Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name: Apurba Koirala

Reg no: 22BCE3799

Subject Code: BCSE303P

Course Title: Operating Systems Lab

Lab Slot: L39 + L40

Guided by: Dr. Anto S

Lab Assessment 2

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Exercise 3 – Deadlock, Synchronization and Memory Allocation

Question 1

(a) Write a C program to implement deadlock avoidance using Banker's Algorithm with the below mentioned details.

(b) What is the objective of deadlock detection? Implement the deadlock detection using any one of the method which is meant for the same.

	Allocation	Max	Available
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

AIM:

To explain the objective of deadlock detection and implement the deadlock detection method.

ALGORITHM:

The objective of deadlock detection is to identify if the system is in a deadlocked state where processes are waiting indefinitely for resources. The system should detect this state to prevent it from continuing indefinitely, allowing for corrective measures to be taken.

Input: Initialize Available vector (Work), Allocation matrix, Request matrix, n processes, m resources.

Initialize: Work = Available; Finish[i] = false for all processes.

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40
Mark: If a process has no allocated resources, set Finish[i] = true.

Find: Look for a process i with Finish[i] == false and Request ≤ Work.

Simulate: If found, add Allocation of i to Work; set Finish[i] = true; repeat step 4.

Check: If all Finish[i] are true, no deadlock; if any false, deadlock exists.

Output: Report safe state or identify deadlocked processes.

