



LaSalle
Universidad

INGENIERÍA DE SOFTWARE
SISTEMAS OPERATIVOS

Filósofos Comensales
Elvis David Minaya Mamani
10/05/2021

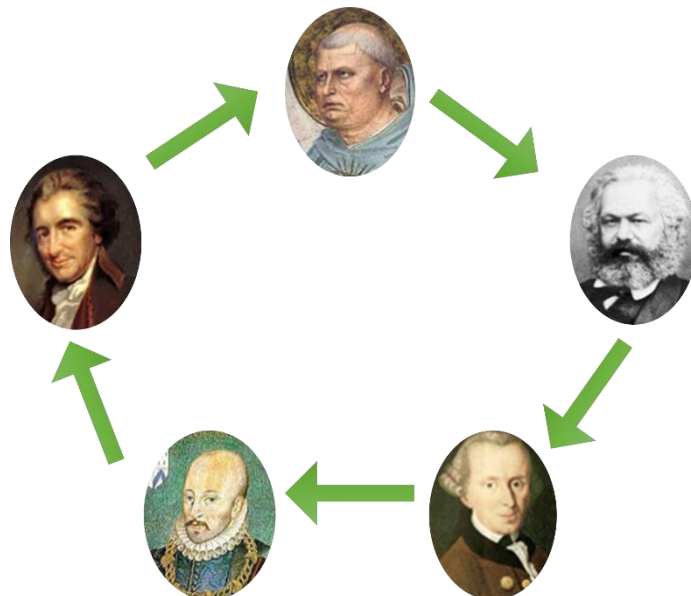
FILÓSOFOS COMENSALES

1. Planteamiento del problema.

- Introducción: Alrededor de una mesa se sientan 5 filósofos, los cuales pasan su vida pensando y cenando. Para cada filósofo hay un plato de fideos, y un tenedor a su izquierda, para como los fideos es necesario ocupar dos tenedores y cada filósofo solo puede tomar el que está a su izquierda y derecha. Si cualquier filósofo toma un tenedor y el otro está ocupado este se quedara esperando con el tenedor en la mano, hasta que pueda coger el otro tenedor para poder empezar a comer.
- **Restricciones:**
 - a) Si dos filósofos juntos toman el mismo tenedor se produce una condición de carrera ya que ambos competirán por el mismo tenedor y uno de ellos tendrá que quedarse sin comer.
 - b) Si todos los filósofos toman el mismo tenedor al mismo tiempo, entonces todos se quedaran esperando eternamente, ya que nadie liberara el tenedor que les falta, se interpreta de tal forma que todos los filósofos, morirán de hambre al mismo tiempo se produce un interbloqueo.

2. Solución por turno cíclico.

- **Cíclico:** Proceso repetitivo, presentamos a los 5 filósofos y le asignamos un tenedor a cada uno, también cada uno tendrá un tiempo determinado para poder comer, cada vez que uno termine el siguiente continuara y así sucesivamente.



3. SOLUCION POR COLAS.

- **Colas:** Como podemos observar tenemos a los 5 filósofos en la mesa, el filósofo número 1 empieza a comer ya que tiene dos tenedores al igual que el filósofo número 3, el filósofo 5 se pone en turno de espera porque solo tiene un tenedor, y los filósofos restantes están pensando. Y así sucesivamente.



4. Solución y ejecución en lenguaje C.

- **Algoritmo 1**
 - **Estructura de filósofo.**

Tenemos la estructura filosofo que contiene un nombre, asignamos el * para el mejor uso de memoria, cantidad de comida, por defecto pusimos 6 (modificable) y dos atributos de tipo tenedor.

```
struct filosofo{  
    char * nombre;  
    int cantComida;  
    struct tenedor * ten1;  
    struct tenedor * ten2;  
};
```

- **Estructura tenedor.**

La estructura tenedor contiene un estado, que será una bandera a él filósofo, cuando estado==1 libre, estado==2 ocupado.

```
struct tenedor{
    int estado;
};
```

- **Función comer.**

Creamos la función comer con el objetivo de repartir de manera equitativa los tenedores, de manera cíclica podemos asignar y rotar los tenedores entre todos los filósofos para que todos puedan comer satisfactoriamente.

```
void * comer( void * h1){
    struct filosofo * fil;
    fil = (struct filosofo*) h1;
    printf("%s %s \n", fil->nombre, "esta pensando");
    while(fil->cantComida > 0){
        if(fil->ten1->estado == 0 && fil->ten2->estado == 0){
            printf("%s %s \n", fil->nombre, "tiene hambre");
            fil->ten1->estado = fil->ten2->estado = 1;
            printf("%s %s \n", fil->nombre, "agarro los 2 tenedores");
            while(fil->cantComida > 0){
                fil->cantComida--;
                printf("%s %s \n", fil->nombre, "esta comiendo");
            }
        }
        else{
            printf("%s %s \n", fil->nombre, "no puede comer");
        }
    }
    fil->ten1->estado = fil->ten2->estado = 0;
    printf("%s %s \n", fil->nombre, "termino de comer");
}
```

