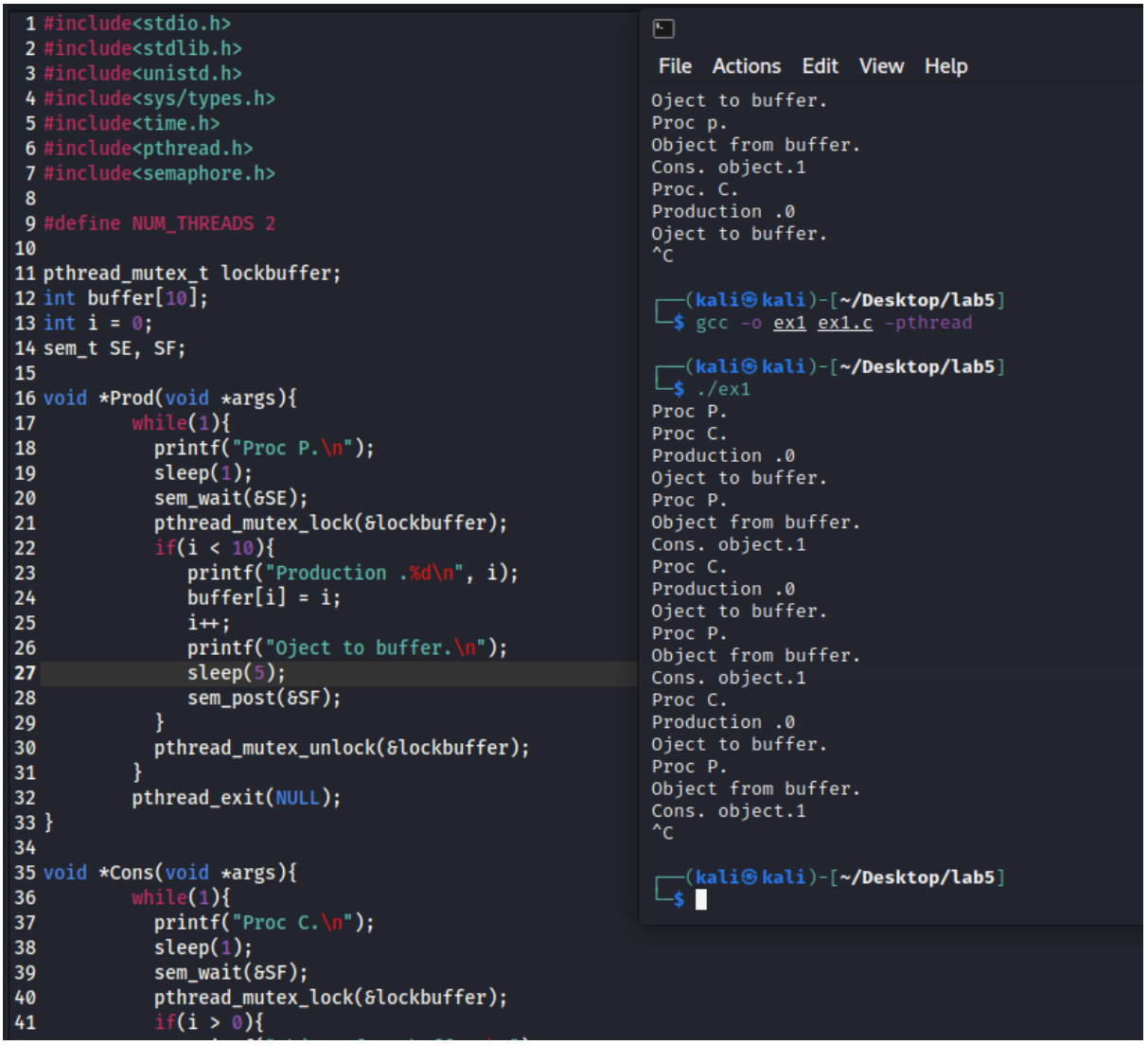
Concurso 5 - Threads/Semaphores

Leonardo Moreira 71512

Ex 1.



```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<time.h>
#include<pthread.h>
#include<semaphore.h>

#define NUM_THREADS 2

pthread_mutex_t lockbuffer;
int buffer[10];
int i = 0;
sem_t SE, SF;

void *Prod(void *args){
        while(1){
          printf("Proc P.\n");
          sleep(1);
          sem_wait(&SE);
          pthread_mutex_lock(&lockbuffer);
          if(i < 10){
             printf("Production .%d\n", i);
             buffer[i] = i;
             i++;
             printf("Oject to buffer.\n");
             sleep(5);
             sem_post(&SF);
          }
          pthread_mutex_unlock(&lockbuffer);
        }
        pthread_exit(NULL);
}

void *Cons(void *args){
        while(1){
          printf("Proc C.\n");
          sleep(1);
          sem_wait(&SF);
          pthread_mutex_lock(&lockbuffer);
          if(i > 0){
```

```
File  Actions  Edit  View  Help
Oject to buffer.
Proc p.
Object from buffer.
Cons. object.1
Proc. C.
Production .0
Oject to buffer.
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ gcc -o ex1 ex1.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ ./ex1
Proc P.
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$
```

```c
            pthread_mutex_lock(&lockbuffer);
            if(i > 0){
               printf("Object from buffer.\n");
               printf("Cons. object.%d\n", i);
               buffer[i-1] = i;
               i--;
               sleep(10);
               sem_post(&SE);
            }
            pthread_mutex_unlock(&lockbuffer);
        }
        pthread_exit(NULL);
}

int main(int args, char *argv[]){
        pthread_t threads[NUM_THREADS];
        srand(time(NULL));
        pthread_mutex_init(&lockbuffer, NULL);
        sem_init(&SE, 0, 10);
        sem_init(&SF, 0, 0);
        int t;
        for(t = 0; t < NUM_THREADS; t++){
            if(t % 2 == 0){
                if(pthread_create(&threads[t], NULL, &Prod, NULL) != 0){
