Fundamentos de Sistemas de Operação MIEI 2013/2014

Exame de recurso, 3 de janeiro de 2014, 3 horas

Sem consulta e sem esclarecimento de dúvidas; indique eventuais hipóteses assumidas nas sua respostas.

Questão 1 (1.0 valores) Para suportar o que no livro OSTEP é chamado de *limited direct execution*, a arquitectura hardware deve suportar os três seguintes mecanismos: instruções privilegiadas, proteção de memória, e interrupção de relógio (*timer*). Explique porque é que todos eles são necessários.

Questão 2 (1.0 valores) Uma arquitectura hardware como a do Pentium tem uma instrução chamada *return from interrupt*, cuja mnemónica é *iret*. Esta instrução, entre outras coisas, muda o modo de operação do CPU de modo sistema (ou kernel) para modo utilizador. Explique porque é que a instrução *iret* existe e em que circunstâncias é usada.

Questão 3 (1.5 valores) Num sistema UNIX um processo tenta ler dados da consola fazendo uma chamada ao sistema read:

```
1: n = read( 0, line, MAX);

2: tot = tot + n;
```

em que *line* é um vector de caracteres com *MAX* posições. Diga porque estados passa o processo desde que começa a executar a linha 1 até acabar a linha 2. Indique claramente que eventos fazem o processo mudar de estado. Considere dois casos possíveis:

a) Já existem MAX bytes disponíveis para ler.

b) Não existem ainda bytes disponíveis., mas o utilizador irá introduzi-los em breve.

Questão 4 (1.5 valores) Um escalonador de CPU usa uma única fila de processos prontos (*READY queue*) e uma estratégia de escalonamento *Round-Robin* com um *time slice T*. Suponha que um processo estava no estado RUNNING e o seu *time slice* terminou. Explique o que é que o sistema operativo faz nesta situação.

Questão 5 (1.5 valores) Considere que três processos A, B e C são os únicos no sistema, tendo entrado na fila de processos prontos no instante de tempo 0, pela ordem A, B, C. Os processos realizam as seguintes acções:

- Processo A: Usa o CPU durante 100 ms e termina
- Processos B e C: Executam 10 vezes um ciclo em que usam o CPU durante 2 ms e de seguida fazem uma operação de entrada/saída que demora 8 ms

Supondo que a troca de contexto tem uma duração desprezável, diga em que instante de tempo terminam os processos A, B e C para os dois seguintes algoritmos de escalonamento:

a) *Round Robin* com uma fatia de tempo de 1 ms
b) *Round Robin* com uma fatia de tempo de 100 ms

Questão 6 (1.0 valores) O que acontece se executarmos o seguinte programa num sistema LINUX/UNIX?

```
do{
    ;
} while (fork() >= 0);
exit( 0 );
```

Justifique a sua resposta.

Questão 7 (1.0 valores) Suponha que se pretende escrever um programa com múltiplos threads que partilham variáveis. Para suportar o acesso concorrente a essas variáveis partilhadas foram implementadas as seguintes funções: *enterCS( ) leaveCS( )* e *initCS( )*. Após um dos threads invocar *initCS( )* , todos os threads chamam *enterCS( )* antes de ter entrar na secção crítica e invocam *leaveCS( )* após terminar a secção crítica.

```
1: void initCS( int *lock){
2:    *lock = 0;
3: }
```

```
1:void enterCS (int *lock){
2:    while(*lock == 1);
3:    *lock = 1;
4:}
```

```
1: void leaveCS (int *lock){
2:    *lock = 0;
3: }
```

Suponha que as funções acima são usadas num sistema operativo que suporta multi-programação e que é executado num hardware com um único CPU. A implementação acima apresentada funciona correctamente? Justifique a sua resposta.

