Project 2: Dinning Philosophers’ problem  
Due Date: April 29, 2002  
(2 person project)

Objective
To have hand-on experiences on inter-process communication and synchronization mechanisms.

Description
A dinning philosophers’ problem is a classic synchronization problem. Though, it does not notably represent a real-world problem, it provides a significant learning value, particularly in process synchronization. It is defined in our textbook as follows:

There are 5 philosophers who spend their time just thinking and eating. They sit at the table and each has his/her own chair. There is a rice bowl in the center of the table and there are 5 chopsticks which of each is laid next to the philosopher’s hand as shown below. When a philosopher thinks, he will not interact with others. Once in a while, a philosopher is hungry and tries to pick up chopsticks on his left and right-hand sides. A philosopher can pick only one chopstick at a time. When a hungry philosopher has chopsticks on both hands, he can start eating. When he is done eating, he puts both chopsticks down and starts thinking again.

![Diagram of philosophers at table](image)

*Note: This picture is originally from Silberchatz’s book or instructor’s material.*

Implement the above problem (5 philosophers) by creating 5 processes and using semaphore for synchronization.

However, care must be taken to prevent a deadlock problem. One possible solution to alleviate the deadlock is known as “an asymmetric solution”, that is, an odd philosopher picks up first a left chopstick and then the right one, while an even philosopher picks up first a right chopstick and then the left one.

Your program should take a number of philosophers’ eating’s from the command line.

```
% dphil 10 // each philosopher will eat 10 times before existing.
```
**Bonus (5%)**

Implement the solution to take the “n” number of philosophers.
```
% dphil 10 7 // each philosopher will eat 10 times before existing. There are 7 philosophers.
```

**What to Hand in**

Submit a tar file using the following command
```
%tar cvf p2.tar README typescript your_codes *.c *.C *.h Makefile
```

To extract you can use “tar xvf p2.tar”

1. A README file with:
   1. Your name and your partner's name
   2. details of your logic/functions
2. All the source files needed to compile, run and test your code (Makefile, .c or c++ files, optional test scripts). Do not submit object or executable files.
3. Output from your testing of your program. Make sure to demonstrate all activities of philosophers (eating, thinking)

Philosopher 0 is thinking...
Philosopher 1 is eating...
Philosopher 3 is thinking...
Philosopher 4 is thinking...

: 

**Suggestions:**

**Useful Unix System Calls or convenient functions**

fork-join: create a new child process and wait for it to exit:

```
A random number generator srand() for simulating eating and thinking time.
```

```
// declare timeval tp for random number seeding
struct timeval tp;

// set things up by seeding with the time of day
getimeofday(&tp,NULL);
srand(tp.tv_sec);
```

```
// in your eating code
cout << "Philosopher : " << I <<" is eating..." << endl;
sleep(rand() % 10); // generate random number from 1-10
```

**A set of semaphore functions from Steven’s book (sample.tar)**
```
int chopstick[5]; // array of sem_id
for (I =0; i < 5 < i++) {
    chopstick[i] = sem_create(BASEKEY + I, 1);
}