LABORATORIO 10:
SEMAFOROS: PROBLEMA DE LA COMIDA DE FILOSOFOS

Propuesto por Dijkstra en 1965, muestra un problema básico de interbloqueo y coordinación de recursos no compartibles de un sistema. El problema se puede enunciar de la siguiente manera:

Cinco filósofos dedican sus vidas a pensar y comer (estas dos acciones son finitas en el tiempo). Los filosos comparten una mesa rodeada de cinco sillas, cada una identificada con el nombre de un filósofo. En el centro de la mesa hay espagueti que es constantemente rellenado, y en la mesa cinco palillos y cinco platos a la izquierda de cada filosofo (figura 1).

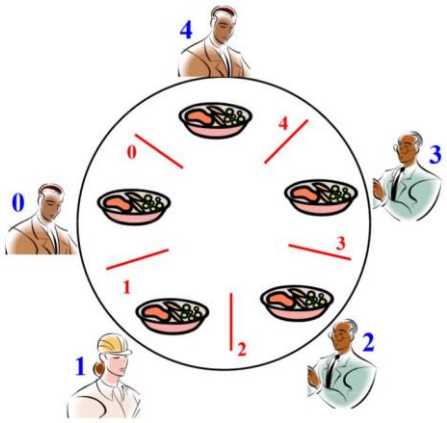


Figura 1. Problema de la comida de filósofos

Los filosos se sientan en sentido anti horario alrededor de la mesa, y cuando un filósofo siente hambre, se dirige a una silla y trata de coger los dos palillos que están más cerca de él. Cuando un filoso tiene sus dos palillos simultáneamente, come sin dejar sus palillos. Cuando acaba de comer, vuelve dejar los dos palillos y empieza a pensar de nuevo.

Cada par de palillos estará disponible por filósofo, siendo usada una sola vez, si otro filosofo lo desea, deberá de esperar hasta que un par de palillos estén disponibles.

La solución al problema, por lo tanto, consiste en inventar un algoritmo cuando un filósofo se dirigiera a el comedor y tome un palillo que estaba a su izquierda y lo sumieran en el espagueti. Pero tal era naturaleza enredada del espagueti que un segundo tenedor era requerido para llevarlo a la boca. El filósofo por ende tenia que también tomar los palillos a su derecha. Cuando terminaban debían bajar ambos palillos, levantarse de la silla y continuar pensando.

En resumen, se presenta el siguiente algoritmo a resolver:

- Un filósofo toma palillos a su izquierda.
- Después toma palillos en a su derecha.
- Come.
- Baja los pares de palillos.

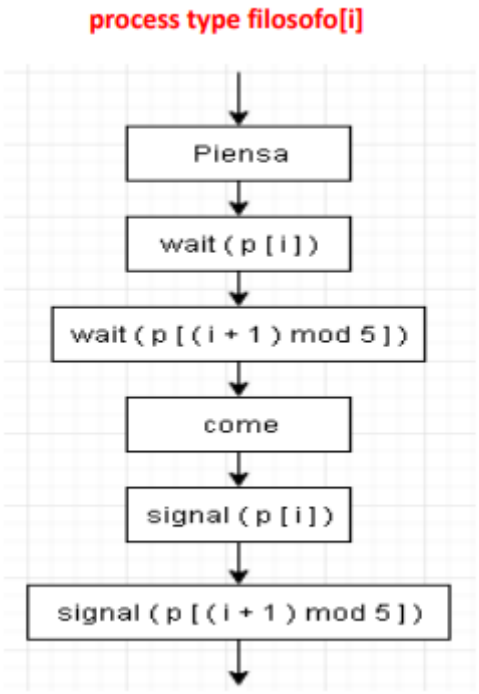
Pero se tiene la siguiente secuencia de eventos:

1) Filósofo 1 comienza el algoritmo, tomando palillos a su izquierda.
2) Filósofo 2 comienza el algoritmo, tomando palillos a su izquierda.
3) Filósofo 3 comienza el algoritmo, tomando palillos a su izquierda.
4) Filósofo 4 comienza el algoritmo, tomando palillos a su izquierda.
5) Filósofo 5 comienza el algoritmo, tomando palillos a su izquierda.
6) ... ? Todos los palillos han sido tomados, pero nadie puede comer!