SOURCE CODE :

```
#include <stdio.h>

#include <stdbool.h>


#define MAX_PROCESS 10

#define MAX_RESOURCES 10


void calculateNeed(int need[MAX_PROCESS][MAX_RESOURCES], int
max[MAX_PROCESS][MAX_RESOURCES], int
alloc[MAX_PROCESS][MAX_RESOURCES], int np, int nr) {

    for (int i = 0; i < np; i++)

        for (int j = 0; j < nr; j++)

            need[i][j] = max[i][j] - alloc[i][j];

}


bool isSafe(int processes[], int avail[], int max[MAX_PROCESS][MAX_RESOURCES], int
alloc[MAX_PROCESS][MAX_RESOURCES], int np, int nr) {

    int need[MAX_PROCESS][MAX_RESOURCES];

    calculateNeed(need, max, alloc, np, nr);
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    bool finish[MAX_PROCESS] = { false };

    int safeSeq[MAX_PROCESS], work[MAX_RESOURCES];


    for (int i = 0; i < nr; i++) work[i] = avail[i];


    int count = 0;

    while (count < np) {

        bool found = false;

        for (int p = 0; p < np; p++) {

            if (!finish[p]) {

                int j;

                for (j = 0; j < nr; j++) {

                    if (need[p][j] > work[j]) break;

                }

                if (j == nr) {

                    for (int k = 0; k < nr; k++)

                        work[k] += alloc[p][k];

                    safeSeq[count++] = p;

                    finish[p] = true;

                    found = true;

                }

            }

        }

        if (!found) {

            printf("System is not in a safe state.\n");

            return false;
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
        }

    }


    printf("System is in a safe state.\nSafe sequence is: ");

    for (int i = 0; i < np; i++) {

        printf("%d ", safeSeq[i]);

    }

    printf("\n");

    return true;

}


void requestResources(int processes[], int avail[], int
max[MAX_PROCESS][MAX_RESOURCES], int
alloc[MAX_PROCESS][MAX_RESOURCES], int np, int nr) {

    int process_num, request[MAX_RESOURCES];

    printf("Enter the process number making the request (0-%d): ", np - 1);

    scanf("%d", &process_num);


    printf("Enter the requested resources: ");

    for (int i = 0; i < nr; i++) {

        scanf("%d", &request[i]);

    }


    for (int i = 0; i < nr; i++) {

        if (request[i] > avail[i]) {

            printf("Requested resources exceed available resources. Request cannot be
granted.\n");
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
            return;

        }

    }


    for (int i = 0; i < nr; i++) {

        if (request[i] > max[process_num][i] - alloc[process_num][i]) {

            printf("Requested resources exceed maximum claim for process. Request cannot be
grant-ed.\n");

            return;

        }

    }


    for (int i = 0; i < nr; i++) {

        avail[i] -= request[i];

        alloc[process_num][i] += request[i];

        max[process_num][i] -= request[i];

    }


    if (isSafe(processes, avail, max, alloc, np, nr)) {

        printf("Request can be granted. Resources have been allocated.\n");

    } else {

        printf("Request cannot be granted as it leads to an unsafe state. Rolling back the
request.\n");

        for (int i = 0; i < nr; i++) {

            avail[i] += request[i];

            alloc[process_num][i] -= request[i];

            max[process_num][i] += request[i];
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

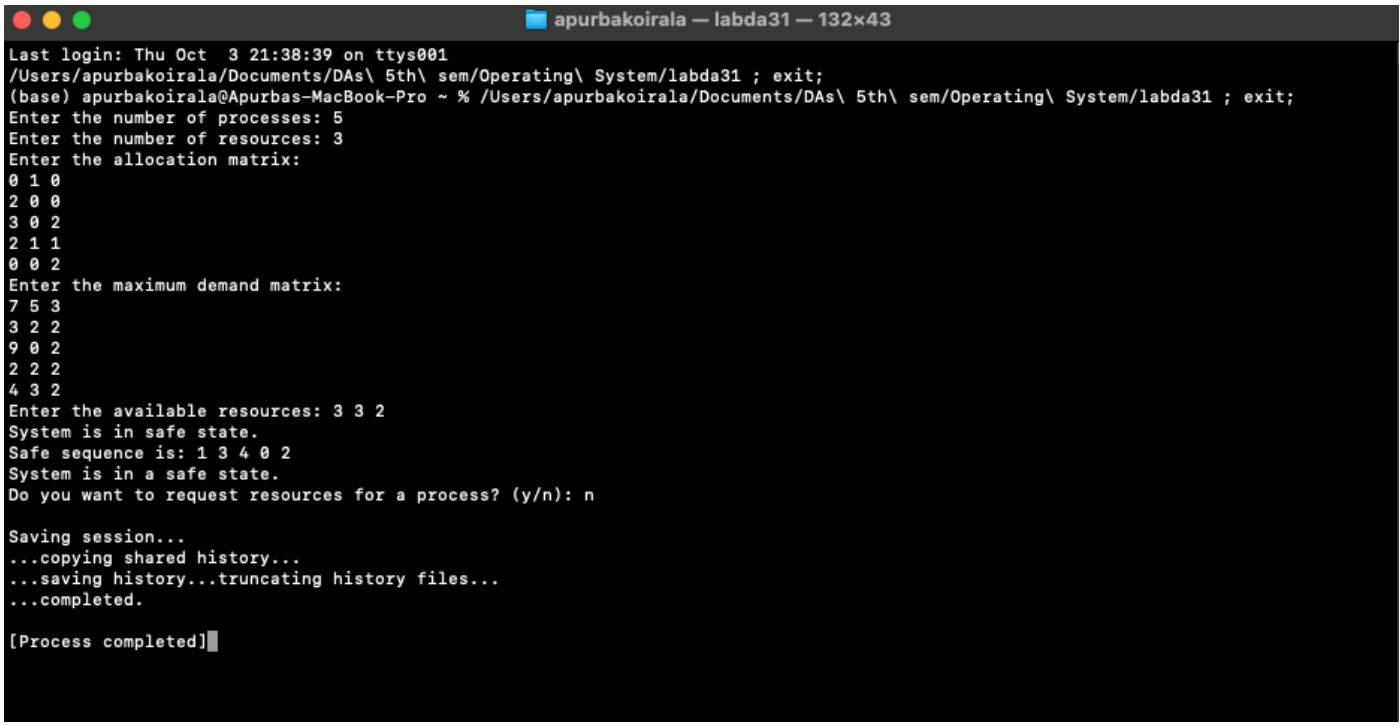
```
        }

    }

}


int main() {

    int np, nr;

    int processes[MAX_PROCESS], avail[MAX_RESOURCES];

    int max[MAX_PROCESS][MAX_RESOURCES],
alloc[MAX_PROCESS][MAX_RESOURCES];


    printf("Enter the number of processes: ");

    scanf("%d", &np);

    printf("Enter the number of resources: ");

    scanf("%d", &nr);

    printf("Enter the allocation matrix:\n");

    for (int i = 0; i < np; i++) {

        processes[i] = i;

        for (int j = 0; j < nr; j++) {

            scanf("%d", &alloc[i][j]);

