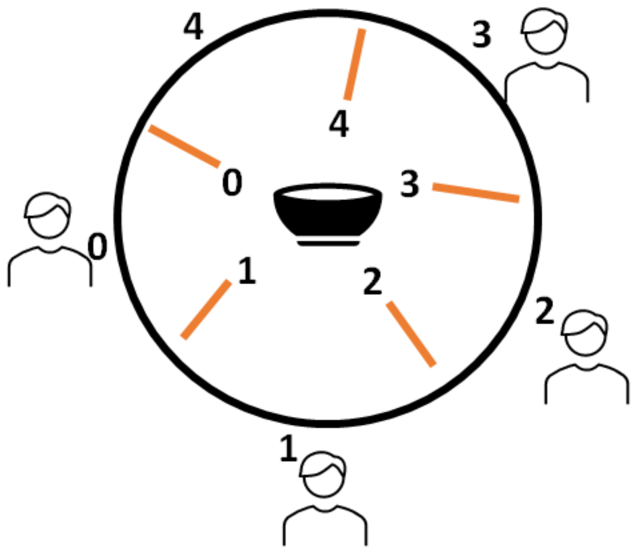
Program on Dining Philosopher Problem

Programs / By Baljit Singh Saini

In this post we are going to understand the C program implementing the solution to the Dining Philosopher Problem. The Dining Philosopher Problem states that there are five philosophers which do two thinks: think and eat. They share a table having a chair for each one of them. In the center of the table there is a bowl of rice and the table is laid with 5 single chopsticks (Refer Figure Below).



When a philosopher thinks, he does not interact with others. When he gets hungry, he tries to pick up the two chopsticks that are near to him. For example, philosopher 1 will try to pick chopsticks 1 and 2. But the philosopher can pickup only one chopstick at a time. He can not take a chopstick that is already in the hands of his neighbour. The philosopher stars to eat when he has both his chopsticks in his hand. After eating the philosopher puts down both the chopsticks and starts to think again.

Solution to Dining Philosopher Problem

Represent each chopstick with a semaphore. Each philosopher first picks up the left chopstick and then the right chopstick using the wait() operation each semaphore. After eating he puts down the chopsticks by using the signal() operation on each chopstick.

```
#include<stdio.h>
```

```
sem_t chopstick[5];
void * philos(void *);
void eat(int);
int main()
 {
         int i,n[5];
         pthread_t T[5];
         for(i=0;i<5;i++)
         sem_init(&chopstick[i],0,1);
         for(i=0;i<5;i++){
                 n[i]=i;
                 pthread_create(&T[i],NULL,philos,(void *)&n[i]);
                 }
         for(i=0;i<5;i++)
                 pthread_join(T[i],NULL);
 }
void * philos(void * n)
 {
         int ph=*(int *)n;
         printf("Philosopher %d wants to eat\n",ph);
         printf("Philosopher %d tries to pick left chopstick\n",ph);
         sem_wait(&chopstick[ph]);
         printf("Philosopher %d picks the left chopstick\n",ph);
         printf("Philosopher %d tries to pick the right chopstick\n",ph);
         sem_wait(&chopstick[(ph+1)%5]);
         printf("Philosopher %d picks the right chopstick\n",ph);
         eat(ph);
         sleep(2);
         printf("Philosopher %d has finished eating\n",ph);
         sem_post(&chopstick[(ph+1)%5]);
         printf("Philosopher %d leaves the right chopstick\n",ph);
         sem_post(&chopstick[ph]);
         printf("Philosopher %d leaves the left chopstick\n",ph);
 }
 void eat(int ph)
 {
         printf("Philosopher %d begins to eat\n",ph);
 }
```

How it works?

