SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GENERAL SEMESTER 2023-24

B.Tech - CSE

BCSE303P: Operating Systems Lab

5. Implement the below concepts:

a) Dynamic memory allocation algorithms - First-fit, Best-fit, Worst-fit algorithms.
b) Page Replacement Algorithms - FIFO, LRU and Optimal.

a)

PURVA SHARMA 21BCE0169

main.c

```c
#include <stdio.h>

#define MAX_MEMORY_SIZE 100

// Memory block structure
typedef struct {
    int start;
    int size;
    int allocated;
} MemoryBlock;

// Function prototypes
void initializeMemory();
void displayMemory();
int firstFit(int processSize);
int bestFit(int processSize);
int worstFit(int processSize);

// Global variables
MemoryBlock memory[MAX_MEMORY_SIZE];

int main() {
    int choice, processSize, allocatedIndex;

    // Initialize memory
    initializeMemory();

    do {
        // Display menu
        printf("\n--- Memory Allocation Menu ---\n");
        printf("1. First Fit\n");
        printf("2. Best Fit\n");
        printf("3. Worst Fit\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter process size: ");
                scanf("%d", &processSize);
                allocatedIndex = firstFit(processSize);
                if (allocatedIndex != -1)
                    printf("Process allocated at index %d\n", allocatedIndex);
                else
                    printf("No suitable memory block found\n");
                displayMemory();
                break;

            case 2:
                printf("Enter process size: ");
                scanf("%d", &processSize);
                allocatedIndex = bestFit(processSize);
                if (allocatedIndex != -1)
                    printf("Process allocated at index %d\n", allocatedIndex);
                else
                    printf("No suitable memory block found\n");
                displayMemory();
                break;

            case 3:
                printf("Enter process size: ");
                scanf("%d", &processSize);
                allocatedIndex = worstFit(processSize);
                if (allocatedIndex != -1)
                    printf("Process allocated at index %d\n", allocatedIndex);
                else
                    printf("No suitable memory block found\n");
                displayMemory();
                break;
```

```c
            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

// Initialize memory blocks
void initializeMemory() {
    int i;

    for (i = 0; i < MAX_MEMORY_SIZE; i++) {
        memory[i].start = i * 10;  // Assuming each block size is 10
        memory[i].size = 10;
        memory[i].allocated = 0;
    }
}

// Display memory blocks
void displayMemory() {
    int i;

    printf("\n--- Memory Blocks ---\n");
    printf("Start\tSize\tAllocated\n");
    for (i = 0; i < MAX_MEMORY_SIZE; i++)
        printf("%d\t%d\t%d\n", memory[i].start, memory[i].size, memory[i].allocated);
}
```

```c
// First Fit allocation algorithm
int firstFit(int processSize) {
    int i;

    for (i = 0; i < MAX_MEMORY_SIZE; i++) {
        if (memory[i].allocated == 0 && memory[i].size >= processSize) {
            memory[i].allocated = 1;
            return i;
        }
    }

    return -1;
}

// Best Fit allocation algorithm
int bestFit(int processSize) {
    int i, bestFitIndex = -1, bestFitDiff = MAX_MEMORY_SIZE;

    for (i = 0; i < MAX_MEMORY_SIZE; i++) {
        if (memory[i].allocated == 0 && memory[i].size >= processSize) {
            int diff = memory[i].size - processSize;
            if (diff < bestFitDiff) {
                bestFitDiff = diff;
                bestFitIndex = i;
            }
        }
    }

    if (bestFitIndex != -1)
        memory[bestFitIndex].allocated = 1;

    return bestFitIndex;
}
```

```c
// Worst Fit allocation algorithm
int worstFit(int processSize) {
    int i, worstFitIndex = -1, worstFitDiff = -1;

    for (i = 0; i < MAX_MEMORY_SIZE; i++) {
        if (memory[i].allocated == 0 && memory[i].size >= processSize) {
            int diff = memory[i].size - processSize;
            if (diff > worstFitDiff) {
                worstFitDiff = diff;
                worstFitIndex = i;
            }
        }
    }

    if (worstFitIndex != -1)
        memory[worstFitIndex].allocated = 1;

    return worstFitIndex;
}
```

OUTPUT:

FIRST FIT-

PURVA SHARMA 21BCE0169

Console: connected (Running: 30 seg)

```
--- Memory Allocation Menu ---
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1
Enter process size: 2
Process allocated at index 0

--- Memory Blocks ---
Start   Size    Allocated
0       10      1
10      10      0
20      10      0
30      10      0
40      10      0
50      10      0
60      10      0
70      10      0
80      10      0
90      10      0
100     10      0
110     10      0
```

BEST FIT-

PURVA SHARMA 21BCE0169

Console: connected (Running: 12 seg)

```
--- Memory Allocation Menu ---
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2
Enter process size: 2
Process allocated at index 0

--- Memory Blocks ---
Start   Size    Allocated
0       10      1
10      10      0
20      10      0
30      10      0
40      10      0
50      10      0
60      10      0
70      10      0
80      10      0
90      10      0
100     10      0
```

