

Número:

Nome:

## LEIC/LETI – 2015/16 - 1º Teste de Sistemas Operativos

29 de Outubro de 2016

Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.

Duração: 1h

### Grupo I [10 Val]

1. Considere o seguinte programa sequencial, em que  $f$  é uma função demorada.

```
1 int main() {
2     int i, x, r;
3     for (i=0; i<3; i++) {
4         x = lerInteiroDoStdin();
5         r = f(x);
6         printf("f(%d%)=%d\n", x, r);
7     }
8 }
```

- a. [2,5v] Construa uma versão paralela do programa acima em que as linhas 5-6 de cada iteração devem ser executadas dentro de um novo processo filho criado para essa iteração. Após o ciclo *for*, o processo pai deve esperar até que todos os processos filho terminem. Assim que o processo pai observa que um processo filho terminou, o pai deve imprimir "Filho *pid* terminou" no *stdout* (em que *pid* denota o identificador do filho). Na sua solução pode omitir o tratamento de chamadas sistema sem sucesso.

b. Considere as seguintes situações hipotéticas. Para cada uma, diga se pode ou não acontecer, e justifique.

i. [1v] Executando-se a solução paralela (construída na alínea anterior), foram recebidos pelo *stdin* os inteiros 0, 1, 2 (por esta ordem); no *stdout* foram impressas as seguintes linhas (por esta ordem):

```
f(1)=[...]  
f(0)=[...]  
f(2)=[...]
```

ii. [1v] O processo pai imprimiu “Filho 1230 terminou” mas o processo 1230 nunca chegou a imprimir o resultado da sua execução (linha 6) no ecrã.

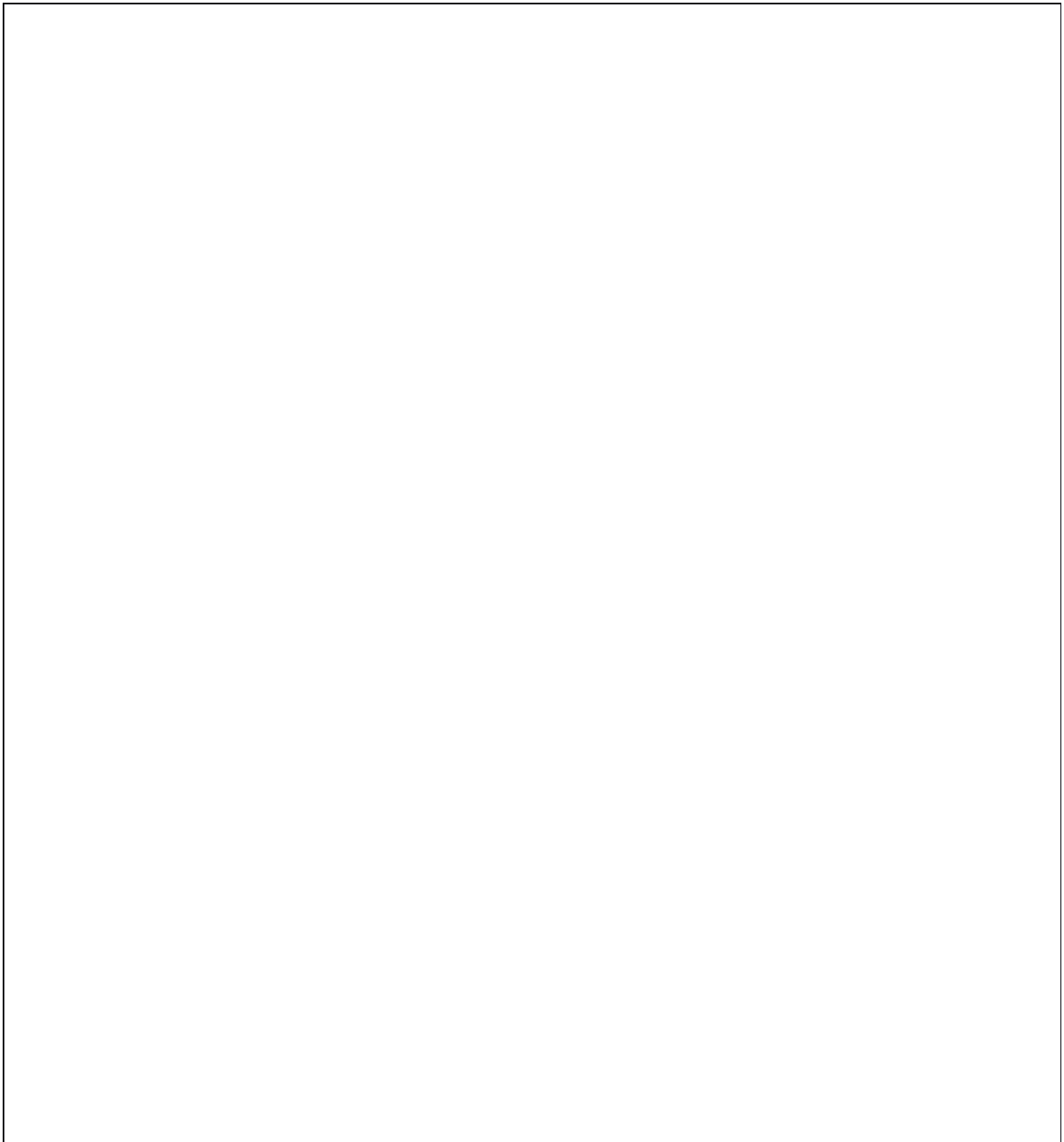
c. [2,5v] A função *f* tem um *bug* e por vezes entra num ciclo infinito. Complemente a sua resposta à alínea a) para que, caso os processos filho não terminem ao fim de 60 segundos, o processo pai os termine de forma abrupta enviando um sinal SIGKILL para cada processo filho.

