

Módulo 1 - Lista de Exercícios (2020/2 REMOTO)

Computação Concorrente (MAB-117)

Prof. Silvana Rossetto

¹DCC/IM/UFRJ

13 de abril de 2021

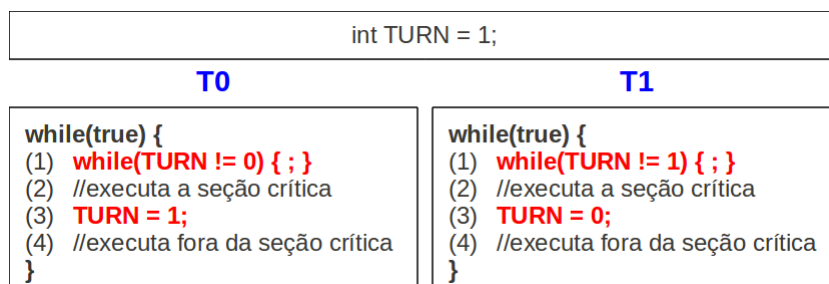
Questão 1 (2 pontos) Responda as questões abaixo, justificando todas as respostas:

- (a) O que caracteriza que um programa é concorrente e não sequencial?
- (b) Qual será a *aceleração máxima* de uma aplicação que possui 5 tarefas que consomem o mesmo tempo de processamento, das quais 4 poderão ser executadas de forma concorrente e 1 precisará continuar sendo executada de forma sequencial, depois que as 4 primeiras terminarem?
- (c) O que é *seção crítica* do código em um programa concorrente?
- (d) Como funciona a *sincronização por exclusão mútua*?

Questão 2 (2 pontos) Uma aplicação dispara três threads (T1, T2 e T3) para execução (códigos mostrados abaixo). Verifique se os valores -3, -1, 1 e 3 podem ser impressos na saída padrão quando essa aplicação é executada. Em caso afirmativo, mostre uma sequência de execução das threads que gere o valor correspondente. Justifique suas respostas.

```
int x=0; //variavel global
(0)      T1:                T2:                T3:
(1)      x++;                x--;                x++;
(2)      x--;                x--;                x++;
(3)      if (x == 0)         if (x == -2)         if (x == 2)
(4)      printf("%d", x);    printf("%d", x);    printf("%d", x);
```

Questão 3 (2 pontos) O código abaixo apresenta uma proposta de implementação de seção crítica com espera ocupada (ao invés de serem bloqueadas, as threads executam um loop de entrada na seção crítica). A solução proposta prevê apenas duas threads (T0 e T1). (a) Essa implementação garante exclusão mútua? Se sim, argumente justificando sua resposta. Se não, descreva cenários de execução que mostrem que a solução é incorreta. (b) Essa solução atende aos demais requisitos de implementação de seção crítica? (Ver texto Cap2.)



Questão 4 (2 pontos) Considere uma aplicação com as três threads mostradas abaixo (T1, T2 e T3). Usando *locks*, reescreva o código das threads para eliminar as condições de corrida ruins. Minimize os trechos de seção atômica, possibilitando o máximo de execução concorrente. Justifique suas decisões/respostas.

```
int x=0, y=0; //variaveis globais
1) void *T1(void *id){          | void *T2(void *id){          | void *T3(void *id){
2)  int a=0;                   |  int a=2;                   |  x--;
3)  while(a<2) {               |  while(a>0) {               |  x++;
4)    x++;                     |    x++;                     |  y++;
5)    x--;                     |    x--;                     |  }
6)    if(x==0)                 |    if(x==0)                 |
7)      {printf("x=%d\n",x);} |    {printf("x=%d\n",x);} |
8)    a++;                     |    a--;                     |
9)    printf("a=%d\n", a);    |    fprintf(file,"a=%d\n", a);|
    } }                       |  } }                       |
```

Questão 5 (2 pontos) Analise o código mostrado abaixo. (a) Ao final da execução desse código, qual conteúdo estará escrito no arquivo bar? (b) Há *condição de corrida* nesse código? Justifique suas respostas.

```
#include<stdio.h>
#include<pthread.h>
void * tarefa(void *arg) {
    FILE *fd_bar, *fd_foo;
    fd_bar = fopen("bar", "a+");
    if(fd_bar) {
        fprintf (fd_bar, "%s", "mundo!\n");
        fd_foo = fopen("foo", "r");
        if(!fd_foo){
            fprintf(fd_bar, "%s", "foo nao existe\n");
        } else fclose(fd_foo);
        fclose (fd_bar);
    }
    pthread_exit (NULL);
}

int main(void) {
    FILE *fd;
    pthread_t tid;
    fd = fopen("foo", "w");
    if (!fd) return 1;
    fprintf(fd, "%s", "Ola ");
    fclose(fd);
    if (pthread_create(&tid, NULL, tarefa, NULL)) return 2;
    rename ("foo", "bar");
    pthread_join(tid, NULL);
    return 0;
}
```