CS311 Operating Systems Project Report

Project Title: System Call – Dining Philosophers Problem

Course: CS311 Operating Systems

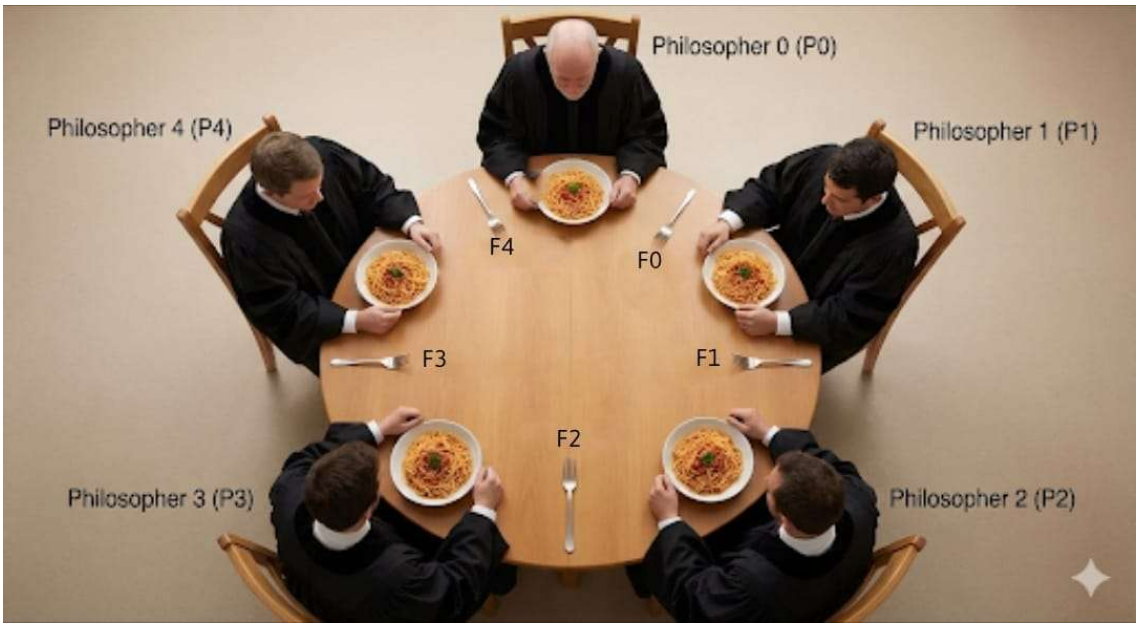
Semester: Fall 2025

Team Information

- **Name:** Abdul Wadood Khan | **Roll No:** 2023029
- **Name:** Rayyan Mehmood | **Roll No:** 2023602
- **Name:** Syed Murtaza Salman | **Roll No:** 2023694

1. Introduction to the Problem

The Dining Philosophers Problem



The Dining Philosophers problem is a classic synchronization scenario in Operating Systems used to illustrate the challenges of resource allocation and concurrency control. The problem visualizes five philosophers sitting around a circular table. They spend their lives alternating between two states: Thinking and Eating1.

There are five forks (resources) placed between each pair of philosophers. To eat, a philosopher must acquire **two forks**—specifically, the one to their immediate left and the one to their immediate right. The core constraints are:

- **Mutual Exclusion:** No two philosophers can hold the same fork at the same time.
- **Deadlock & Starvation:** The solution must prevent a situation where every philosopher picks up one fork and waits indefinitely for the second (deadlock), or where a philosopher never gets a chance to eat (starvation).

Our Kernel-Level Solution

Standard solutions often use user-space threads and mutexes (e.g., `pthread_mutex`). In this project, we implemented a unique solution by moving the resource management directly into the Linux Kernel (v6.8.1). We created a custom system call that utilizes Kernel Semaphores to strictly manage the forks. By entering the kernel (Ring 0) to request resources, we ensure atomic access and robust synchronization that cannot be bypassed by user-space logic.

---

2. The Unique Function

We implemented a new system call named **`sys_dining_philosophers`**. This function serves as the interface between the user program and the kernel's resource management[2].

