main
		trap (via `exit()`)
Free memory of process		
Remove from process list		

#3 Qual a diferença entre estes dois regimes de execução do CPU: modo utilizador e modo supervisor? Diga para que serve esta distinção.

@3 O modo utilizador e o modo supervisor diferenciam-se principalmente quanto às instruções máquina que executam. O modo supervisor tem a possibilidade de correr qualquer instrução da máquina quer esta seja de natureza normal ou privilegiada (manipulação de hardware: controladores, interrupções, gestão de memória). Já o modo utilizador apenas consegue correr instruções normais. Caso este precise de acesso a mais memória, acesso a um periférico deverá explicitamente requisitar ao SO através de uma interrupção de software (trap). Esta distinção tem como objectivo melhorar a segurança do SO, uma vez que desta forma o utilizador não tem acesso, por exemplo, a registos que de forma alguma devem ser alterados/suprimidos.

#4 Explique em que medida o mecanismo de interrupção de programas é essencial para o SO conseguir implementar o regime de operação de multiprogramação.

@4 Num regime de operação de multiprogramação é necessário o mecanismo de Interrupções para que o SO tenha a possibilidade de colocar o CPU a correr diversas actividades concorrentes, ou seja, sem este a execução concorrente é impossível.

#5 Quais as diferenças entre monoprogramação e multiprogramação?

@5 Num sistema de monoprogramação apenas é possível correr um processo de cada vez, isto é, uma vez que um processo "ganha" o CPU, este apenas o liberta quando a sua execução termina. Um exemplo de desvantagem deste método é que se um processo fica bloqueado à aguardar I/O o processador fica sem correr qualquer outro processo. Para colmatar esta situação foi criado o sistema de multiprogramação que através de interrupções de sistema permite que o CPU desenrole diversas actividades concorrentes. Usando o mesmo exemplo, caso um processo necessite de I/O este perde de imediato o CPU, dando lugar a outro processo para usufruir do mesmo, sendo assim possível melhorar a performance do CPU.

#6 O que é o contexto de execução de um programa? Descreva os seus principais elementos?

@6Execução de um processo: _start (endereço guardado no start address,

especificado pelo compilador C) obtém argumentos e variáveis de ambiente para

o mapa do processo, iniciando a pilha, e lança a função main.

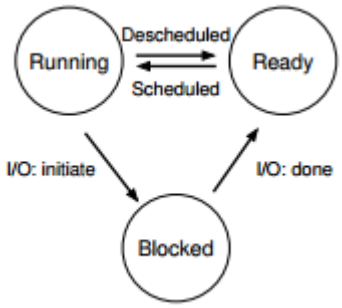
#7 Explique as principais acções que o SO efectua quando um utilizador ao terminal selecciona uma aplicação (num ficheiro executável) para activação? Indique também que estruturas de dados devem existir no SO, para implementar aquelas acções.

OS	Program
Create entry for process list	
Allocate memory for program	
Load program into memory	
Set up stack with argc/argv	
Clear registers	
Execute **call** main()	
	Run main()
	Execute **return** from main
Free memory of process	
Remove from process list	

#8 Considere um Sistema de Operação (SO) da família Unix.

a) Desenhe o diagrama de estados lógicos de um processo e todas as transições entre estados, durante a execução do processo num sistema Unix, desde que é criado até que é terminado. Indique os nomes dos diversos estados, defina as situações a que corresponde cada estado e, para cada transição do diagrama, indique as condições que a desencadeiam.

- **Running**: In the running state, a process is running on a pro cessor. This means it is executing instructions.
- **Ready**: In the ready state, a process is ready to run but for som reason the OS has chosen not to run it at this given moment.
- **Blocked**: In the blocked state, a process has performed som kind of operation that makes it not ready to run until som other event takes place. A common example: when a proces initiates an I/O request to a disk, it becomes blocked and thu some other process can use the processor.



b) Para cada uma das chamadas ao sistema Unix indicadas em b1) e b2), descreva os seus argumentos e os resultados da sua invocação, diga quais as acções internas que desencadeia a nível hardware e SO e quais as transições de estados desencadeadas no diagrama que apresentou na alínea a).

b1) wait

wait() faz o parente esperar que o filho acabe de executar as suas instruções.

Quando child acaba, wait() retorna para o pai.

b2) fork

fork() (system call) cria um novo processo.