- **CUERPO (MAIN).**

Creamos los n filósofos deseados y asignamos un tenedor cada uno (struct filósofo, struct tenedor) asignamos a todos los tenedores como estado=0 porque empiezan siendo libres.

Enviamos a nuestra función comer los n filósofos y ahí la magia del algoritmo.

```
int main() {
    pthread_t thread1, thread2, thread3, thread4, thread5;
    struct tenedor * ten1 = (struct tenedor *) malloc (sizeof(struct tenedor));
    struct tenedor * ten2 = (struct tenedor *) malloc (sizeof(struct tenedor));
    struct tenedor * ten3 = (struct tenedor *) malloc (sizeof(struct tenedor));
    struct tenedor * ten4 = (struct tenedor *) malloc (sizeof(struct tenedor));
    struct tenedor * ten5 = (struct tenedor *) malloc (sizeof(struct tenedor));
    struct filosofo * fil1 = (struct filosofo *) malloc (sizeof(struct filosofo));
    struct filosofo * fil2 = (struct filosofo *) malloc (sizeof(struct filosofo));
    struct filosofo * fil3 = (struct filosofo *) malloc (sizeof(struct filosofo));
    struct filosofo * fil4 = (struct filosofo *) malloc (sizeof(struct filosofo));
    struct filosofo * fil5 = (struct filosofo *) malloc (sizeof(struct filosofo));

    ten1->estado = ten2->estado = ten3->estado = ten4->estado = ten5->estado = 0;
    fil1->nombre = "Fujimori";
    fil1->cantComida = comida;
    fil1->ten1 = ten1;
    fil1->ten2 = ten2;
    fil2->nombre = "Castillo";
    fil2->cantComida = comida;
    fil2->ten1 = ten2;
    fil2->ten2 = ten3;
    fil3->nombre = "Acuna";
    fil3->cantComida = comida;
    fil3->ten1 = ten3;
    fil3->ten2 = ten4;
    fil4->nombre = "Ollanta";
    fil4->cantComida = comida;
    fil4->ten1 = ten4;
    fil4->ten2 = ten5;
    fil5->nombre = "Llica";
    fil5->cantComida = comida;
    fil5->ten1 = ten5;
    fil5->ten2 = ten1;
    int iret1, iret2, iret3, iret4, iret5;
    iret1 = pthread_create( &thread1, NULL, comer, (void*) fil1);
    iret2 = pthread_create( &thread2, NULL, comer, (void*) fil2);
    iret3 = pthread_create( &thread3, NULL, comer, (void*) fil3);
    iret4 = pthread_create( &thread4, NULL, comer, (void*) fil4);
    iret5 = pthread_create( &thread5, NULL, comer, (void*) fil5);
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
    pthread_join( thread3, NULL);
    pthread_join( thread4, NULL);
    pthread_join( thread5, NULL);
    printf("Thread 1 returns: %d\n", iret1);
    printf("Thread 2 returns: %d\n", iret2);
    printf("Thread 3 returns: %d\n", iret3);
    printf("Thread 4 returns: %d\n", iret4);
    printf("Thread 5 returns: %d\n", iret5);
    return 0;
}
```