Now let us understand how the code works

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
sem_t chopstick[5];
void * philos(void *);
void eat(int);
```

In the main() function there are three for loops. The first loop is used to initialize each semaphore variable with initial value to 1. The second for loop is used to create five threads which will act as the five philosophers. The third for loop uses the pthread_join function which makes the parent program wait for each of the thread to finish.

Next is the *philos* function. The philos function when called by each thread receives the same value as the thread number. For example if thread one runs, the variable ph in the philos function is assigned the value n. This is done because each philosopher n before eating will pick two chopstick, n and (n+1)%5.
Next, the sem_wait function is used on the left chopstick first

```
sem_wait(&chopstick[ph]);
```

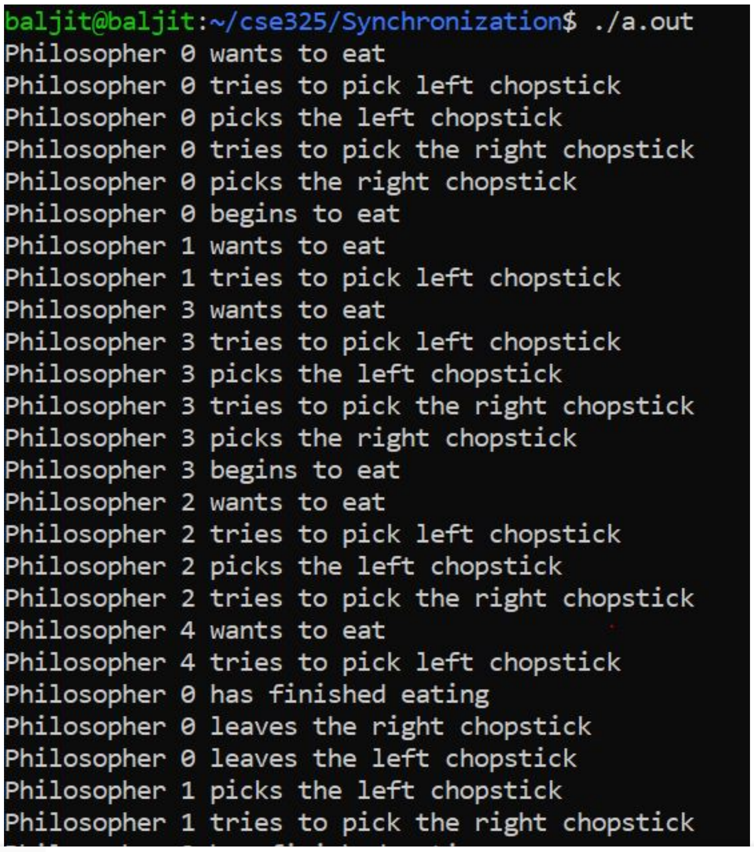
If successful, the thread executes the sem_wait function on the right chopstick

```
sem_wait(&chopstick[(ph+1)%5]);
```

These two operations are equivalent to picking the left chopstick and then the right chopstick. If both these operations are successful this means that the philosopher is able to pick both the chopsticks and hence will start to eat by calling the *eat()* function. After eating both the chopsticks are release by using the *sem_post()* function.

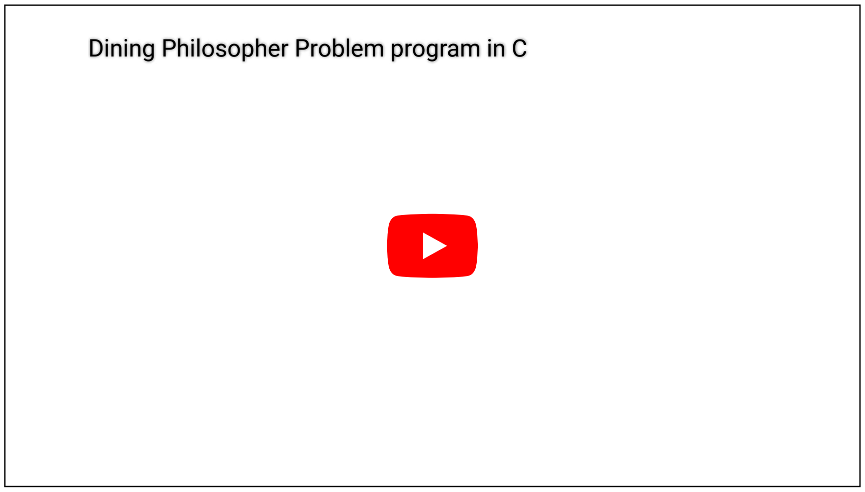
Output

Here's a sample output. Your output might differ each type you run the program. This is because the sequence of execution of threads will be different. Try to understand the output below and then relate it with what you get.



