Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente - 2019/2

Relatório de Entrega de Atividades

Aluno(s): Ana Luísa
Gabriel Sylar
Atividade: Laboratorio 6 - Deadlocks

1.1.1. A partir do extrato código abaixo, e de uma possível sequência de eventos, represente cada estado do sistema após a ocorrência de cada evento.

```
// Processo A
requisita(K)
requisita(L)
libera(L)
libera(K)
```

```
// Processo B
requisita(L)
requisita(M)
libera(M)
libera(L)
```

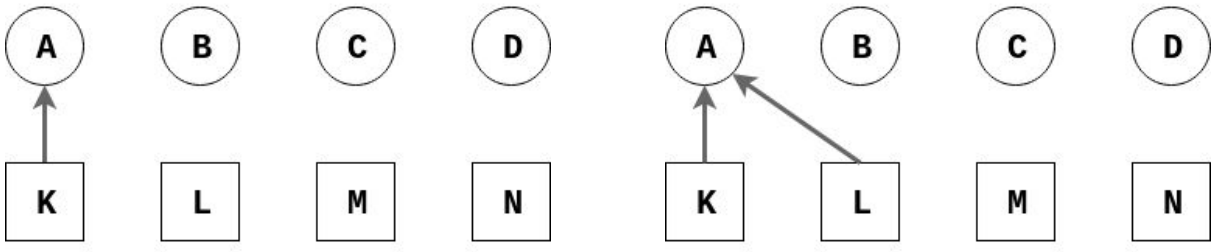
```
// Processo C
requisita(L)
requisita(K)
requisita(M)
libera(M)
libera(K)
libera(L)
```

```
// Processo D
requisita(M)
requisita(N)
libera(N)
libera(M)
```

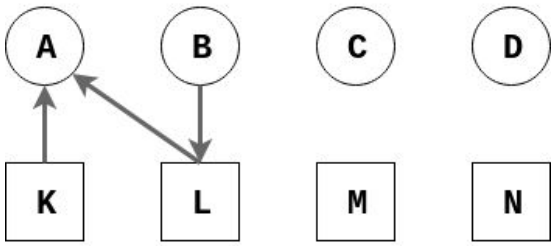
Sequência de eventos:

```
A requisita K
A requisita L
B requisita L
B requisita M
C requisita L
A libera L
A libera K
C requisita K
```

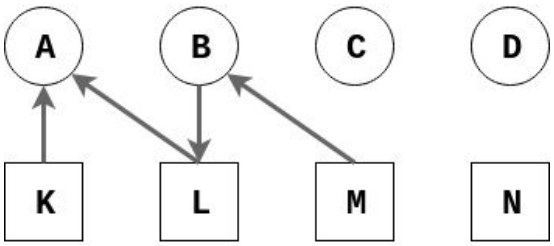
Resposta:



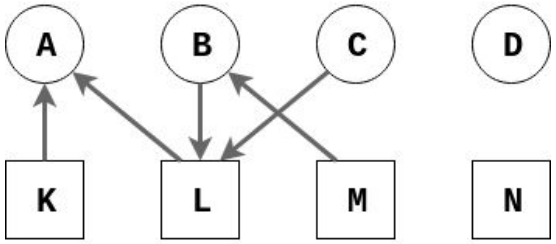
(1) `A requisita K` (2) `A requisita L`



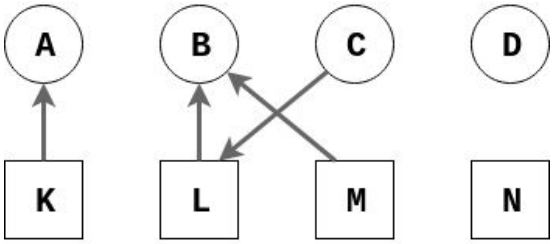
(3) `B requisita L`



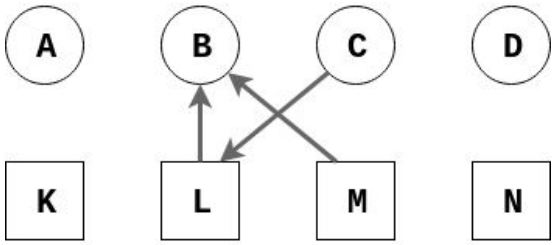
(4) `B requisita M`



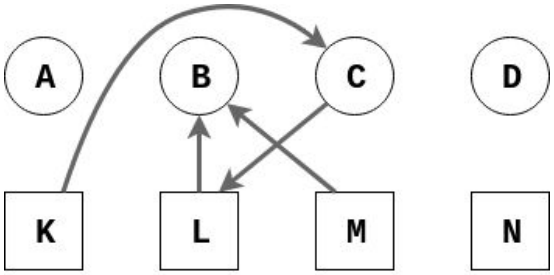
(5) `C requisita L`



(6) `A libera L`



(7) `A libera K`

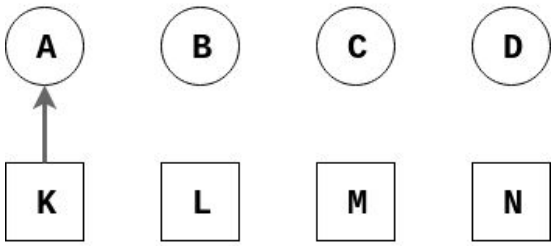


(8) `C requisita K`

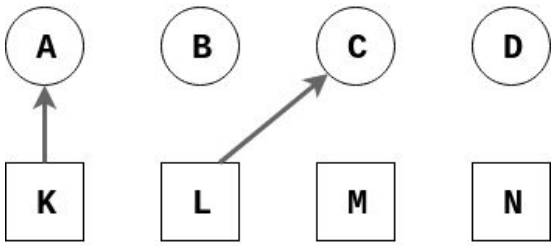
1.1.2. Utilizando o extrato de código anterior, identifique se é possível que ocorra algum deadlock. Se for possível, descreva a sequência de eventos para que este ocorra, bem como a sua representação gráfica.

```
A requisita K
C requisita L
A requisita L
C requisita K
```

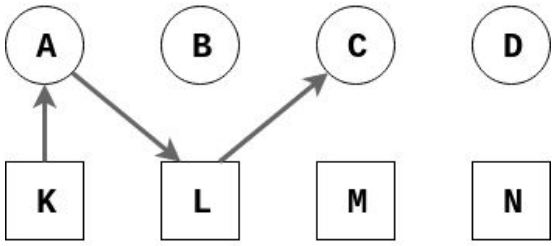
Resposta:



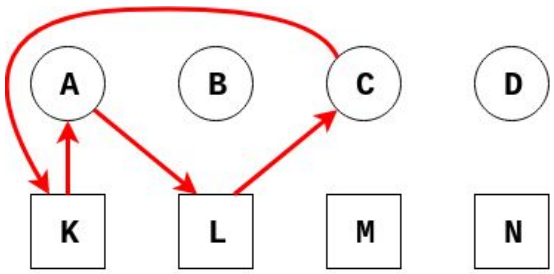
(1) `A requisita K`



(2) `C requisita L`



(3) `A requisita L`



(4) `C requisita K; DEADLOCK`

As quatro condições para deadlock foram satisfeitas, sendo elas Exclusão Mútua, Posse e Espera, Não-Preempção e Espera Circular.

2.1.1. Implementar uma Solução para o Jantar dos Filósofos

Sua solução deverá funcionar da seguinte maneira:

- Cada filósofo será representado por uma thread.
- Continuamente, cada filósofo deverá Pensar (dormir por um segundo), pegar garfo e soltar garfo.
- Um filósofo poderá encontrar-se em um de três estados: PENSANDO, COMENDO e COM FOME.
- Quando o filósofo pegar um garfo, indica-se que este está com fome.
- Ao pegar o garfo, este deverá verificar se os garfos da esquerda e da direita, um por vez, em qualquer ordem, estão disponíveis. Uma dica para essa verificação, é olhar o próprio estado dos filósofos próximos e se ambos estiverem pensando ou com fome, este é um indício que os garfos ainda estão disponíveis. Assim, o filósofo passa para o estado de comendo e dorme por dois segundos.
- Ao soltar o garfo, o filósofo passa para o estado de pensando, indicando que este liberou o(s) garfo(s).

Resposta:

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define N 5
#define ESQUERDO (id+4)%N   // pega garfo do lado esquerdo
#define DIREITO (id+1)%N    // pega garfo do lado direito

#define PENSANDO 0
#define COM_FOME 1
#define COMENDO 2
int estado[N];

pthread_mutex_t acessar_estado;
sem_t empanturrado[N];

void comer_ou_nao_comer_eis_a_questao (long int id) {
        if ((estado[id] == COM_FOME) && (estado[ESQUERDO] != COMENDO) &&
(estado[DIREITO] != COMENDO)) {
        printf("Filosofo %ld pegou os garfos %ld e %ld\n", id, ESQUERDO, DIREITO);
        estado[id] = COMENDO;
        printf("Filosofo %ld esta comendo\n\n", id);
        sleep(2);
        sem_post(&empanturrado[id]);          // se comeu solta os garfos
        }
}

void soltarGarfo (long int id) {
        pthread_mutex_lock(&acessar_estado);

        printf("Filosofo %ld soltou os garfos %ld e %ld\n", id, ESQUERDO, DIREITO);
        estado[id] = PENSANDO;
        printf("Filosofo %ld esta pensando\n\n", id);