        }

    }

    printf("Enter the maximum demand matrix:\n");

    for (int i = 0; i < np; i++) {

        for (int j = 0; j < nr; j++) {

            scanf("%d", &max[i][j]);

        }

    }
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    printf("Enter the available resources: ");
    for (int i = 0; i < nr; i++) {
        scanf("%d", &avail[i]);
    }
    if (isSafe(processes, avail, max, alloc, np, nr)) {
        printf("System is in a safe state.\n");
    } else {
        printf("System is not in a safe state.\n");
    }
    char ch;
    printf("Do you want to request resources for a process? (y/n): ");
    scanf(" %c", &ch);
    if (ch == 'y' || ch == 'Y') {
        requestResources(processes, avail, max, alloc, np, nr);
    }
    return 0;
}
```

OUTPUT SCREEN SHOT:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40



RESULTS: The deadlock avoidance and deadlock detection successfully implemented and desired output received.

Question 2

Write a C program to demonstrate the working of multiprocessor synchronization with Dining Philosopher problem.

AIM:

To demonstrate the working of multiprocessor synchronization with Dining Philosopher problem.

ALGRITHM:

- Initialize: Declare a variable isSuccessful as a boolean.
- Repeat: Begin an infinite loop.
- Set: Assign isSuccessful = false.
- Attempt: Check if both forks are available:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

- If yes, pick up the forks one by one.
- Set isSuccessful = true.

Check: If isSuccessful is false, block process Pi.

Eat: If forks were successfully picked up, proceed to eat.

Release: Put down both forks after eating.

Neighbor Check:

- If the left neighbor is waiting for the right fork, wake them up.
- If the right neighbor is waiting for the left fork, wake them up.

Think: Return to thinking state.

Repeat Forever: Continue the loop indefinitely.

SOURCE CODE :

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>


#define NUM_PHILOSOPHERS 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (id + 4) % NUM_PHILOSOPHERS

#define RIGHT (id + 1) % NUM_PHILOSOPHERS


int status[NUM_PHILOSOPHERS];

int philosopherID[NUM_PHILOSOPHERS] = { 0, 1, 2, 3, 4 };


sem_t accessLock;
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
sem_t sync[NUM_PHILOSOPHERS];


void check(int id) {
   if (status[id] == HUNGRY && status[LEFT] != EATING && status[RIGHT] != EATING)
{
      status[id] = EATING;
      sleep(2);
      printf("Philosopher %d picks up forks %d and %d\n", id + 1, LEFT + 1, id + 1);
      printf("Philosopher %d is Eating\n", id + 1);
      sem_post(&sync[id]);
   }
}


void pickFork(int id) {
   sem_wait(&accessLock);
   status[id] = HUNGRY;
   printf("Philosopher %d is Hungry\n", id + 1);
   check(id);
   sem_post(&accessLock);
   sem_wait(&sync[id]);
   sleep(1);
}


void releaseFork(int id) {
   sem_wait(&accessLock);
   status[id] = THINKING;
   printf("Philosopher %d puts down forks %d and %d\n", id + 1, LEFT + 1, id + 1);
```