Pid = process identifier

cria mapa de memória do filho

start address do filho := PC = Eret

guarda start address do filho em tabela de processos do SO

copia informação do pai

insere filho na fila prontos

empilha pid-filho na pilha do pai

empilha pid 0 na pilha do filho

retorna ao programa do pai

#9 Considere a execução concorrente de um programa (P1) com comportamento CPU-bound e de outro programa (P2) com comportamento IO-bound, num sistema Unix para um computador que só tem um CPU.

a) Explique as características do comportamento IO-bound

Sistemas que fazem uso intensivo de inputs e outputs.

Explique as características do comportamento CPU-bound num SO como o Unix, em que pontos da sua execução pode um programa perder o controlo do CPU? Para cada caso explique as correspondentes acções desencadeadas pelo hardware e/ou pelo sistema de operação.

Quando várias threads são criadas, elas podem não ser corridas pela ordem pela qual elas foram chamadas, chamando as threads A,B,C o scheduler pode correr primeiro a thread B depois a A, depois a C. Uma das soluções é esperar que 1 thread termine para começar a outra, mas isso implica esperar que cada thread termine.

Outro problema é quando várias threads utilizam e modificam variáveis global partilhada situadas em zonas criticas ao mesmo tempo, que irão gerar respostas não determinísticas.

Isto acontece devido ao scheduler e o acesso a zona critica não ser controlado.

Solução: bloquear acesso a variável global partilhada via mutex/locks.

a) Apresente, em pseudo-código, todas as acções desencadeadas quando um programa executado em modo utilizador, invoca uma instrução para fazer a chamada ao sistema de operação fork() no Unix. Deve indicar quais as acções suportadas automaticamente pelo hardware do computador e quais as acções suportadas pelo SO e o seu efeito nas estruturas relevantes do núcleo do sistema.

cria mapa de memória do filho

start address do filho := PC = Eret

guarda start address do filho em tabela de processos do SO

copia informação do pai

insere filho na fila prontos

empilha pid-filho na pilha do pai

empilha pid 0 na pilha do filho

retorna ao programa do pai

b) Apresente, em pseudo-código, todas as acções desencadeadas (seja por hardware, seja pelo SO), num sistema de multiprogramação, para efecutar a comutação de contextos entre dois processos P1 e P2, no caso em que o processo P1, quando em execução, atinge o limite do seu Time-slice, e sendo P2 o novo processo que o SO escolherá para execução. Desenhe, no diagrama de estados dos processos, quais as transições de estados sofridas por P1 e P2 neste caso.

OS @ boot (kernel mode)	Hardware
initialize trap table	
	remember addresses of...
	syscall handler
	timer handler
start interrupt timer	
	start timer
	interrupt CPU in X ms

OS @ run (kernel mode)	Hardware	Program (user mode)
		Process A
		...
	timer interrupt	
	save regs(A) to k-stack(A)	
	move to kernel mode	
	jump to trap handler	
Handle the trap		
Call `switch()` routine		
save regs(A) to proc-struct(A)		
restore regs(B) from proc-struct(B)		
switch to k-stack(B)		
return-from-trap (into B)		
	restore regs(B) from k-stack(B)	
	move to user mode	
	jump to B's PC	
		Process B
		...

#10 Explique a diferença entre um programa com comportamento CPU-bound e outro com um comportamento IO-bound. Explique de que modo um sistema de operação lida com esses comportamentos, em cada um dos seguintes casos:

CPU bound é quando o tempo de processamento depende do processador.

IO bound fazem uso extensivo de IO.

a) um sistema não interactivo (batch) de monoprogramação

b) um sistema interactivo de monoprogramação

c) um sistema não interactivo (batch) de multiprogramação

O CPU corre as instruções prontas a executar durante o intervalo de tempo que espera pelos inputs.

d) um sistema interactivo de multiprogramação

O CPU corre os programas pelo Schedule ou se é interrompido por um IO.

a) Explique o conceito de canal virtual ou file descriptor no sistema Unix. Dê um exemplo do seu interesse prático e indique quais as chamadas ao sistema que o manipulam. b) Explique o que é a multiprogramação e diga quais as vantagens de um sistema de operação que a suporta. c) Quando um programa é executado em modo utilizador, quais as acções que lhes estão proibidas? Explique porquê. d) Dê um exemplo de um caso em que, num sistema de operação com um único utilizador, o conceito de multiprogramação seja importante.

#11 Explique as diferenças entre um sistema de operação com monoprogramação e um sistema de operação com multiprogramação e os efeitos de cada um desses tipos de sistemas nos principais factores de eficiência de um sistema de operação (taxa de utilização de CPU e periféricos, débito de trabalhos e tempo de resposta ao utilizador).

Mono-programming corre só um programa de cada vez.

Multi-programming corre vários programas de cada vez.

