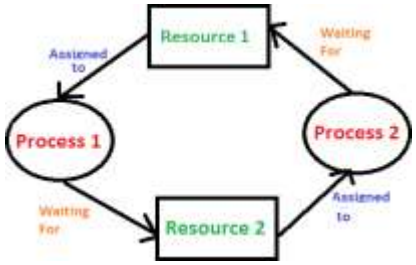
CS-362L Operating System Lab 9

Banker's Algorithms

Objectives: To understand banker's algorithm and implement it.

What is deadlock?

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process. As in the diagram below, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.



Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm. It is used for cases where we have multiple instances of a single resource type. When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a **safe state**. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Safe State: A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.

Safe Sequence: A system is in a safe state only if there exists a safe sequence. The sequences in which, when the processes will execute, will not leave the system in deadlock.

Safety Algorithm: The safety algorithm is used for finding out whether or not a system is in a safe state.

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

Processing steps:

Step 1: Input the number of resources, say **"m"**.

Step 2: Create an array to store the number of instances of each resource type. Call it **totalResources**.

Step 3: Input the number of processes, say **"n"**.

Step 4: Use 2D array to get allocated resources for each process. Here rows represents number of processes [n] and columns represents number of resources [m].
Step 5: Get Max need of process using 2D array.

Step 6: Calculate the need of each process. The formula to calculate need is Need [i,j] = Max[i,j] – Allocation [i,j]

Step 7: Next calculate available resources for each process. To calculate available resources, follow the steps below:

1. Generate the sum of values in the column for each each allocated resource.
2. Create 2D array equal to the size of **"m"**(number of resources).
3. Calculate available resources using formula: available = totalResources - sum of allocated resources. Store it in array called **available**.

Step 7: Create a 2D array called **"Work"**. Initialize this as *Work == Available*. Don't forget to create array called **Finish** to keep track of status of processes.

Step 8: Use boolean function called **comparefunc** to compare the condition $Need_i \leq Work$

1. Update work array: work = work + allocation
2. Update finish for that particular process.

Use a loop to continue again the checking all processes once the array of processes end

Step 9: Print safe sequence and a table containing your values.

Conclusion: At the end of this lab, student will be able to implement banker's algorithms.

LAB TASK

1. Implement Banker's Algorithm to determine the safe state for a given set of processes in C/C++.

Note: You need to prepare a word document containing your code along with the screenshot of output for each program.