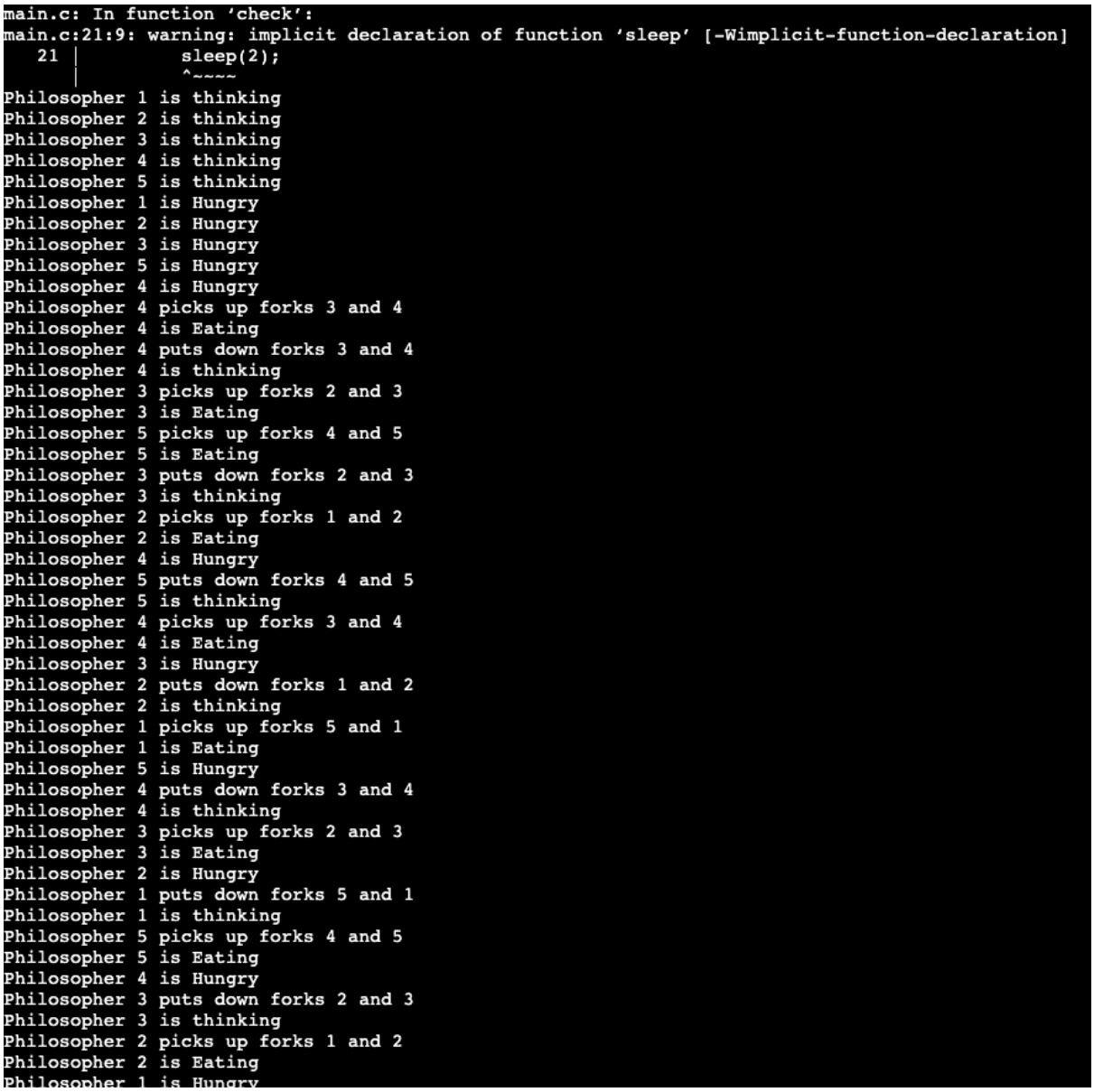
Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    printf("Philosopher %d is thinking\n", id + 1);

    check(LEFT);

    check(RIGHT);

    sem_post(&accessLock);

}


void* philosopher(void* num) {

    while (1) {

        int* id = num;

        sleep(1);

        pickFork(*id);

        sleep(0);

        releaseFork(*id);

    }

}


int main() {

    int i;

    pthread_t thread_id[NUM_PHILOSOPHERS];

    sem_init(&accessLock, 0, 1);

    for (i = 0; i < NUM_PHILOSOPHERS; i++)

        sem_init(&sync[i], 0, 0);

    for (i = 0; i < NUM_PHILOSOPHERS; i++) {

        pthread_create(&thread_id[i], NULL, philosopher, &philosopherID[i]);

        printf("Philosopher %d is thinking\n", i + 1);

    }
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    for (i = 0; i < NUM_PHILOSOPHERS; i++)

        pthread_join(thread_id[i], NULL);

}
```

OUTPUT SCREEN SHOT:



```
main.c: In function 'check':
main.c:21:9: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
   21 |         sleep(2);
      |         ^~~~~
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 picks up forks 3 and 4
Philosopher 4 is Eating
Philosopher 4 puts down forks 3 and 4
Philosopher 4 is thinking
Philosopher 3 picks up forks 2 and 3
Philosopher 3 is Eating
Philosopher 5 picks up forks 4 and 5
Philosopher 5 is Eating
Philosopher 3 puts down forks 2 and 3
Philosopher 3 is thinking
Philosopher 2 picks up forks 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
Philosopher 5 puts down forks 4 and 5
Philosopher 5 is thinking
Philosopher 4 picks up forks 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 puts down forks 1 and 2
Philosopher 2 is thinking
Philosopher 1 picks up forks 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 puts down forks 3 and 4
Philosopher 4 is thinking
Philosopher 3 picks up forks 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 puts down forks 5 and 1
Philosopher 1 is thinking
Philosopher 5 picks up forks 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 puts down forks 2 and 3
Philosopher 3 is thinking
Philosopher 2 picks up forks 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
Philosopher 5 puts down forks 4 and 5
Philosopher 5 is thinking
Philosopher 4 picks up forks 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 puts down forks 1 and 2
Philosopher 2 is thinking
Philosopher 1 picks up forks 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 puts down forks 3 and 4
Philosopher 4 is thinking
Philosopher 3 picks up forks 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 puts down forks 5 and 1
Philosopher 1 is thinking
Philosopher 5 picks up forks 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 puts down forks 2 and 3
Philosopher 3 is thinking
Philosopher 2 picks up forks 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 puts down forks 4 and 5
Philosopher 5 is thinking
Philosopher 4 picks up forks 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 puts down forks 1 and 2
Philosopher 2 is thinking
Philosopher 1 picks up forks 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 puts down forks 3 and 4
Philosopher 4 is thinking
Philosopher 3 picks up forks 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 puts down forks 5 and 1
Philosopher 1 is thinking
Philosopher 5 picks up forks 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 puts down forks 2 and 3
Philosopher 3 is thinking
Philosopher 2 picks up forks 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 puts down forks 4 and 5
Philosopher 5 is thinking
```

RESULTS: The working of multiprocessor sync with Dining Philosophers problem demonstrated

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Question 3

Write a C program to implement the following memory allocation strategies

(i) First Fit
(ii) Best Fit
(iii) Worst Fit

Choose a minimum of six memory partitions and five processes (Minimum) with sizes of your choice.

Rank the strategies in terms of how efficiently they use memory.

AIM: To implement First fit, Best fit, Worst fit

ALGORITHM:

Initialize arrays for memory blocks (blockSize[]), process sizes (processSize[]), and block assignments (allocation[]).

In First-Fit, copy blockSize to tempBlockSize and allocate each process to the first block that fits, reducing the block size after allocation.

In Best-Fit, copy blockSize to tempBlockSize and allocate each process to the smallest fitting block, then decrease the size of that block.

In Worst-Fit, copy blockSize to tempBlockSize and allocate each process to the largest fitting block, reducing that block's size accordingly.

Calculate the total memory used by summing the sizes of all allocated processes in the allocation[] array.

Print the allocation results, showing which block each process is allocated to or indicating if not allocated.

In the main function, input the number of memory blocks and processes along with their sizes.

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

Perform First-Fit, Best-Fit, and Worst-Fit allocations for the input processes and blocks.

Display the allocation results and calculate total memory used for each allocation strategy.

Compare the memory usage of the three strategies and print the ranking based on the highest memory usage.

SOURCE CODE :

```
#include <stdio.h>

#include <string.h>


void firstFit(int blockSize[], int m, int processSize[], int n, int allocation[]) {
    int tempBlockSize[m];
    memcpy(tempBlockSize, blockSize, sizeof(int) * m);


    for (int i = 0; i < n; i++) allocation[i] = -1;


    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (tempBlockSize[j] >= processSize[i]) {
                allocation[i] = j;
                tempBlockSize[j] -= processSize[i];
                break;
            }
        }
    }
}
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
void bestFit(int blockSize[], int m, int processSize[], int n, int allocation[]) {
    int tempBlockSize[m];
    memcpy(tempBlockSize, blockSize, sizeof(int) * m);

    for (int i = 0; i < n; i++) allocation[i] = -1;