#12 Considere, no sistema Unix, a chamada ao SO fork que é utilizada no seguinte programa: (1) int p1, p2, p3, p4; (2) p1 = fork(); (3) p2 = getpid(); (4) p3 = fork(); (5) p4 = getpid(); (6) exit(0); Diga quantos processos são criados pela execução deste programa. Justifique a sua resposta e, para cada processo criado, indique quais as instruções que esse processo executa, desde a primeira até à última.

Considere agora o seguinte programa no Unix: (1) int p1, p2, p3; (2) p1 = fork(); (3) if (p1 > 0) { (4) p2 = fork(); (5) } else { p3 = fork();} (6) exit(0); Diga quantos processos são criados pela execução deste programa. Justifique a sua resposta e, para cada processo criado, indique quais as instruções que esse processo executa, desde a primeira até à última.

Perguntas de testes preparados pelo Prof. Paulo Lopes em 2013

#1 Considere o programa que se segue

```
#include <stdio.h>
#include <stdlib.h>
#define DOIS 2

int x = 5;
int main( int argc, char *argv[] ){
  int y;
  int *z;
  if( argc != DOIS ) { printf( "%s num\n", argv[0] );}

  y = atoi( argv[1] );
  z = &x;
  printf( "%d\n", *z + y );
  return 0;
}
```

a) Quando se usa o compilador (assuma Linux e gcc) para produzir a versão executável, quais são as fases distintas do processo que se sucedem até estar produzido o "ficheiro executável"? Para cada fase diga: o nome da fase (ou, em alternativa, o nome do programa que

"implementa" essa fase), quais são o(s) ficheiro(s) de entrada, e quais são o(s) de saída, de que tipo são esses ficheiros (e.g., texto, binário, etc.).

b) Considere que o programa executável é lançado em execução; faça um esboço do mapa do espaço de endereçamento do processo, identificando nesse espaço as zonas em que ficam as variáveis (x, y, z), os argumentos (argc, argv), as funções usadas (printf, atoi) e o código gerado para o programa principal (main).

#2 Considere a seguinte estrutura de dados que suporta uma lista ligada, e as respectivas operações (das quais apenas uma tem o código visível):

```
typedef struct item { int valor; struct item *seguinte } Elemento;
Elemento *cabeca=NULL;

void insereNaCabeca(int val) {
 Elemento tmp= malloc(sizeof(Elemento));
 tmp->valor=val; tmp->seguinte=NULL;
 if (cabeca) tmp->seguinte= cabeca;
 cabeca=tmp;
}

void PercorreLista(void) {...}

int ProcuraNaLista(int val) {...}

void Apaga(int val) {...}
```

Considere um programa multi-threaded em que as funções definidas anteriormente (insereNaCabeca, Apaga, PercorreLista, ProcuraNaLista) são executadas concorrentemente pelas diferentes threads.

Assuma que a lista é global, isto é, a variável cabeça é global.

Usando como exemplo a função insereNaCabeca, mostre o que nela tem de fazer para que a execução concorrente dê resultados correctos, usando unicamente construções (estruturas de dados e

funções) da norma Pthreads. Nota: se precisar de declarar variáveis globais, declare-as junto da estrutura Elemento; se achar necessário, pode alterar a estrutura. Nota: não escreva o código das outras funções!!!

Perguntas de testes preparados pelo Prof. Pedro Medeiros

#1 Considere que pretende instalar um sistema operativo (SO) que suporta múltiplos utilizadores e em que cada utilizador pode lançar múltiplos processos (exemplo: Linux, Windows). Indique que tipo de características terá de ter o hardware para que seja possível instalar este tipo de SO.

#2 Considere um sistema operativo que suporta multi-programação e que utiliza uma fila única de processos prontos; o algoritmo de atribuição de CPU aos processos é o round-robin (RR).

Suponha que existem 5 processos na fila de processos prontos que têm sempre características cpu-bound. Explique porque é que esta estratégia RR garante uma distribuição equilibrada do CPU pelos 5 processos.

@2a Uma vez que os processos cpu-bound tem por defeito consumir apenas os recursos do CPU é necessário que este seja libertado para que outros processos possam usufruir do mesmo. Uma vez que temos 5 processos cpu-bound a concorrer por um único CPU a utilização do Round Robin é de facto uma forma de distribuição equilibrada uma vez que este método tem como definição a partilha do CPU com base em Time Slices, ou seja, o relógio do CPU de x em x tempo lança uma interrupção onde se verifica se o time slice atribuído ao processo já passou, caso não, o processo continua a correr até esgotar o seu time slice (verificado em posterior interrupção); caso sim, o contexto de execução do processo é gravado e é dado CPU ao processo que se encontre em fila de espera (será previamente carregado o contexto de execução deste processo). Desta forma todos os processos correm durante o mesmo tempo, tendo assim uma partilha justa do CPU.

a) Considere que no sistema aparecem 2 processos que têm sempre características io-bound. Explique como poderia modificar o algoritmo RR para beneficiar este tipo de processos.

