Sistemas Operativos (2013/14) **Exame: Com "cheat-sheet"**
13 de Junho de 2014 **Duração: 120 min**

1- [8v] Em cada alínea desta pergunta deverá indicar se a afirmação feita é verdadeira ou falsa e explicar a sua resposta como exemplificado em seguida.
No caso da afirmação ser **verdadeira, deverá explicar a razão de assim ser**. Por exemplo, dada a afirmação:

> A instalação dum módulo no *kernel* do Linux **não** pode ser realizada por um utilizador comum.

uma possível resposta é:

> Verdadeiro. Após a sua instalação um módulo executa em modo priveligiado, podendo executar qualquer ação. Assim, se qualquer utilizador pudesse instalar um módulo, poderia executar qualquer ação incluindo, p.ex., ler o conteúdo de ficheiros de outros utilizadores.

No caso da afirmação ser **falsa, deverá alterar a afirmação de forma a torná-la verdadeira de forma não trivial e explicar a razão de assim ser**. Por exemplo, dada a afirmação:

> A instalação dum módulo no *kernel* do Linux pode ser realizada por qualquer utilizador.

uma possível resposta é:

> Falso. Um utilizador comum não pode instalar um módulo do *kernel* do Linux. Após a sua instalação um módulo executa em modo priveligiado, podendo executar qualquer ação. Assim, se qualquer utilizador pudesse instalar um módulo, poderia executar qualquer ação incluindo, p.ex., ler o conteúdo de ficheiros de outros utilizadores.

Como ilustrado por estes exemplos, a resposta não deverá exceder 3 ou 4 frases (uma resposta demasiado longa poderá ser penalizada).

1.1- As funções *wrapper* das chamadas ao sistema da `libc` não são portáveis, i.e. dependem da ISA do processador em que executam.

1.2- Com o advento de plataformas com centenas de processadores, um processo estará apenas em 1 de 2 estados, pelo que será possível eliminar o 3º do diagrama de estados apresentado nas aulas.

1.3- Em algoritmos de escalonamento baseados em *quantuns*, o tamanho destes é especialmente relevante quando a carga do sistema é predominantemente *I/O bound*.

1.4- Instruções de permissão/inibição de interrupção são inúteis em multi-processadores.

1.5- Monitores, ao contrário de *mutexes*, permitem sincronização sem espera ativa (*busy waiting*).

1.6- Se o tempo de execução duma operação de E/S for da mesma ordem de grandeza que a execução duma instrução do CPU, E/S com *polling* é mais eficiente do que E/S com interrupções.

1.7- Se um processo tentar aceder a uma página cuja *page-table entry* tem o respetivo *valid-bit* inactivo, Unix/Linux gera uma exceção do tipo *segmentation fault*.

1.8- Num sistema de memória virtual paginada, se a área de *swap* for maior do que a memória RAM é pouco provável que ocorra *thrashing*.

1.9- *Linear Block Addressing (LBA)* abstrai a geometria dos discos e consequentemente torna inútil qualquer tentativa do SO para evitar *seeks*.

1.10- A API das chamadas ao sistema de acesso a ficheiros de Unix/Linux (`read()` e `write()`) pressupõem um acesso sequencial.

2- **[2.5v]** Sobre o projecto de desenvolvimento dum *device driver* para a porta série em Linux.

2.1- Durante esse projeto usou uma *shell script* cujas principais linhas eram as seguintes:

```
insmod -f ./${module}.ko || exit 1
major=‘cat /proc/devices | awk "\\$2==\"$module\" {print \\$1}"| head -n 1‘
mknod /dev/${module} c $major 0
```

Explique para que serve cada uma delas e justifique a sua necessidade.

2.2- Considere a função `read` da estrutura `struct fileops`. Descreva de forma sintética a sua implementação (pode enumerar os passos), fazendo referência às estruturas de dados do *kernel* usadas.
IMP.: A cotação desta alínea depende da implementação descrita ser baseada em *polling* (80%), ou em interrupções (100%).

3- **[2v]** Os segmentos de código seguintes ilustram um padrão usado na sincronização entre processos com sinais POSIX.

```
...                                              ...
A: sigdelset(&mask, SIG_USR1);                   1: kill(pid, SIG_USR1)
B: sigprocmask(SIG_SET_MASK,&mask,&oldmask);     ...
C: pause()
...
```

Assuma, que o argumento `pid` na linha `1` é o PID do processo que executa o conjunto de instruções da esquerda.

3.1- Explique em que medida estes segmentos permitem a sincronização entre os processos correspondentes. Mostre que este código pode sofrer duma *race-condition*.
Dica: Use uma sequência de execução de instruções para ajudá-lo na sua explicação.