WORST FIT-

PURVA SHARMA 21BCE0169

Console: connected (Running: 16 seg)

```
--- Memory Allocation Menu ---
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3
Enter process size: 2
Process allocated at index 0

--- Memory Blocks ---
Start   Size    Allocated
0       10      1
10      10      0
20      10      0
30      10      0
40      10      0
50      10      0
60      10      0
70      10      0
80      10      0
90      10      0
100     10      0
```

b)

PURVA SHARMA 21BCE0169

main.c

```c
#include <stdio.h>

#define MAX_PAGES 100
#define MAX_FRAMES 10

// Function prototypes
void initializeFrames();
int isPageInFrames(int page, int frameCount);
int getPageIndex(int page, int frameCount);
int getFifoPageIndex(int frameCount);
int getLruPageIndex(int frameCount, int usedCount[MAX_FRAMES]);
int getOptimalPageIndex(int frameCount, int pages[], int pageIndices[], int currentIndex);

// Global variables
int frames[MAX_FRAMES];
int pageFaults = 0;

int main() {
    int pageReferences[MAX_PAGES];
    int pageIndices[MAX_PAGES];
    int frameCount, referenceCount;
    int i, j;

    // Read input
    printf("Enter the number of frames: ");
    scanf("%d", &frameCount);

    printf("Enter the number of page references: ");
    scanf("%d", &referenceCount);

    printf("Enter the page reference string: ");
    for (i = 0; i < referenceCount; i++) {
        scanf("%d", &pageReferences[i]);
        pageIndices[i] = -1;
    }

```

```c
    // FIFO algorithm
    initializeFrames();

    for (i = 0; i < referenceCount; i++) {
        if (!isPageInFrames(pageReferences[i], frameCount)) {
            int index = getFifoPageIndex(frameCount);
            frames[index] = pageReferences[i];
            pageFaults++;
        }
    }

    printf("\nFIFO Page Replacement:\n");
    printf("Page Faults: %d\n", pageFaults);

    // LRU algorithm
    initializeFrames();
    pageFaults = 0;
    int usedCount[MAX_FRAMES] = {0};

    for (i = 0; i < referenceCount; i++) {
        if (!isPageInFrames(pageReferences[i], frameCount)) {
            int index = getLruPageIndex(frameCount, usedCount);
            frames[index] = pageReferences[i];
            pageFaults++;
        }
        usedCount[getPageIndex(pageReferences[i], frameCount)] = i + 1;
    }

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

```c
    // LRU algorithm
    initializeFrames();
    pageFaults = 0;
    int usedCount[MAX_FRAMES] = {0};

    for (i = 0; i < referenceCount; i++) {
        if (!isPageInFrames(pageReferences[i], frameCount)) {
            int index = getLruPageIndex(frameCount, usedCount);
            frames[index] = pageReferences[i];
            pageFaults++;
        }
        usedCount[getPageIndex(pageReferences[i], frameCount)] = i + 1;
    }

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

    // Optimal algorithm
    initializeFrames();
    pageFaults = 0;

    for (i = 0; i < referenceCount; i++) {
        if (!isPageInFrames(pageReferences[i], frameCount)) {
            int index = getOptimalPageIndex(frameCount, &pageReferences[i], &pageIndices[i], i);
            frames[index] = pageReferences[i];
            pageFaults++;
        }
    }

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

    return 0;
}
```

```c
// Initialize frames
void initializeFrames() {
    int i;

    for (i = 0; i < MAX_FRAMES; i++)
        frames[i] = -1;
}

// Check if page is present in frames
int isPageInFrames(int page, int frameCount) {
    int i;

    for (i = 0; i < frameCount; i++) {
        if (frames[i] == page)
            return 1;
    }

    return 0;
}

// Get the index of a page in frames
int getPageIndex(int page, int frameCount) {
    int i;

    for (i = 0; i < frameCount; i++) {
        if (frames[i] == page)
            return i;
    }

    return -1;
}

// Get the index of the page to replace using FIFO algorithm
int getFifoPageIndex(int frameCount) {
    return (pageFaults % frameCount);
}
```

```c
// Get the index of the page to replace using LRU algorithm
int getLruPageIndex(int frameCount, int usedCount[MAX_FRAMES]) {
    int i, minIndex = 0, minValue = usedCount[0];

    for (i = 1; i < frameCount; i++) {
        if (usedCount[i] < minValue) {
            minValue = usedCount[i];
            minIndex = i;
        }
    }

    return minIndex;
}

// Get the index of the page to replace using Optimal algorithm
int getOptimalPageIndex(int frameCount, int pages[], int pageIndices[], int currentIndex) {
    int i, j, maxIndex = 0, maxValue = -1;

    for (i = 0; i < frameCount; i++) {
        if (pageIndices[i] == -1)
            return i;

        int found = 0;

        for (j = currentIndex + 1; j < MAX_PAGES; j++) {
            if (pages[j] == frames[i]) {
                if (j > maxValue) {
                    maxValue = j;
                    maxIndex = i;
                }
                found = 1;
                break;
            }
        }

        if (!found)
            return i;
    }
    return maxIndex;
}
```

OUTPUT:

PURVA SHARMA 21BCE0169

Console: connection closed (Running: 15 seg)

```
Enter the number of frames: 6
Enter the number of page references: 4
Enter the page reference string: 2 3 7 6

FIFO Page Replacement:
Page Faults: 4

LRU Page Replacement:
Page Faults: 4

Optimal Page Replacement:
Page Faults: 4
```