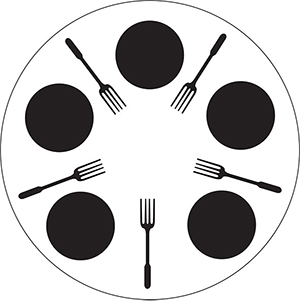
Dining Philosopher-Kernel System Call





Group Members

Avinash 17K-3892

Essam 17K-3827

Haseebullah 17k-3764

OBJECTIVE:

To implement Dining Philosopher System Call in Kernel Space to Re Allocate Resources to a Process

Introduction to Problem:

Dining Philosopher is one of the Problem which we see in Process Synchronization which states that states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

There are three states of Philosopher:

* Thinking: When Philosopher is not Eating
* Hungry: When Philosopher wants to Eat but chopstick is acquired
* Eating: When Philosopher is Eating

and a philosopher will be in either one state.

Conditions Can Occur:

* Deadlock
* Starvation

Deadlock:

 Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process

Starvation:

If a resource is never allocated to a process then the condition known as the Starvation. If a Philosopher never become able to eat the Food the Philosopher will be in Starvation

Solution of The Dining Philosopher Problem:

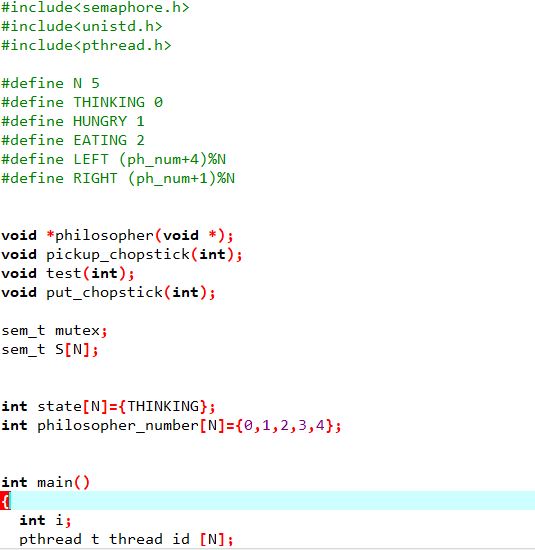
* Semaphores

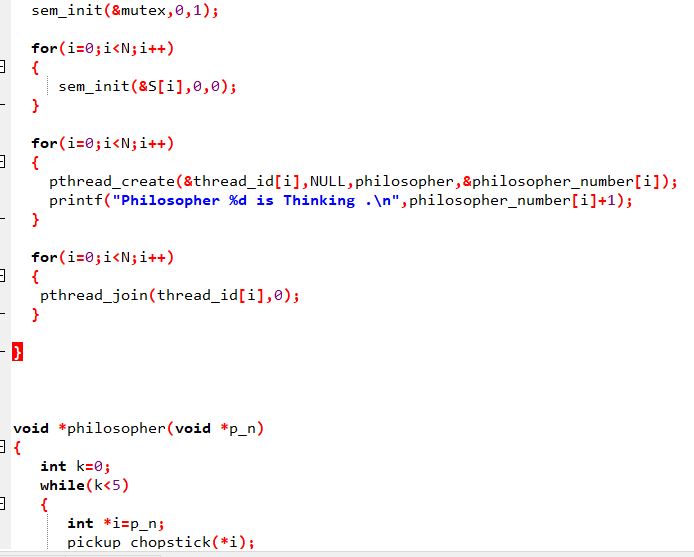
The Philosopher cannot Pick any one of the sticks if any of its neighbor is

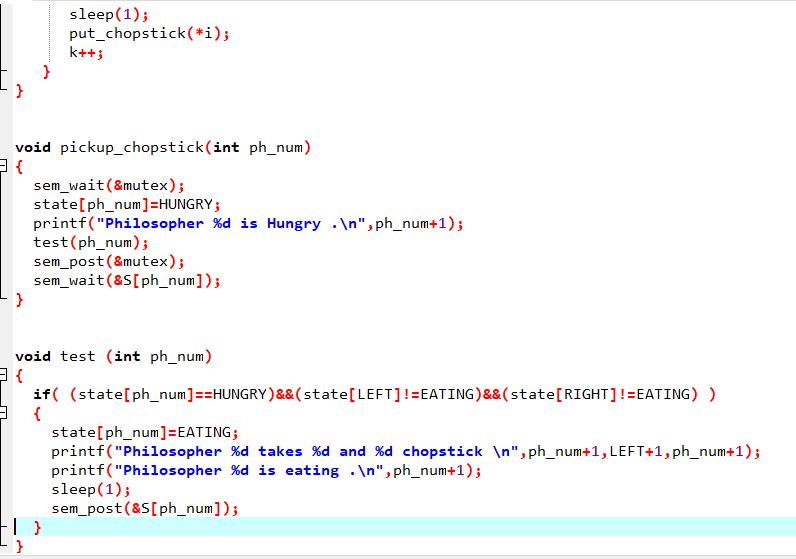
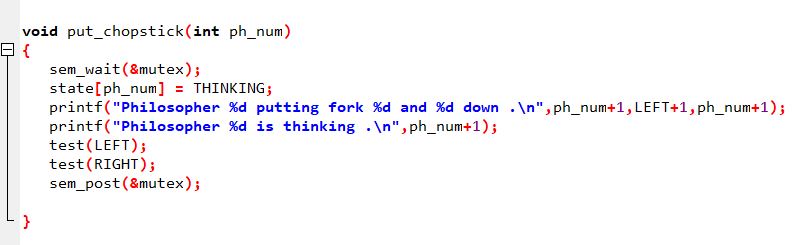
Eating. To Handle all the allocation, we use the Semaphores which ensure that which

