MC514—Sistemas Operacionais: Teoria e Prática

Profa. Islene Calciolari Garcia

2 de maio de 2006

Questão	Nota
1	
2	
3	
4	
5	
Total	

Nome:	RA:

Instruções: Você pode fazer a prova a lápis (desde que o resultado final seja legível :-) e utilizar o verso das folhas para rascunho ou para completar a resolução das questões. Não é permitida consulta a qualquer material manuscrito ou impresso. Em caso de fraude, todos os envolvidos receberão nota zero.

Funções:

```
int  pthread_create(pthread_t  *thread,
         pthread_attr_t *attr,
         void * (*start_routine)(void *),
         void *arg);

int pthread_join(pthread_t th,
        void **thread_return);

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t * sem);
int sem_post(sem_t * sem);
```

1 (2.0) Analise o código abaixo:

```
#define N_THR 5

void* f_thread(void *v) {
  int thr_id;
  thr_id = *(int *) v;
  printf("Thread %d. ", thr_id);
  return NULL;
}

int main() {
  pthread_t thr[N_THR];
  int i;

  for (i = 0; i < N_THR; i++)
    pthread_create(&thr[i], NULL, f_thread, (void*) &i);

  for (i = 0; i < N_THR; i++)
    pthread_join(thr[i], NULL);

  return 0;
}
```

A seguinte saída seria possível? Em caso afirmativo, apresente resumidamente uma seqüência de execução das threads que levaria a esta saída. Em caso negativo, explique o motivo.

```
Thread 0. Thread 0. Thread 0. Thread 0. Thread 4.
```

2 (2.0) No algoritmo da padaria proposto por Lamport, duas ou mais threads podem escolher a mesma senha e o desempate é feito por meio do identificador da thread. Um programador achou isto injusto e preferiu tentar implementar um esquema no qual cada thread escolheria garantidamente uma senha diferente. A idéia é que, durante a operação de máximo, se uma thread estiver escolhendo um número, a outra aguarda. Analise o código abaixo e responda se este código poderia apresentar algum problema.

```
/* Variáveis globais */
escolhendo[N] = { false, false, ..., false }
num[N] = { 0, 0, ..., 0 }

Thread i:
  /* max_i e j são variáveis locais */
  escolhendo[i] = true;
  max_i = 0;
  for (j = 0; j < N; j++)
    if (j != i)
      while (escolhendo[j]) ; /* Espera j terminar de escolher */
    if (num[j] > max_i)
        max_i = num[j]
  num[i] = max_i + 1;
  escolhendo[i] = false;

  /* Espera threads com senhas menores saírem da região crítica */
  for (j = 0; j < N; j++)
    while (num[j] != 0 && num[j] < num[i]) ;

  regiao_critica();
  num[i] = 0;
```

3 (2.0) A função `pthread_join()` tem por objetivo esperar uma thread terminar de executar e receber seu o valor de retorno. Supondo que esta função não existisse, como você faria para simular o seu comportamento **sem utilizar espera ocupada**? Implemente apenas a espera pelo término da execução. Não precisa se preocupar com o valor de retorno.

```
int pthread_join(pthread_t th, void **thread_return);
```

```
/* Declaração de variáveis globais */
```

```
/* Código a ser incluído na função executada pela thread */
```

```
/* Código a ser incluído no ponto de espera pelo término da execução da thread */
```

```
pthread_mutex_t lock;
pthread_cond_t cond;
volatile int nl = 0; /* Número de leitores */
volatile int ne = 0; /* Número de escritores */

void *leitor(void* v) {
  pthread_mutex_lock(&lock);
  while (ne > 0)
    pthread_cond_wait(&cond, &lock);
  nl++;
  pthread_mutex_unlock(&lock);

  leitura(v);

  pthread_mutex_lock(&lock);
  nl--;
  pthread_cond_broadcast(&cond);
  pthread_mutex_unlock(&lock);

  return NULL;
}
```

```
void *escritor(void *v) {
  pthread_mutex_lock(&lock);
  while (nl > 0)
    pthread_cond_wait(&cond, &lock);
  ne++;
  pthread_mutex_unlock(&lock);

  escrita(v);

  pthread_mutex_lock(&lock);
  ne--;
  pthread_cond_broadcast(&cond);
  pthread_mutex_unlock(&lock);

  return NULL;
}
```

Responda as questões abaixo, justificando-as.

(a) Há garantia de que apenas um escritor executa a operação de escrita em um dado instante?

(b) Vários leitores podem executar simultaneamente a operação de leitura em um dado instante?

(c) Há possibilidade de starvation de leitores?

(d) Há possibilidade de starvation de escritores?

5 (2.0) Suponha que um processo invoca a função `fork()`. Imediatamente depois, este processo pai instala um tratador de sinais e vai dormir com o comando `pause()` (que interrompe a execução de um processo até que este receba um sinal). Quando começa a executar, o filho envia um sinal SIGALRM para acordar o pai. Quando recebe o sinal, o pai exibe uma mensagem e termina sua execução.

```
void trata_SIGALRM(int signum) {
  printf("Ai que sono! Queria dormir mais...\n");
}

int main() {

  if ((pid = fork()) != 0) {
    signal(SIGALRM, trata_SIGALRM); /* Instalação do tratador de sinal */
    pause();  /* Pai espera ser acordado pelo filho */
  }
  else
    kill (getppid(), SIGALRM); /* Filho envia sinal para acordar o pai */

  return 0;
}
```

Descreva **dois** cenários em que este código apresentaria uma comportamento diferente do descrito acima.