National University of Computer & Emerging Sciences

Department of Computer Science

Operating Systems Lab

Lab # 11

Instructions:

1. Make a word document with the convention “SECTION_ROLLNO _LAB-NO”. In addition, paste all of your work done at the LINUX prompt.
2. You have to submit a Word File.
3. Plagiarism is strictly prohibited; negative marks would be given to students who cheat.

Task 1

Consider a situation having three Coffee loving persons as threads and one agent thread.
Each person is required to drink coffee but to do that; the person thread needs three ingredients: coffee, water, and milk. Each thread has infinite supply of one specific element, e.g. the first one has infinite supply of coffee, the second one has infinite supply of water and the last one has infinite supply of milk. Similarly, the agent has infinite supply of all ingredients and it chooses two out of three ingredients at random repeatedly. Afterwards, agent puts selected ingredients on table and the person having deficient ingredients can take the two ingredients and make coffee to drink.
For example, if agent selects water and milk randomly then the only thread who can drink is person thread 1. The agent then puts out another two of the three ingredients, and the cycle repeats. Above mentioned behavior is required to be simulated using monitors only. Also use meaningful variable and functions names.

Task 2

Banker's Algorithm

Description

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

Pseudo Code

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R}_j$

Max :

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P}_i$ may request at most **'k'** instances of resource type $\mathbf{R}_j$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P}_i$ is currently allocated **'k'** instances of resource type $\mathbf{R}_j$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P}_i$ currently need **'k'** instances of resource type $\mathbf{R}_j$ for its execution.

- Need [i, j] = Max [i, j] – Allocation [i, j]

Allocation specifies the resources currently allocated to process P_i and Need$_i$ specifies the additional resources that process P_i may still request to complete its task. Banker's algorithm consists of Safety algorithm and Resource request algorithm.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

```
1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) if Finish [i] = true for all i
then the system is in a safe state
```

Lab Exercise

1. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance using the details given in Lecture.