Fundamentos de Sistemas de Operação MIEI 2014/2015

Exame de recurso, 16 de janeiro de 2014, 3h (1h30m+1h30m)

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

1ª Parte (correspondente ao 1º teste)

Questão 1 Como é sabido, para o sistema operativo funcionar correctamente o CPU deve ter dois modos: utilizador e sistema.

a) O que diferencia estes dois modos?

b) Dê exemplos de algumas instruções máquina que só podem estar acessíveis em modo sistema e diga porquê.

c) Diga em que situações é que o CPU comuta de modo utilizador para modo sistema.

d) E em que situações comuta de modo sistema para modo utilizador?

Questão 2 Considere vários programa que todos eles chamam 10000 vezes uma função *f( ).*

a) Admita que num programa A *f( )* é uma função que faz uma computação aritmética que demora X milisegundos e que num progama B, *f( )* faz uma chamada ao sistema que dentro do kernel demora os mesmos X milisegundos. Qual dos programas demora mais tempo a executar? Justifique.

b) Considere agora um programa C que depois de abrir um ficheiro para leitura chama 10000 vezes a chamada ao sistema *read( )* para ler um byte do ficheiro e um programa D que faz o mesmo que o programa C mas usando a função da biblioteca *stdio* do C *fread( )*. Qual dos programas demora mais tempo a executar? Justifique.

Questão 3 Considere um sistema operativo que suporta múltiplos processos em execução simultânea. Descreva as acções efectuadas pelo SO operativo quando um processa P tenta ler N bytes do teclado e não há dados para ler.

Questão 4 Considere o seguinte programa que é executado num ambiente UNIX.

```
int main ( int argc, char *argv[ ]){
        int x = 5;
        int child = fork ( );
        if( child == 0) {
                x = x + 5;
        }
        else {
                child = fork ( );
                x = x + 10;
                if ( child ) x = x + 5;
        }
}
```

Quantas cópias diferentes da variável *x* existem? E para cada um dos processos, qual é o valor de *x* quando cada um dos processos termina?

Questão 5 Quer quando se faz um *fork( )* quer quando se faz um *pthread_create( )* é criada uma nova máquina virtual com um CPU virtual, memória e uma tabela de canais abertos. Para os dois casos explique como são inicializadas os três componentes da máquina virtual do novo processo criado.

Questão 6 O CPU Pentium tem uma instrução máquina chamada LOCK XCHG que troca o conteúdo de duas variáveis de forma indivisível. Isto significa que após ser executado o seguinte troço de programa em C

```
int x = 4
int y = 1;
LOCK XCHG (x, y)
```

a variável *x* contém o valor 1 e a variável *y* contém o valor 4. Esta operação é feita impedindo o acesso de outros processos às posições de memória envolvidas.

Considere o seguinte código incompleto que usa as funções de criação de threads da biblioteca Pthreads.

```
int b; // variável global partilhada por todos os processos
void *func( void * arg){
   int l;
   for( j = 0; j < 1000000; j++){
            // código em falta que garante que só entra um processo de cada vez na secção crítica
            // início da secção crítica
            ....
           // fim da secção crítica
           // código em falta que liberta o acesso à secção crítica
   }
}
int main(){
   b = 0;
   for( i=0; i , N; i++) pthread_create( …, NULL, func, NULL):
   for( i=0; i , N; i++) pthread_join( …,):
}
```

Apresente o código em falta. Tem de utilizar a instrução LOCK XCHG e não pode usar *mutexes,* condições ou semáforos.

Questão 7 Considere o código seguinte que usa a biblioteca Pthreads

```
// declarações em falta
void *func1( void * arg){
   while(1){
            // código em falta
            printf("a"); fflush(stdout);
           // código em falta
   }
}
void *func2( void * arg){
    while(1){
            // código em falta
            printf("b"); fflush(stdout);
           // código em falta
    }
}
int main(){
    pthread_t t1, t2;
   // código em falta
   // cria dois threads
```

```
    pthread_create(&t1 NULL, func1, NULL): pthread_create(&t2, NULL, func2, NULL):
    pthread_join(t1, …): pthread_join(t2, …):
}
```

Apresente o código em falta de forma a garantir que vão sendo escritos no canal standard de saída alternadamente "a" e "b". Pode usar qualquer mecanismo de sincronização da biblioteca de Pthreads, mas sugere-se o uso de semáforos.

Questão 8 Considere que tem disponível uma versão incompleta da biblioteca *Pthreads* na qual existem as funções habituais sobre mutexes e condições, mas não existem semáforos. Suponha que se pretendem suportar semáforos sobre esta bibliotecas com um funcionamento idêntico à definição habitual das operações wait( ) e post( ) / signal( ) sobre semáforos. Complete o seguinte código.

```
#include <pthread.h>
typedef struct {
        int cont;
        pthread_mutex_t ex;
        pthread_cond_t cond;
} mySem;
void initSem(  mySem *s, int value){
        // código em falta
        pthread_mutex_init( &s->ex, NULL);
        pthread_cond_init( &s->cond, NULL);
}
void waitSem( mySem * s){
        // código em falta
}
void postSem( mySem * s){
        // código em falta
}
```

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)
```

Mutexes

Inicialização

```
int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) ou
pthread_mutex_t mutex  = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock (pthread_mutex_t *mutex)
int pthread_mutex_unlock (pthread_mutex_t *mutex)
```

Condition Variables

Inicialização

```
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr)  ou
pthread_cond_t cond  = PTHREAD_COND_INITIALIZER;
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)
int pthread_cond_signal(pthread_cond_t *cond)
```

Semaphores

Inicialização