        // se havia filosofos esperando com fome, entao libera garfos pra eles comerem
        comer_ou_nao_comer_eis_a_questao(ESQUERDO);
        comer_ou_nao_comer_eis_a_questao(DIREITO);

        pthread_mutex_unlock(&acessar_estado);
}

void pegarGarfo (long int id) {
        pthread_mutex_lock(&acessar_estado);
```

```
        estado[id] = COM_FOME;
        printf("Filosofo %ld esta com fome\n\n", id);
        comer_ou_nao_comer_eis_a_questao(id);

        pthread_mutex_unlock(&acessar_estado);
        sem_wait(&empanturrado[id]);          // se nao comeu, fica esperando
}

void* filosofo (void* n_filosofo) {
        long int id = (long int)n_filosofo;

        while (1) {
        sleep(1);
        pegarGarfo(id);
        sleep(1);
        soltarGarfo(id);
        }
}

int main () {
        pthread_t thread[N];

        pthread_mutex_init(&acessar_estado, NULL);
        for (long int i=0; i < N; i++)
        sem_init(&empanturrado[i], 0, 0);

        for (long int i=0; i < N; i++) {
        pthread_create(&thread[i], NULL, filosofo, (void*)i);
        printf("Filosofo %ld esta pensando\n\n", i);
        }
        for (long int i=0; i < N; i++)
        pthread_join(thread[i], NULL);

        pthread_mutex_destroy(&acessar_estado);
        for (long int i=0; i < N; i++)
        sem_destroy(&empanturrado[i]);

        return 0;
}
```

3.1.1. Localização Deadlocks. Dado o código abaixo:

```
#include <pthread.h>
int var = 0;
void* contador ( void* arg ) {
      var++;
}

int main ( void ) {
      pthread_t t[10];
      for (int i = 0; i < 10; i++) {
            pthread_create(&t[i], NULL, contador, NULL);
      }
      var++; // A main tambem ira incrementar
      for (int i = 0; i < 10; i++) {
            pthread_join(t[i], NULL);
      }

      return 0;
}
```

Utilize a ferramenta Valgrind para detectar condições de corrida dentro do código. Exiba o relatório apresentado pela ferramenta, bem como uma extração dos pontos problemáticos no código.

Resposta:

Utilizando o comando *info thread* da ferramenta Valgrind, obtivemos :

```
(gdb) info thread
  Id   Target Id         Frame
  1    Thread 0x7ffff7f9b700 (LWP 6149) "311" 0x00007ffff7905e51 in clone ()
   from /lib64/libc.so.6
* 2    Thread 0x7ffff7818700 (LWP 6150) "311" 0x00000000004005d1 in contador ()
  3    Thread 0x7ffff7017700 (LWP 6151) "311" 0x00007ffff7905e51 in clone ()
   from /lib64/libc.so.6
```

Utilizando o Helgrind, para detecção de condições de corrida, obtivemos o seguinte relatório:

```
aluno@MO2017:~/Área de trabalho> valgrind --tool=helgrind ./311
==6378== Helgrind, a thread error detector
==6378== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==6378== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==6378== Command: ./311
==6378==
```

```
==6378== ---Thread-Announcement----------------------------------------
==6378==
==6378== Thread #3 was created
==6378==    at 0x5146E4E: clone (in /lib64/libc-2.22.so)
==6378==    by 0x4E443AF: create_thread (in /lib64/libpthread-2.22.so)
==6378==    by 0x4E45ECA: pthread_create@@GLIBC_2.2.5 (in /lib64/libpthread-2.22.so)
==6378==    by 0x4C314E7: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x40061E: main (in /home/aluno/Área de trabalho/311)
==6378==
==6378== ---Thread-Announcement----------------------------------------
==6378==
==6378== Thread #2 was created
==6378==    at 0x5146E4E: clone (in /lib64/libc-2.22.so)
==6378==    by 0x4E443AF: create_thread (in /lib64/libpthread-2.22.so)
==6378==    by 0x4E45ECA: pthread_create@@GLIBC_2.2.5 (in /lib64/libpthread-2.22.so)
==6378==    by 0x4C314E7: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x40061E: main (in /home/aluno/Área de trabalho/311)
==6378==
==6378== ----------------------------------------------------------------
==6378==
==6378== Possible data race during read of size 4 at 0x601044 by thread #3
==6378== Locks held: none
==6378==    at 0x4005D5: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
==6378==
==6378== This conflicts with a previous write of size 4 by thread #2
==6378== Locks held: none
==6378==    at 0x4005DE: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
==6378==  Address 0x601044 is 0 bytes inside data symbol "var"
==6378==
==6378== ----------------------------------------------------------------
==6378==
==6378== Possible data race during write of size 4 at 0x601044 by thread #3
==6378== Locks held: none
==6378==    at 0x4005DE: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
==6378==
==6378== This conflicts with a previous write of size 4 by thread #2
==6378== Locks held: none
==6378==    at 0x4005DE: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
```

