Estudo Dirigido 1 (Lista 1)

Questão 1

a)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

long double pi(int n){
    long double term;
    term = (4.0 / ((8.0 * n) + 1.0)) -
           (2.0 / ((8.0 * n) + 4.0)) -
           (1.0 / ((8.0 * n) + 5.0)) -
           (1.0 / ((8.0 * n) + 6.0));

    return term * (1.0 / pow(16.0, n));
}
int main(int argc, char*argv[]){
    if (argc < 2)
    {
        printf("Error! digite %s <n de iteracoes>", argv[0]);
        return 1;
    }
    int n = atoi(argv[1]);

    long double valor_pi = 0.0;
    for(int i=0; i<n; i++){
        valor_pi += pi(i);
    }
    printf("pi = %Lf\n", valor_pi);
    return 0;
}
```

b)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pthread.h>

typedef struct{
    short int nthreads;
    int n;
    short int id;
```

```c
} t_args;

long double calcula_pi(int n){
    long double term;
    term = (4.0 / ((8.0 * n) + 1.0)) -
           (2.0 / ((8.0 * n) + 4.0)) -
           (1.0 / ((8.0 * n) + 5.0)) -
           (1.0 / ((8.0 * n) + 6.0));

    return term * (1.0 / pow(16.0, n));
}

void* task(void* args){
    t_args* arg = (t_args*) args;

    long double *soma= (long double*) malloc (sizeof(long double));
    *soma = 0.0;
    for (int i = arg->id; i < arg->n; i+= arg->nthreads)
    {
        *soma += calcula_pi(i);
    }
    free(args);
    pthread_exit((void*) soma);
}


int main(int argc, char* argv[]){
    if (argc < 3){
        printf("Erro! Digite <Num Thredas> e <Num iteracoes>");
        return -1;
    }

    short int nthreads = atoi(argv[1]);
    int n = atoi(argv[2]);
    if (nthreads > n) nthreads = n;

    pthread_t tids[nthreads];

    for(int i = 0; i < nthreads; i++){
        t_args *arg = (t_args*) malloc (sizeof(t_args));

        arg->id = i;
        arg->n = n;
        arg->nthreads = nthreads;

        if (pthread_create(&tids[i], NULL, task, (void *) arg)){
            printf("--ERRO: pthread_create()\n"); exit(-1);
        }
    }

    long double *result_indv, resultado = 0.0;
    for (long int i = 0; i < nthreads; i++)
    {
        if (pthread_join(tids[i],(void *) &result_indv))
```

```
        {
            printf("--ERRO: pthread_join()\n"); exit(-1);
        }
        resultado += *result_indv;
        free(result_indv);
    }

    printf("\n\t pi = %Lf", resultado);
    return 0;
}
```

Questão 2

```
void* task(void* args){
    //tret* ret = (tret*) malloc(sizeof(tret));
    targ *arg = (targ*) args;

    int fatia = arg->tam / arg->nthreds;
    int inicio = arg->id * fatia;
    int fim = (arg->id == (arg->nthreds-1))? arg->tam : inicio + fatia;
    int min_local = min;
    int max_local = max;
    for(int i = inicio; i < fim; i++){
        //printf("thread %ld -> %d\n", arg->id, vetor[i]);
        if(vetor[i] < min_local) min_local = vetor[i];
        if(vetor[i] > max_local) max_local = vetor[i];
    }
    pthread_mutex_lock(&mutex);
    if(min_local < min) min = min_local;
    if(max_local > max) max = max_local;
    pthread_mutex_unlock(&mutex);

}
```

Questão 3

a)

Seção crítica é uma área do código onde é compartilhada entre as threads, mas que devem ser executadas de forma atômica, ou seja, não pode ser executada por mais de uma thread simultaneamente para evitar inconsistências

b)

Corrida de dados é quando um mais de um fluxo de execução se entrelaça na hora da escrita, ou seja, mais de uma thread acessam a mesma posição da memória ao mesmo tempo.
ocorre quando há dois ou mais acessos concorrentes a uma mesma posição de memória, e pelo menos um desses acessos é de escrita.

c)

A violação de atomicidade ocorre quando uma sequência de operações que deveria ser executada como um bloco único e indivisível é interrompida por outra thread.

d)

Violação de ordem se dá quando duas execuções distintas ocorrem de ordens invertidas, ou seja, quando a ordem de precedência não respeitada.

e)

Visa garantir que **os trechos de código em cada thread que acessam objetos compartilhados não sejam executados ao mesmo tempo**, ou que uma vez iniciados, são executados até o fim sem que outra thread inicie a execução do trecho equivalente.

f)

Garante o bloqueios de uma ou mais threads até um sinal de liberação seja mandado .
Visa garantir que **uma thread seja retardada enquanto uma determinada condição lógica da aplicação não for satisfeita**

g)

Para evitar os problema que surgem com a programação concorrente, tais como, corrida de dados e violação de atomicidade e ordem.

h)

É criado variáveis em comum que é compartilhada entre as threads, tal trecho de código onde há esse compartilhamento é chamado de seção crítica. E visando evitar problemas como corrida de dados, queremos que a ações executadas na seção crítica sejam atômicas, ou seja, uma por vez, e isso é feito através do pthread_mutex.

Questão 4

a)

a thread 1 é iniciada, S1 é executado, então a thread 1 é pausada, então a thread 2 é iniciada, S3 é executado, thread 2 termina, thread 1 é despausada S2 é executado e a thread 1 termina.
S1 -> S3 -> S2

b)

Adicionar pthread_mutex nas seções S1 e S2

c)

Adiconamos uma variavel estado = 0 e junto com ela implementar pthread_cond. Enquanto a variavel estador for igual a 0, a thread 2 é bloqueada. Uma vez que a thread 1 terminar de inicializar mThread, mudamos a variavel estado para 1 e desbloqueamos a thread 2.