Sugestão: recorra à função sistema *alarm*, que as *man pages* descrevem da seguinte forma:

```
#include <unistd.h>  
unsigned int alarm(unsigned int seconds);  
Description  
alarm() arranges for a SIGALRM signal to be delivered to the calling  
process in seconds seconds.
```

Na sua solução pode omitir o tratamento de chamadas sistema sem sucesso. Desde que o faça de forma clara, pode apresentar apenas as partes da sua solução que são modificadas.

---



- d. [1v] Uma solução alternativa teria sido executar cada iteração numa nova tarefa criada para o efeito, dentro do mesmo processo inicial. Ou seja, multi-tarefa em vez de multi-processo. Indique uma vantagem dessa nova solução.



2. Na disciplina estudou estas 2 formas de implementar um trinco lógico (entre outras): Solução A: algoritmo de Lamport (algoritmo *Bakery*); Solução B: trinco lógico com suporte do núcleo.

- a. [1v] Assuma que a tarefa t1 chama *fechar(trinco)* numa situação em que o trinco está fechado, logo t1 espera até que a tarefa na posse do trinco o abra. Essa espera é feita em espera ativa? Justifique.

Solução A:

Solução B:

- b. [1v] Alguma das soluções recorre a trincos *hardware*? Se sim, indique em que situação.

Solução A:

Solução B:

**Grupo II [10 Val]**

1. Considere a seguinte solução do problema clássico designado produtores-consumidores, apresentada em pseudo-código. Nesta solução, cada fio de execução (e.g., tarefa reais num dado processo): executa a função produtor se for produtor, executa a função consumidor se for consumidor

Programa prod-cons.c

```
1. int buf[N];
2. int prodptr=0, consptr=0;
3. trinco_t trinco;
4. semaforo_t pode_prod = criar_semaforo(N),
5. pode_cons = criar_semaforo(0);
```

```
1. produtor() {
2.     while(TRUE) {
3.         int item = produz();
4.         esperar(pode_prod);
5.         fechar(trinco);
6.         buf[prodptr] = item;
7.         prodptr = (prodptr+1)%N;
8.         abrir(trinco);
9.         assinalar(pode_cons);
10.    }
11. }
```

```
1. consumidor(){
2.     while(TRUE) {
3.         int item;
4.         esperar(pode_cons);
5.         fechar(trinco);
6.         item = buf[consptr];
7.         consptr = (consptr+1)%N;
8.         abrir(trinco);
9.         assinalar(pode_prod);
10.        consome(item);
11.    }
12. }
```

- a. [1 Val] Quais as instruções que estão directamente envolvidas na exclusão mútua? (Indique os números das linhas.)

- b. [1 Val] Altere o código para considerar apenas um produtor. Apresente a solução que use menos recursos. (Indique no espaço em baixo quais as linhas que são alteradas e como.)

- c. [1,5 Val] Considere agora que, na função `consumidor`, a linha 10 passa a constar entre as linhas 7 e 8. Qual o impacto no paralelismo da solução? Justifique a sua resposta.

- d. [1,5 Val] Considere o código originalmente apresentado nesta pergunta. Diga se a situação seguinte pode suceder: um produtor a inserir um item na posição 10 do *buffer*, e um consumidor a retirar um item da posição 5 do *buffer*. Justifique a sua resposta.

- e. Considere agora o código originalmente apresentado mas em que o semáforo `pode_prod` é inicializado com um valor  $M > N$ .

- i. [2 Val] Poderia ocorrer alguma situação errónea com alguma tarefa do tipo produtor? Justifique a sua resposta detalhadamente apresentando um cenário de execução.

- ii. [2 Val] Poderia ocorrer uma situação na qual um consumidor não ficasse bloqueado quando não existem itens no buffer? Justifique a sua resposta.

- f. [1 Val] Considere o código apresentado assim como os vários estados em que uma tarefa se pode encontrar. Diga em que estado ou estados é que uma tarefa produtora se pode encontrar. Para cada um do(s) estado(s) que menciona, indique uma linha na função `produtor` que lhe(s) corresponda.