Questão 8 (1.5 valores) Considere o seguinte código.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int x;
void *t1(void *arg) {
   int i;
   for (i = 0; i < 1000000; i++){
        pthread_mutex_lock( &ex);
       x = x + 1;
       pthread_mutex_unlock( & ex);
   }
}

int main(int argc, char *argv[]) {
  pthread_t p1, p2, p3;
  pthread_mutex_t ex;

  pthread_mutex_init( &ex, NULL);
  x = 0;
  pthread_create(&p1, NULL, t1, NULL);pthread_create(&p2, NULL, t1, NULL);
  pthread_create(&p3, NULL, t1, NULL);
  pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);
  printf("x = %d\n", x);
  return 0;
}
```

a) Qual é o valor de *x* que o programa escreve? Justifique. Porque é que é necessário usar o *mutex ex* ?

b) Considere uma nova versão do programa em apenas se muda o código de t1 para

```
void *t1(void *arg) {
   int i, totp = 0;
   for (i = 0; i < 1000000; i++) totp = totp + 1;
    pthread_mutex_lock( &ex);
   x = x + totp;
   pthread_mutex_unlock( & ex);
}
```

Este programa produz o mesmo resultado do que o da alínea a) ? Justifique.

c) Verificou-se que a versão b) executa mais rapidamente do que a versão a) ? Diga porquê.

Questão 9 (1.0 valores) Considere o seguinte código que está incompleto.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void f1( ){
   …
}
void f2( ){
   …
}
void f3( ){
   …
}
void *w1(void *arg) {  // código incompleto

     f1();

}
void *w2(void *arg) { // código incompleto

     f2();

}
void *w3(void *arg) { // código incompleto

     f3();

}
int main(int argc, char *argv[]){
  pthread_t p1, p2, p3;

  // código em falta

  pthread_create(&p1, NULL, w1, NULL);pthread_create(&p2, NULL, w2, NULL);
  pthread_create(&p3, NULL, w3, NULL);
  pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);
  return 0;
}
```

Utilizando *mutexes* e *condições*, pretende-se que acrescente o código necessário para garantir que a função f1( ) apenas é executada **depois** de f2( ) **e** f3( ) executarem.

Questão 10 (1.5 valores) Um sistema em que os endereços virtuais têm 28 bits, usa uma MMU que suporta páginas com dimensão 4 KBytes (2^12).

a) Quantas entradas tem a tabela de páginas de cada processo? Justifique.

b) Para um dado processo em execução, as primeiras entradas da tabela de páginas são as seguintes:

Nº página virtual	Nº página física (em base 16)
0	Inválida
1	0xDEAD
2	0xCAFE
3	0xDADA

As outras entradas são todas inválidas. Indique, justificando, os endereços físicos que correspondem aos seguintes endereços virtuais. Responda "Endereço Inválido" se o endereço virtual for inválido.

b1) 0x0002020 b2) 0x0011001 b3) 0x0000301 b4) 0x0001211

Questão 11 (1.0 valores) Explique como funciona a paginação a pedido e quais são as suas vantagens e inconvenientes. Indique também quais são os requisitos mínimos que o hardware tem de ter para que este mecanismo funcione.

Questão 12 (1.0 valores) Num sistema UNIX/LINUX um processo pode requisitar mais memória ao sistema usando a chamada *sbrk();* esta chamada é feita por exemplo quando o utilizador invoca a função *malloc( )* da biblioteca standard do C. Ver detalhes sobre o sbrk() na última página.

a) Explique como é que o sistema operativo poderia implementar esta função se usasse uma MMU com suporte para segmentos.

