

OPERATING SYSTEM

PROJECT REPORT

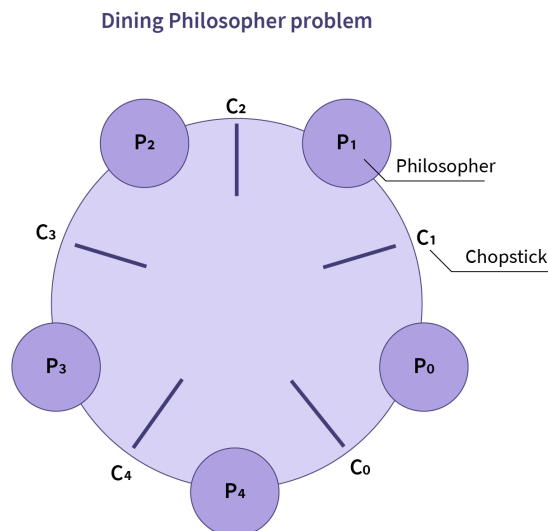
DINING PHILOSOPHERS PROBLEM

1. PROJECT DESCRIPTION

The dining philosophers problem is considered a classic synchronization problem. This problem deals with resource allocation, it is an example of a large class of concurrency-control problems. It was originally formulated in 1965 by Edsger Dijkstra as a student exam exercise, presented in terms of computers competing for access to tape drive peripherals. Soon after, Tony Hoare gave the problem its present formulation.

2. PROBLEM STATEMENT

Five silent philosophers sit at a round table around a bowl of spaghetti. Chopsticks are placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right chopsticks. Only one philosopher can hold each chopstick so a philosopher can use the chopstick only if another philosopher is not using it. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right and the one on their left as they become available, but cannot start eating before getting both chopsticks. Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed. The problem is how to design a discipline of behavior such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, without any deadlock.



NUMBER OF WAYS TO RESOLVE THIS PROBLEM:

- Allows at most four philosophers who sit together at one table.
- Allows a philosopher to take chopsticks only if both chopsticks are there.
- Using asymmetric solutions, philosophers in odd numbers took a left chopstick first and then a right chopstick. While philosophers in even numbers take the right chopstick first and then the left chopstick.

3. OPERATING SYSTEM IMPLEMENTATION

We have utilized Semaphore and Threads to exhibit and take care of the issue. One straightforward arrangement is to address every chopstick with a semaphore. A philosopher attempts to get a chopstick by executing a wait () procedure on that semaphore so that not more than one philosopher can eat at a time. The philosopher delivers his chopsticks by executing the signal () procedure on the semaphores. Philosophers in odd numbers take the left chopstick first and then the right chopstick. Whereas, philosophers in even numbers take the right chopstick first and then the left chopstick.