Philosopher will be eating. We use a semaphore mutex which acquires lock and only ensure that no two philosophers can put and pick chopstick at the same time.

The Philosopher are executed as thread and as each thread wants to Eat food the semaphore must ensure that it satisfy all conditions other wise that thread will go in to waiting. So, in this way The Dining Philosopher Problem can be Handled

**User Level Code:**





**Important Functions:**

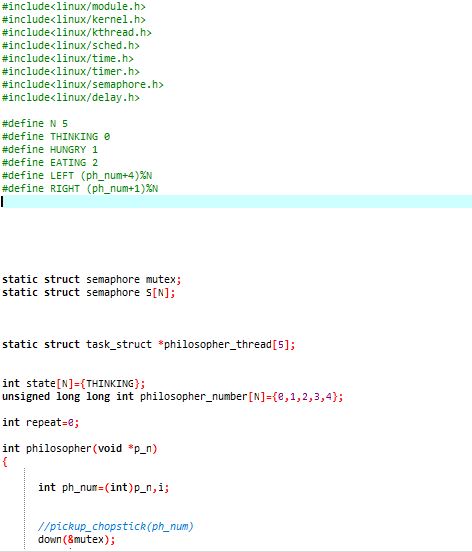
Sem\_wait(): To acquire lock

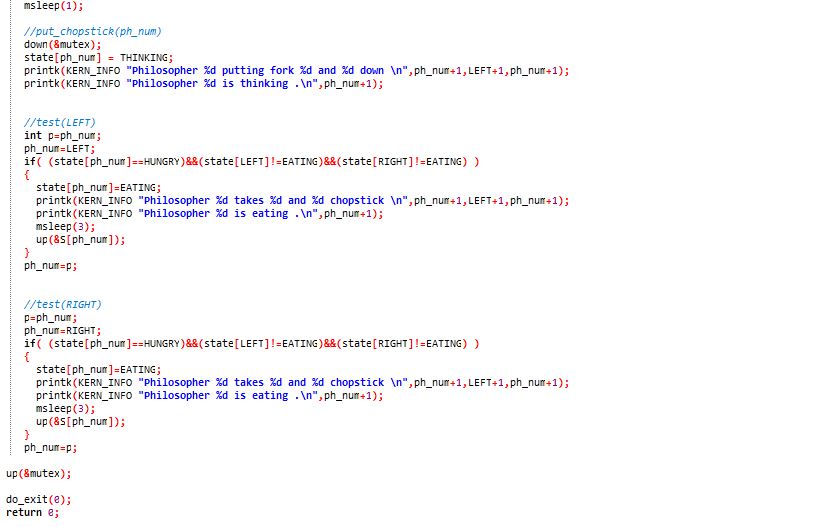
Sem\_post(): To free a lock

pthread\_create():To create a thread

pthread\_join(): To execute threads

Kernel Level Code







SSSSS

**Important Functions:**

Kthread\_create() : To create Thread

Kthread\_stop : To stop thread

Struct Seamphore : Structure of Seamphore

**Kernel Compilation:**

**After writing the kernel level code now we have to compile the system call in to kernel .**

**The following are the kernel compilation commands to compile system call in kernel**:

* sudo apt-get update
* sudo apt-get upgrade
* apt-get install gcc
* apt-get install python-pip python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev
* apt-get install libncursesw5-dev
* wget <https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.0.9.tar.gz> or You can download it manually
* cd /usr/src/linux-5.0.9
* Create a directory having name of your system call and in that directory we create c code file and paste our system call code
* mkdir dining
* cd dining
* touch dining.c
* touch Makefile
* In makefile we paste obj-y := dining.o
* After that we go to main makefile of linux and add our system call
* After that we go to syscalls.h present in /include/linux we give our system call prototype
* After that we go to arch/x86/entry/syscalls and in syscall\_64.tbl