```
int sem_init( sem_t *sem, int type, int initial_value)  // type is always 0 when using Pthreads
int sem_wait( sem_t  *sem )
int sem_post( sem_t * sem )
```

2ª Parte (correspondente ao 2º teste)

Questão 9 Considere um SO como o Windows ou o Linux em que o escalonamento dos processos procura garantir que os utilizadores têm tempos de resposta curtos.

a) Nestes sistemas, os algoritmos de escalonamento de processos favorecem os processos *I / O Bound* face aos processos *CPU Bound*. Explique porque é que este favorecimento existe.

b) Supondo um algoritmo de escalonamento Round Robin com uma única fila como é que pode ser feito o favorecimento dos processo *I /O Bound* ?

c) De que forma é que o SO determina que um dado processo P1 é *I /O Bound* e outro processo P2 é *CPU Bound*?

Questão 10 Três processos são criados simultaneamente num sistema computacional:

- O processo A chega em primeiro lugar no instante 0 e usa o CPU durante 25 milissegundos antes de terminar
- O processo B é criado depois mas ainda no instante 0. Este processo executa 2 vezes um ciclo: em cada ciclo usa o CPU por 3 milisegundos e depois realiza uma operação de entrada / saída que demora 5 milisegundos
- O processo C também é criado no instante 0, mas depois de B. A sequência de acções efectuadas é idêntica à de B.

Usa-se uma estratégia de escalonamento Round Robin com fatia de tempo de 5 ms. Supondo que o tempo de troca de contexto é desprezável, indique, justificando, o tempo em que acaba a execução de cada um dos processos A, B e C.

Questão 11 Considere uma MMU que suporta endereços de 32 bits e páginas de 4 Kbytes.

a) Indique o número de página virtual e o deslocamento correspondentes aos endereços virtuais 20, 4100 e 8300.

b) Supondo que a página virtual 2 está carregada na página física 1, a que endereço físico corresponde o endereço virtual 8200?

c) Este tipo de gestão de memória está sujeito a fragmentação externa? Justifique a sua resposta.

Questão 12 Suponha um sistema de operação que executa numa máquina com uma MMU baseada em páginas e em que é usada paginação a pedido e uma estratégia de substituição de páginas LRU. Suponha que um dado processa tem atribuídas 4 páginas físicas e que inicialmente não tem nenhuma página carregada. Diga, justificando, quantas faltas de página ocorrem quando o processo faz a seguinte sequência de referências a páginas virtuais: 1, 3, 2, 4, 2, 1, 5, 6, 2, 6, 1, 7, 5, 6, 1.

Questão 13 Suponha um sistema de operação que executa numa máquina com uma MMU baseada em páginas. Considere as chamadas ao sistema *fork( )* e *exec( )* usadas nos sistemas operativos da família UNIX. Explique de que forma é que um MMU baseada em páginas pode servir para diminuir o tempo gasto na criação de um processo quando há um *fork(*) e para diminuir o número de páginas ocupadas na memória física.

Questão 14 Considere um disco com 32768 blocos (2^15). Cada bloco tem 2K bytes. Pretende-se calcular qual é o máximo tamanho (em blocos) de um ficheiro para duas formatações lógicas diferentes do disco.

a) O sistema de ficheiros colocado no disco usa o formato FAT-16. Isto significa que os blocos que constituem o ficheiro estão descritos por uma lista ligada que começa na entrada da directoria e que vai ocupando entradas na FAT. Cada entrada da FAT tem 16 bits. Qual é o tamanho máximo de um ficheiro guardado neste diso com o sistema de ficheiros FAT-16?

b) O sistema de ficheiros colocado no disco usa um formato à moda do UNIX. Cada i-node tem 10 endereços em que os primeiros 8 são endereços directos e o nono e o décimo endereços apontam para blocos com endereços. Qual é o tamanho máximo de um ficheiro guardado neste disco com o sistema de ficheiros descrito?

Questão 15 Considere um sistema de ficheiros com a organização habitual do UNIX.

a) Quantos acessos a disco são necessários para obter o i-node do ficheiro */x/y/z/w*? Assuma que o kernel só guarda o *i-node* da directoria raiz e que as directorias ocupam apenas um bloco. Justifique a sua resposta.

b) O comando do "shell" *cp /x/f1 /y/f1* faz uma cópia do 1º ficheiro para outro com o nome do 2º argumento do comando. Diga, justificando, que acessos são feitas ao disco quando é feita esta cópia; inclua os acessos à zonas de dados e de meta-dados do disco. Suponha as seguintes hipóteses simplificativas

- */x/f1* existe e ocupa apenas um bloco
- */y/f2* não existe
- as directorias envolvidas ocupam apenas um bloco
- o bitmap de blocos ocupa apenas um bloco

Questão 16 Considere o sistema de ficheiros usado no trabalho prático nº 2 que tem as seguintes caracterísiticas principais:

- o bloco 0 é o superbloco e contém o número mágico, o número de blocos do disco e o número de blocos ocupado pela FAT (*nblocosFAT*)
- O bloco1 contém a única directoria existente. Cada entrada desta directoria tem
  - Uma marca a dizer se está ocupada ou livre
  - O nome do ficheiro
  - Um inteiro com o comprimentos do ficheiro em bytes
  - O número do bloco onde começa o ficheiro
- Os blocos 2 a *nblocosFAT+1* contêm a FAT que tem a listas ligadas de blocos onde os estão guardados os ficheiros
- Os restantes blocos são blocos de dados

Suponha que quando um disco é montado se pretende verificar se o sistema de ficheiros está coerente e se forem descobertos erros se tenta corrigi-los. O que seria necessário fazer ?