Fundamentos de Sistemas de Operação MIEI 2015/2016

Exame de recurso, 8 de janeiro de 2016, 2h40m (1h20m+1h20m)

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

1ª Parte (correspondente ao 1º teste)

Questão 1 (1 valores) Como é sabido, para o sistema operativo funcionar correctamente o CPU deve ter dois modos: utilizador e sistema. Explique porque é que é necessária a existência destes dois modos.

Questão 2 (1 valores) Considere as chamadas ao sistema UNIX

pid_t wait(int * loc)
void exit(int status)

Explique o funcionamento destes duas chamadas e diga como é que elas se relacionam entre si. Explique por que estados passa o processo que executa *exit( )*. Neste contexto, em que situação é que o processo que executa *exit( )* fica no estado *Zombie*?

Questão 3 (1 valor) Considere as chamadas ao sistema *fork( )* e *execvp(char *executable_file, char *argv[ ])* . Ambas executam operações sobre máquinas virtuais que executam o código associado ao processo filho. Como é que a chamada *fork( )* inicializa a máquina virtual associada ao processo filho? E como é que a chamada *execvp( )* preenche o estado da máquina virtual associada a um processo?

Questão 4 (1 valor) O CPU NV tem uma instrução máquina **indivisível** e que garante o acesso exclusivo ao conteúdo da posição de memória cujo endereço é o 1º argumento da instrução:

`int atomic_xchg( int *p, int val )` que afeta o conteúdo da posição apontada por *p* com o valor *val*, retornando o conteúdo anterior de **p.* Em *C* a descrição será

```
int atomic_xchg( int *p, int val ){
      int temp = *p; *p = val;
      return temp;
}
```

Diga como poderia usar estas instruções para conseguir executar secções críticas (isto é em que se garante exclusão mútua). Mais concretamente o que se pretende é que, usando a instrução `atomic_xchg`, apresente é o código que implementa as operações

`InitLock( unsigned int *ad )`

`Lock( unsigned int *ad )`

`Unlock( unsigned int *ad )`

Questão 5 (1.5 valores) Complete o seguinte programa que pretende utilizar 4 *threads* para calcular o histograma de uma imagem em tons de cinzento. A imagem é representado por um vetor com SIZE*SIZE posições em que cada posição pode tomar um valor entre 0 e 255. Recorde-se que o histograma determina quantas ocorrências de cada valor existem no vetor.

```
#include <stdio.h>
#include <pthread.h>
#define SIZE 1024
#define NO_GREY_LEVELS 256
#define NTHREADS 4
unsigned char image[SIZE*SIZE];
unsigned int histogram[NO_GREY_LEVELS];
```

```
void *worker( void * arg ){
…
}
int main(int argc, char *argv[]){
// declarações em falta
       int i;
       for(i = 0; i < NTHREADS; i++) pthread_create( … );
       for(i = 0; i < NTHREADS; i++) pthread_join( … );
       for(i = 0; i < NO_GREY_LEVELS; i++) printf(“ %d\n”, histogram[i]);
       return 0;
}
```

Complete as funções *main( )* e *worker( )*.

Questão 6 (1 valor) Considere o protótipo da função *pthread_cond_wait( )*. Explique a necessidade do argumento *pthread_mutex_t *mutex.*

Questão 7 (1.5 valores) Considere o seguinte fragmento de programa que usa Pthreads.

```
#include <stdio.h>
#include <pthread.h>
void func1(){ … };
void func2(){ … };
void *worker1( void * arg ){
…
}
void *worker2( void * arg ){
…
}
int main(int argc, char *argv[]){
// declarações e inicializações em falta

       pthread_create( … ,…, worker1, …); pthread_create( … ,…, worker2, …);
       pthread_join( … ); pthread_join( … );

       return 0;
}
```

Suponha que só o thread *worker1* pode executar *func1( )* e que só o thread *worker2( )* pode executar *func2( )*. Pretende garantir a seguinte sequência de execuções

- 2 execuções de func1( )
- 2 execuções de func2( )
- após esta fase, execução alternada de *func1( )* e *func2( )*.