    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (tempBlockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || tempBlockSize[bestIdx] > tempBlockSize[j])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            tempBlockSize[bestIdx] -= processSize[i];
        }
    }
}

void worstFit(int blockSize[], int m, int processSize[], int n, int allocation[]) {
    int tempBlockSize[m];
    memcpy(tempBlockSize, blockSize, sizeof(int) * m);

    for (int i = 0; i < n; i++) allocation[i] = -1;
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    for (int i = 0; i < n; i++) {
        int worstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (tempBlockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || tempBlockSize[worstIdx] < tempBlockSize[j])
                    worstIdx = j;
            }
        }
        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            tempBlockSize[worstIdx] -= processSize[i];
        }
    }
}

int totalMemoryUsed(int blockSize[], int m, int allocation[], int processSize[], int n) {
    int memoryUsed = 0;
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1) {
            memoryUsed += processSize[i];
        }
    }
    return memoryUsed;
}
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
void printAllocation(const char *strategy, int processSize[], int n, int allocation[]) {
    printf("\n%s Allocation:\n", strategy);
    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

int main() {
    int m, n;

    printf("Enter number of memory blocks (minimum 6): ");
    scanf("%d", &m);
    int blockSize[m];
    printf("Enter the sizes of %d memory blocks: ", m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &blockSize[i]);
    }

    printf("Enter number of processes (minimum 5): ");
    scanf("%d", &n);
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    int processSize[n];

    printf("Enter the sizes of %d processes: ", n);

    for (int i = 0; i < n; i++) {

        scanf("%d", &processSize[i]);

    }


    int allocationFirstFit[n], allocationBestFit[n], allocationWorstFit[n];


    firstFit(blockSize, m, processSize, n, allocationFirstFit);

    bestFit(blockSize, m, processSize, n, allocationBestFit);

    worstFit(blockSize, m, processSize, n, allocationWorstFit);


    printAllocation("First-Fit", processSize, n, allocationFirstFit);

    printAllocation("Best-Fit", processSize, n, allocationBestFit);

    printAllocation("Worst-Fit", processSize, n, allocationWorstFit);


    int totalMemoryFirstFit = totalMemoryUsed(blockSize, m, allocationFirstFit, processSize,
n);

    int totalMemoryBestFit = totalMemoryUsed(blockSize, m, allocationBestFit, processSize,
n);

    int totalMemoryWorstFit = totalMemoryUsed(blockSize, m, allocationWorstFit,
processSize, n);


    printf("\nMemory Usage Efficiency Ranking (highest memory usage is ranked best):\n");

    if (totalMemoryFirstFit >= totalMemoryBestFit && totalMemoryFirstFit >=
totalMemoryWorstFit)

        printf("1. First-Fit\n");
```

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
    else if (totalMemoryBestFit >= totalMemoryFirstFit && totalMemoryBestFit >=
totalMemoryWorstFit)

        printf("1. Best-Fit\n");

    else

        printf("1. Worst-Fit\n");



    if ((totalMemoryFirstFit >= totalMemoryBestFit && totalMemoryFirstFit <
totalMemoryWorstFit) ||

        (totalMemoryFirstFit < totalMemoryBestFit && totalMemoryFirstFit >=
totalMemoryWorstFit))

        printf("2. First-Fit\n");

    else if ((totalMemoryBestFit >= totalMemoryFirstFit && totalMemoryBestFit <
totalMemoryWorstFit) ||

            (totalMemoryBestFit < totalMemoryFirstFit && totalMemoryBestFit >=
totalMemoryWorstFit))

        printf("2. Best-Fit\n");

    else

        printf("2. Worst-Fit\n");



    printf("3. The remaining strategy\n");



    return 0;

}
```

OUTPUT SCREEN SHOT:

Reg No : 22BCE3799
Name : Apurba Koirala
Slot : L39+L40

```
Enter number of memory blocks (minimum 6): 6
Enter the sizes of 6 memory blocks: 100
200
300
400
500
600
Enter number of processes (minimum 5): 5
Enter the sizes of 5 processes: 100
150
200
250
300

First-Fit Allocation:
Process No.     Process Size    Block No.
1               100             1
2               150             2
3               200             3
4               250             4
5               300             5

Best-Fit Allocation:
Process No.     Process Size    Block No.
1               100             1
2               150             2
3               200             3
4               250             4
5               300             5

Worst-Fit Allocation:
Process No.     Process Size    Block No.
1               100             6
2               150             5
3               200             6
4               250             4
5               300             5

Memory Usage Efficiency Ranking (highest memory usage is ranked best):
1. First-Fit
2. Worst-Fit
3. The remaining strategy
```

RESULTS: Required memory allocation strategies implemented and output screenshot attached