Una solución sencilla consiste en representar cada palillo por un semáforo. Un filósofo trata de coger el palillo ejecutando una operación wait sobre el semáforo:

```
var palillo: array[0..4] of semaphore;
```

Donde los cinco semáforos están inicializados a 1. La estructura de un proceso filósofo será:

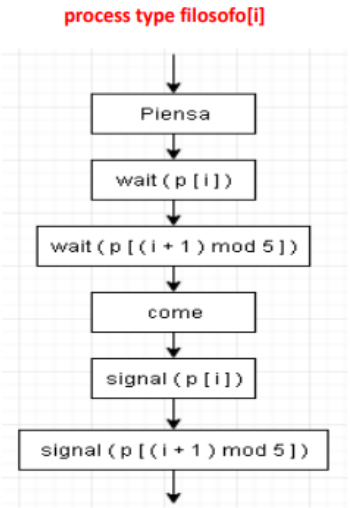


Cada filósofo toma primero el palillo de su izquierda, y después, el de su derecha. Cuando un filósofo termina de comer, devuelve los dos palillos a la mesa. Esta solución garantiza que no hay dos vecinos comiendo simultáneamente; sin embargo, es rechazada porque hay peligro de interbloqueo. Supongamos que los cinco filósofos se sienten hambrientos a la vez, y que cada uno coge el palillo de su izquierda y todos intentan coger el otro palillo, que no estará disponible. Para solucionar este problema de interbloqueo veremos las siguientes propuestas:

SOLUCION 1:

Esta primera solución consiste en representar cada palillo como un semáforo, cuando un filósofo trata de coger un palillo ejecuta la operación WAIT sobre el semáforo.

- Estructuras:
  P: array[0..4] of semaphore
- Inicialización
  For k:=0 to 4 do
  Initial (p[k],1);



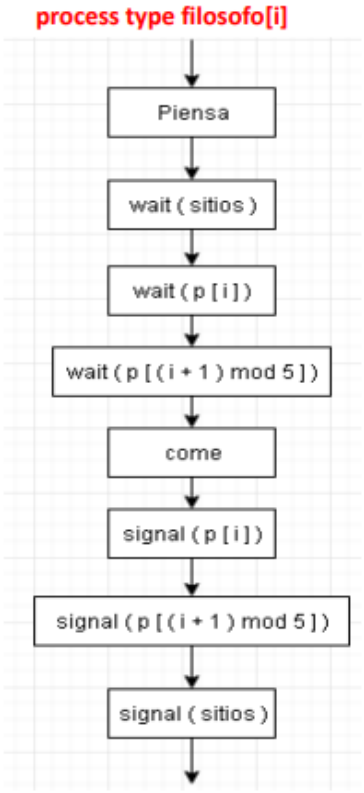

```
program filosofos;
var palillo: array[0..4] of semaphore;
process type filosofo(i: integer);
begin
      repeat
            writeln(i,' : Piensa');
            wait(palillo[i]);
            wait(palillo[(i+1)mod 5]);
            writeln(i,' : Come');
            signal(palillo[i]);
            signal(palillo[(i+1)mod 5]);
      forever
end;
var k,i : integer;
      F: array[0..4] of filosofo;
begin
            for k:=0 to 4 do
            initial(palillo[k],1);
      cobegin
            for k:=0 to 4 do
                  F [k](k);
      coend
end.
```

Sin embargo; esta solución no es correcta, ya que, si todos los procesos dejan de pensar al mismo tiempo, y cogen los palillos de su izquierda se produce un interbloqueo de todos los filósofos.

SOLUCION 2:

Consiste en inicializar la cantidad de filósofos que pueden comer simultáneamente a 4. Debido a que no se podrán ocupar los 5 palillos, no será posible un interbloqueo total.