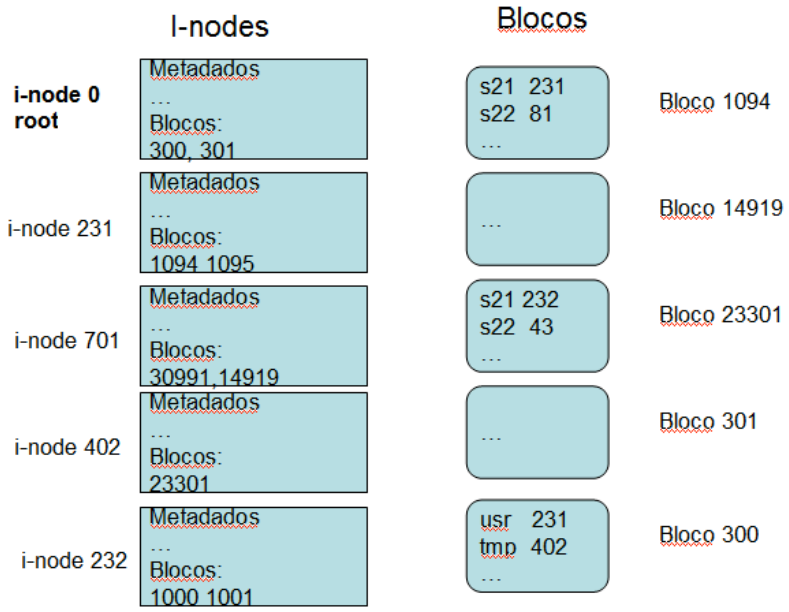
b) E se a MMU for baseada em páginas?

Questão 13 (1.0 valores) Seja *Tr* o tempo de acesso a uma posição de memória quando a respectiva página está carregada em RAM e *Td* o mesmo tempo quando é preciso ir buscar a página ao disco.

a) Como relaciona o tempo de acesso efectivo á RAM (*Te*) com *Tr* e *Td* ?

b) Um sistema de memória virtual está sujeito ao fenómeno de *thrashing*. Para um dado processo P, diga em que consiste esta situação e como é que ela se relaciona com Te. Relacione Te com a relação entre o nº de páginas virtuais que P consegue ter em RAM e o nº total de páginas da virtuais da imagem do processo.

Questão 14 (1.0 valores) A figura seguinte representa uma parte de um sistema de ficheiros UNIX.



No kernel do sistema existe a indicação de que a directoria raiz corresponde ao i-node 0; a tabela de i-nodes já foi lida para RAM quando o disco foi montado. Diga que blocos de disco são lidos e por que ordem, quando um processo faz a chamada ao sistema `f = open("/tmp/s21", O_RDONLY)`

Questão 15 (1.5 valores) Para um sistema de ficheiros UNIX, indique, justificando, a sequência de leituras e escritas feitas na zona de meta-dados do disco quando se executa a chamada ao sistema *unlink ("/tmp/xxx")*. Suponha que o ficheiro existe e que o processo que faz a chamada tem permissões para alterar a directoria */tmp*.

Questão 16 (1.0 valores) Suponha uma variante do sistema de ficheiros UNIX em que os blocos de dados têm 4Kbytes e os endereços de blocos têm 2 bytes. Na parte do *i-node* que indica os blocos que contêm dos dados dos ficheiros há:

- 10 endereços directos
- 1 endereço indirecto - isto é que contém o endereço de um bloco com endereços
- 1 endereço duplamente indirecto - contém endereços de blocos que contêm endereços de blocos com endereços

Diga, justificando, qual é o maior número de blocos de dados que um ficheiro pode ter neste sistema.

Questão 17 (1.0 valores) Suponha um disco com as seguintes características:

- seek time médio: 9 ms
- tempo de uma rotação: 5 ms
- 100 sectores por pista
- Um sector demora 40 micro-segundos a ser lido

Apresente, justificando detalhadamente, uma estimativa para o tempo que demora a leitura de 50 blocos supondo que os blocos estão todos contíguos na mesma pista.

Algumas chamadas ao sistema UNIX/Linux

`int fork( )`	
`int execvp( char *executable_file, char * args[ ] )`	
`int wait( int *status)`	
`int brk (void * addr)`	Muda, se possível, a localização do *program break* (fim da zona de dados do programa) para o endereço *addr*
`void * sbrk( intptr_t increment)`	Adiciona *increment* bytes à zona de dados do programa; isto é adiciona *increment* bytes ao *program break*

Algumas funções da biblioteca de Pthreads

`int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
`int pthread_join (pthread_t thread, void **retval)`
`int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`
`int pthread_mutex_lock (pthread_mutex_t *mutex)`
`int pthread_mutex_unlock (pthread_mutex_t *mutex)`
`int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr)`
`int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)`
`int pthread_cond_signal(pthread_cond_t *cond)`