Experiment - 10

Write a code and compare different page replacement algorithms (FIFO, LRU, Optimal) in C. Evaluate their performance using various reference string inputs to understand their impact on memory management.

Algorithm for implementation of page replacement algorithms

Step 1: Define Constants and Include Libraries

1. Define the maximum size of arrays using a macro *#define MAX 100.*
2. Include necessary header files:
    - *#include <stdio.h>* for input/output operations.
    - *#include <limits.h>* for constants like *INT_MAX*.

Step 2: Input Handling

1. Initialize variables:
    - *referenceString* to hold the sequence of page requests.
    - *n* to store the number of pages in the reference string.
    - *frames* to store the number of memory slots available.
2. Prompt the user to input the values:
    - Number of pages *(n)*.
    - The reference string (a series of page numbers).
    - Number of frames available in memory.

Step 3: Implement Helper Functions

1. Function *isPageInMemory()*:

- Iterate over the memory slots.
- Check if the requested page is already in memory.
- If found, return 1 (true); otherwise, return 0 (false).

2. Function *fifo()* (First In First Out):

- Given the *memory* array, return the *front* index, which tracks the oldest page in the memory.

3. Function *lru()* (Least Recently Used):

- Initialize an array *least_recent[]* to track the most recent usage of each page.
- For each page in memory, search backward in the reference string to find its most recent use.
- Identify the page that was least recently used and return its index.

4. Function *optimal()*:

- Initialize an array *future[]* to predict the future use of each page in memory.
- For each page in memory, search forward in the reference string to determine when it will be used next.
- Identify the page that will not be used for the longest time and return its index.

Step 4: Initialize Simulation

1. Create a simulation function called simulate():
    - This function takes in the *referenceString*, the number of pages (n), the number of frames (frames), and the algorithm type (FIFO, LRU, Optimal).
    - Initialize an empty *memory[]* array with -1 to indicate empty slots.
    - Initialize a variable *pageFaults* to count the number of page faults.
    - Initialize variables *pos* (position to replace a page) and *front* (used for FIFO).

Step 5: Simulate Page Replacement

1. Iterate through each page in the reference string:

- **Check if the page is already in memory** using *isPageInMemory()*:
  - If it is already in memory, continue to the next page.
  - If it's not in memory, a page fault occurs:
    1. Increment *pageFaults*.
    2. Depending on the chosen algorithm, decide which page to replace:
       - **FIFO**: Use *fifo()* to get the index of the oldest page. Update *front*.
       - **LRU**: Use *lru()* to get the index of the least recently used page.
       - **Optimal**: Use *optimal()* to get the index of the page that won't be used for the longest time.
    3. Replace the selected page in memory.
- Print the current state of memory after each page replacement.

Step 6: Display Simulation Results

1. Print the total number of page faults after completing the simulation for the selected algorithm.

2. Repeat **Step 5** for each algorithm (FIFO, LRU, and Optimal).

Step 7: Main Function

1. Initialize the main function.

3. Collect user inputs for:

    • Number of pages in the reference string (*n*).

    • The actual reference string.

    • Number of memory frames.

3. Call the *simulate()* function three times:

    • Once for *FIFO*.

    • Once for *LRU*.

    • Once for *Optimal*.