- Estructuras:
  P: array[0..4] of semaphore
  sitios: semaphore
- Inicialización
  Initial (sitios,4) {semáforo del recurso compartido}
  For k:=0 to 4 do
  Initial (p[k],1);



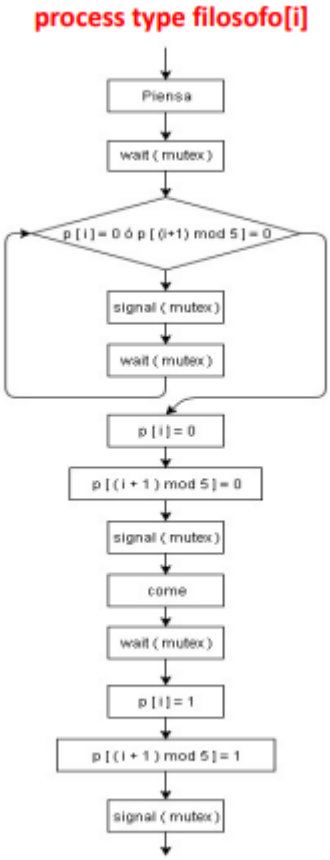

```
program filosofo2;
var palillo: array[0..4] of semaphore;
var sitio : semaphore;

process type filosofo(i: integer);
begin
   repeat
         writeln(i,' : Piensa');
         wait(sitio);
         wait(palillo[i]);
         wait(palillo[(i+1)mod 5]);
         writeln(i,' : Come');
         signal(palillo[i]);
         signal(palillo[(i+1)mod 5]);
         signal(sitio);
   forever
end;
var k,i : integer;
   F: array[0..4] of filosofo;
begin
         initial(sitio,4);
         for k:=0 to 4 do
         initial(palillo[k],1);
   cobegin
         for k:=0 to 4 do
               F[k](k);
   coend
end.
```

Este algoritmo hace que sea vuelva más lento el proceso, pero garantiza que al menos un filósofo puede comer.

SOLUCION 3:

Esta solución consiste en que cada filósofo tome sus palillos solamente si ambos están libres.

- Estructuras:
  P: array[0..4] of integer {puede tomar dos valores los palillos; 1=libre y 0=ocupado}
  mutex: semaphore
- Inicialización
  Initial (mutex,1)
  for k:=0 to 4 do
  p[k]=1;




```
program filosofo3;
var palillo: array[0..4] of integer;
var mutex : semaphore;
process type filosofo(i: integer);
begin
      repeat
            writeln(i,' : Piensa');
            wait(mutex);
            while( (palillo[i]=0) or (palillo[(i+1)mod
5]=0) ) do
            begin
                  signal(mutex);
                  wait(mutex);
            end;
            palillo[i]:= 0;
            palillo[(i+1)mod 5]:=0;
            signal(mutex);
            writeln(i,' : Come');
            wait(mutex);
            palillo[i]:=1;
            palillo[(i+1)mod 5]:=1;
            signal(mutex);
      forever
end;
var k,i : integer;
      F: array[0..4] of filosofo;
begin
            initial(mutex,1);
            for k:=0 to 4 do
```

```
        palillo[k]:=1;
cobegin
        for k:=0 to 4 do
                F[k](k);
coend
        end.
```

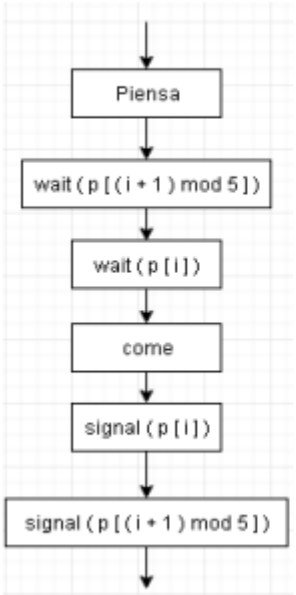
Este algoritmo se basa en espera ocupada, no es eficiente.

