MIT World Peace University

Operating Systems
Second Year B. Tech, Semester 3

Simulation of Bankers Algorithm using C

Assignment 4
Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 9, 2022

1 Code

```c
#include <stdio.h>

// Bankers Algorithm
int allocation[5][3] = {
    {0, 1, 0},
    {3, 0, 2},
    {3, 0, 2},
    {2, 1, 1},
    {0, 0, 2}};
int max[5][3] = {
    {7, 5, 3},
    {3, 2, 2},
    {9, 0, 2},
    {2, 2, 2},
    {4, 3, 3}};
int available[5][3] = {
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0}};
int need[5][3] = {
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0},
    {0, 0, 0}};

int work[3] = {0, 0, 0};
int max_resources[3] = {10, 5, 7};
int finish[5] = {0, 0, 0, 0, 0};
void calc_work()
{
    int temp = 0;
    for (int j = 0; j < 3; j++)
    {
        temp = 0;
        for (int i = 0; i < 5; i++)
        {
            temp += allocation[i][j];
        }
        work[j] = max_resources[j] - temp;
    }
}

void calc_need()
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            need[j][i] += max[j][i] - allocation[j][i];
        }
    }
}

```

```c
// Displays a variable length 2 Dimensional Matrix
void display_mat(int *matrix, int rows, int cols)
{
    printf("\n");
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("%d ", matrix[i * cols + j]);
        }
        printf("\n");
    }
    printf("\n");
}

int checkFinish()
{
    for (int i = 0; i < 5; i++)
    {
        if (finish[i] < 1)
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int t = 0;
    int request[3] = {0, 0, 0};
    calc_work();
    calc_need();
    printf("At what time instant do you want some random process to need some resources? \n");
    scanf("%d", &t);
    for (int i = 0; i < 3; i++)
    {
        scanf("%d", &request[i]);
    }

    printf("The need matrix is: \n");
    display_mat(&need[0][0], 5, 3);
    printf("\nThe Safe State is: \n");
    int block = 0;
    do
    {
        for (int j = 0; j < 5; j++)
        {
            if (t == j + 1 && block != 1)
            {
                if (request[0] <= work[0] && request[1] <= work[1] && request[2] <= work[2])
                {
                    work[0] -= request[0];
                    work[1] -= request[1];
                    work[2] -= request[2];
                    block = 1;
                }
            }
            if (!finish[j])
            {
                if (need[j][0] <= work[0] && need[j][1] <= work[1] && need[j][2] <= work[2])
                {
                    finish[j] = 1;
                    printf(" %d ", j);
                    work[0] += allocation[j][0];
                    work[1] += allocation[j][1];
                    work[2] += allocation[j][2];
                }
            }
        }
    } while (!checkFinish());

    return 0;
}
```

Listing 1: Assignment 4.Cpp

2 Input and Output

```
At what time instant do you want some random process to need some resources?
1
1 0 2
The need matrix is:

7 4 3
0 2 0
6 0 0
0 1 1
4 3 1


The Safe State is:
 1  3  4  2  0
```

Listing 2: Input and Output.Cpp

OS -FAQ- Assignment - 4

8|11|2022

(*) Banker's Algorithm

Krishnaraj P.T.
1032210088
PA20 - A1

Q.1 What is meant by deadlock? What are the necessary conditions for a deadlock situation?

→ A situation where a finite number of resources are to be distributed among a number of competing processes.

→ A. set of blocked processes each holding ~~a~~ resources and waiting to acquire a resource held by another process in the set.

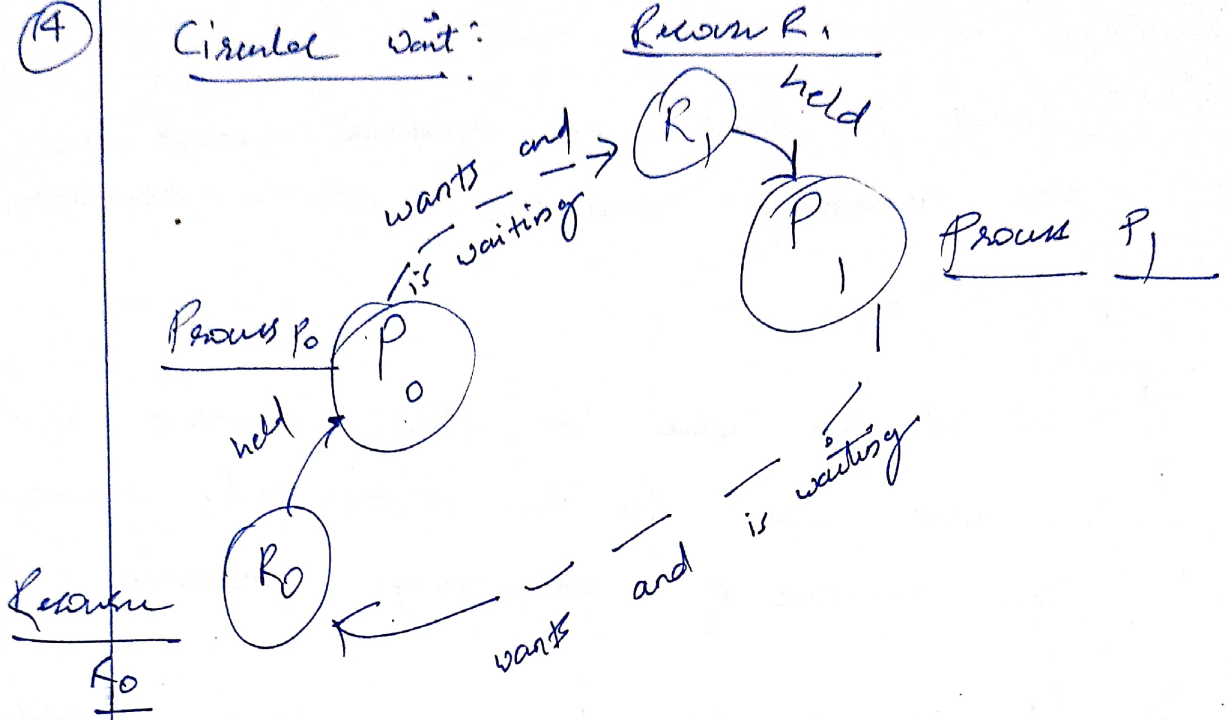
→ It can happen if and only if these conditions occur simultaineously.

