POLITÉCNICO DE PORTALEGRE

Escola Superior de Tecnologia, Gestão e Design

OPERATING SYSTEMS

Exercise 2 - Tasks and Synchronization Mechanisms

Delivery Date: 24-Nov-2024

Objective

- Describe what tasks are and how they are created in programs.
- Identify mutual exclusion and synchronization issues.
- Apply synchronization mechanisms (mutexs and semaphores) to solve problems.

1. Task creation

a) Copy the ficha2-eg1.tgz file to your desktop and unzip its contents:
`tar -xvf ficha2-eg1.tgz`

b) Study, compile and run the application. Understand the result.

Tarefas

Criação de Tarefas

Código Fonte

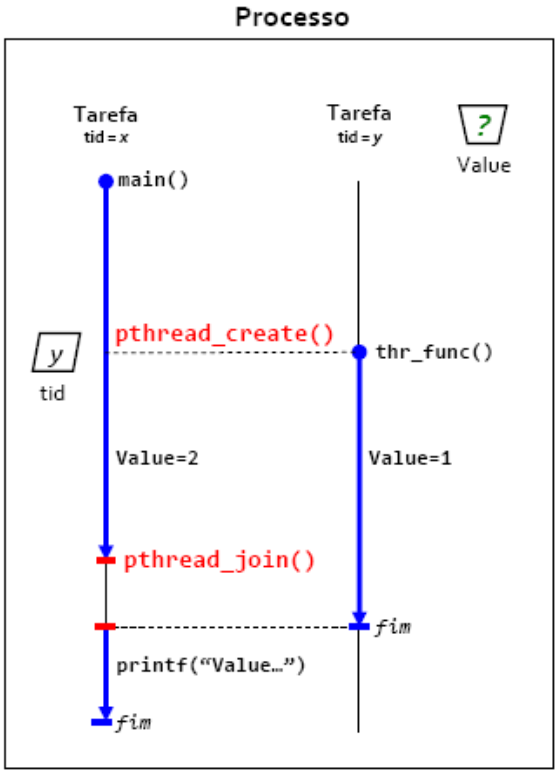
```
int Value = 0;

void* thr_func(void* ptr)
{
   Value = 1;
   return NULL;
}

void main()
{
   int tid;
   pthread_create(&tid, NULL,thr_func,NULL);
   Value = 2;
   pthread_join(tid, NULL);
   printf("Value=%d\n",Value);
}
```

Output da Execução

```
Value=?
```



c) Why is the content of the global variable undetermined? `Value`?

d) Set up an experiment that proves the statement in c). (*suggestion*: use the system call `sleep`).

2. Identifying and resolving mutual exclusion problems

a) Copy the file ficha2-eg2.tgz to your desktop and unzip its contents:
`tar -zxvf ficha2-eg2.tgz`

b) Study, compile and run the application. Understand the result.

Tarefas

Partilha de Dados

Código Fonte

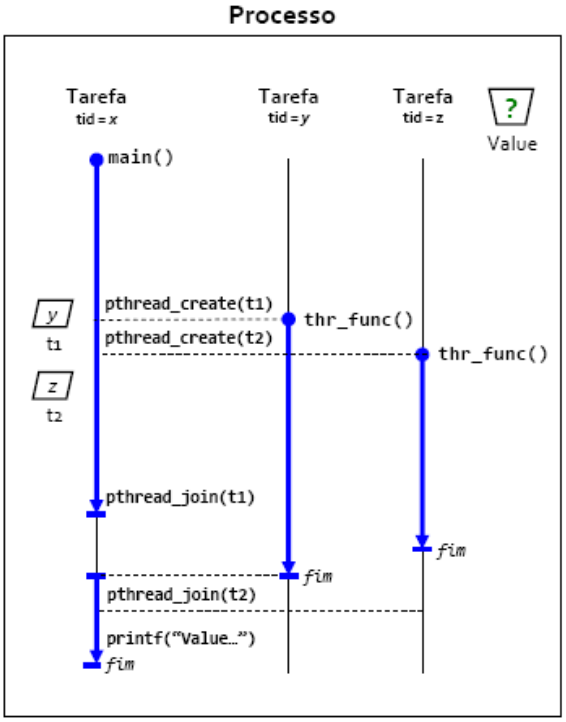
```
int Value = 10;

void* thr_func(void* ptr)
{
    if (Value >= 10) {
       Value -= 10;
    }
   return NULL;
}

void main()
{
   int t1,t2;
   pthread_create(&t1, NULL,thr_func,NULL);
   pthread_create(&t2, NULL,thr_func,NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   printf("Value=%d\n",Value);
}
```

Output da Execução

```
Value=?
```



c) What values are possible for the variable `Value` at the end of execution? What's the problem? Set up an experiment to prove this statement.

d) Fix the problem in the previous example using a mutex. Check that, with this change, the experience set up in the previous paragraph produces a correct result.

