Lab 8: Implementation of Page Replacement Algorithms

CS363 • Operating System (Lab)

Date: 14 - 10 - 2025

Paging and Page Table Concepts

What is Paging ?

When a process is too large to fit completely in the main memory (RAM), the operating system divides both **main memory** and **processes** into fixed-size blocks.

- Each block of a process is called a **Page**.
- Each block of main memory is called a **Frame**.

Example:

If the page size = 4 kB and the main memory = 16 kB,
then the memory has 4 frames (4 kB × 4 = 16 kB).

Pages of a process can be loaded into any available frames, not necessarily in order.

What is a Page Table ?

A **Page Table** is used to keep track of which page of a process is stored in which frame of memory.

Page Number	Frame Number	Valid/Invalid Bit
0	2	V
1	5	I
2	0	V

Table 1: Example of a Page Table

Legend:

- **V (Valid):** The page is currently in physical memory.
- **I (Invalid):** The page is not currently in memory.

What is a Page Fault?

A **Page Fault** occurs when a process tries to access a page that is not present in the main memory.

When this happens, the operating system performs the following steps:

1. Pauses the running process.
2. Finds the required page on the disk.
3. Loads it into memory (in a free frame).
4. Updates the page table.
5. Resumes the process.

What is Page Replacement ?

Page Replacement is a memory-management technique used when a process requests a page that is not present in physical memory (RAM). This situation is known as a **Page Fault**. To handle it, the operating system must load the required page from secondary storage (disk) into a free frame in main memory.

Need for Page Replacement

If no free frame is available, the operating system must decide **which existing page to remove** to make space for the new one. This decision is made using a **Page Replacement Algorithm**.

Steps in Page Replacement

The operating system follows the below steps when a page fault occurs and page replacement is required:

1. Find the location of the required page on the disk.
2. If a free frame exists, load the page into that frame.
3. If no free frame exists, apply a **page-replacement algorithm** to select a victim page.
4. Write the victim page to the disk if it was modified (dirty).
5. Read the desired page into the freed frame and update the page and frame tables.
6. Restart the process that caused the page fault.

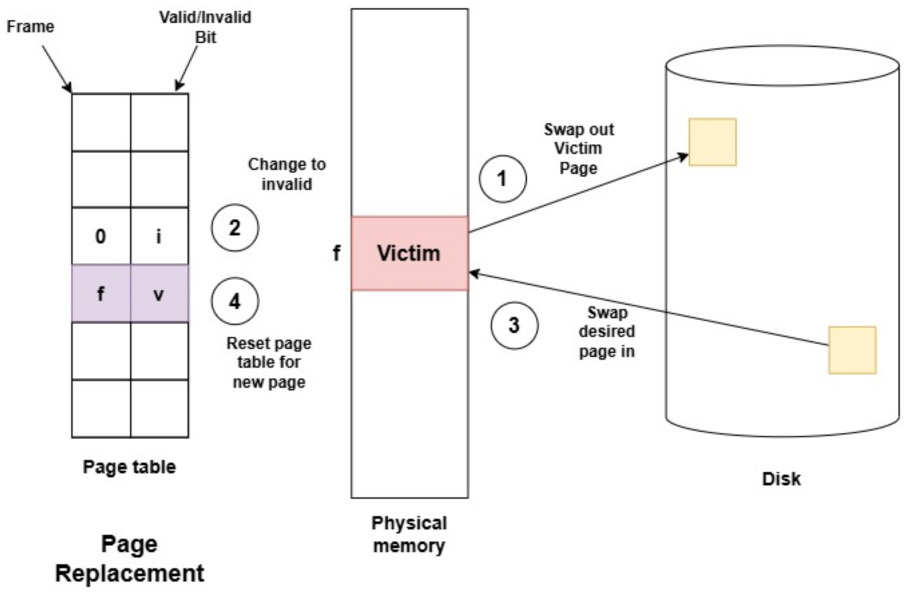

Figure 1: Simplified Page Replacement Process

Page Fault Service Time

When a page fault occurs and no free frame is available:

- Two disk transfers take place:
  - One **out** (victim page written to disk)
  - One **in** (required page read from disk)
- This effectively **doubles the page fault service time**.

