

OS LAB 5
TENIDHAR REDDY CHINTHAM
23BCT0227
30/10/25

Q1.) Implement Dynamic Memory Allocation algorithms - First-fit, Best-fit, Worst-fit in C.

CODE.)

```
#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    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 (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\nFirst Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1)
            printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
        else
            printf("Process %d -> Not Allocated\n", i + 1);
    }
}

void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    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 (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
}
```

```
}
```

```
printf("\nBest Fit Allocation:\n");
for (int i = 0; i < n; i++) {
    if (allocation[i] != -1)
        printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
    else
        printf("Process %d -> Not Allocated\n", i + 1);
}
}
```

```
void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) allocation[i] = -1;

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

```
printf("\nWorst Fit Allocation:\n");
for (int i = 0; i < n; i++) {
    if (allocation[i] != -1)
        printf("Process %d -> Block %d\n", i + 1, allocation[i] + 1);
    else
        printf("Process %d -> Not Allocated\n", i + 1);
}
}
```

```
int main() {
    int m, n;
    printf("Enter number of blocks: ");
    scanf("%d", &m);
    int blockSize[m];
    printf("Enter size of each block:\n");
    for (int i = 0; i < m; i++) scanf("%d", &blockSize[i]);

    printf("Enter number of processes: ");
    scanf("%d", &n);
    int processSize[n];
    printf("Enter size of each process:\n");
    for (int i = 0; i < n; i++) scanf("%d", &processSize[i]);
}
```

```

int b1[m], b2[m], b3[m];
for (int i = 0; i < m; i++) b1[i] = b2[i] = b3[i] = blockSize[i];

firstFit(b1, m, processSize, n);
bestFit(b2, m, processSize, n);
worstFit(b3, m, processSize, n);

return 0;
}

```

OUTPUT.)

```

• matlab@sjt317scope021:~/23bct0227$ gcc dyna.c -o ./dyna
• matlab@sjt317scope021:~/23bct0227$ ./dyna
Enter number of blocks: 4
Enter size of each block:
100
200
350
50
Enter number of processes: 3
Enter size of each process:
50
175
150

First Fit Allocation:
Process 1 -> Block 1
Process 2 -> Block 2
Process 3 -> Block 3

Best Fit Allocation:
Process 1 -> Block 4
Process 2 -> Block 2
Process 3 -> Block 3

Worst Fit Allocation:
Process 1 -> Block 3
Process 2 -> Block 3
Process 3 -> Block 2
• matlab@sjt317scope021:~/23bct0227$ █

```

SIGN.) ATTACHED AT END

Q2.) Implement Page Replacement Algorithms FIFO, LRU and Optimal in C.

CODE.)

```
#include <stdio.h>

int findLRU(int time[], int n) {
    int min = time[0], pos = 0;
    for (int i = 1; i < n; ++i) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}

void FIFO(int pages[], int n, int framesCount) {
    int frames[framesCount], front = 0, count = 0, pageFaults = 0;
    for (int i = 0; i < framesCount; i++) frames[i] = -1;

    for (int i = 0; i < n; i++) {
        int page = pages[i], found = 0;
        for (int j = 0; j < framesCount; j++)
            if (frames[j] == page) found = 1;

        if (!found) {
            frames[front] = page;
            front = (front + 1) % framesCount;
            pageFaults++;
        }
    }
    printf("\nFIFO Page Faults: %d\n", pageFaults);
}

void LRU(int pages[], int n, int framesCount) {
    int frames[framesCount], time[framesCount];
    int pageFaults = 0, counter = 0;
    for (int i = 0; i < framesCount; i++) frames[i] = -1;

    for (int i = 0; i < n; ++i) {
        int found = 0;
        for (int j = 0; j < framesCount; ++j) {
            if (frames[j] == pages[i]) {
                counter++;
                time[j] = counter;
                found = 1;
                break;
            }
        }
    }
}
```

```

if (!found) {
int pos;
for (pos = 0; pos < framesCount; ++pos)
if (frames[pos] == -1) break;
if (pos == framesCount) pos = findLRU(time, framesCount);

```

```

frames[pos] = pages[i];
counter++;
time[pos] = counter;
pageFaults++;
}
}

```

```

printf("\nLRU Page Faults: %d\n", pageFaults);
}

```

```

void Optimal(int pages[], int n, int framesCount) {
int frames[framesCount];
int pageFaults = 0;
for (int i = 0; i < framesCount; i++) frames[i] = -1;

```

```

for (int i = 0; i < n; i++) {
int page = pages[i], found = 0;
for (int j = 0; j < framesCount; j++)
if (frames[j] == page) found = 1;

```

```

if (!found) {
int replaceIndex = -1, farthest = i + 1;
for (int j = 0; j < framesCount; j++) {
int k;
for (k = i + 1; k < n; k++)
if (frames[j] == pages[k]) break;
if (k == n) {
replaceIndex = j;
break;
}
if (k > farthest) {
farthest = k;
replaceIndex = j;
}
}
if (replaceIndex == -1) replaceIndex = 0;
frames[replaceIndex] = page;
pageFaults++;
}
}

```

```

printf("\nOptimal Page Faults: %d\n", pageFaults);
}

```

```

int main() {

```

```

int n, frames;
printf("Enter number of pages: ");
scanf("%d", &n);
int pages[n];
printf("Enter the page reference string:\n");
for (int i = 0; i < n; i++) scanf("%d", &pages[i]);

printf("Enter number of frames: ");
scanf("%d", &frames);

FIFO(pages, n, frames);
LRU(pages, n, frames);
Optimal(pages, n, frames);

return 0;
}

```

OUTPUT.)

```

• matlab@sjt317scope021:~/23bct0227$ gcc pagerep.c -o ./pagerep
• matlab@sjt317scope021:~/23bct0227$ ./pagerep
Enter number of pages: 20
Enter the page reference string:
3 2 1 4 2 5 2 3 4 5 2 5 4 1 4 2 1 3 2 1
Enter number of frames: 4

FIFO Page Faults: 9

LRU Page Faults: 8

Optimal Page Faults: 6
• matlab@sjt317scope021:~/23bct0227$ █

```

SIGN.) ATTACHED AT END