- **System Call Number:** 548 (Added to system call table)
- **Kernel Version:** 6.8.1-u2023694 (Custom compiled kernel)

Function Prototype:

C

```
asmlinkage long sys_dining_philosophers(int id, int action);
```

Logic Description:

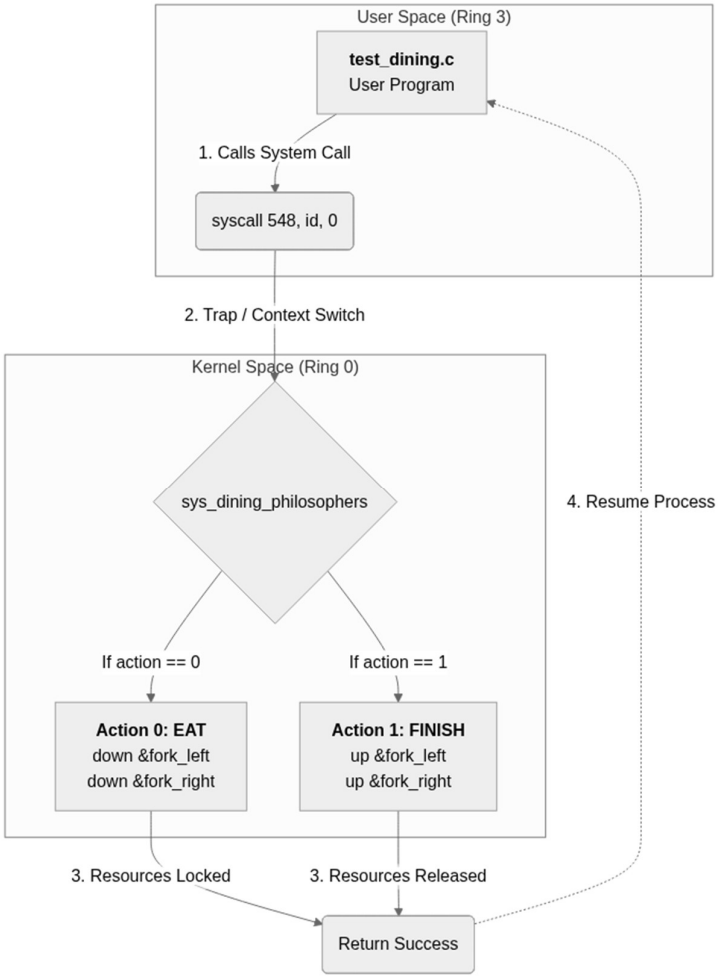
The function accepts two arguments:

1. **id**: The identifier of the philosopher (0–4).
2. **action**: A control flag determining the operation.
    - **Action 0 (EAT):** The kernel executes down() operations on the semaphores corresponding to the philosopher's left and right forks. This strictly locks the resources. If a fork is busy, the process is put to sleep until it becomes available.
    - **Action 1 (FINISH):** The kernel executes up() operations on the corresponding semaphores, releasing the forks for neighbors to use.

---

3. System Call Flow Diagram

The following diagram illustrates the transition from User Space to Kernel Space when the program executes.



---

4. Execution Commands

To execute the project, the following sequence of commands is used in the Linux terminal[4]:

1. Verification of Custom Kernel:

First, we confirm that the system has booted into our custom-compiled kernel containing the new system call.

Bash

```
uname -r
```

```
# Expected Output: 6.8.1-u2023694
```

2. Compilation:

We compile the user-space test program using gcc, linking the pthread library for threading support.

Bash

```
gcc test_dining.c -o test_dining -lpthread
```

3. Execution:

We run the compiled executable to start the simulation.

Bash

```
./test_dining
```

4. Verification of Kernel Logs:

To verify that the system call is executing inside the kernel, we check the kernel ring buffer logs.

Bash

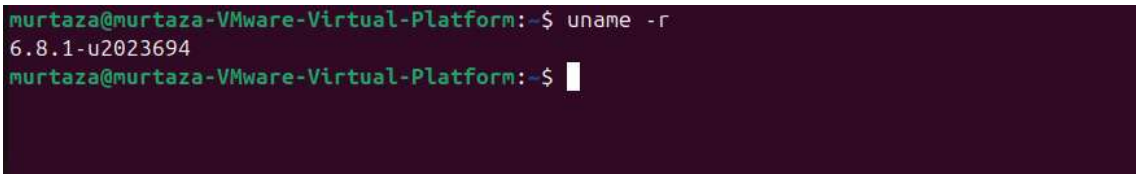
```
sudo dmesg | tail -n 20
```

---

5. Proof of Execution (Screenshots)

Figure 1: User Space Output

Description: This screenshot demonstrates that the system is running on the custom kernel (6.8.1-u2023694) and the user program is successfully issuing requests to the kernel.