@2b O algoritmo Round Robin não é eficaz para processos io-bound uma vez que estes raramente ocupam muito tempo de CPU, podendo libertar este recurso muito rapidamente. Desta forma o RR deverá ser alterado usando recurso a um sistema de prioridades. Os processos io-bound deverão ter uma prioridade superior (irão correr primeiro) que os cpu-bound, uma vez que mais rapidamente irão libertar o cpu para estes últimos utilizarem. Um exemplo de algoritmo que actua desta forma é o MLFQ (Multi-Level Feedback Queue)

#3 Considere um sistema operativo que suporta multi-programação, isto é, tem em simultâneo vários processos carregados na RAM. O CPU suporta dois modos de operação: modo supervisor – em que é possível executar todas as instruções máquina incluindo as privilegiadas – e modo utilizador em que as instruções privilegiadas não podem ser usadas. Diga que tipos de instruções máquina é que só podem ser executadas em modo supervisor.

@3 Manipulação de Hardware, Controladores, Interrupções e Gestão de Memória.

#4 Suponha um sistema operativo que suporta multi-programação e em que um processo pode perder o CPU porque se esgota a fatia de tempo que lhe está atribuída. Como é que o sistema operativo garante que, quando o processo volta a correr, a RAM que lhe está atribuída não foi alterada por outros processos?

@4 Ao perder o CPU, é necessário que seja gravado o estado da computação, para que quando este volte ao estado Execução se mantenha tal e qual como o deixou, desta forma os registos do CPU são guardados no descritor do processo, o SO impede que qualquer outro processo altere o conteúdo da imagem (RAM reservada ao processo) e também é preservada a tabela de canais abertos. Ou seja, a RAM atribuída não é alterada por outros processos visto o SO assegurar que nenhum outro processo tem acesso a este espaço de endereçamento.

#5 Considere um sistema operativo que suporta multi-programação e suporta escalonamento Round Robin com fatia de tempo. Que informação está guardada no descritor de um processo (PCB)?

@5 No descritor de processo estará gravada toda a informação necessária para recolocar o processo em estado Executar. Ou seja, Conteudo dos Registos do CPU; descrição do endereço inicial da memória e respectiva dimensão; Tabela de Canais Abertos e directoria corrente; bem como o estado do processo e o respectivo PID (Id do Processo).

#6 Considere um sistema de escalonamento com múltiplas filas com a estratégia MLFQ (multi-level feedback queue). A cada fila corresponde um valor numérico (prioridade). A fila mais prioritária está associada à prioridade 0 e a menos prioritária à prioridade 200. Um processo com a prioridade X só obtém CPU quando todas as filas com prioridade entre 0 e X-1 estiverem vazias. Quando um processo é criado é-lhe atribuída a prioridade 100. De 1 em 1 segundo o SO recalcula a prioridade de um processo pelo algoritmo if (processo passou mais de 20 ms no estado bloqueado no último segundo) prioridade = prioridade – 5 else prioridade = prioridade + 5 Diga que tipo de processos é favorecido por este algoritmo de escalonamento. Quais as vantagens, do ponto de vista do SO, desse favorecimento?

@6 O tipo de processo favorecido por este algoritmo é o cpu-bound, uma vez que poderão voltar a usufruir do CPU mais rapidamente do que se se mantivesse sempre em prioridade mínima. Por exemplo se um processo cpu-bound estiver em prioridade mínima e o sistema estiver constantemente a receber processos io-bound o primeiro nunca teria a oportunidade de executar. Assim com um boost de prioridade o mesmo poderá executar e libertar o CPU mais rapidamente.

#7 Quando um programa P é executado no sistema operativo X que suporta apenas um processo em memória de cada vez (mono-programação) é produzido um ficheiro de resultados R. Considere agora que o mesmo programa P é executado sobre o sistema de operação Y que suporta múltiplos processos em memória (multi-programação) e que usa uma estratégia de atribuição de CPU aos processos baseada em fatias de tempo (time slices). Diga como é que o sistema operativo Y garante que, quando P é executado, é produzido um ficheiro com conteúdo igual ao de R.