- **LIBRERIAS USADAS DE LENGUAJE C**

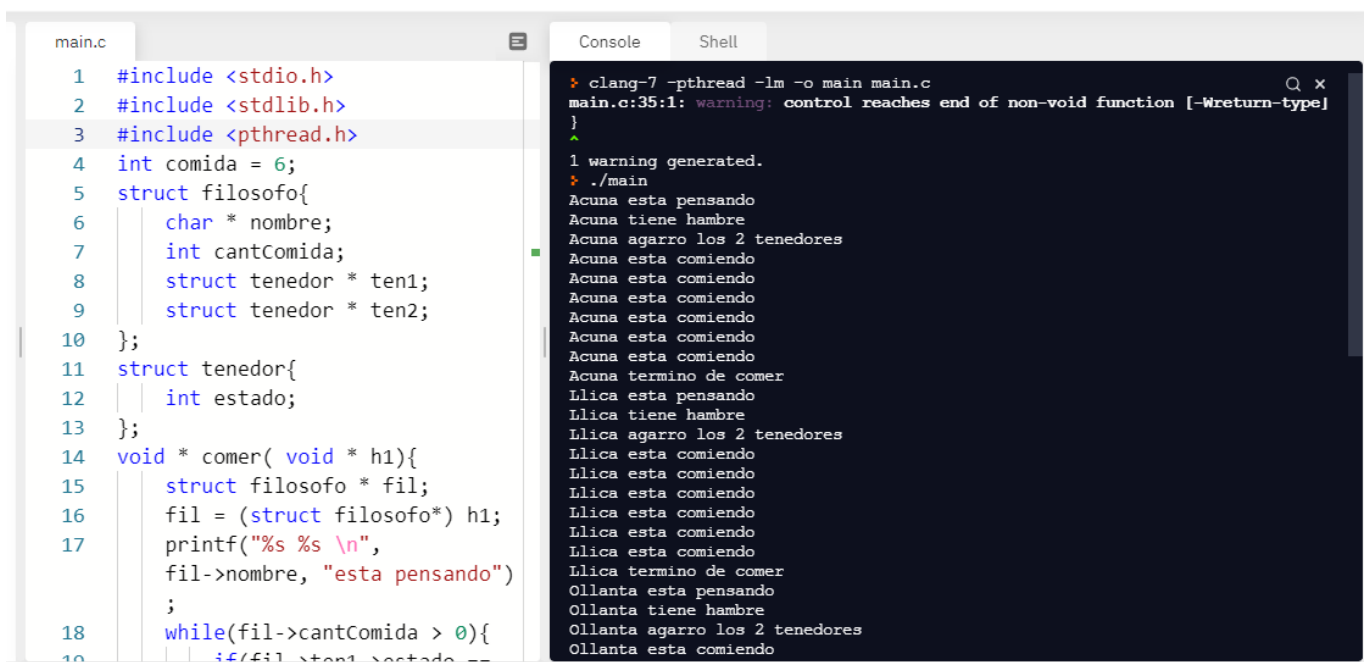
-

`#include <stdio.h>` -> Standard input-output header" (cabecera estándar E/S). Flujo de de datos, consola.

`#include <stdlib.h>` -> Standard library o biblioteca estándar). Es el archivo de cabecera de la biblioteca estándar.

`#include <pthread.h>` -> Languages provide the POSIX thread(**pthread**) standard API(Application program Interface).

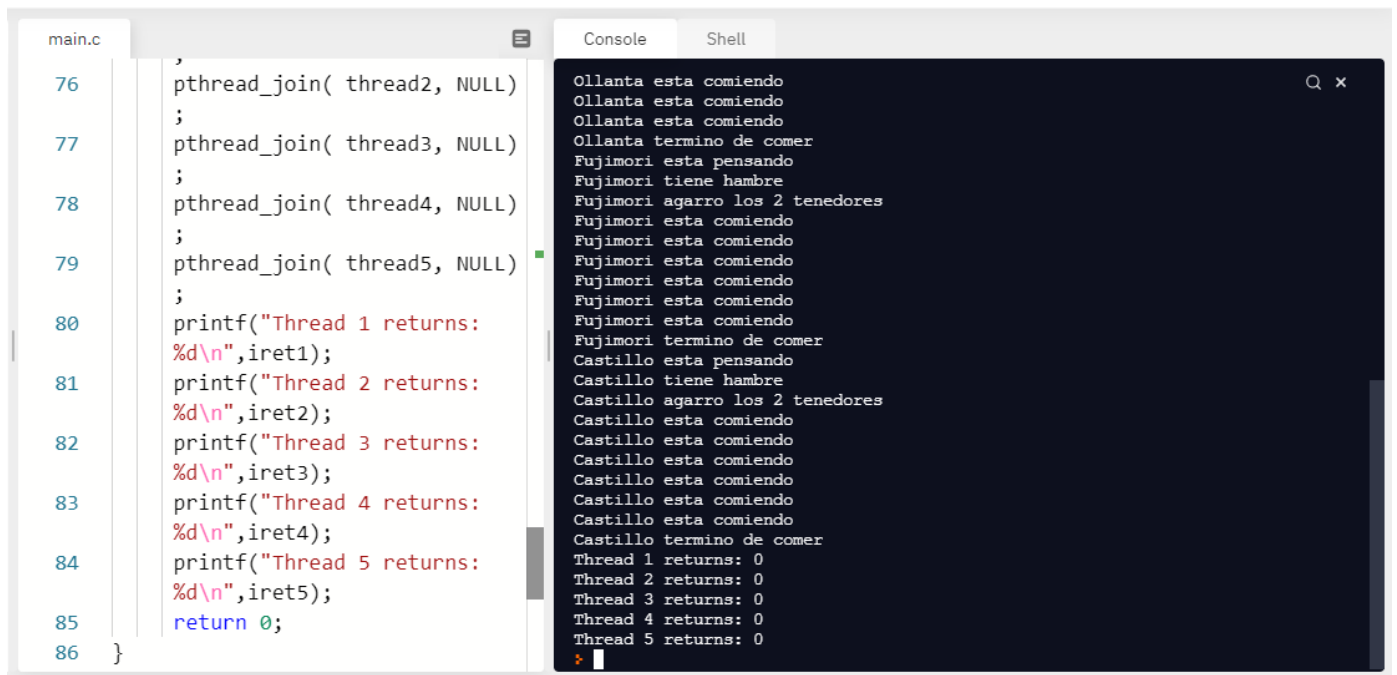
- **IMÁGENES DE COMPILACION.**
COMPILADOR REPLIT C
IDE DEV C++
SUBLIME TEXT WITH MINGW