```
baljit@baljit:~/cse325/Synchronization$ ./a.out
Philosopher 0 wants to eat
Philosopher 0 tries to pick left chopstick
Philosopher 0 picks the left chopstick
Philosopher 0 tries to pick the right chopstick
Philosopher 0 picks the right chopstick
Philosopher 0 begins to eat
Philosopher 1 wants to eat
Philosopher 1 tries to pick left chopstick
Philosopher 3 wants to eat
Philosopher 3 tries to pick left chopstick
Philosopher 3 picks the left chopstick
Philosopher 3 tries to pick the right chopstick
Philosopher 3 picks the right chopstick
Philosopher 3 begins to eat
Philosopher 2 wants to eat
Philosopher 2 tries to pick left chopstick
Philosopher 2 picks the left chopstick
Philosopher 2 tries to pick the right chopstick
Philosopher 4 wants to eat
Philosopher 4 tries to pick left chopstick
Philosopher 0 has finished eating
Philosopher 0 leaves the right chopstick
Philosopher 0 leaves the left chopstick
Philosopher 1 picks the left chopstick
Philosopher 1 tries to pick the right chopstick
```

Video Link



Viva Questions on Program on Dining Philosopher Problem

Q1. Which function is used to create the threads?

Q2. What is the use of sem_init() function?

Q3. Why is the sem_wait() function used twice in the philos function?

Relevant Programs

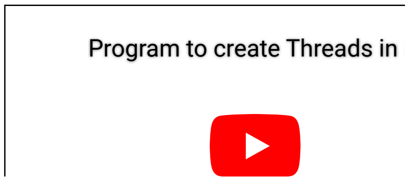
Program to create Threads

Using the semaphore variables

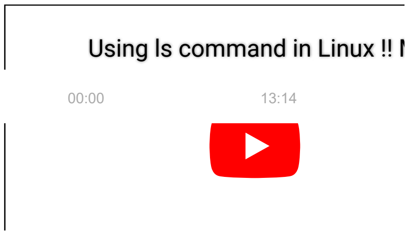
← Previous Post Next Post →

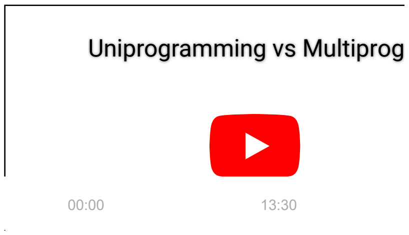
Search …

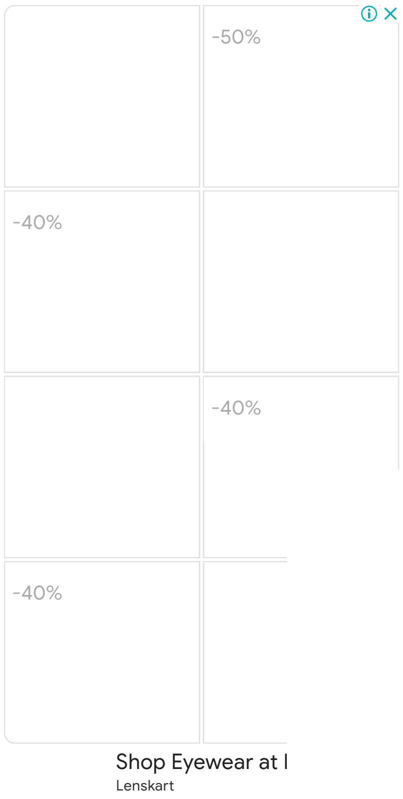
OS Lab



00:00 09:29


Using ls command in Linux !! I
00:00
13:14


Uniprogramming vs Multiprog
00:00
13:30


-50%
-40%
-40%
-40%
Shop Eyewear at I
Lenskart

Categories

- Memory Management
- Practice Problems
- Process
- Process Synchronization
- System calls
- Thread

- Programs
- Question Hub
- UGC-NET MCQ's for OS
- Uncategorized