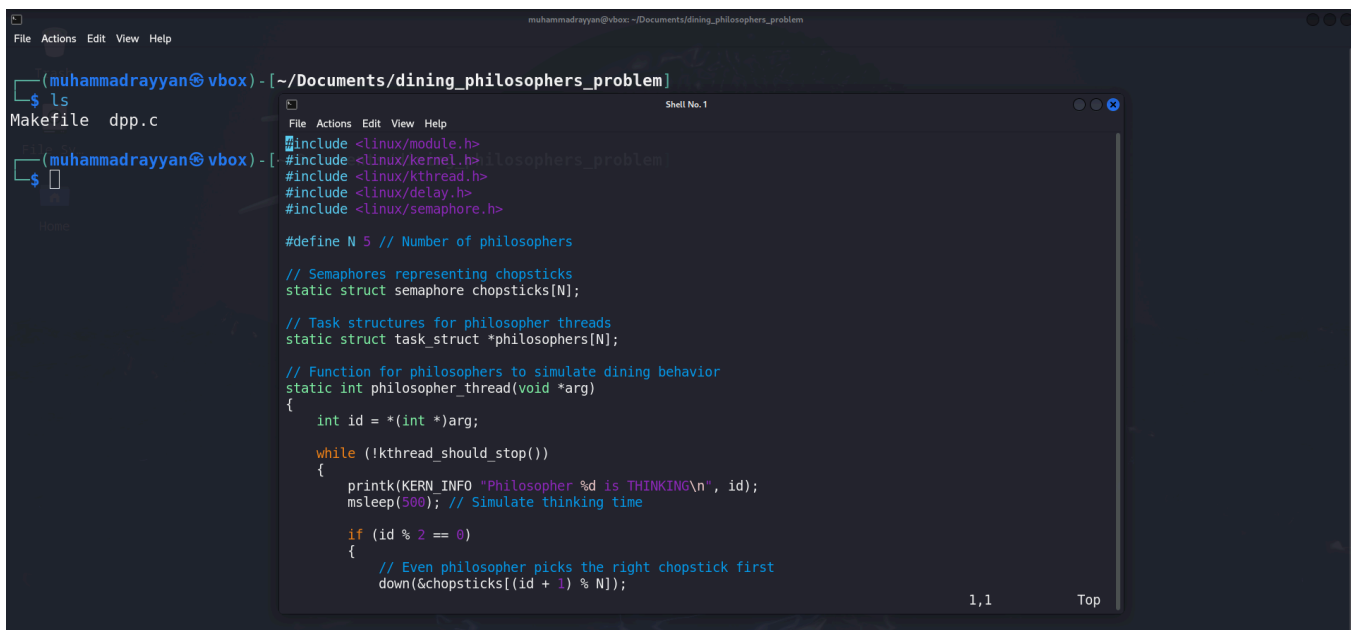
4. PROJECT OUTCOME

The main objective of this project was to learn the use of semaphores and threads and to understand the concept of deadlock, race conditions, resource allocation, and process synchronization.

5. PROJECT RECORDING

[operating_system_project/dining_philosophers_problem](#)

6. PROJECT SNIPPETS



```
(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ ls
Makefile  dpp.c

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ cat dpp.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/kthread.h>
#include <linux/delay.h>
#include <linux/semaphore.h>

#define N 5 // Number of philosophers

// Semaphores representing chopsticks
static struct semaphore chopsticks[N];

// Task structures for philosopher threads
static struct task_struct *philosophers[N];

// Function for philosophers to simulate dining behavior
static int philosopher_thread(void *arg)
{
    int id = *(int *)arg;

    while (!kthread_should_stop())
    {
        printk(KERN_INFO "Philosopher %d is THINKING\n", id);
        msleep(500); // Simulate thinking time

        if (id % 2 == 0)
        {
            // Even philosopher picks the right chopstick first
            down(&chopsticks[(id + 1) % N]);
        }
    }
}
```

```
muhammadrayyan@vbox: ~/Documents/dining_philosophers_problem
File Actions Edit View Help

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ ls
Makefile dpp.c

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ obj-m += dpp.o
```

```
(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ ls
Makefile dpp.c

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ make -C /lib/modules/$(uname -r)/build M=$PWD
make: Entering directory '/usr/src/linux-headers-6.11.2-amd64'
  CC [M] /home/muhammadrayyan/Documents/dining_philosophers_problem/dpp.o
  MODPOST /home/muhammadrayyan/Documents/dining_philosophers_problem/Module.symvers
  CC [M] /home/muhammadrayyan/Documents/dining_philosophers_problem/dpp.mod.o
  LD [M] /home/muhammadrayyan/Documents/dining_philosophers_problem/dpp.ko
  BTF [M] /home/muhammadrayyan/Documents/dining_philosophers_problem/dpp.ko
Skipping BTF generation for /home/muhammadrayyan/Documents/dining_philosophers_problem/dpp.ko due to unavailability of vmlinux
make: Leaving directory '/usr/src/linux-headers-6.11.2-amd64'

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$
```

```
(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ ls
Makefile Module.symvers dpp.c dpp.ko dpp.mod dpp.mod.c dpp.mod.o dpp.o modules.order

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ sudo insmod dpp.ko
[sudo] password for muhammadrayyan:

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$
```

```
(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ dmesg | tail -n 30
[ 6776.423756] Philosopher 1 picks up right chopstick 2
[ 6776.423757] Philosopher 1 is EATING
[ 6776.423759] Philosopher 4 picks up right chopstick 0
[ 6776.423760] Philosopher 4 picks up left chopstick 4
[ 6776.423761] Philosopher 4 is EATING
[ 6776.423762] Philosopher 3 picks up left chopstick 3
[ 6776.935437] Philosopher 1 puts down left chopstick 1
[ 6776.935447] Philosopher 1 puts down right chopstick 2
[ 6776.935451] Philosopher 1 is THINKING
[ 6776.935474] Philosopher 4 puts down left chopstick 4
[ 6776.935477] Philosopher 4 puts down right chopstick 0
[ 6776.935479] Philosopher 4 is THINKING
[ 6776.937370] Philosopher 3 picks up right chopstick 4
[ 6776.937378] Philosopher 3 is EATING
[ 6776.937386] Philosopher 0 picks up right chopstick 1
[ 6776.937389] Philosopher 0 picks up left chopstick 0
[ 6776.937391] Philosopher 0 is EATING
[ 6777.446432] Philosopher 3 puts down left chopstick 3
[ 6777.446439] Philosopher 3 puts down right chopstick 4
[ 6777.446441] Philosopher 3 is THINKING
[ 6777.446443] Philosopher 2 picks up right chopstick 3
[ 6777.446444] Philosopher 2 picks up left chopstick 2
[ 6777.446445] Philosopher 2 is EATING
[ 6777.446448] Philosopher 0 puts down left chopstick 0
[ 6777.446449] Philosopher 0 puts down right chopstick 1
[ 6777.446494] Philosopher 0 is THINKING
[ 6777.446497] Philosopher 1 picks up left chopstick 1
```

```

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ sudo rmmod dpp

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ make -C /lib/modules/$(uname -r)/build M=$PWD clean
make: Entering directory '/usr/src/linux-headers-6.11.2-amd64'
CLEAN /home/muhammadrayyan/Documents/dining_philosophers_problem/Module.symvers
make: Leaving directory '/usr/src/linux-headers-6.11.2-amd64'

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$ ls
Makefile dpp.c

(muhammadrayyan@vbox) - [~/Documents/dining_philosophers_problem]
$