4. Print results for each algorithm.

Step 8: END

```c
#include <stdio.h>
#include <limits.h>

#define MAX 100

// Function to check if a page is present in memory
int isPageInMemory(int page, int memory[], int frames) {
    for (int i = 0; i < frames; i++) {
        if (memory[i] == page) {
            return 1;
        }
    }
    return 0;
}

// Function to find the position of the page to be replaced in FIFO
int fifo(int memory[], int front, int frames) {
    return front;
}
```

```c
// Function to find the position of the page to be replaced in LRU
int lru(int memory[], int frames, int referenceString[], int n, int index) {
    int least_recent[MAX], i, j;

    for (i = 0; i < frames; i++) {
        least_recent[i] = -1;
        for (j = index - 1; j >= 0; j--) {
            if (referenceString[j] == memory[i]) {
                least_recent[i] = j;
                break;
            }
        }
    }

    int min = least_recent[0], pos = 0;
    for (i = 1; i < frames; i++) {
        if (least_recent[i] < min) {
            min = least_recent[i];
            pos = i;
        }
    }
    return pos;
}
```

```c
// Function to find the position of the page to be replaced in Optimal
int optimal(int memory[], int frames, int referenceString[], int n, int index) {
    int future[MAX], i, j;

    for (i = 0; i < frames; i++) {
        future[i] = INT_MAX;
        for (j = index + 1; j < n; j++) {
            if (referenceString[j] == memory[i]) {
                future[i] = j;
                break;
            }
        }
    }

    int max = future[0], pos = 0;
    for (i = 1; i < frames; i++) {
        if (future[i] > max) {
            max = future[i];
            pos = i;
        }
    }
    return pos;
}
```

```c
void simulate(int referenceString[], int n, int frames, const char *algo) {
    int memory[MAX], i, j, pageFaults = 0, pos = 0, front = 0;

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

    for (i = 0; i < n; i++) {
        if (!isPageInMemory(referenceString[i], memory, frames)) {
            pageFaults++;
            if (algo == "FIFO") {
                pos = fifo(memory, front, frames);
                front = (front + 1) % frames;
            } else if (algo == "LRU") {
                pos = lru(memory, frames, referenceString, n, i);
            } else if (algo == "Optimal") {
                pos = optimal(memory, frames, referenceString, n, i);
            }
            memory[pos] = referenceString[i];
        }
        printf("Memory after inserting %d: ", referenceString[i]);
        for (j = 0; j < frames; j++) {
            if (memory[j] == -1)
                printf(" - ");
            else
                printf(" %d ", memory[j]);
        }
        printf("\n");
    }
    printf("Total page faults using %s: ↓ ", algo, pageFaults);
}
```

```c
int main() {
    int referenceString[MAX], n, frames, i;

    printf("Enter the number of pages in the reference string: ");
    scanf("%d", &n);

    printf("Enter the reference string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &referenceString[i]);
    }

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

    printf("\n----- FIFO -----\n");
    simulate(referenceString, n, frames, "FIFO");

    printf("\n----- LRU -----\n");
    simulate(referenceString, n, frames, "LRU");

    printf("\n----- Optimal -----\n");
    simulate(referenceString, n, frames, "Optimal");

    return 0;
}
```

```
Enter the number of pages in the reference string: 12
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3
Enter the number of frames: 3
```

```
----- FIFO -----
Memory after inserting 7:  7  -  -
Memory after inserting 0:  7  0  -
Memory after inserting 1:  7  0  1
Memory after inserting 2:  2  0  1
Memory after inserting 0:  2  0  1
Memory after inserting 3:  2  3  1
Memory after inserting 0:  2  3  0
Memory after inserting 4:  4  3  0
Memory after inserting 2:  4  2  0
Memory after inserting 3:  4  2  3
Memory after inserting 0:  0  2  3
Memory after inserting 3:  0  2  3
Total page faults using FIFO: 9
```

```
----- LRU -----
Memory after inserting 7:  7  -  -
Memory after inserting 0:  7  0  -
Memory after inserting 1:  7  0  1
Memory after inserting 2:  2  0  1
Memory after inserting 0:  2  0  1
Memory after inserting 3:  2  0  3
Memory after inserting 0:  2  0  3
Memory after inserting 4:  4  0  3
Memory after inserting 2:  4  2  3
Memory after inserting 3:  4  2  3
Memory after inserting 0:  0  2  3
Memory after inserting 3:  0  2  3
Total page faults using LRU: 8
```

```
----- Optimal -----
Memory after inserting 7:  7  -  -
Memory after inserting 0:  7  0  -
Memory after inserting 1:  7  0  1
Memory after inserting 2:  2  0  1
Memory after inserting 0:  2  0  1
Memory after inserting 3:  2  3  1
Memory after inserting 0:  2  3  0
Memory after inserting 4:  4  3  0
Memory after inserting 2:  4  2  0
Memory after inserting 3:  4  2  3
Memory after inserting 0:  0  2  3
Memory after inserting 3:  0  2  3
Total page faults using Optimal: 7
```