HW1 : SUBHAJIT SAHU : 2018801013

THE DINING PHILOSOPHERS PROBLEM

There are 5 philosophers sitting around a table. Each philosopher thinks, or when hungry eats. Each philosopher has a bowl of noodles, but there are only 5 forks between each philosopher. However, in order to eat, a philosopher needs both forks. A deadlock occurs when all philosophers pick up one fork, but are waiting for the other fork to be put down.

One idea to solve the dependency cycle is to alter the order in which the last philosopher picks up forks.

CODE: https://repl.it/@wolfram77/dining-philosophers-problem#Philosopher.java

```
import java.util.concurrent.*;


class Philosopher extends Thread {
int i;
static int N = 5;
static Semaphore[] fork;
static Philosopher[] philosopher;


Philosopher(int id) {
i = id;
}


public void run() {
try {
while(true) {
System.out.println(i+": thinking");
Thread.sleep(100);
int j = (i+1) % N;
int p = Math.min(i, j);
int q = Math.max(i, j);
fork[p].acquire();
fork[q].acquire();
System.out.println(i+": eating");
Thread.sleep(100);
fork[q].release();
fork[p].release();
}
```

```
}
catch(InterruptedException e) {
e.printStackTrace();
}
}


public static void main(String[] args) {
fork = new Semaphore[N];
philosopher = new Philosopher[N];
for(int i=0; i<N; i++) {
fork[i] = new Semaphore(1);
philosopher[i] = new Philosopher(i);
}
for(int i=0; i<N; i++) {
System.out.println(i+": started");
philosopher[i].start();
}
}
}
```

OUTPUT: https://dining-philosophers-problem.wolfram77.repl.run

```
0: started
1: started
2: started
1: thinking
0: thinking
3: started
2: thinking
4: started
3: thinking
4: thinking
1: eating
3: eating
1: thinking
0: eating
3: thinking
2: eating
0: thinking
4: eating
2: thinking
1: eating
4: thinking
...
```