

Fundamentos de Sistemas de Operação MIEI 2014/2015

1º Teste, 20 Outubro de 2014, 2h – versão A

Nome:

Nº

Avisos: Sem consulta; a interpretação do enunciado é da responsabilidade do aluno; se necessário indique a sua interpretação. No fim deste enunciado encontra os protótipos de funções que lhe podem ser úteis.

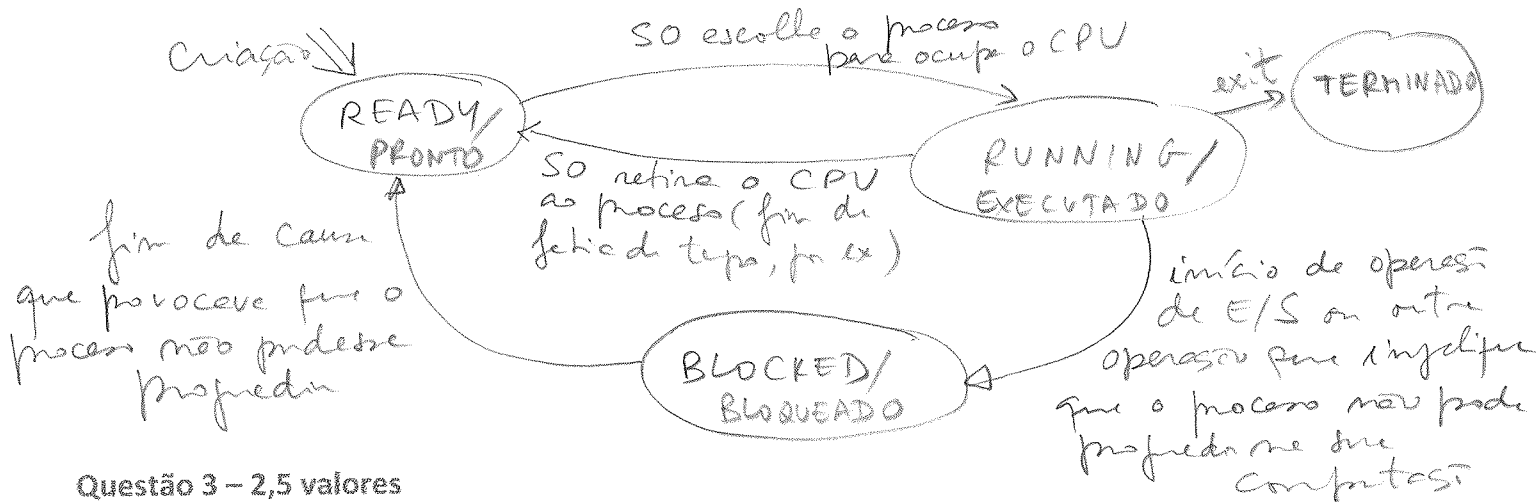
Questão 1 – 2,5 valores

Explique porque é que, para o sistema operativo conseguir cumprir as suas funções, o CPU tem de ter dois modos de funcionamento (sistema, utilizador). Indique quando é que se verifica uma mudança de modo do CPU.

- O SO reserva para si a manipulação do hardware. As instruções que manipulam o hardware só estão acessíveis quando o CPU está em modo sistema.
- O CPU muda para modo sistema quando há um interrupto.
- O CPU volta ao modo utilizador quando executa um instrução mediante `return-from-interrupt`.

Questão 2 – 2,5 valores

Considere um sistema operativo que suporta múltiplos processos carregados em memória em que em cada processo existe apenas um fluxo de execução (um único thread). Faça um diagrama de estados em que mostre os estados em que um processo pode existir ao longo da sua vida. Nos arcos que ligam os estados, indique abreviadamente o que é que provoca a mudança de estado em causa.



Questão 3 – 2,5 valores

Considere um sistema operativo que suporta múltiplos processos carregados em memória em que em cada processo existe apenas um fluxo de execução (um único thread). Explique como é que o sistema operativo preserva o estado da computação do processo enquanto este está nos estados PRONTO (READY) e BLOQUEADO (WAITING).