@7 Será garantido o mesmo resultado R em ambos os SO (X e Y) uma vez que no sistema Y quando um time slice termina o estado da computação do processo P é salvaguardado e só depois será executado um novo processo. Quando for novamente vez do processo P executar o estado da computação previamente guardado é retomado, dando a ilusão ao processo que nunca foi interrompido, fazendo com que o resultado final seja R tal como no SO X.

#8 Considere um sistema operativo que suporta multi-programação e que utiliza uma fila única de processos prontos; o algoritmo de atribuição de CPU aos processos é o round-robin (RR). Admita uma situação em que o sistema operativo tem de repartir o CPU entre 5 processos que fazem computação intensiva (CPU bound). Explique porque é que esta estratégia RR garante uma distribuição equilibrada do CPU pelos 5 processos.

@8 Ver resposta @2a.

#9 Considere um sistema de operação monolítico como o LINUX.

a) Explique a finalidade das chamadas ao sistema

e a relação entre as chamadas ao sistema e a utilização

dos modos supervisor e utilizador.

@9a As chamadas de sistema tem como função solicitar explicitamente ao SO a execução de funções privilegiadas. Uma vez que o modo utilizador não acesso a este tipo de funções é através deste tipo de chamadas que é possível gerar uma interrupção de software (trap) que muda o modo do CPU para Modo Supervisor e desta forma será possível executar a função privilegiada necessária.

b) Compare a chamada de um procedimento local com a chamada de uma

função de sistema, nomeadamente em termos do tempo dispendido em cada uma

das operações.

@9b Em termos de tempo dispendido uma função de sistema é mais lenta que uma chamada de um procedimento local, uma vez que bastantes mais instruções terão de ser executadas para que esta forneça o resultado esperado.

#10

a) Compare, quanto aos objectivos, a política de gestão do processador

num sistema dedicado ao processamento de trabalhos não-interactivos (batch) com a de um sistema utilizado por múltiplos utilizadores interactivos (time--sharing)

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.

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 optimizar a utilização do processador.

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.

b) Um sistema de operação usa um algoritmo de escalonamento do

processador que privilegia os processos que consumiram recentemente pouco

tempo de CPU. Explique porque é que este algoritmo favorece os processos

I/O bound'' sem penalizar demasiado os processos CPU/Bound''.

Porque reduz o time slice, o que é vantajoso porque os processos IO são muito rápidos e assim liberta mais tempo para os processos que utilizam o CPU.

c) Compare as políticas de escalonamento de processos usadas

num sistema interactivo multi-utilizador com as que são usadas num sistema

de tempo real que é usado num computador que trata do controlo de tráfego

aéreo de um aeroporto.

Sistema interactivo multi-utilizador permite vários utilizadores usarem os recursos do CPU simultaneamente. O SO tem de garantir que o uso é equilibrado de modo a que cada progama usa os recursos separadamente e suficientemente para que no caso que haja algum problema, não afecte toda a comunidade de utilizadores.

Sistema em tempo real gere os recursos do computador de modo que cada operação executa em precisamente o mesmo tempo, e possui uma fraca user-interface.

#11 Indique que tipo de informação deve estar guardada no descritor (PCB) de

um processo ``pesado''. Idem para o descritor de um processo leve.

Pid, registos do programa (como o counter), status, prioridade, stack pointer, frame pointer, adress space.

#13 Compare as políticas de escalonamento de processos usadas

num sistema interactivo e multi-utilizador,

com as que são utilizadas num sistema vocacionado para

o processamento de trabalhos não-interactivos.

Processos não interativos, isola-se do utilizador, aloca recursos, e controla os programas que correm no sistema.

Um sistema interativo multi-utilizador são todos eles que pedem uma fatia do CPU e todos eles usao o computador.

#14 Considere um sistema de operação monolítico como o LINUX. Indique, em detalhe, o que acontece quando:

a) Um processo faz uma chamada ao sistema do tipo \emph{getpid()}?

@14a Quando o sistema faz uma chamada do tipo getpid() irá receber um valor inteiro que tem como significado o identificador do processo cujo sobre o qual a chamada foi executada.

b) Um processo chama a função malloc()?

@14b Quando chamada a função malloc() são alocados size bytes e é retornado um apontador para a memória alocada. A memória não é limpa.

c) Um processo faz uma chamada ao sistema \emph{read({ ...)} e não há dados disponíveis.

@14c Quando feito a chamada read({…}) o processo que está em estado Pronto é colocado em estado Executar e posteriormente no estado Bloqueado para receber informação, contudo não estando esta ainda disponível, o processo será recolocado na fila de processos Prontos e recomeça o ciclo. O processo ficará neste ciclo até que os dados estejam finalmente disponível para a chamada os ler.