                        perror("ERROR");
                }
                else if(pthread_create(&threads[t], NULL, &Cons, NULL) != 0){
                        perror("ERROR");
                }
            }
        }
        for(t = 0; t < NUM_THREADS; t++){
             if(pthread_join(threads[t], NULL) != 0){
                perror("ERROR JOIN");
             }
        }
        sem_destroy(&SF);
        sem_destroy(&SE);
        pthread_mutex_destroy(&lockbuffer);
        return 0;
}
```

```
┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ gcc -o ex1 ex1.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ ./ex1
Proc P.
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
Proc C.
Production .0
Oject to buffer.
Proc P.
Object from buffer.
Cons. object.1
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$
```

Ex 2.

1)

O problema do jantar dos filósofos baseia-se em vários conceitos.
Podemos começar por perceber que temos numa mesa redonda vários filósofos sentados uns ao lado dos outros, imaginando que cada um tem um prato à sua frente com comida e, tanto à sua direita como à esquerda, têm um garfo apenas.

Depois podemos pensar nisto como estados, ou seja, ou eles estão a comer, ou eles estão a pensar, não podem fazer as duas coisas ao mesmo tempo, portanto, temos 2 estados.

Por exemplo, se o filósofo verde quiser comer, o azul e o vermelho não podem, visto que é sempre preciso 2 garfos para comer (e têm que ser garfos adjacentes ao filósofo).
Atenção: para comer não podem pegar em 2 garfos ao mesmo tempo, têm que pegar num e só depois em outro, ou seja, imaginando que o verde está a comer (com os 2 garfos obrigatoriamente) tanto o vermelho como o azul, não podem, pois só têm 1 garfo disponível cada um. Só o poderão fazer assim que o verde acabar de comer e pousar os 2 garfos na mesa.

Então, podemos concluir, que o problema aqui é que, sendo que os garfos são limitados, 2 filósofos adjacentes não podem comer ao mesmo tempo, isto é, não podem aceder ao mesmo garfo.