A computação que o processo está a fazer é representada por três componentes: CPU virtual, memória virtual e tabela de canais abertos. Cada um deles é preservado da seguinte forma:

- CPU virtual: o SO guarda o conteúdo dos registos no descritor do processo.
- Memória virtual: o SO programa o hardware de protecção de memória de forma a impedir o acesso à memória reservada ao processo.
- Tabela de canais: guarda a memória do SO; acessível via descritor.

Questão 4 – 2,5 valores

Quando, num sistema UNIX, um processo (pai) executa a chamada ao sistema `fork()` o sistema operativo cria um novo processo (filho) e associa-lhe uma máquina virtual nova. Explique em detalhe qual é o estado inicial da máquina virtual criada, e como é que esse estado é preenchido.

CPU virtual: é inicializado com o estado do CPU virtual do processo pai. A única diferença é o conteúdo do registo que contém o valor retornado pelo `fork`.

Memória virtual: são reservadas zonas de memória de dimensão e conteúdo iguais às do pai.

Tabela de canais abertos: o processo filho recebe a tabela de canais abertos que é uma cópia da do pai.

Questão 5 – 2,5 valores

Pretende-se lançar em execução os programas que estão armazenados nos ficheiros executáveis `"prog1"`, `"prog2"` e `"prog3"` que estão guardados na directoria corrente e que são invocados sem argumentos. Apresente o código que garante que:

- Os processos que executam `"prog1"` e `"prog2"` são lançados em execução simultânea
- O processo que executa `"prog3"` só começa a executar depois de ambos os processos que executam `"prog1"` e `"prog2"` terminarem
- O programa só termina depois de `"prog3"` terminar

```
p1 = fork();
if (p1 == 0)
    ..., execvp("prog1", ...);
else {
    p2 = fork();
    if (p2 == 0)
        ..., execvp("prog2", ...);
    else {
        wait(...);
        wait(...);
    }
}

p3 = fork();
if (p3 == 0)
    ..., execvp("prog3", ...)
else
    wait(...);
```

Questão 6 – 2,5 valores

Considere o código seguinte

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int cont = 0;
void *worker(void *arg) {
    int i;
    int n = (int)arg;
    for( i = 0; i < n; i++ )
        cont ++;
}

int main(int argc, char *argv[]) {
    pthread_t p1, p2, p3;
    pthread_create(&p1, NULL, worker, (void*)1000000);
    pthread_create(&p2, NULL, worker, (void*)1000000);
    pthread_create(&p3, NULL, worker, (void*)1000000);
    pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);
    printf("%d\n", cont);
    return 0;
}
```

- a) O código acima apresentado produz sempre o resultado 3000000? Justifique a resposta.

cont é uma variável global e o seu incremento não é uma ação atômica. Assim sendo, podem se perder increments e obtidos resultados diferentes

- b) Suponha que se acrescentou à biblioteca Pthreads uma função que corresponde a uma instrução máquina que **executa de forma indivisível**

```
int exchange( int *v1, int *v2){
    // esta função é executada de forma indivisível mesmo com mais de um CPU
    int t = *v2; *v2 = *v1;
    return t;
}
```

Utilizando esta função, modifique o programa anterior de forma a garantir que ele produz sempre resultados correctos.

É preciso garantir a exclusão mútua, fazendo lock(--) antes do incremento de cont e unlock(--) depois

| | | |
|--|---|--|
| <pre>int l; // Inicializar void init(){ l = 0; }</pre> | <pre>void lock(){ int one = 1; while (exchange(&one, &l) == 1); }</pre> | <pre>void unlock(){ l = 0; }</pre> |
|--|---|--|

Questão 7 – 2,5 valores

Considere uma variante do problema do produtor consumidor em que o buffer cresce à medida que é necessário; os itens produzidos e consumidos são do tipo *char*. Complete o código apresentado.

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

