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

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ
23 de dezembro de 2020

Questão 1 Considere o programa mostrado abaixo, que contabiliza a quantidade de números negativos presentes em um vetor de inteiros e responda as questões colocadas:

```
//variaveis globais/compartilhadas
//numero de threads e tamanho do vetor
int nthreads, tam;
//vetor de elementos
int *vet;
//vetor de resultados
int *resultados;

//funcao executada pelas threads
void *negativos (void *tid) {
  int id = * (int *) tid;
  //intervalo de elementos processados por cada thread
  int inicio, fim, j;
  int tam_bloco = tam/nthreads;
  //variavel local para a qtde de negativos encontrados pela thread
  int qtde=0;

  //calcula o intervalo de elementos da thread
  inicio = id * tam_bloco;
  //o ultimo fluxo trata os elementos restantes
  if (id<nthreads-1) fim = inicio + tam_bloco;
  else fim = tam;
  for(j=inicio; j<fim; j++) {
     if(vet[j] < 0) qtde++;
  }

  //carrega o resultado parcial para o vetor de resultados
  resultados[id] = qtde;

  free(tid);
  pthread_exit(NULL);
}

int main() {
  pthread_t *tid_sistema; //vetor identificadores das threads no sistema
  int *tid; //identificadores das threads no programa
  int t; //variavel contadora
  int qtde_negativos=0; //contabiliza a qtde total de numeros negativos encontrados
  tam = atoi(argv[1]); //tamanho do vetor
  nthreads = atoi(argv[2]); //numero de threads

  //limita o numero de threads ao tamanho do vetor
  if(nthreads>tam) nthreads = tam;

  //aloca espaco para o vetor de identificadores das threads no sistema
  tid_sistema = (pthread_t *) malloc(sizeof(pthread_t) * nthreads);
  if(tid_sistema==NULL) {
      printf("--ERRO: malloc()\n"); exit(-1);
  }

  //aloca espaco para o vetor de resultados
  resultados = (int *) malloc(sizeof(int) * nthreads);
  if(resultados==NULL) {
      printf("--ERRO: malloc()\n"); exit(-1);
  }
```

```
  //Parte 1
  ...aloca espaço e inicializa o vetor de entrada

  //Parte 2: contabiliza a qtde de negativos no vetor
  //cria as threads
  for(t=0; t<nthreads; t++) {
    tid = malloc(sizeof(int));
    if(tid==NULL) { printf("--ERRO: malloc()\n"); exit(-1); }
    *tid = t;
    if (pthread_create(&tid_sistema[t], NULL, negativos, (void*) tid)) {
       printf("--ERRO: pthread_create()\n"); exit(-1);
    }
  }

  //espera todas as threads terminarem e calcula o valor de saida
  for(t=0; t<nthreads; t++) {
     if (pthread_join(tid_sistema[t], NULL)) {
        printf("--ERRO: pthread_join()\n"); exit(-1);
     }
     //atualiza o valor de saida
     qtde_negativos += resultados[t];
  }
  //Parte 3
  ...exibe os resultados e libera os espaços de mémoria alocados
}
```

(a) Esse programa está correto (i.e., ele calcula corretamente a quantidade de valores negativos no vetor)?
(b) É possível executar o programa com sucesso passando um número qualquer de threads na linha de comando?
(c) A carga de trabalho será sempre balanceada entre as threads, independente do número de threads informado?
(d) Há *condição de corrida* nesse código?

Justifique todas as respostas.

Questão 2 Responda as questões abaixo:

(a) O que caracteriza que um programa é concorrente e não sequencial?
(b) O que é seção crítica do código?
(c) O que significa uma operação ser atômica?
(d) Como funciona a sincronização por exclusão mútua?

Justifique todas as respostas.

Questão 3 Uma aplicação dispara três threads (T1, T2 e T3) para execução (códigos mostrados abaixo). (a) Verifique se os valores $-3, -2, 0, 2, 3$ 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.

```
int x=0; //variavel global
         T1:                T2:            T3:
(1)      x++;               x--;           x--;
(2)      x--;               x++;           x++;
(3)      x++;                              x--;
(4)      if (x == 1)                         if(x == -1)
(5)         printf("%d",x);                   printf("%d",x);
(6)
```

Questão 4 O código abaixo apresenta uma proposta de implementação de exclusão mútua 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.

```
boolean queroEntrar_0 = false, queroEntrar_1 = false;
```

T0

```
while(true) {
(1)  while(queroEntrar_1) { ; }
(2)  queroEntrar_0 = true;
(3)  //executa a seção crítica
(4)  queroEntrar_0 = false;
(5)  //executa fora da seção crítica
}
```

T1

```
while(true) {
(1)  while(queroEntrar_0) { ; }
(2)  queroEntrar_1 = true;
(3)  //executa a seção crítica
(4)  queroEntrar_1 = false;
(5)  //executa fora da seção crítica
}
```

Questão 5 O código abaixo apresenta outra proposta de implementação de exclusão mútua com espera ocupada. 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.

```
boolean queroEntrar_0 = false, queroEntrar_1 = false; int TURN;
```

T0

```
while(true) {
(1)  queroEntrar_0 = true;
(2)  TURN = 1;
(3)  while(queroEntrar_1 &&
                TURN == 1) { ; }
(4)  //executa a seção crítica
(5)  queroEntrar_0 = false;
(6)  //executa fora da seção crítica
}
```

T1

```
while(true) {
(1)  queroEntrar_1 = true;
(2)  TURN = 0;
(3)  while(queroEntrar_0 &&
                TURN == 0) { ; }
(4)  //executa a seção crítica
(5)  queroEntrar_1 = false;
(6)  //executa fora da seção crítica
}
```