The screenshot shows a code editor with a file named `main.c` and a terminal window. The code in `main.c` defines a philosopher structure and a function to simulate eating and thinking. The terminal shows the compilation command `clang-7 -pthread -lm -o main main.c` and the execution output, which displays the sequence of actions for three philosophers: Acuna, Illica, and Ollanta.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 int comida = 6;
5 struct filosofo{
6     char * nombre;
7     int cantComida;
8     struct tenedor * ten1;
9     struct tenedor * ten2;
10 };
11 struct tenedor{
12     int estado;
13 };
14 void * comer( void * h1){
15     struct filosofo * fil;
16     fil = (struct filosofo*) h1;
17     printf("%s %s \n",
18         fil->nombre, "esta pensando")
19     ;
20     while(fil->cantComida > 0){
21         if(fil->ten1->estado ==
```

```
clang-7 -pthread -lm -o main main.c
main.c:35:1: warning: control reaches end of non-void function [-Wreturn-type]
}
^
1 warning generated.
$ ./main
Acuna esta pensando
Acuna tiene hambre
Acuna agarro los 2 tenedores
Acuna esta comiendo
Acuna esta comiendo
Acuna esta comiendo
Acuna esta comiendo
Acuna esta comiendo
Acuna esta comiendo
Acuna termino de comer
Illica esta pensando
Illica tiene hambre
Illica agarro los 2 tenedores
Illica esta comiendo
Illica esta comiendo
Illica esta comiendo
Illica esta comiendo
Illica esta comiendo
Illica termino de comer
Ollanta esta pensando
Ollanta tiene hambre
Ollanta agarro los 2 tenedores
Ollanta esta comiendo
```

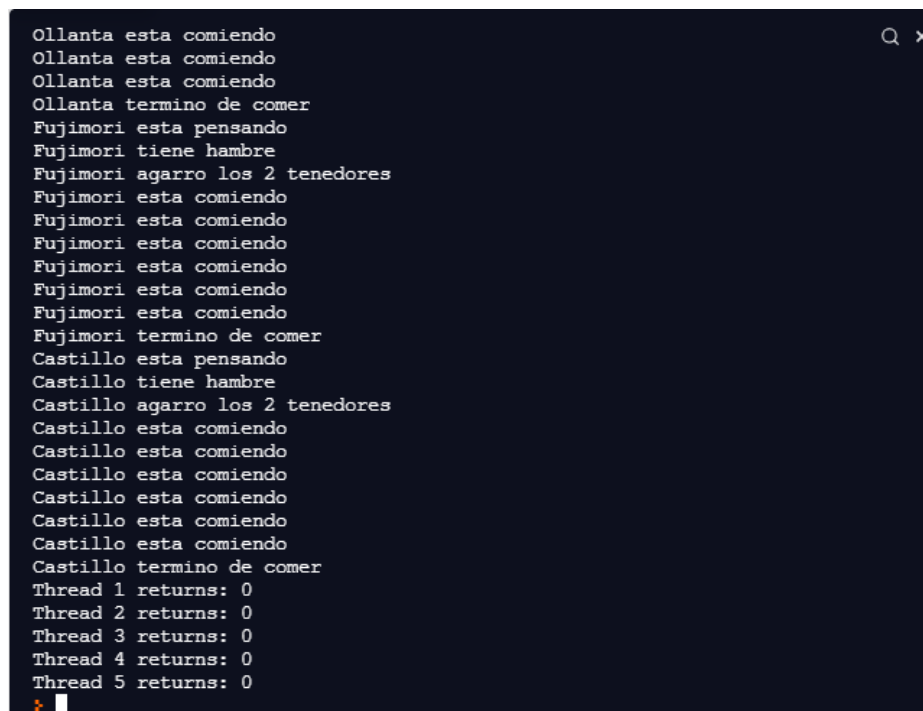



The image shows a code editor with a file named `main.c` and a console window. The code defines 5 threads (1-5) that perform a sequence of actions: thinking, getting hungry, picking up 2 forks, eating, and finishing. The console output shows the execution of these threads in a non-deterministic order, with some threads finishing before others start. The final output shows that all threads have returned successfully.