```
#define CHUNKSIZE 16
char *start;
int currentBufSize;
int nitems;
int put;
int get;
pthread_mutex_t ex = PTHREAD_MUTEX_INITIALIZER;
// missing code
```

```
pthread_cond_t empty = PTHREAD_COND_INITIALIZER;
```

```
void initBuf(){
    start = malloc(CHUNKSIZE); currentBufSize = CHUNKSIZE; put = 0; get = 0; nitems = 0;
}
```

```
void putBuf( char item){
    pthread_mutex_lock(&ex);
    if (put == currentBufSize){
        start = realloc(start, currentBufSize+ CHUNKSIZE); // adds CHUNKSIZE bytes to the buffer
        currentBufSize = currentBufSize + CHUNKSIZE;
    }
    start[put] = item; put++; nitems++;
    // missing code
}
```

```
pthread_mutex_unlock(&ex);
}
```

```
char getBuf(){
    // missing code
```

```
pthread_mutex_lock(&ex);
while (nitems == 0) pthread_cond_wait(&empty, &ex);
char t = start[get];
get++; nitems--;
return t;
}
```

```
void *producer(void *arg) {
    FILE *f;
```

```
    f = fopen((char *)arg, "r");
    while( !feof(f) ) putBuf( (char)getc(f) );
    putBuf(127); fclose(f);
    return NULL;
}
```

```
void *consumer(void *arg) {
    char c;
    do{
        c = getBuf();
        if(c < 127) putchar(c);
    }while ( c != (unsigned char)127); // 127 is used to mark the end of file
    return NULL;
}
```

Questão 8 – 2,5 valores

Considere o código seguinte.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
int cont = 0;
sem_t ex;

void *fp1(void *arg) {
    sem_wait( &ex); cont = cont + 1; sem_post( &ex);
    return NULL;
}

void *fp2(void *arg) {
    sem_wait( &ex); cont = cont * 8; sem_post( &ex);
    return NULL;
}

void *fp3(void *arg) {
    sem_wait( &ex); cont = cont + 2; sem_post( &ex);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p1, p2, p3;
    sem_init( &ex, 0, 1 );

    pthread_create(&p1, NULL, fp1, NULL); pthread_create(&p2, NULL, fp2, NULL);
    pthread_create(&p3, NULL, fp3, NULL);
    pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);
    printf("%d\n", cont);
    return 0;
}
```

- a) Diga, justificando, se o programa produz sempre o mesmo resultado.

O resultado produzido pelo programa depende da ordem de execução das funções $fp1()$, $fp2()$ e $fp3()$

- b) Altere o código para que seja sempre impresso o valor 10. A ordem de execução deve ser

- b1) O que deve acrescentar na declaração de variáveis globais ?

$sem_t\ s12;$
 $sem_t\ s23;$

$fp1$ antes de $fp2$
 $fp2$ antes de $fp3$;

- b2) O que deve acrescentar no $main()$?

$sem_init(&s12, 0, 0);$
 $sem_init(&s23, 0, 0);$

- b3) Qual é o novo código para $fp1()$?

```
void *fp1(void *arg) {
    sem_wait( &ex); cont = cont + 1; sem_post( &ex);
    sem_post(&s12);
    return NULL;
}
```

b4) Qual é o novo código para *fp2()* ?

```
void *fp2(void *arg) {  
    sem_wait(&S12);  
    sem_wait( &ex); cont = cont * 8; sem_post( &ex);  
    sem_post(&S23);  
    return NULL;  
}
```

b5) Qual é o novo código para *fp3()* ?

```
void *fp3(void *arg) {  
    sem_wait(&S23);  
    sem_wait( &ex); cont = cont + 2; sem_post( &ex);  
  
    return NULL;  
}
```

Algumas chamadas ao sistema UNIX/Linux

```
int fork( )  
int execvp( char *executable_file, char * args[ ] )  
int wait( int *status)
```

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 )
```