DIVISÃO CÓDIGO

Aplicação.c:

* thread1\_func(void \*arg)
* thread2\_func(void \*arg)
* main(void)

my\_semaphore.c:

* TLista \*tlista\_inicia()
* lista\_inicia(TGrafo \*pgrafo, int nvertices)
* lista\_insere(TGrafo \*pgrafo, int u, int v)
* tgrafo\_inicia(TGrafo \*pgrafo, int NVertices)
* tgrafo\_inserearesta(TGrafo \*pgrafo, int u, int v)
* criaPilha(void)
* PilhaEhVazia(TPilha\* p)
* Empilha(TPilha\* p, int v)
* Desimpilha(TPilha\* p)
* dfs(int v)
* temCiclo()
* converteVertice(long valor\_do\_vertice)
* sem\_wait(sem\_t \*sem)

ISADORY:

Aplicação.c:

* + thread1\_func(void \*arg)
  + thread2\_func(void \*arg)
  + main(void)

my\_semaphore.c:

* + TLista \*tlista\_inicia()
  + lista\_inicia(TGrafo \*pgrafo, int nvertices)
  + lista\_insere(TGrafo \*pgrafo, int u, int v)
  + dfs(int v)
  + temCiclo()

DAY:

Aplicação.c:

my\_semaphore.c:

* + tgrafo\_inicia(TGrafo \*pgrafo, int NVertices)
  + tgrafo\_inserearesta(TGrafo \*pgrafo, int u, int v)
  + criaPilha(void)
  + PilhaEhVazia(TPilha\* p)
  + Empilha(TPilha\* p, int v)
  + Desimpilha(TPilha\* p)
  + converteVertice(long valor\_do\_vertice)
  + sem\_wait(sem\_t \*sem)

DIVISÃO VÍDEO

ISADORY:

aplicacao.c

Agora eu vou explicar o programa que executa de fato a verificação da ocorrência do deadlock.

Para a execução do programa, as bibliotecas <semaphore.h> e <pthread.h> foram fundamentais, por conterem os tipos de variáveis necessárias para a implementação.

Iniciando a explicação, nas linhas 6 e 7 são declaradas as variáveis do tipo sem\_t, que são do tipo semáforo.

Já nas próximas linhas, tem-se a implementação das threads que serão executadas. Entre as linhas 10 e 25 foi implementada a thread 1. Quando é realizada a sua execução, é impressa uma mensagem no terminal, de modo a informar qual thread está sendo executada, além das informações do semáforo ( seu tamanho em Bytes) e o ID da thread em execução. Em seguida, nas linhas 14 e 15 executa-se a função sem\_wait, reescrita neste trabalho, para fazer a detecção do deadlock, mas não bloquear o semáforo em mutex1 e mutex2. Nas linhas 16 e 17, a função sem\_post é aplicada nas variáveis de semáforo mutex1 e mutex2, a fim de desbloquear os semáforos. No seu retorno, se o valor do semáforo torna-se maior do que zero, então outro processo ou thread bloqueado em uma [chamada sem\_wait](https://man7.org/linux/man-pages/man3/sem_wait.3.html) será ativado e prosseguido para bloquear o semáforo. Já para valor igual a 0, a operação é bem sucedida. Por fim, é aplicada a função pthread\_exit, que termina o thread de chamada.

Nas linhas 27 a 43, é implementada a thread 2, que tem o mesmo processo de implementação explicado anteriormente.

Da linha 46 até 67 é executada a função main do programa. Na linha 44 é criado um vetor que armazenará as duas threads a serem usadas.

Nas linhas 49 a 57 é mostrado um cabeçalho com a identificação de quem fez o trabalho, no caso eu, Isadora e a Daiana.

Já nas linhas 59 e 60 é executada a função sem\_init, que inicializa os semáforos do programa, com valor inicial igual a 1. Sobre essa função, o valor do segundo argumento passado, no caso o valor 0, permite o compartilhamento entre as threads de um processo.

Em seguida, nas linhas 61 e 62 é executada a função pthread\_create, que cria as threads 1 e 2 do programa.

E nas linhas 63 e 64 são executadas a função pthread\_join, que têm a função de esperar a thread especificada na função terminar, no caso a thread da posição 0 e 1 do vetor. Se ela terminou, a função retorna imediatamente, e a thread especificada na função deve ser juntável.

Execução:

Para a execução do programa, foi criado um arquivo .sh, como sugerido no enunciado do trabalho. Então executando…

Finalização

Bom, depois de explicar o que foi implementado e testá-lo, agora eu vou falar um pouco sobre a divisão de como foi feito.Ao longo do trabalho, todos os processos realizados foram feitos tanto por mim, Isadora, quanto pela Daiana de forma igualitária, em que nós duas conversamos sobre a programação do código por completo.

Auto-avaliação

Fazendo uma auto-avaliação de como foi o grupo, sobre o que nós achamos do trabalho. De início tivemos um pouco de dificuldade quanto ao que se pedia, em como relacionar os conhecimentos explicados nas aulas para solucionar o problema de trabalho, mas que é uma fase normal do trabalho, por ter ocorrido processo semelhante no trabalho anterior. Outro ponto é que, na nossa percepção, esse foi um trabalho mais complexo, em relação ao primeiro, por envolver conceitos mais complexos na construção do programa, além de também exigir um bom conhecimento de estruturas de dados. Porém, pelo fato do trabalho ser em dupla, a troca de informações entre nós duas tornou a realização do projeto menos massivo de ser feito. E foi isso, espero que tenham entendido a explicação do nosso projeto, e muito obrigada por assistir esse vídeo.