3.2- Elimine esta *race-condition* usando a chamada ao sistema `sigsuspend()`, i.e. Faça as alterações necessárias ao código apresentado na alínea anterior para eliminar esta *race-condition*. Justifique.
Dica: Use a chamada ao sistema `sigsuspend()`

4- **[2v]** Sobre memória virtual paginada.

4.1- A instrução `MOVSD` da IA-32, a qual tem um único byte, permite transferir uma *double word* duma posição de memória, apontada por um par de registos internos, para outra posição de memória, apontada por outro par de registos internos. Qual é o **número máximo** de conversões de endereços virtuais em endereços físicos que o processador precisa realizar na execução desta instrução? Justifique os "passos" usados na elaboração da sua resposta.
IMP. Na arquitetura IA-32 os endereços não precisam estar alinhados.

4.2- Assumindo que um processador IA-32 foi configurado para usar uma tabela de páginas de 2 níveis (páginas de 4 KiByte, 10 bits para o primeiro nível e 10 bits para o segundo) quantos acessos à memória poderão ser necessários na execução daquela instrução. Explique.

4.3- Explique como é que a implementação de memória virtual paginada reduz significativamente o número médio de acessos à memória necessários para realizar a conversão de endereços.

5- [2v] Sobre sistemas de ficheiros. Considere um sistema de ficheiros baseado numa *file allocation table* (FAT) que suporta até 4 TiByte de espaço dedicado a blocos de dados, cada um dos quais com 4 KiBytes.

5.1- Explique o interesse de manter uma cópia da FAT em memória principal.

5.2- Assumindo um espaço de endereçamento virtual de 32 bits, considera esta solução exequível?

5.3- O sistema de ficheiros Unix clássico baseado em *inodes* sofre do mesmo problema? Justifique.

6- [3.5v] Suponha que tem um conjunto de ficheiros à sua disposição que representam acessos a páginas *web*, onde os campos estão separados por vírgulas segundo o seguinte formato: `<TIMESTAMP>,<UID>,<URL>`. Por exemplo:

```
1402601645,3a9ece95744c48128e3fb0344a3fdfd8,www.fe.up.pt
1402536663,3a9ece95744c48128e3fb0344a3fdfd8,www.fe.up.pt/~ee99999
1402536663,a508f4aa3d2b4ded89c443786dc0d01d,https://www.google.pt/?q=operating+s
ystems
```

Pretende-se calcular o total de entradas de um dado utilizador. P.ex., nas linhas acima, duas entradas correspondem a acessos do utilizador `3a9ece95744c48128e3fb0344a3fdfd8`.
IMP: Se nas alíneas que se seguem tiver dificuldade em implementar alguma da funcionalidade necessária, procure isolar essa funcionalidade em funções, e invoque-as do seu código. Por favor, inclua comentários explicando o que faz cada uma das funções usadas mas não implementadas.

6.1- Escreva um programa que tome como argumentos o identificador do utilizador e o nome dos ficheiros (por esta ordem) e, para cada um desses ficheiros, imprime, num ficheiro cujo nome resulta da concatenação do nome do ficheiro e do identificador do utilizador separados por um ponto, uma linha com o número de entradas desse utilizador .
Nota: O `<TIMESTAMP>` tem sempre 10 dígitos. O `<UID>` tem sempre 32 caractéres. O `<URL>` nunca excede os 255 caractéres. Cada linha termina com o carácter `\n`.

6.2- Escreva uma versão multiprocesso do programa da alínea anterior. Nesta versão, para cada ficheiro de entrada deverá criar um processo que execute o programa desenvolvido na alínea anterior sobre um único ficheiro. Quando todos os processos filhos terminarem deverá ler cada um dos ficheiros de saída por eles gerados e imprimir na saída padrão (*stdout*) o nº total de entradas do utilizador especificado.

Protótipos

Processos

```
pid_t fork(void);
int execve(const char *filename, char *const argv[], char *const envp[]);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
void _exit(int status);
WIFEXITED(); // macro
WEXITSTATUS(); // macro
```

Threads

```
int  pthread_create(pthread_t  *  thread, pthread_attr_t * attr,
                    void *(*start_routine)(void *), void * arg);
void pthread_exit(void *retval);
int pthread_join(pthread_t th, void **thread_return);
pthread_t pthread_self(void);
```

Mutexes/Locks

```
pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
int  pthread_mutex_init(pthread_mutex_t  *mutex,
                        const pthread_mutex_attr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Variáveis de Condição

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int    pthread_cond_init(pthread_cond_t    *cond,    pthread_condattr_t
       *cond_attr);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);
```

Semáforos

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
int sem_destroy(sem_t *sem);
```

Sistema de Ficheiros

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
        flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_APPEND
        mode: S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP,
              S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fildes, off_t offset, int whence);
       whence: SEEK_SET, SEEK_CUR, SEEK_END
```

Sinais

```
int kill(pid_t pid, int sig);
int pause(void);
int sigsuspend(const sigset_t *mask);
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
```