1. Mutual Exclusion: Resource must be held in a non-sharable mode

2. Hold and wait: a process holding atleast one resource is waiting to ~~aq~~ acquire additional resources held by other processes.

(3) No Preemption: a resource can be released only voluntarily by the process holding it; only after completing its task.

(14) Circular wait:



P_0 waits for P_1 to finish

P_1 waits for P_0.

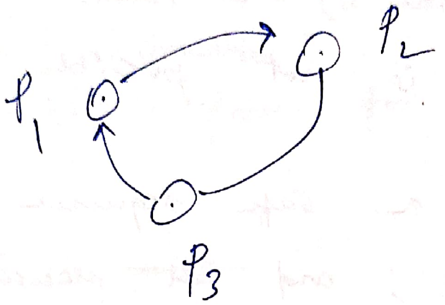
So no on finishes.

(Q.2) Deadlock vs/ Starvation

Deadlock	Starvation
→ Resources waiting for each other to release critical resources; so none get executed	→ High priority processes keep ~~waiting~~ executing; but low priority ones keep getting blocked / starved.
→ Resources are blocked not used	→ Resources are used by high priority processes.

→ ~~B~~ Need hold & wait, circular wait, mutual exclusion, No preemption

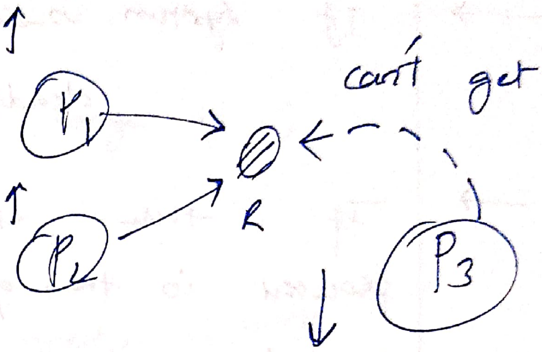
→ solved by eliminating/ avoiding the problem that causes it.



Need different priorities assigned to processes.

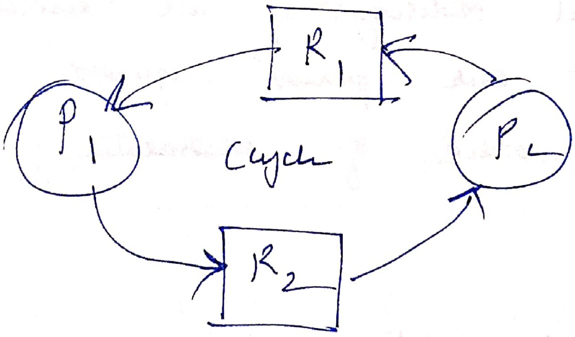
↑ priority to long ones

→ solved by aging or introducing some time quantum.



Q.3 Explain difference between Deadlock detection, avoidance & prevention.

① Deadlock Detection :



Resource allocation graph

→ write an algorithm & check for presence of cycle in the Resource allocation graph.

Execute it periodically to get if deadlock occured

→

```
if ( cycle_is_graph ( )) return deadlock
else return no_deadlock ;
```

② Deadlock Avoidance :

→ Method to ~~oo~~ prevent deadlock.

→ Requires system to have some prior information

→ Processes can declare the max ~~the~~ resources they need.

→ uses concept of safe state.

→ If system.is_safe_state() == True:

deadlock is not possible.

→ If there exists a safe sequence for all process in the queue ; and all process will eventually get the resources they want ~~?~~ , Such a state is called as a safe state

③ Deadlock Prevention

→ ensure required condition to cause deadlock are never met.

→ Impose a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration ; to avoid

circular ~~~~ wait

→ If a resource is holding some resources & requests another resource that cannot be immediately allocated to it ; then all resources currently being held are released ; to avoid No Preemption

→ to avoid hold and wait, require process to request and be allocated all its resources before it begins execution.

★ Q.4 What is a Safety Algorithm ?

→ Algorithm to detect if the system is in safe mode

1. → work = available ;

finish [i] = false for i in range n ;

2 → Find i ~~such that~~ so

Finish [i] = false && $need_i \le work$

if i not found ; skip to step 4

3 . finish [i] = true

goto 2

4 if finish [i] == true for i in range n ;

System in safe mode ✓

else : System not in safe mode. X