Tarefas

Trincos

Código Fonte

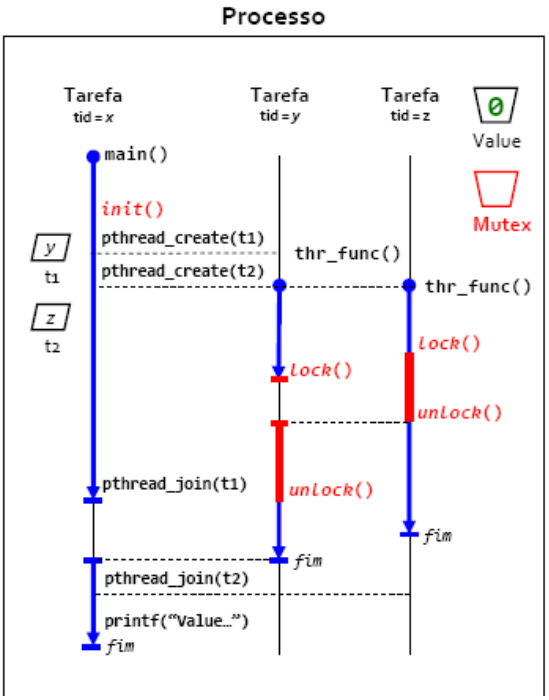
```
int Value = 10;
pthread_mutex_t Mutex;

void* thr_func(void* ptr)
{
   pthread_mutex_lock(&Mutex);
   if (Value >= 10) {
      Value -= 10;
   }
   pthread_mutex_unlock(&Mutex);
   return NULL;
}

void main()
{
   int t1,t2;
   pthread_mutex_init(&Mutex,NULL);
   pthread_create(&t1, NULL,thr_func,NULL);
   ...
}
```

Output da Execução

```
Value=0
```



e) The solution would work if the variable `mutex` was local to the function `thr_func` ?

Exercises

Counting Tasks

Implement an application that concurrently increments a variable `Counter` from 0 to 20 using tasks. You can use the structure as a base skeleton ficha2-eg3.tgz. Suggestion: create individual versions of the solution for each item.

1. Competing tasks

a) Use two tasks, implemented by two functions - `thr_inc_even` and `thr_inc_odd` . The first increases the value if `Counter` is even, the other if `Counter` is odd. Properly synchronize shared variable access with mutexs. Tasks must print a message each time they increment the shared variable. For example, the result should be the following:

```
...
[E] Counter=11
[O] Counter=12
[E] Counter=13
[O] Counter=14
...
```

b) Generalize the previous exercise so that the variable is increased by N tasks (N is defined by a constant). Each task receives an identifier id (between 0 and N-1). Each task only increments Counter when id=Counter%N. Instead of E or O, the identification of the task that increments the variable must be printed. Tip: Only use a thr_incrementer function that takes the function identifier as an argument.

Example of passing arguments to threads:

```
/*
* argthread.c - example passing example into threads
*/

#include <stdio.h>
#include <pthread.h>


void* thr_arg_pass_example(void* ptr)
{
int* arg = (int*)ptr;
printf("Argument: %d\n", *arg);
   return NULL;
}

int main()
{
   pthread_t tid;

   int arg = 1234;

   if (pthread_create(&tid, NULL, thr_arg_pass_example, (void*)&arg) != 0) {
      printf("Error creating thread.\n");
      return -1;
   }

   if(pthread_join(tid, NULL) != 0) {
      printf("Error joining thred.\n");
      return -1;
   }

   printf("Finished.\n");
   return 0;
}
```

i) Faça testes com diferentes valores de N.

ii) Experiment by inserting instructions `sleep` in the critical region to confirm that the synchronization works properly.

iii) Why was the section defined this way? Why not smaller or larger?

c) Modify the previous solution using a semaphore to replace the mutex.

Example of using semaphores:

```
/*
* semaphore.c - example semaphore declaration and usage
*/

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>


int main()
{
sem_t semaphore;

printf("Initialization with 1 unit. \n");
    sem_init(&semaforo, 0, 1);

printf("Wait.\n");
sem_wait(&semaforo);

printf("Tick. \n");
sem_post(&semaforo);

printf("Destroy the semaphore. \n");
sem_destroy(&semaforo);

return 0;
}
```

i) What is the meaning of the value 1 at the traffic light initialization?

ii) If the semaphore initialization value were different from 1, would it have similar behavior to the mutex approach?

2. Cooperative Tasks

Now use two cascading tasks - `thr_inc_low` and `thr_inc_high` , respectively. The first increments `Counter` between 0 and 10, the second increments `Counter` between 11 and 20. Both are launched simultaneously, but the task `thr_inc_high` must be blocked until the task `thr_inc_low` finish. Use a semaphore to synchronize tasks.

a) What value should the semaphore be initialized with?

b) What function does the traffic light perform in terms of synchronization?

c) In this case, do you have any mutual exclusion problems that you need to resolve? Why?