```

```

(muhammadrayyan@vbox) - [~/Documents/dpp]
$ ls
dpp.c

(muhammadrayyan@vbox) - [~/Documents/dpp]
$

```

```

File Actions Edit View Help
Shell No. 1

#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);
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);
}
1,1 Top

```

```

(muhammadrayyan@vbox) - [~/Documents/dpp]
$ ls
dpp.c

(muhammadrayyan@vbox) - [~/Documents/dpp]
$ gcc dpp.c -lpthread

(muhammadrayyan@vbox) - [~/Documents/dpp]
$ ls
a.out dpp.c

(muhammadrayyan@vbox) - [~/Documents/dpp]
$

```

```
File Actions Edit View Help
(muhammadraysan@vbox) - [~/Documents/dpp]
$ ./a.out
Philosopher 0 wants to eat
Philosopher 0 tries to pick left chopstick
Philosopher 1 wants to eat
Philosopher 1 tries to pick left chopstick
Philosopher 1 picks the left chopstick
Philosopher 1 tries to pick the right chopstick
Philosopher 1 picks the right chopstick
Philosopher 1 begins to eat
Philosopher 2 wants to eat
Philosopher 2 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 4 wants to eat
Philosopher 4 tries to pick left chopstick
Philosopher 0 picks the left chopstick
Philosopher 0 tries to pick the right chopstick
Philosopher 3 has finished eating
Philosopher 4 picks the left chopstick
Philosopher 4 tries to pick the right chopstick
Philosopher 3 leaves the right chopstick
Philosopher 3 leaves the left chopstick
Philosopher 1 has finished eating
Philosopher 1 leaves the right chopstick
```