Complete o código do *main( )* , *worker1( )* e *worker2( ).* Sugere-se o uso de semáforos.

Questão 8 (1 valor) Considere um sistema operativo concebido para ser executado num computador dedicado ao uso interativo e que suporta múltiplos processos. Indique as vantagens e inconvenientes do algoritmo de escalonamento *Multiple Level Feedback Queue*.

Questão 9 (1 valor) Considere um sistema operativo tempo real destinado a ser executado num computador com um único CPU e que supervisiona o funcionamento de uma caldeira industrial. O sistema informático possui sensores que medem parâmetros de funcionamento da caldeira e actuadores que permitem modificar esse valores. Neste sistema correm vários processos e o algoritmo de escalonamento é baseado em prioridades, executando-se em cada momento o processo com a prioridade mais elevada. Supondo que a prioridade dos processos é fixa e definida no momento da sua criação, faça um diagrama de estados dos processos neste sistema; indique o que é que provoca transições de estado.

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 10 (1 valor) Diga quais são as vantagens dos mecanismos de gestão da memória física baseados em páginas face às abordagens que usam partições contíguas.

Questão 11 (1 valor) Como funciona a ligação dinâmica de bibliotecas? Como relaciona com a paginação a pedido e a chamada ao sistema *mmap( ).*

Questão 12 (1 valor) Explique como é que a paginação a pedido permite executar programas cuja imagem é superior ao tamanho da memória física disponível para os programas utilizadores.

Questão 13 (1 valor) Considere o algoritmo *2nd chance* ou *clock* para substituição de páginas. Explique porque é que este algoritmo é aproximação razoável do algoritmo LRU (*Least Recently Used*) .

Questão 14 (1 valor) Num sistema de memória virtual que suporta paginação a pedido, num dado instante a imagem de um processo tem N páginas e P páginas estão em RAM e N-P em disco.

a) Como relaciona o tempo médio de acesso à memória com o valor P/N e com as características de localidade de referências que o programa exibe ?

b) O que é uma situação de *thrashing*? Como relaciona essa situação com o tempo médio de acesso à memória ?

Questão 15 (2.5 valores) Considere um sistema de ficheiros com a organização habitual do UNIX.

a) A chamada ao sistema *int lstat(char * symbolic_name, struct stat *buf)* preenche a estrutura *buf* com um conjunto de informação retirada do *i-node* do ficheiro cujo nome é *symbolic_name*. O programa utilizador pode usar esta chamada da seguinte forma:

```
#include <sys/stat.h>
struct stat b;
…
lstat("/home/rita/xpto.c", &b); printf("comprimento de xpto.c = %d\n", b.st_size);
```

Supondo que o processo que executa o pedaço de código acima tem os direitos de acesso necessários para obter o comprimento do ficheiro, descreva em detalhe os acessos ao disco feitos pela chamada ao sistema *lstat*.

b) Considere o seguinte pedaço de código

```
int f = open("/home/rita/blabla.txt", O_RDWR | O_CREAT); // criação do ficheiro blabla.txt
write(f, "12345", 6);
close( f );
```

Suponha que o processo que executa o pedaço de código acima tem os direitos de acesso necessários para criar o ficheiro na diretoria indicada e que o ficheiro ainda não existe. Descreva as operações efectuadas pelas chamadas ao sistema. Pretende-se que descreva os acessos feitos ao disco e as alterações feitas aos meta-dados em disco e às tabelas do sistema.

Questão 16 (1 valor) Explique as vantagens que existem em ter uma cache de blocos associada ao sistema de ficheiros; discuta separadamente os casos da leitura e escrita. Por outro lado, que consequências tem o uso da cache de blocos na manutenção da consistência do sistema de ficheiros?

Questão 17 (1.5 valores) Considere um sistema de ficheiros com a organização habitual do UNIX e que não utiliza *journaling*. Suponha que houve uma falha de energia e que após o retorno desta se está a verificar a consistência de um discos antes de o montar. Diga quais são as principais operações efectuadas para verificar a consistência do sistema de ficheiros que reside no disco que está a ser examinado.