Nome: João Vitor de Freitas Barbosa
DRE: 117055449

Computação Concorrente
Lista de exercícios 1

Questão 1:

a) O fato do mesmo programa executar diferentes tarefas simultaneamente

b) Bom sabemos que o ganho de velocidade da execução é dado por

$$\frac{T_{sequencial}}{t_s + t_p}$$

Temos então 5 tarefas que consomem o mesmo tempo t de processamento.
No programa sequencial, teríamos um tempo total de $5t$ para o processamento das 5 tarefas.
No concorrente, teríamos 4 tarefas executando de forma concorrente em um tempo t e após isso também temos a última tarefa executando de forma sequencial em um tempo t, portanto:

$$\frac{5t}{t+t} = \frac{5t}{2t} = \frac{5}{2} = 2.5$$ de ganho de desempenho

c) É uma região do programa onde estão variáveis compartilhadas por mais de um fluxo de execução (cada um lendo e escrevendo nessas variáveis), podendo assim um fluxo interferir negativamente no outro

d) Através de algum mecanismo o acesso a determinado bloco de código (seção crítica) fica limitado a uma única thread por vez

Questão 2:

Os valores -3, -1, 1 e 3 podem ser impressos na saída padrão quando essa aplicação é executada.
-3: Podemos pensar que a T2 começou o primeiro decremento e salvou o valor 0 no registrador mas antes de terminar, a T1 começou a fazer o seu incremento no x++ no valor atual de x que não foi alterado então temos 0+1= 1 e guardou de volta em x. O controle voltou para a T2, mas ela já tinha guardado anteriormente no registrador o valor 0, então ela fez o decremento de 0 duas vezes, tendo -2 e entra no if na linha 3 mas antes de executar o print, o controle volta para a T1 e faz o decremento mais uma vez na linha 2 tendo -3 como resultado. Depois disso o controle volta novamente para T2 e executa o print.

-1: Podemos pensar que a T1 começou e executou as duas primeiras operações, tendo como resultado 0 e entrando no if(x==0), mas antes de executar o print, o controle vai para T2 e executa o primeiro decremento, tendo como resultado -1. Após isso o controle retorna para T1 e executa o print de -1.

1: Podemos pensar que a T1 começou e executou as duas primeiras operações, tendo como resultado 0 e entrando no if(x==0), mas antes de executar o print, o controle vai para T3 e executa o primeiro incremento, tendo como resultado 1. Após isso o controle retorna para T1 e executa o print, exibindo 1.

3: Podemos pensar que a T3 começou e executou as duas primeiras operações de incremento, tendo como resultado 2, e entrando no if(x==2), mas antes de executar o print da linha 3, o controle vai para T1 e executa o primeiro incremento, tendo como resultado 3. Após isso, o controle retorna para T3 e executa o print, exibindo o número 3.

Questão 3:

Sim, essa implementação garante a exclusão mútua, pois a variável TURN é modificada somente após a execução da seção crítica, enquanto há um WHILE travando a execução de outra seção crítica.
Essa solução não atende aos demais requisitos de implementação de seção crítica, pois segundo a definição encontrada no texto cap2, para que as threads possam executar corretamente e chegar ao final das suas execuções, é necessário que toda thread que entre na seção crítica saia dela em algum momento, isto é, uma thread não deve executar um loop infinito ou terminar a sua execução dentro de uma seção crítica. No caso temos um loop infinito while(true)

Questão 4:

Fiz uso do lock para as secções críticas do código, onde há a leitura e escrita de variável global, como por exemplo no x++ e x--. Optei por não utilizar o mesmo no if antes do print pois é apenas chamada de leitura. Como a variável ***a*** é variável local em ambas as threads T1 e T2, não há necessidade de utilizar lock nos trechos em que ela é modificada.

```
int x = 0; //variavel global
int y = 0; //variavel global
pthread_mutex_t lock; //variavel especial para sincronizacao de exclusao mutua

void * T1(void *arg) {
   int a = 0;
   while(a < 2){
         //entrada de secao critica
         pthread_mutex_lock(&lock);
         x++; //incrementa a variavel compartilhada
         x--; //decrementa a variavel compartilhada
         pthread_mutex_unlock(&lock);
         //saida de secao critica
         if(x==0){
               printf("T1 x=%d\n", x);
               a++;
         }
         printf("T1 a=%d\n", a);
   }

   pthread_exit(NULL);
}

void * T2(void *arg) {
   int a = 2;
   while(a >0){
         //entrada de secao critica
         pthread_mutex_lock(&lock);
         x++; //incrementa a variavel compartilhada
         x--; //decrementa a variavel compartilhada
         pthread_mutex_unlock(&lock);
         //saida de secao critica
         if(x==0){
               printf("T2 x=%d\n", x);
         }
         a--;
         printf("T2 a=%d\n", a);
   }
   pthread_exit(NULL);
}
```

```
void * T3(void *arg) {
   //entrada de secao critica
   pthread_mutex_lock(&lock);
   x--;
   x++;
   y++;
   pthread_mutex_unlock(&lock);
   //saida de secao critica
   pthread_exit(NULL);
}
```

Questão 5:

a)
Ao final da execução desse código, o conteúdo escrito no arquivo bar é o seguinte:
Ola mundo!
foo nao existe

b) A condição de corrida se encontra no momento em que a thread principal, a da main acessa o arquivo *foo* e depois modifica seu nome para *bar*. Nisso, quando a thread que executa a função tarefa, tenta abrir o arquivo *foo* para leitura, ele não encontra e acaba escrevendo “foo nao existe” em bar.
“Ola” é originalmente escrito em *foo,* após isso, o arquivo foo é renomeado para bar.
A thread que executa tarefa, escreve “mundo!” em *bar* e logo após também escreve “foo nao existe” em *bar* pois não encontrou *foo*.
Ambas tentam acessar o mesmo arquivo, sendo que uma exclui o mesmo antes da outra conseguir o acesso.