

Operating Systems (CSE316)

Case Study

Santosh Padhi

11706018

Section: K1640, Roll No.: 37

santoshpadhi1616@gmail.com

[OS-CASESTUDY-B37](https://github.com/santosh0798/OS-CASESTUDY-B37)

(<https://github.com/santosh0798/OS-CASESTUDY-B37>)

Question:-

Q3. Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

The Problem: To write a multi-threaded program to implement the banker's algorithm such that the created threads request and release resources to the banker and the banker grants a request only if the system stays in a safe state. A safe state is a state of the system in which deadlock is not possible.

Algorithm:-

getSafeSequence(PROCESSES,AVAILABLE,REQUIRED,ALLOCATED)

//This simulates the current state of the system to find a safe sequence and returns an array.

	Complexity
isPseudoSafe = 0	
sequence = [] // Empty array	1
finished = []	1
available = AVAILABLE.copy() // Copy the array	1
remaining = PROCESSES.length() // n number of processes	1
While remaining>0 loop,	n
isPseudoSafe = 0	1
For p = 0 to PROCESSES.length() loop,	n
If finished[p]==1 then, continue; //Process finished in simulation	1
For r = 0 to available.length() loop, // r number of resources	r
If REQUIRED[p][r]==available[r] then, break;	1
[End for loop]	
If r== available.length() then,	1
// This process can allocate all required resources, finish it.	
For r = 0 to available.length() loop, // r number of resources	r
available[r] = available[r]+ALLOCATED[p][r]	1
[End for loop]	
remaining = remaining-1	1
sequence = addelement(sequence,p)	1
isPseudoSafe = 1	1
[End if]	
[End for loop]	
If isPseudoSafe==0 then, break;	1
[End while loop]	
If remaining>0 then, return []; // Return empty array as no safe sequence.	1
return sequence;	1

Total complexity = $4 + n(1 + n(1 + r + 1 + r + 3) + 1)$

$= n^2(2r + 5) + 2n + 4$

$= O(n^2 * r)$, where n: number of processes, r: number of resources

requestResources(processID,resourceID,count,PROCESSES,AVAILABLE,REQUIRED,AL
LOCATED)

//This returns true if the request from 'processID' for 'count' no. of 'resourceID' is granted.

	Complexity
ALLOCATED [processID][resourceID]+=count	1
AVAILABLE [resourceID] -= count	1
REQUIRED [processID][resourceID]-=count	1
sequence = getSafeSequence(PROCESSES,AVAILABLE,REQUIRED,ALLOCATED)	$O(n^2 * r)$
If sequence.length()>0 then, return true;	1
Else,	
ALLOCATED [processID][resourceID]-=count	1
AVAILABLE [resourceID] += count	1
REQUIRED [processID][resourceID]+=count	1
[End else]	

Total Complexity = $3 + (n^2 * r) + 3$

= $O(n^2 * r)$, where n: number of processes, r: number of resources

Constraints:-

The requested resources must be available, i.e. $AVAILABLE[resourceID] \geq count$

If(resourceCount>banker->availableResourcesArray[resourceIndex])returnCode = -5;

The requested resources must be less than the max requirement specified by the process, i.e.

if(resourceCount+banker->resourcesAllocatedMatrix[processIndex][resourceIndex]>(banker->resourcesDemandMatrix[processIndex][resourceIndex]) || resourceCount<0)returnCode = -3;

$count + ALLOCATED[processID][resourceID]$
 $\leq MAXDEMAND[processID][resourceID]$

```

if(returnCode<1)
{
    // Unlock the mutex lock to allow other threads to allocate resources.
    pthread_mutex_unlock(&(banker->concurrencyLock));
    return returnCode;
}

```

Purpose of use: Banker's algorithm is used as a deadlock avoidance algorithm to prevent the condition of deadlock in a system with concurrent resource access.

