UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Ciência, Tecnologia e Ensino Superior

U.C. 21111

Sistemas Operativos

02 de julho de 2019

INSTRUÇÕES

- Leia estas instruções na totalidade antes de iniciar a resolução do teste.
- O enunciado do teste é constituído por 2 grupos de questões, tem 4 páginas e termina com a palavra FIM.
- Se o seu exemplar não estiver completo ou nele se verificar qualquer outra deficiência, por favor dirija-se ao professor vigilante.
- O teste deve ser resolvido na sua totalidade em folhas de respostas.
- Nas respostas, tenha a preocupação de utilizar uma letra legível.
- Todas as respostas devem ser escritas unicamente com caneta azul ou preta.
- O teste é SEM CONSULTA. Todos os elementos necessários à resolução são fornecidos no enunciado.
- Não é permitido utilizar máquina de calcular.
- As cotações são indicadas por grupo e nas próprias questões.
- Nas questões de escrita de programas, a sua correção terá em conta critérios de proficiência e compreensibilidade do código (legibilidade, indentação, estrutura, comentários e explicação geral).
- O não cumprimento das instruções implica a anulação das respetivas questões.
- O tempo de realização do teste é de 150 minutos.

Grupo I [12 valores]

1.1. [1.2] Uma das principais funções de um SO é a gestão de recursos de um sistema complexo constituído por muitas partes. Explique porquê. Dê exemplos de gestão espacial e de gestão temporal.

1.2. [1.2] Em que geração de SO foi introduzida a técnica de spooling ?

1.3. [1.2] Explique porque razão em UNIX a criação de um subprocesso diferente do processo pai é uma operação efectuada em dois passos.

1.4. [1.2] Indique justificando as três razões principais da existência de tarefas.

1.5. [1.2] Observe o seguinte programa com dois processos:

```
/* pseudo-codigo */
semaforo recurso1, recurso2;

void processoA(void){
   down(&recurso1);
   down(&recurso2);
   utilizar_ambos_os_recursos();
   up(&recurso2);
   up(&recurso1);
}

void processoB(void){
   down(&recurso2);
   down(&recurso1);
   utilizar_ambos_os_recursos();
   up(&recurso1);
   up(&recurso2);
}
```

Este programa pode potencialmente levar a uma situação de impasse (deadlock)? Justifique.

1.6. [1.2] Explique os conceitos de recursos preemptíveis e não preemptíveis. Dê um exemplo de cada.

1.7. [1.2] Considere um sistema com memória virtual mononível. O espaço de endereçamento virtual é de 20 bits e a dimensão da página virtual é de 2KB. Quantas entradas tem a Tabela de Páginas ?

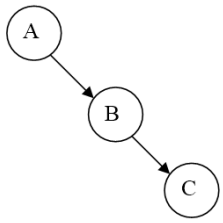
1.8. [1.2] Explique em que consiste o problema da paginação excessiva (trashing) e em que condições ocorre.

1.9. [1.2] O que é uma ligação simbólica ?

1.10. [1.2] De um ponto de vista genérico, quais são as principais responsabilidades do SO relativamente a dispositivos de entrada/saída (I/O) ?

Grupo II [8 valores]

2.1. [3] Escreva um programa em linguagem C padrão que crie uma hierarquia de processos em árvore como mostra a figura. Cada processo deve imprimir na saída padrão uma mensagem do tipo "Processo X", onde X significa A, B ou C conforme o caso. Cada processo deve esperar que os seus processos filhos (directos) terminem, antes dele próprio terminar. O programa não necessita de testar erros nas chamadas às funções.



2.2. [5] Escreva um programa multitarefa em linguagem C segundo a norma POSIX que determine o valor máximo dos elementos de um vector int x[] de dimensão int nx. O vector e a sua dimensão constituem variáveis globais ao programa e admite-se que foram devidamente inicializados.

A tarefa principal deve criar duas sub-tarefas em que cada uma em paralelo (ou em pseudo-paralelismo) analiza metade do vector x[]. A tarefa principal deve esperar que ambas as tarefas terminem e depois deve imprimir o resultado final e terminar.

Nota: planeie cuidadosamente como é dividido o trabalho entre as sub-tarefas e como é efectuada a sincronização e a comunicação da informação necessária à resolução do problema entre as três tarefas.

Formulário

```
#include <stdlib.h>
int system(char *string);

#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
pid_t getpid(void);
pid_t getppid(void);

#include <unistd.h>
unsigned int sleep(unsigned int seconds);
extern char **environ;
int execl(char *path, char *arg, ...);
int execlp(char *file, char *arg, ...);
int execle(char *path, char *arg , ..., char *envp[]);
int execv(char *path, char *argv[]);
int execvp(char *file, char *argv[]);
int execve(char *path, char *argv [], char *envp[]);

#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);

#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
   void *(*start_routine)(void*), void *arg);
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
#define PTHREAD_CREATE_DETACHED
#define PTHREAD_CREATE_JOINABLE
int pthread_join(pthread_t thread, void **value_ptr);
int pthread_mutex_init(pthread_mutex_t *mutex,
   pthread_mutexattr_t * attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

FIM