7. CODE

- **Kernel Module Code:**

```
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/kthread.h>

#include <linux/delay.h>

#include <linux/semaphore.h>

#define N 5 // Number of philosophers

// Semaphores representing chopsticks
static struct semaphore chopsticks[N];

// Task structures for philosopher threads
static struct task_struct *philosophers[N];

// Function for philosophers to simulate dining behavior
static int philosopher_thread(void *arg)
{

```

```

int id = *(int *)arg;

while (!kthread_should_stop())
{
    printk(KERN_INFO "Philosopher %d is THINKING\n", id);
    msleep(500); // Simulate thinking time

    if (id % 2 == 0)
    {
        // Even philosopher picks the right chopstick first
        down(&chopsticks[(id + 1) % N]);
        printk(KERN_INFO "Philosopher %d picks up right chopstick %d\n", id, (id + 1) % N);

        down(&chopsticks[id]);
        printk(KERN_INFO "Philosopher %d picks up left chopstick %d\n", id, id);
    }
    else
    {
        // Odd philosopher picks the left chopstick first
        down(&chopsticks[id]);
        printk(KERN_INFO "Philosopher %d picks up left chopstick %d\n", id, id);

        down(&chopsticks[(id + 1) % N]);
        printk(KERN_INFO "Philosopher %d picks up right chopstick %d\n", id, (id + 1) % N);
    }

    printk(KERN_INFO "Philosopher %d is EATING\n", id);
    msleep(500); // Simulate eating time

    // Release chopsticks
    up(&chopsticks[id]);

```

```

    printk(KERN_INFO "Philosopher %d puts down left chopstick %d\n", id, id);

    up(&chopsticks[(id + 1) % N]);
    printk(KERN_INFO "Philosopher %d puts down right chopstick %d\n", id, (id + 1) % N);
}

return 0;
}

// Module initialization
static int __init dpp_init(void)
{
    int i;

    printk(KERN_INFO "Initializing Dining Philosophers Problem Module\n");

    // Initialize semaphores for chopsticks
    for (i = 0; i < N; i++)
    {
        sema_init(&chopsticks[i], 1);
    }

    // Create threads for philosophers
    for (i = 0; i < N; i++)
    {
        philosophers[i] = kthread_create(philosopher_thread, &i, "Philosopher-%d", i);
        if (philosophers[i])
        {
            wake_up_process(philosophers[i]);
            printk(KERN_INFO "Philosopher %d thread created\n", i);
        }
    }
}

```

```

    else
    {
        printk(KERN_ERR "Failed to create thread for Philosopher %d\n", i);
    }
    msleep(10); // Small delay to avoid race conditions during thread creation
}

return 0;
}

// Module cleanup
static void __exit dpp_exit(void)
{
    int i;

    printk(KERN_INFO "Cleaning up Dining Philosophers Problem Module\n");

    // Stop philosopher threads
    for (i = 0; i < N; i++)
    {
        if (philosophers[i])
        {
            kthread_stop(philosophers[i]);
            printk(KERN_INFO "Philosopher %d thread stopped\n", i);
        }
    }
}

module_init(dpp_init);
module_exit(dpp_exit);

```



```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("OpenAI");
MODULE_DESCRIPTION("Dining Philosophers Problem as a Linux Kernel Module");
```

- **User Module Code:**

```
#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);
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);
}

```

PROFILE LINKS

- [linkedin.com/muhammadravayyan](https://www.linkedin.com/in/muhammadravayyan)

- github.com/muhammadraysan

CERTIFICATION





Statement of Achievement

MUHAMMAD RAYYAN

has successfully achieved student level credential for completing the Linux Unhatched course, provided by Cisco Networking Academy in collaboration with NDG.

The graduate is able to proficiently:

- Understand the basics of using the Linux command line.
- Navigate home and system directories and listing files in various locations.
- Create, move and delete files and directories under the home directory.
- Search and extract data from files in the home directory.
- Turn repetitive commands into simple scripts.
- Describe where various types of information are stored on a Linux system.
- Query vital networking configuration.
- Manage various types of users on a Linux system.
- Understand and manipulate file permissions and ownership settings.



Scan to Verify



Laura Quintana
Vice President and General Manager
Cisco Networking Academy

Issued on: Dec 17, 2024