SOLUCION 4:

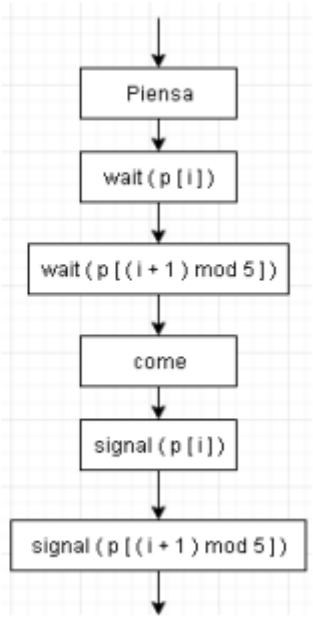
Solución asimétrica, es decir un filósofo impar toma primero su palillo de la izquierda y luego el palillo de la derecha, mientras que un filósofo par elige su palillo de la derecha y luego el de la izquierda.

- Estructuras:
  P: array[0..4] of semaphore
- Inicialización
  for k:=0 to 4 do
  Initial(p[k],1);

process type filosofopar[i]



process type filosofoimpar[i]




```
program filosofo4;
var palillo: array[0..4] of semaphore;
process type filosofopar(i: integer);
begin
   repeat
         writeln(i,' : Piensa');
         wait(palillo[(i+1)mod 5]);
         wait(palillo[i]);
         writeln(i,' : Come');
         signal(palillo[i]);
         signal(palillo[(i+1)mod 5]);
   forever
end;
process type filosofoimpar(i: integer);
```

```
begin
   repeat
         writeln(i,' : Piensa');
         wait(palillo[i]);
         wait(palillo[(i+1)mod 5]);
         writeln(i,' : Come');
         signal(palillo[i]);
         signal(palillo[(i+1)mod 5]);
   forever
end;
var k,i : integer;
   FP: array[0..4] of filosofopar;
   FI: array[0..4] of filosofoimpar;
begin
   for k:=0 to 4 do
         initial(palillo[k],1);
   cobegin
         for k:=0 to 4 do
         begin
               if ((k = 0) or (k = 1))then
                     FP [k](k)
               else
                     FI [k](k)
         end
   coend
end.
```

Actividad propuesta

Realizar los casos del problema Filósofos de Pascal FC a C++ usando como referencia el siguiente codigo:

```
#include <iostream>
#include <mutex>
#include <thread>

using namespace std;

int main()
{
   const int no_of_philosophers = 5;
   // estructura de palillos
   struct Chopstics
   {
   public:
      Chopstics(){;}
      std::mutex mu;
   };

   auto eat = [](Chopstics &left_chopstics, Chopstics& right_chopstics, int philosopher_number) {
```

```
        std::unique_lock<std::mutex> llock(left_chopstics.mu);
        std::unique_lock<std::mutex> rlock(right_chopstics.mu);

        cout << "Filosofo " << philosopher_number << " esta comiendo" << endl;

        std::chrono::milliseconds timeout(1500);
        std::this_thread::sleep_for(timeout);

        cout << "Filosofo " << philosopher_number << " ha terminado de comer" << endl;
    };

    //crear palillos
    Chopstics chp[no_of_philosophers];

    //create Filosofos
    std::thread philosopher[no_of_philosophers];

    //Filosofo empieza a leer
    cout << "Filosofo " << (0+1) << " esta leyendo.." << endl;
    philosopher[0] = std::thread(eat, std::ref(chp[0]), std::ref(chp[no_of_philosophers-
1]), (0+1));

    for(int i = 1; i < no_of_philosophers; ++i) {
        cout << "Filosofo " << (i+1) << " esta leyendo.." << endl;
        philosopher[i] = std::thread(eat, std::ref(chp[i]), std::ref(chp[i-1]), (i+1));
    }

    for(auto &ph: philosopher) {
        ph.join();
    }

    return 0;
}
```