To reduce this delay, a **Modify (Dirty) Bit** is used:

- If **Modify Bit = 1**, the page was changed - it must be written to disk.
- If **Modify Bit = 0**, the page was not changed - writing to disk can be skipped, thus reducing service time.

Page Replacement Algorithms

1. FIFO (First-In, First-Out) Algorithm

Idea: Replace the oldest page in memory (the one loaded earliest).

Working:

- Maintain a queue of pages in memory.
- When a page fault occurs and a new page must be loaded:
  - The page at the **front** of the queue is removed.
  - The new page is added to the **rear** of the queue.

Advantages:

- Simple to implement.
- Requires minimal book-keeping.

Disadvantages:

- May lead to **Belady's Anomaly**: increasing the number of frames can increase the number of page faults.

2. Optimal Page Replacement Algorithm

Idea: Replace the page that will not be used for the longest time in the future.

Working:

- When a page fault occurs, look ahead in the reference string.
- Replace the page whose next use is farthest in the future.

Advantages:

- Guarantees the lowest possible page fault rate.
- Never suffers from **Belady's Anomaly**.

Disadvantages:

- Requires future knowledge of page references - mainly used for theoretical comparison.

3. LRU (Least Recently Used) Algorithm

Idea: Replace the page that has not been used for the longest time in the past.

Working:

- Associate each page with the time of its last use.
- When a page must be replaced, choose the one that was least recently used.

Advantages:

- Practical and performs close to the Optimal algorithm.
- Based on a realistic assumption - past behavior predicts future use.

Disadvantages:

- Requires maintaining usage order or timestamps, which increases overhead.

Implementation and Programs

Program 1: FIFO Page Replacement

Listing 1: C Program for FIFO Page Replacement

```c
#include <stdio.h>

int main() {
    int frames[10], pages[30];
    int n, m, pageFaults = 0;
    int i, j, k, pos = 0, flag;

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

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

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

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

    printf("\nPage Replacement Process (FIFO):\n");
    printf("--------------------------------\n");

    for(i = 0; i < m; ++i) {
        flag = 0;

        for(j = 0; j < n; ++j) {
            if(frames[j] == pages[i]) {
                flag = 1; // Page hit
                break;
            }
        }

        if(flag == 0) {
            frames[pos] = pages[i];
            pos = (pos + 1) % n;
            pageFaults++;

            printf("For page %2d : ", pages[i]);
            for(k = 0; k < n; ++k) {
                if(frames[k] == -1)
                    printf(" - ");
                else
                    printf("%2d ", frames[k]);
            }
            printf(" (Page Fault)\n");
        } else {
            printf("For page %2d : ", pages[i]);
            for(k = 0; k < n; ++k) {
                if(frames[k] == -1)
                    printf(" - ");
                else
```

```c
                    printf("%2d ", frames[k]);
                }
                printf(" (Hit)\n");
            }
        }

    printf("\nTotal Page Faults = %d\n", pageFaults);
    printf("Total Page Hits   = %d\n", m - pageFaults);

    return 0;
}
```