We define our system call and index number

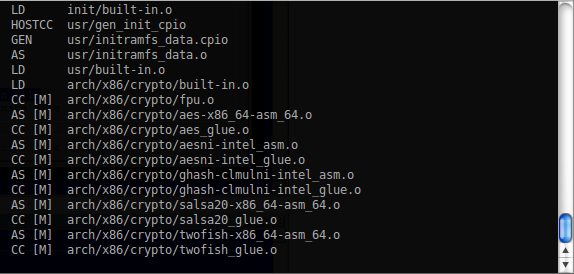
After that we run some commands

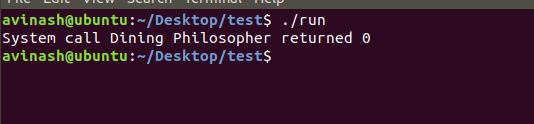
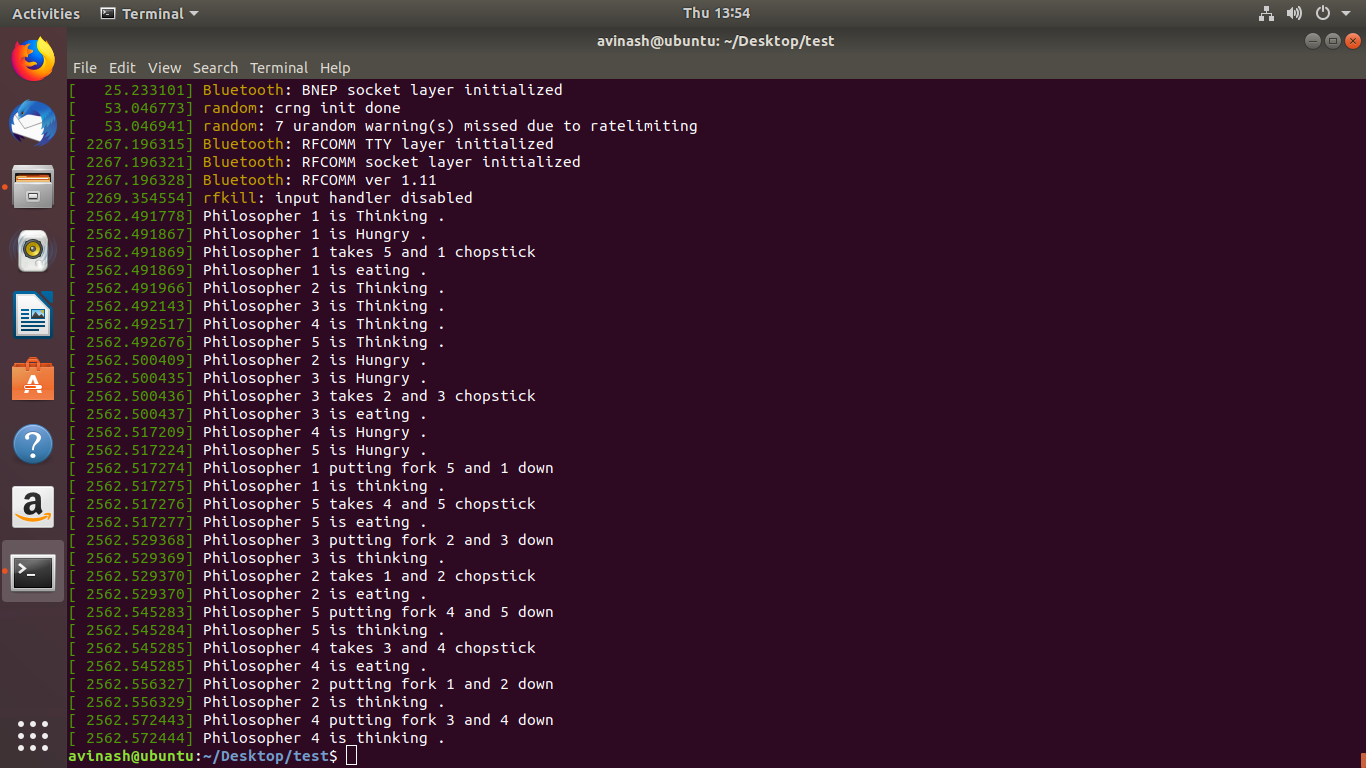
* Apt install flex && apt install bison
* Make menuconfigs
* Make oldconfig
* Make -j4

After Compilation …

make modules\_install install

* After that we using syscall(index\_number) function we call our system call



**Output:**

**Conclusion:**

**As the Dining Philosopher call has been implemented and now it it will be easier for kernel to allocate the resources to the process and also there will be no condition of starvation and deadlock.**