Q1

1 / 1

Q1. (a) Programação concorrente é aquela que lida com vários fluxos ou threads. A diferença do sequencial é a possibilidade de vários linhas de comando serem executados ao mesmo tempo.

(b)
Necessidade de lidar com dispositivos de forma assincronizada, e.g.: ao receber mensagens de clientes a partir de um servidor.
Necessidade de separar a estrutura lógica de um problema, e.g.: abas do navegador.
Necessidade de aumentar o desempenho das aplicações, como jogos.

(c) ~~Prog.~~ Aceleração máxima, segundo a lei de Amdahl, é o maior desempenho possível dado um número ilimitado de processadores.
Sendo assim, $ace_{máx} = \frac{5}{2} = 2,5$

(d) Parte compartilhada do código entre as threads, como um struct ou conexão de internet.

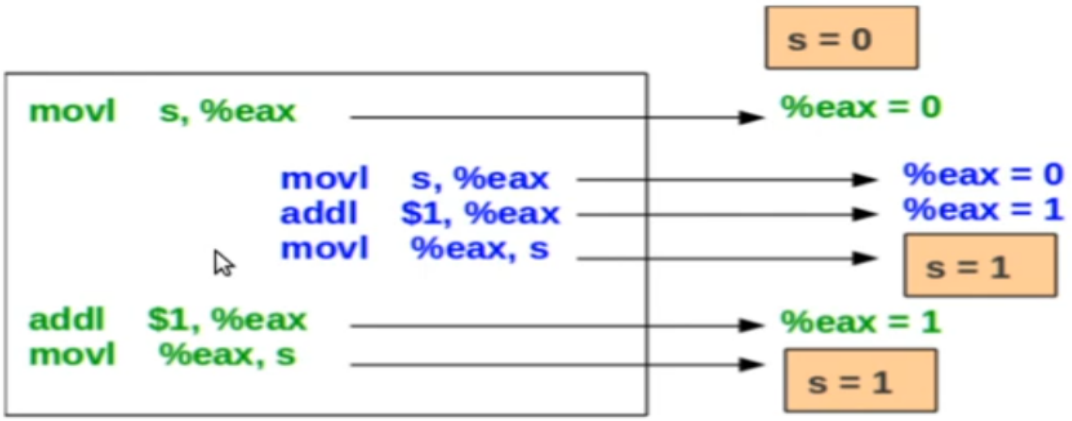
(e) Explicitamente bloqueia parte do código para uma thread, até que a seção seja liberada pelo thread anterior

Q2

```c
typedef struct{
    int id;
} Args;

int num_threads; //número de threads
int n; //último termo do somatório

void *tarefa(void *p){
    Args *args = (Args *) p;

    //evitar perda ao sair do escopo
    //vai precisar liberar dps, como feito no laboratório 3
    double *psoma = (double*) malloc(sizeof(double));

    for (int i = (args->id + 1); i <= n; i += num_threads){
       if(i%2){ //para evitar usar pow(-1,i);
             *psoma -= ((double) 4)/(2*i+1);
       }else{
             *psoma += ((double) 4)/(2*i+1);
       }
    }
    pthread_exit((void *) psoma);
}
```

Q3

Irei fazer para cada valor separadamente:
(TM)(N) com M sendo a thread e N sendo a linha.
Os comandos que estiverem na mesma linha irão representar isso:



-1:

(T1)(1) x-1;
(T1)(2) x+1;
(T1)(3) x-1;
(T1)(4) if(x==-1)
(T1)(5) printf("-1");

0:
(T1)(1) x-1;
(T1)(2) x+1;
(T1)(3) x-1;
(T1)(4) if(x==-1)
(T2)(1) x+1;
(T1)(5) printf("0");

2:
(T3)(1) x+1;
(T3)(2) if(x==1)
(T2)(1) x+1;
(T3)(3) printf("2");

-2
(T1)(1) x-1;
(T1)(2) x+1;
(T1)(3) x-1; (T2)(1) x+1;
(T1)(4) if(x==-1)
(T2)(2) x-1;
(T1)(5) printf("-2");

3
(T3)(1) x+1;
(T3)(2) if(x==1)
(T2)(1) x+1; (T1)(1) x-1;
(T1)(2) x+1;
(T3)(3) printf("3");

-3
não há como

4
não há como

Q4

O problema dessas duas threads é o pequeno período de tempo em que o “queroEntrar_N” é escrito em cada um. Mesmo que a T0 entre primeiro, existe chance do T1 executar o queroEntrar_1 imediatamente depois, impedindo as duas threads. Vice-versa tmb.
Mesmo com o tempo suficientemente grande entre as duas threads, ao ponto de passar pela condição do while, o fato de haver um segundo while(true) no início eventualmente

garante comportamento caótico, persistindo com o problema… Mas a seção crítica seria executada.
Supondo que o caso seja do **segundo** parágrafo.

(a) sim
(b) sim
(c) Supondo o caso mais próximo do primeiro parágrafo, as duas threads iriam tornar queroEntrar=true ao mesmo tempo, prendendo uma a outra em loop infinito.
(d) Sim. As seções não conseguem ser executadas ao mesmo tempo. Basta observar como o queroEntrar_N influencia na outra thread, não nela mesma. I.e.: while só pode ser passado quando a outra thread permitir.