Expected Output:

```
[202463010@paramshavak ~]$ nano fifo.c
[202463010@paramshavak ~]$ gcc -std=c99 -Wall -O2 fifo.c -o fifo
[202463010@paramshavak ~]$ ./fifo
Enter number of frames: 3
Enter number of pages: 20
Enter the reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page Replacement Process (FIFO):
--------------------------------
For page  7 :  7  -  -  (Page Fault)
For page  0 :  7  0  -  (Page Fault)
For page  1 :  7  0  1  (Page Fault)
For page  2 :  2  0  1  (Page Fault)
For page  0 :  2  0  1  (Hit)
For page  3 :  2  3  1  (Page Fault)
For page  0 :  2  3  0  (Page Fault)
For page  4 :  4  3  0  (Page Fault)
For page  2 :  4  2  0  (Page Fault)
For page  3 :  4  2  3  (Page Fault)
For page  0 :  0  2  3  (Page Fault)
For page  3 :  0  2  3  (Hit)
For page  2 :  0  2  3  (Hit)
For page  1 :  0  1  3  (Page Fault)
For page  2 :  0  1  2  (Page Fault)
For page  0 :  0  1  2  (Hit)
For page  1 :  0  1  2  (Hit)
For page  7 :  7  1  2  (Page Fault)
For page  0 :  7  0  2  (Page Fault)
For page  1 :  7  0  1  (Page Fault)

Total Page Faults = 15
Total Page Hits   = 5
```

Program 2: Optimal Page Replacement

Listing 2: C Program for Optimal Page Replacement

```c
#include <stdio.h>

int main() {
    int frames[10], pages[30];
    int n, m, pageFaults = 0;
    int i, j, k, pos, flag1, flag2, max, future[10];

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

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

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

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

    printf("\nPage Replacement Process (Optimal):\n");
    printf("-----------------------------------\n");

    for(i = 0; i < m; ++i) {
        flag1 = flag2 = 0;

        for(j = 0; j < n; ++j) {
            if(frames[j] == pages[i]) {
                flag1 = flag2 = 1;
                break;
            }
        }

        if(flag1 == 0) {
            for(j = 0; j < n; ++j) {
                if(frames[j] == -1) {
                    frames[j] = pages[i];
                    pageFaults++;
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0) {
            for(j = 0; j < n; ++j) {
                future[j] = -1;
                for(k = i + 1; k < m; ++k) {
                    if(frames[j] == pages[k]) {
                        future[j] = k;
                        break;
                    }
                }
            }

            max = -1;
            pos = -1;
            for(j = 0; j < n; ++j) {
                if(future[j] == -1) {
                    pos = j;
                    break;
                } else if(future[j] > max) {
                    max = future[j];
                    pos = j;
                }
            }

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

        printf("For page %2d : ", pages[i]);
        for(j = 0; j < n; ++j) {
            if(frames[j] == -1)
                printf(" - ");
            else
                printf("%2d ", frames[j]);
        }

        if(flag1 == 0)
            printf(" (Page Fault)\n");
        else
            printf(" (Hit)\n");
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);
    printf("Total Page Hits   = %d\n", m - pageFaults);

    return 0;
}
```

Expected Output:

```
[202463010@paramshavak ~]$ nano optimal.c
[202463010@paramshavak ~]$ gcc -std=c99 -Wall -O2 optimal.c -o optimal
[202463010@paramshavak ~]$ ./optimal
Enter number of frames: 3
Enter number of pages: 20
Enter the reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page Replacement Process (Optimal):
-----------------------------------
For page  7 :  7  -  -  (Page Fault)
For page  0 :  7  0  -  (Page Fault)
For page  1 :  7  0  1  (Page Fault)
For page  2 :  2  0  1  (Page Fault)
For page  0 :  2  0  1  (Hit)
For page  3 :  2  0  3  (Page Fault)
For page  0 :  2  0  3  (Hit)
For page  4 :  2  4  3  (Page Fault)
For page  2 :  2  4  3  (Hit)
For page  3 :  2  4  3  (Hit)
For page  0 :  2  0  3  (Page Fault)
For page  3 :  2  0  3  (Hit)
For page  2 :  2  0  3  (Hit)
For page  1 :  2  0  1  (Page Fault)
For page  2 :  2  0  1  (Hit)
For page  0 :  2  0  1  (Hit)
For page  1 :  2  0  1  (Hit)
For page  7 :  7  0  1  (Page Fault)
For page  0 :  7  0  1  (Hit)
For page  1 :  7  0  1  (Hit)

Total Page Faults = 9
Total Page Hits   = 11
```