```
main.c
76 pthread_join( thread2, NULL)
77 ;
78 pthread_join( thread3, NULL)
79 ;
80 pthread_join( thread4, NULL)
81 ;
82 pthread_join( thread5, NULL)
83 ;
84 printf("Thread 1 returns:
85 %d\n",iret1);
86 printf("Thread 2 returns:
87 %d\n",iret2);
88 printf("Thread 3 returns:
89 %d\n",iret3);
90 printf("Thread 4 returns:
91 %d\n",iret4);
92 printf("Thread 5 returns:
93 %d\n",iret5);
94 return 0;
95 }
```

Console

```
Ollanta esta comiendo
Ollanta esta comiendo
Ollanta esta comiendo
Ollanta termino de comer
Fujimori esta pensando
Fujimori tiene hambre
Fujimori agarro los 2 tenedores
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori termino de comer
Castillo esta pensando
Castillo tiene hambre
Castillo agarro los 2 tenedores
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo termino de comer
Thread 1 returns: 0
Thread 2 returns: 0
Thread 3 returns: 0
Thread 4 returns: 0
Thread 5 returns: 0
```



The console window displays the output of the program, showing the execution of 5 threads (1-5) in a non-deterministic order. The output shows the threads performing actions like thinking, getting hungry, picking up 2 forks, eating, and finishing. The final output shows that all threads have returned successfully.

```
Ollanta esta comiendo
Ollanta esta comiendo
Ollanta esta comiendo
Ollanta termino de comer
Fujimori esta pensando
Fujimori tiene hambre
Fujimori agarro los 2 tenedores
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori esta comiendo
Fujimori termino de comer
Castillo esta pensando
Castillo tiene hambre
Castillo agarro los 2 tenedores
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo esta comiendo
Castillo termino de comer
Thread 1 returns: 0
Thread 2 returns: 0
Thread 3 returns: 0
Thread 4 returns: 0
Thread 5 returns: 0
```

5. PUNTO DE VISTA, PERSONAL.

- Después de haber investigado y analizado los diferentes algoritmos he llegado a la conclusión de que el tiempo va a depender mucho de la cantidad de datos a procesar y con cantidad me refiero a la n filósofos y n tenedores en juego, como también el volumen de comida.
- Como dato adicional acotar que este fue un ejercicio planteando en la RPC de agosto del 2012 planteándolo como un reto de programación actual (hoy) el cual se busca mejorar e crear nuevos algoritmos.
- Por mi parte podría acotar como juego o reto de programación el poder agregar una regla más la cual sería que un filósofo puede robar/coger un tenedor del filósofo de su derecha, siempre y cuando el filósofo de su izquierda no este comiendo. La idea se me ocurrió al recordar una de las pruebas de inteligencia que se aplicaban a una población totalmente pobre para medir sus capacidades de oportunidad, esta idea se me ocurrió en base a la serie 3% en netflix.

- **BIBLIOGRAFIA.**

- <http://darksystem79.blogspot.com/2013/11/filosofos-comensales.html#:~:text=El%20problema%20de%20los%20fil%C3%B3sofos,procesos%20en%20un%20sistema%20operativo.>
- https://es.wikipedia.org/wiki/Problema_de_la_cena_de_los_fil%C3%B3sofos
- <https://www2.infor.uva.es/~cllamas/concurr/pract97/immartin/index.html>
- <https://www.studocu.com/pe/document/universidad-nacional-del-callao/sistemas-neumaticos-electroneumaticosoleohidraulicos-electrohidraulicos/informe/monografia-de-filosofos-comensales/5223748/view>
- <https://books.google.com.pe/books?id=dYZ8DwAAQBAJ&pg=PA297&dq=pthread.h+c%2B%2B&hl=es-419&sa=X&ved=2ahUKEwjspanW7MDwAhUiK7kGHfaRAIAQ6AEwAHoECAAQAg#v=onepage&q=pthread.h%20c%2B%2B&f=false>
-