Poderemos pensar neste problema em termos computacionais como, os filósofos sendo processos e os garfos poderão ser tomados como recursos ou operações etc…
Isto é, temos que perceber que existe um problema quando 2 processos tentam aceder ao mesmo recurso (garfos) ao mesmo tempo, havendo assim, um problema de resource allocation.
Por isso, o objetivo passa por conseguir que os recursos (limitados) sejam partilhados entre processos de uma maneira sincronizada sem quebrar nenhuma regra, por exemplo, 2 processos estarem a utilizar o mesmo recurso ao mesmo tempo.

2)

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


#define numPhils 5
#define thinking 2
#define hunger 1
#define eat 0
#define Left (i + 4) % numPhils
#define right (i + 1) % numPhils

int PhilStates[numPhils];
int phil[numPhils] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t semp[numPhils];

void maint(int i)
{
    if (PhilStates[i] == hunger && PhilStates[Left] != eat && PhilStates[right] != eat) {

        PhilStates[i] = eat;
        sleep(2);

        printf("Philosopher %d picks up fork %d and %d\n", i + 1, Left + 1, i + 1);
        printf("Philosopher %d is eating\n", i + 1);

        sem_post(&semp[i]);
    }
}

```

```
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ gcc -o ex2b ex2b.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ ./ex2b
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 picks up fork 2 and 3
Philosopher 3 is eating
Philosopher 1 picks up fork 5 and 1
Philosopher 1 is eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 picks up fork 3 and 4
Philosopher 4 is eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 picks up fork 1 and 2
Philosopher 2 is eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 picks up fork 4 and 5
Philosopher 5 is eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$
```

```c
}

// take up chopsticks
void Pick_Fork(int i)
{

    sem_wait(&mutex);

    // PhilStates that hunger
    PhilStates[i] = hunger;

    //printf("Philosopher %d is hunger\n", i + 1);

    // eat if neighbours are not eat
    maint(i);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&semp[i]);

    sleep(10);
}

// put down chopsticks
void Drop_Fork(int i)
{

    sem_wait(&mutex);

    // PhilStates that thinking
    PhilStates[i] = thinking;

    printf("Philosopher %d putting fork %d and %d down\n", i + 1, Left + 1, i + 1);
    printf("Philosopher %d is thinking\n", i + 1);

    maint(Left);
    maint(right);

    sem_post(&mutex);
}

```

```
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ gcc -o ex2b ex2b.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ ./ex2b
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 picks up fork 2 and 3
Philosopher 3 is eating
Philosopher 1 picks up fork 5 and 1
Philosopher 1 is eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 picks up fork 3 and 4
Philosopher 4 is eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 picks up fork 1 and 2
Philosopher 2 is eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 picks up fork 4 and 5
Philosopher 5 is eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$
```

```c

void* SELF(void* num)
{

    while (1) {
        int* m = num;
        sleep(1);
        Pick_Fork(*m);
        sleep(0);
        Drop_Fork(*m);
    }
}

int main()
{

    int k;
    pthread_t thread_id[numPhils];

    sem_init(&mutex, 0, 1); // initialize the semaphores

    for (k = 0; k < numPhils; k++)

        sem_init(&semp[k], 0, 0);

    for (k = 0; k < numPhils; k++) {

        // create philosopher processes
        pthread_create(&thread_id[k], NULL, SELF, &phil[k]);

        printf("Philosopher %d is thinking\n", k + 1);
    }

    for (k = 0; k < numPhils; k++)

        pthread_join(thread_id[k], NULL);
}
```

```
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ gcc -o ex2b ex2b.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$ ./ex2b
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 picks up fork 2 and 3
Philosopher 3 is eating
Philosopher 1 picks up fork 5 and 1
Philosopher 1 is eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 picks up fork 3 and 4
Philosopher 4 is eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 picks up fork 1 and 2
Philosopher 2 is eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 picks up fork 4 and 5
Philosopher 5 is eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
^C

┌──(kali㉿kali)-[~/Desktop/lab5]
└─$
```