UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Educação e Ciência

**U.C. 21111**

**Sistemas Operativos**

**26 de setembro de 2016**

## 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 3 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] Indique e descreva resumidamente quais são as duas funções principais de um SO.

**1.2.** [1.2] Explique em que consiste um interpretador de comandos (shell) e o seu modo de operação. Este é considerado como pertencente ao sistema operativo?

**1.3.** [1.2] Explique o conceito de pseudoparalelismo e relacione-o com o conceito de processo.

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

**1.5.** [1.2] Explique em que consiste o escalonamento de um processo e descreva quatro situações que possam levar o SO a efectuá-lo.

**1.6.** [1.2] Explique o que se entende por separação entre mecanismo de escalonamento (scheduling mechanism) e política de escalonamento (scheduling policy), especificando quem é responsável por cada uma das duas vertentes do escalonamento.

**1.7.** [1.2] O conceito de espaço de endereçamento virtual unidimensional permite atribuir um espaço próprio de endereçamento para cada processo ou tarefa (thread) ? Justifique.

**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 entende por um disco fragmentado ? Nessa situação, o que acontece quando é criado um novo ficheiro cuja dimensão ocupe vários blocos ?

**1.10.** [1.2] Como se processa a comunicação entre o SO e um controlador de dispositivo, quer a nível de comandos, quer a nível de dados ?

## Grupo II [8 valores]

**2.1.** [3] Escreva um programa em linguagem C que crie um sub-processo e que com recurso à função execv() execute um comando que produza os mesmos resultados que a execução de "ls -l /home > aaa"na linha de comandos. O comando ls encontra-se na directoria /bin. O processo pai deve esperar que o processo filho termine.

**2.2.** [5] Escreva um programa multitarefa em linguagem C segundo a norma POSIX que converta para letras maiúsculas todos os elementos de um vector char x[1000] que contenham letras minúsculas.

A tarefa principal deve criar 7 sub-tarefas, 5 do tipo analisadora() e as outras 2 do tipo conversora(). Cada tarefa analisadora deve inspeccionar elementos diferentes do vector x[] e quando encontrar uma letra minúscula colocar o respectivo índice num vector auxiliar int idx[], após o que deve dar oportunidade às outras tarefas de serem executadas. Cada tarefa conversora deve retirar todos os índices do vector idx[] e converter as respectivas letras minúsculas

em x[] para maiúsculas, após o que deve dar oportunidade às outras tarefas de serem executadas. A tarefa principal deve esperar que todas as sub-tarefas terminem.

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 todas as 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**