**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**OPERATING SYSTEM LAB REPORT #10**

**SUBMITTED BY:**

Pradeep Dahal

017BSCIT029

2nd year/ 4th sem

|  |  |
| --- | --- |
|  | Signature |
| Mr. Rabin Maharjan  (Lecturer) |  |
| Department of Computer Science | |

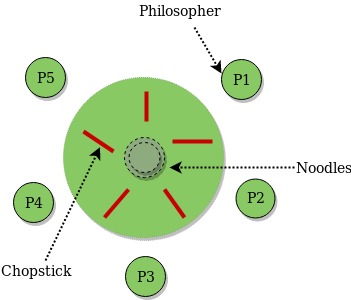
**SUBMITTED TO:**

# Dining Philosopher’s Problem

The dining philosopher’s problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

## Problem Statement

The Dining Philosopher Problem 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.



## SOURCE CODE

#include <unistd.h>

#include<stdio.h>

#include<pthread.h>

pthread\_t philosopher[5]; //defining threads

pthread\_mutex\_t chopstick[5]; //declaring mutex

void \*process(int n)

## { printf("\n philosopher %d Thinking!!",n);

pthread\_mutex\_lock(&chopstick[n-1]);

pthread\_mutex\_lock(&chopstick[(n)%5]);

printf("\n philosopher %d Eating!!-----chopstick[%d]-----chopstick[%d]",n,(n-1),(n%5));

sleep(2);

pthread\_mutex\_unlock(&chopstick[n-1]);

pthread\_mutex\_unlock(&chopstick[(n)%5]);

printf("\n philosopher %d Done!!",n); }

int main()

{ int i; //create a mutex

for(i=0;i<5;i++)

{pthread\_mutex\_init(&chopstick[i],NULL);}

for(i=1;i<=5;i++) //create a thread

{pthread\_create(&philosopher[i],NULL,(void \*)process,(int \*)i );}

for(i=0;i<5;i++)

{pthread\_join(philosopher[i],NULL);}

return 0; }

## OUTPUT