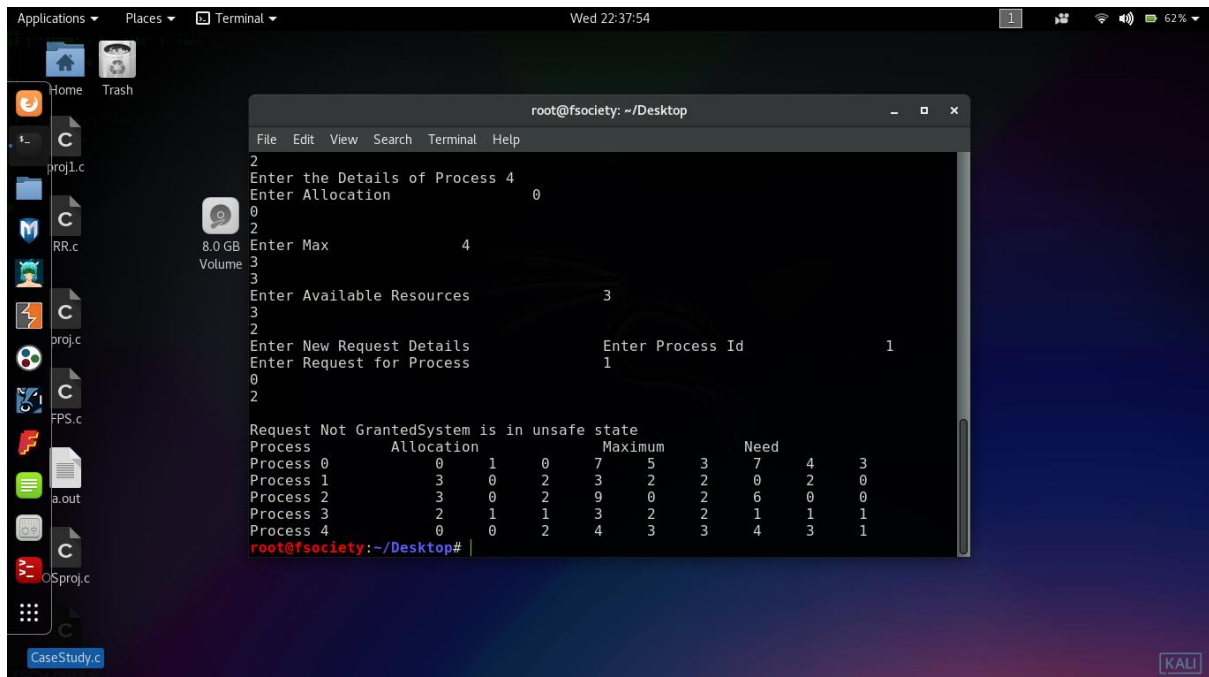
Working of program: The project program uses multiple threads to request and release resources to the banker. Each thread runs a [loop](#) and in each iteration/tick of the loop, it either requests or releases a certain resource. The resource to request/free, its count and the iteration to do so in is all stored in a [threadTickBehaviour](#) structure which is passed to the thread while its creation(see [thread creation](#)).

Additional Code: The project contains additional code for some features required for the demonstration of the Banker's algorithm. It consists of a custom made UI library that helps present the information in a neat and presentable manner. It also contains 'systemRunner.c' containing code that creates and manages the multiple threads used for concurrently requesting resources from the banker. Finally it contains 'systemContextHelper.c' which is used to initialize and manage all the data structures used in the program. The entry function 'int main()' can be found in the 'main.c' file.

Screenshots:-

Main Screen

Thread Behaviour Screen



```
root@fsociety: ~/Desktop
File Edit View Search Terminal Help
2
Enter the Details of Process 4
Enter Allocation 0
0
2
Enter Max 4
3
3
Enter Available Resources 3
3
2
Enter New Request Details Enter Process Id 1
Enter Request for Process 1
0
2
Request Not GrantedSystem is in unsafe state
Process Allocation Maximum Need
Process 0 0 1 0 7 5 3 7 4 3
Process 1 3 0 2 3 2 2 0 2 0
Process 2 3 0 2 9 0 2 6 0 0
Process 3 2 1 1 3 2 2 1 1 1
Process 4 0 0 2 4 3 3 4 3 1
root@fsociety:~/Desktop#
```

Run Simulation Screen