Proof of User Space Execution: The image below explicitly proves the code is running in User Space (Ring 3). This is confirmed by the execution of the standard shell command (./test_dining) and the standard output messages (printf) visible in the terminal before the program interacts with the kernel.

```
murtaza@murtaza-VMware-Virtual-Platform:~$ cd ~/OS_Project_Name
gcc test_dining.c -o test_dining -lpthread
./test_dining
================================================
   DINING PHILOSOPHERS: SYSTEM CALL TEST (1 Cycle)
================================================
[Philosopher 1] is THINKING.
[Philosopher 2] is THINKING.
[Philosopher 2] is HUNGRY and calling kernel for forks...
[Philosopher 4] is THINKING.
[Philosopher 4] is HUNGRY and calling kernel for forks...
[Philosopher 0] is THINKING.
[Philosopher 0] is HUNGRY and calling kernel for forks...
[Philosopher 3] is THINKING.
[Philosopher 3] Cannot eat yet (No forks). Waiting for neighbor...
[Philosopher 1] Cannot eat yet (No forks). Waiting for neighbor...
   >>> [Philosopher 2] ACQUIRED FORKS and is EATING.
   >>> [Philosopher 0] ACQUIRED FORKS and is EATING.
[Philosopher 3] is HUNGRY and calling kernel for forks...
[Philosopher 1] is HUNGRY and calling kernel for forks...
       [Philosopher 2] FINISHED eating. Dropped forks.
--- [Philosopher 2] is FULL and leaving the table. ---
   >>> [Philosopher 4] ACQUIRED FORKS and is EATING.
       [Philosopher 0] FINISHED eating. Dropped forks.
--- [Philosopher 0] is FULL and leaving the table. ---
   >>> [Philosopher 1] ACQUIRED FORKS and is EATING.
       [Philosopher 4] FINISHED eating. Dropped forks.
--- [Philosopher 4] is FULL and leaving the table. ---
   >>> [Philosopher 3] ACQUIRED FORKS and is EATING.
       [Philosopher 1] FINISHED eating. Dropped forks.
```

Figure 2: Kernel Space Logs

Description: This screenshot shows the dmesg logs. It proves that the kernel function sys_dining_philosophers is correctly identifying and locking the specific forks (semaphores) for each philosopher (e.g., "Philosopher 1 picked up left fork (1)").

```
.firefox.firefox" name="/proc/pressure/memory" pid=2834 comm="firefox" requested_mask="r" denied_mask="r" f
d=0
[  130.628552] Dining Philosophers: Forks Initialized.
[  130.628555] Philosopher 2 is hungry.
[  130.628556] Philosopher 2 picked up left fork (2).
[  130.631205] Philosopher 4 is hungry.
[  130.631208] Philosopher 4 picked up left fork (4).
[  130.631243] Philosopher 0 is hungry.
[  130.631244] Philosopher 0 picked up left fork (0).
[  130.730596] Philosopher 2 picked up right fork (3).
[  130.730605] Philosopher 2 is EATING.
[  130.738667] Philosopher 0 picked up right fork (1).
[  130.738673] Philosopher 0 is EATING.
[  131.131603] Philosopher 3 is hungry.
[  131.131705] Philosopher 1 is hungry.
[  131.231106] Philosopher 2 put down forks and is THINKING.
[  131.231303] Philosopher 3 picked up left fork (3).
[  131.239166] Philosopher 0 put down forks and is THINKING.
[  131.239182] Philosopher 4 picked up right fork (0).
[  131.239184] Philosopher 4 is EATING.
[  131.239253] Philosopher 1 picked up left fork (1).
[  131.346792] Philosopher 1 picked up right fork (2).
[  131.346800] Philosopher 1 is EATING.
[  131.739468] Philosopher 4 put down forks and is THINKING.
[  131.739640] Philosopher 3 picked up right fork (4).
[  131.739641] Philosopher 3 is EATING.
[  131.846975] Philosopher 1 put down forks and is THINKING.
[  132.241181] Philosopher 3 put down forks and is THINKING.
[  168.092103] sched: RT throttling activated
```

GitHub Repository

Link: https://github.com/Murtaza436/OS_Dining_Philosophers.git