```
==6378==  Address 0x601044 is 0 bytes inside data symbol "var"
==6378==
==6378== ---Thread-Announcement----------------------------------------
==6378==
==6378== Thread #1 is the program's root thread
==6378==
==6378== ---Thread-Announcement----------------------------------------
==6378==
==6378== Thread #11 was created
==6378==    at 0x5146E4E: clone (in /lib64/libc-2.22.so)
==6378==    by 0x4E443AF: create_thread (in /lib64/libpthread-2.22.so)
==6378==    by 0x4E45ECA: pthread_create@@GLIBC_2.2.5 (in /lib64/libpthread-2.22.so)
==6378==    by 0x4C314E7: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x40061E: main (in /home/aluno/Área de trabalho/311)
==6378==
==6378== ----------------------------------------------------------------
==6378==
==6378== Possible data race during read of size 4 at 0x601044 by thread #1
==6378== Locks held: none
==6378==    at 0x400629: main (in /home/aluno/Área de trabalho/311)
==6378==
==6378== This conflicts with a previous write of size 4 by thread #11
==6378== Locks held: none
==6378==    at 0x4005DE: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
==6378==  Address 0x601044 is 0 bytes inside data symbol "var"
==6378==
==6378== ----------------------------------------------------------------
==6378==
==6378== Possible data race during write of size 4 at 0x601044 by thread #1
==6378== Locks held: none
==6378==    at 0x400632: main (in /home/aluno/Área de trabalho/311)
==6378==
==6378== This conflicts with a previous write of size 4 by thread #11
==6378== Locks held: none
==6378==    at 0x4005DE: contador (in /home/aluno/Área de trabalho/311)
==6378==    by 0x4C316E6: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-amd64-linux.so)
==6378==    by 0x4E45723: start_thread (in /lib64/libpthread-2.22.so)
==6378==  Address 0x601044 is 0 bytes inside data symbol "var"
==6378==
==6378==
==6378== For counts of detected and suppressed errors, rerun with: -v
==6378== Use --history-level=approx or =none to gain increased speed, at
==6378== the cost of reduced accuracy of conflicting-access information
==6378== ERROR SUMMARY: 20 errors from 4 contexts (suppressed: 0 from 0)
```

O ponto problemático é que todas as threads (incluindo a main) estão acessando a variável global var ao mesmo tempo - condição de corrida.

Com utilização de um lock é possível proteger a região crítica (acesso à variável compartilhada), corrigindo a condição de corrida, como demonstrado abaixo:

```
#include <pthread.h>

int var = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* contador ( void* arg ) {
    pthread_mutex_lock(&lock);
    var++;
    pthread_mutex_unlock(&lock);
}

int main ( void ) {
    pthread_t t[10];
    int i;
    for (i = 0; i < 10; i++) {
        pthread_create(&t[i], NULL, contador, NULL);
    }
    pthread_mutex_lock(&lock);
    var++;
    pthread_mutex_unlock(&lock);
    for (i = 0; i < 10; i++) {
        pthread_join(t[i], NULL);
    }
    return 0;
}
```

Após o uso do lock, fizemos uso do Helgrind novamente e nenhum erro foi apontado no relatório:

```
aluno@MO2017:~/Área de trabalho> valgrind --tool=helgrind ./311
==6536== Helgrind, a thread error detector
==6536== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==6536== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==6536== Command: ./311
==6536==
==6536==
==6536== For counts of detected and suppressed errors, rerun with: -v
==6536== Use --history-level=approx or =none to gain increased speed, at
==6536== the cost of reduced accuracy of conflicting-access information
==6536== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 30 from 6)
```