Questão 5

-4 : Não Alcançável
-3 : T3(1) -> T3(2) -> T3 Pausa ->T1(1) Sobrescreve -> T1(2) / T2(2) Sobrescreve -> T1(3) -> T3(3)
-2 : T2(1)/T1(1) T1 Sobrescreve -> T1(2) -> T1(3) -> T1(4) -> T2(2) -> T1(5)
-1 : T1(1) -> T1(2) -> T1(3) -> T1(4) -> T1(5)
0 : T3(1) -> T3(2) -> T1(1) -> T1(2) -> T1(3) -> T3(3)
1 : T3(1) -> T3(2) -> T3(3)
2 : T3(1) -> T3(2) -> T2(1) -> T3(3)
3 : T3(1) -> T3(2) -> T1(2) -> T2(1) -> T3(3)
4 : Não Alcançável

Questão 6

a)

x=0
Uma execução sem interrupções de T1 ou T2

x=-1
Todas as threads começam a executar, T3 começa a executar a linha 2 e pausa, T1 ou T2 executam até a linha 6 e pausa, T3 então termina de executar a linha 2 e sobrescreve, T1 ou T2 executam a linha 7.

x=2
Todas as threads começam a executar, T3 ganha prioridade e executa a linha 2 e é pausada, T1 tem prioridade e executa até a linha 5 e é pausada, T2 executa até a linha 4 e é pausada, T1 retorna e executa a linha 6 e é pausada, T2 retorna junto com T3 e a linha 3 de T3 sobrescreve a linha 5 de T2, T3 termina, T2 vai para o proximo loop, executa a linha 4 e é pausada, T1 retorna e imprime.

x=4
Não alcançável.

b)

Sim, as três threads acessam uma a mesma variável, todas realizam as mesmas operações (incremento e decremento), e há risco de sobrescrita pois não nenhum definição de ordem de execução deixando a cargo do sistema.

Questão 7

Sim, o problema está na leitura da variável meuSaldo, pois pode ocorrer violação de atomicidade e ordem, fazendo com o que uma thread ao passar da condição if, antes que possa executar retira(val), o valor de saldo é atualizado, fazendo com que um novo decremento seja feito no novo valor.

Questão 8

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int estado = 0, aux = 0;
pthread_mutex_t mutex;
```

```c
pthread_cond_t cond_t1, cond_t2;

void* t1(void *arg){
    pthread_mutex_lock(&mutex);

    printf("T1: ola, voce esta acessando a variavel 'aux' agora?\n");
    estado = 1;
    pthread_cond_signal(&cond_t2);

    while (estado < 2) pthread_cond_wait(&cond_t1, &mutex);
    printf("T1: certo, entao vou altera-la, ta?\n");
    estado = 3;
    pthread_cond_signal(&cond_t2);

    while (estado < 4) pthread_cond_wait(&cond_t1, &mutex);
    aux = 14;
    printf("T1: terminei a alteracao da variavel 'aux'\n");
    estado = 5;
    pthread_cond_signal(&cond_t2);


    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}

void* t2(void *arg){
    pthread_mutex_lock(&mutex);

    while(estado < 1) pthread_cond_wait(&cond_t2, &mutex);
    printf("T2: oi, nao, nao estou\n");
    estado=2;
    pthread_cond_signal(&cond_t1);
    while(estado < 3) pthread_cond_wait(&cond_t2, &mutex);
    printf("T2: tudo bem\n");
    estado=4;
    pthread_cond_signal(&cond_t1);
    while(estado < 5) pthread_cond_wait(&cond_t2, &mutex);
    printf("T2: perfeito, recebido!\n");



    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}


int main(void){
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond_t1,NULL);
    pthread_cond_init(&cond_t2,NULL);

    pthread_t thread1, thread2;

    pthread_create(&thread1,NULL, t1, NULL);
```

```
    pthread_create(&thread2,NULL, t2, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_t1);
    pthread_cond_destroy(&cond_t2);

    return 0;
}
```

Questão 9

a)

Cada thread ficou responsável por k elementos, realizando saltos no vetor.

b)

Cada thread termina sua execução quando seu próximo salto for excedente ao tamanho do vetor

c)

não, caso o tamanho do vetor não seja divisivel pelo numero de threads, a divisão de elementos é desigual.

d)

Impede que haja violação de ordem, forçando as thread percorrem da esquerda para direita no vetor.

Questão 10

a)

na tarefa da thread A, a thread não espera a thread B imprimir o valor de X e continua incrementando, causando corrida de dados.

b)

adicionando uma variável para o estado e fazendo a thread A ser bloqueada.

c)

```
int x = 0, estado = 0;
pthread_mutex_t x_mutex;
pthread_cond_t x_cond, condA;
void *A (void *tid) {
    for (int i = 0; i < 100; i++)
    {
```

```
        pthread_mutex_lock(&x_mutex);
        while(estado > 0) pthread_cond_wait(&condA,&x_mutex);
        x++;
        if (x == 50)
        {
            estado = 1;
            pthread_cond_signal(&x_cond);
        }
        pthread_mutex_unlock(&x_mutex);
    }
}
void *B (void *tid) {
    pthread_mutex_lock(&x_mutex);
    while(estado < 1) pthread_cond_wait(&x_cond, &x_mutex);
    printf("X=%d\n", x);
    estado = 0;
    pthread_cond_signal(&condA);
    pthread_mutex_unlock(&x_mutex);
}
```