Tribhuvan University
Faculty of Humanities and Social Sciences

Lab report on:
Operating System Lab 6:
Deadlock & Fixed partition

Submitted to:
Mr. Roshan Maharjan,
Er. Himal Chand Thapa ,
Department of Computer Application,
Himalaya College of Engineering,
Chyasal,Lalitpur

Submitted by:
Sujal Gurung
Roll no: 34
BCA II/IV
September 21, 2023

1 Objectives

- implement deadlock detection program using Banker's algorithm
- understand fixed partitioning with a demo program

2 Introduction

Deadlock is an undesirable situation where multiple processes are endlessly stuck waiting on the other to release resources. As all processes are waiting on one another, none of them finish execution, which can disrupt normal system usage.

Paused processes need to be loaded from disk to main memory. One way to do this is to divide memory into multiple partitions whose size can't be changed. Process is loaded into a partition if partition is large enough to hold it. A partition can only hold one process so remaining area of parition isn't utliized. This is called internal fragmentation.

3 Lab Work

3.1 Write a program to implement deadlock detection using Banker's algorithm

```c
#include <stdio.h>
int main() {
    int n, m, i, j, k;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);
    int alloc[n][m], max[n][m], need[n][m], avail[m], finish[n], safe_seq[n],
    ↪  count = 0;

    // Input allocation matrix
    printf("Enter the allocation matrix: \n");
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
    // Input maximum matrix
    printf("Enter the maximum matrix: \n");
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    // Calculate need matrix
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
```

```
    // Input available matrix
    printf("Enter the available matrix: \n");
    for(i=0; i<m; i++) {
        scanf("%d", &avail[i]);
    }

    // Initialize finish array as false for each process
    for(i=0; i<n; i++) {
        finish[i] = 0;
    }

    // Banker's Algorithm
    for(k=0; k<n; k++) {
        for(i=0; i<n; i++) {
            if(finish[i] == 0) {
                int flag = 1;
                for(j=0; j<m; j++) {
                    if(need[i][j] > avail[j]) {
                        flag = 0;
                        break;
                    }
                }
                if(flag) {
                    for(j=0; j<m; j++) {
                        avail[j] += alloc[i][j];
                    }
                    finish[i] = 1;
                    safe_seq[count++] = i;
                }
            }
        }
    }
    // Check for deadlock
    int deadlock = 0;
    for(i=0; i<n; i++) {
        if(finish[i] == 0) {
            printf("Process %d is deadlocked.\n", i);
            deadlock = 1;
        }
    }
    if(!deadlock) {
        printf("System is in safe state.\n");
        printf("Safe sequence is: ");
        for(i=0; i<n; i++) {
            printf("%d ", safe_seq[i]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
0 1 0
```

```
2 0 0
3 0 2
2 1 1
0 0 2
Enter the maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the available matrix: 3 3 2
System is in safe state.
Safe sequence is: 1 3 4 0 2
```

3.2 Write a program to demonstrate basic fixed size partitioning using 1st fit algorithm

Here, we divide available memory into equal sized partitions.

```c
#include<stdio.h>
void main() {
    int ms, bs, nob, ef,n, mp[10],tif=0; int i,p=0;

    printf("Enter the total memory available (in Bytes) -- ");
    scanf("%d",&ms);
    printf("Enter the block size (in Bytes) -- ");
    scanf("%d", &bs);
    nob=ms/bs;
    ef=ms - nob*bs;
    printf("\nEnter the number of processes -- ");
    scanf("%d",&n);
    for(i=0;i<n;i++) {
        printf("Enter memory required for process %d (in Bytes)-- ",i+1);
        scanf("%d",&mp[i]);
    }
    printf("\nNo. of Blocks available in memory--%d",nob);
    printf("\n\n PROCESS\t MEMORYREQUIRED \t ALLOCATED \t INTERNAL
    ↪  FRAGMENTATION");
    for(i=0;i<n && p<nob;i++) {
        printf("\n %d\t\t%d",i+1,mp[i]);
        if(mp[i] > bs)
            printf("\t\tNO\t\t---");
        else {
            printf("\t\tYES\t%d",bs-mp[i]);
            tif = tif + bs-mp[i];
            p++;
        }
    }
    if(i<n) printf("\nMemory is Full, Remaining Processes cannot be
    ↪  accomodated");
    printf("\n\nTotal Internal Fragmentation is %d",tif);
    printf("\nTotal External Fragmentation is %d",ef);
}
```

Output:

```
Enter the total memory available (in Bytes) -- 1500
Enter the block size (in Bytes) -- 500

Enter the number of processes -- 4
Enter memory required for process 1 (in Bytes)-- 212
Enter memory required for process 2 (in Bytes)-- 417
Enter memory required for process 3 (in Bytes)-- 112
Enter memory required for process 4 (in Bytes)-- 426

No. of Blocks available in memory--3

 PROCESS         MEMORYREQUIRED  ALLOCATED   INTERNAL FRAGMENTATION
 1               212             YES         288
 2               417             YES         83
 3               112             YES         388
Memory is Full, Remaining Processes cannot be accomodated

Total Internal Fragmentation is 759
Total External Fragmentation is 0%
```

4 Conclusion

Thus, we learned how deadlock might be detected in a system & how it can be avoided by choosing the right sequence of processes to run. We also understood a basic version of fixed paritioning in memory by implementing its program.