Program 3: LRU Page Replacement

Listing 3: C Program for LRU Page Replacement

```c
#include <stdio.h>

int main() {
    int frames[10], pages[30], counter[10];
    int n, m, pageFaults = 0;
    int time = 0, flag1, flag2, i, j, pos, min;

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

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

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

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

    printf("\nPage Replacement Process (LRU - Counter Method):\n");
    printf("--------------------------------------------------\n");

    for(i = 0; i < m; ++i) {
        flag1 = flag2 = 0;

        for(j = 0; j < n; ++j) {
            if(frames[j] == pages[i]) {
                time++;
                counter[j] = time; // Update recent use time
                flag1 = flag2 = 1;
                break;
            }
        }

        if(flag1 == 0) {
            for(j = 0; j < n; ++j) {
                if(frames[j] == -1) {
                    time++;
                    frames[j] = pages[i];
                    counter[j] = time;
                    pageFaults++;
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0) {
            min = counter[0];
            pos = 0;
            for(j = 1; j < n; ++j) {
                if(counter[j] < min) {
                    min = counter[j];
                    pos = j;
                }
            }
            time++;
            frames[pos] = pages[i];
            counter[pos] = time;
            pageFaults++;
        }

        printf("For page %2d : ", pages[i]);
        for(j = 0; j < n; ++j) {
            if(frames[j] == -1)
                printf(" - ");
            else
                printf("%2d ", frames[j]);
        }

        if(flag1 == 1)
            printf(" (Hit)\n");
        else
            printf(" (Page Fault)\n");
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);
    printf("Total Page Hits   = %d\n", m - pageFaults);

    return 0;
}
```

Expected Output:

```
[202463010@paramshavak ~]$ nano lru_counter.c
[202463010@paramshavak ~]$ gcc -std=c99 -Wall -O2 lru_counter.c -o lru_counter
[202463010@paramshavak ~]$ ./lru_counter
Enter number of frames: 3
Enter number of pages: 20
Enter the reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page Replacement Process (LRU - Counter Method):
-------------------------------------------------
For page  7 :  7  -  -  (Page Fault)
For page  0 :  7  0  -  (Page Fault)
For page  1 :  7  0  1  (Page Fault)
For page  2 :  2  0  1  (Page Fault)
For page  0 :  2  0  1  (Hit)
For page  3 :  2  0  3  (Page Fault)
For page  0 :  2  0  3  (Hit)
For page  4 :  4  0  3  (Page Fault)
For page  2 :  4  0  2  (Page Fault)
For page  3 :  4  3  2  (Page Fault)
For page  0 :  0  3  2  (Page Fault)
For page  3 :  0  3  2  (Hit)
For page  2 :  0  3  2  (Hit)
For page  1 :  1  3  2  (Page Fault)
For page  2 :  1  3  2  (Hit)
For page  0 :  1  0  2  (Page Fault)
For page  1 :  1  0  2  (Hit)
For page  7 :  1  0  7  (Page Fault)
For page  0 :  1  0  7  (Hit)
For page  1 :  1  0  7  (Hit)

Total Page Faults = 12
Total Page Hits   = 8
```

Conclusion

Page replacement plays a vital role in efficiently managing limited main memory by deciding which pages to retain and which to replace when a page fault occurs. Among the algorithms, FIFO is simple but can suffer from Belady's anomaly, Optimal provides the theoretical best performance but is impractical, and LRU offers a realistic balance between efficiency and implementability, closely approximating Optimal behavior in real systems.

ASSIGNMENT:
Implement FIFO, Optimal, and LRU (using both Counter and Stack methods), and record page faults for different reference strings and frame sizes.