# ST. XAVIER’S COLLEGE

(Affiliated to Tribhuvan University)
Maitighar, Kathmandu

**OS Lab Assignment #7**

Page Replacement Algorithm

**SUBMITTED BY:**

Milan Rawal

018BSCIT019

2nd Year/4th Sem

**SUBMITTED TO:**

| | |
|---|---|
| **Er. Rajan Karmacharya (Coordinator)** | |
| **Er Rabin Maharjan (Lecturer)** | |

**Department of Computer Science**

## Page replacement algorithm:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

**Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, the Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults. There are different type of page replacement algorithm but here we are discussing about four different type of algorithm and they are:

1. FIFO (First In First Out)
2. Optimal Replacement Algorithm
3. Least recently used (LRU)
4. Least Frequently Used(LFU)

### 1. First In First Out(FIFO)

**Idea:** This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.

When a page needs to be replaced, the page in the front of the queue is selected for removal.

**ALGORITHM**

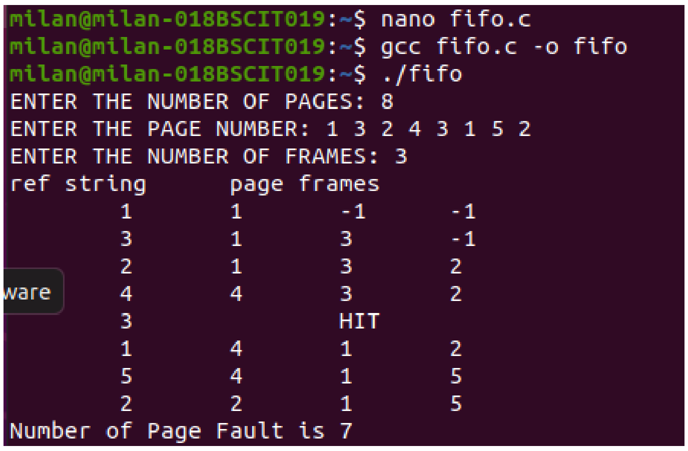
1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Forma queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

**Source code:**

```
#include<stdio.h>
int main(void){
        int i,j=0,n,a[50],frame[10],no,k,avail,count=0;
        printf("ENTER THE NUMBER OF PAGES: "); scanf("%d",&n);
        printf("ENTER THE PAGE NUMBER: ");
        for(i=1;i<=n;i++) scanf("%d",&a[i]);
        printf("ENTER THE NUMBER OF FRAMES: "); scanf("%d",&no);
        for(i=0;i<no;i++) frame[i]= -1;
```

```
	printf("ref string\tpage frames\n");
	for(i=1;i<=n;i++){
		printf("\t%d\t",a[i]); avail=0;
		for(k=0;k<no;k++)
			if(frame[k]==a[i]) {avail=1; printf("\tHIT");}
		if (avail==0){
			frame[j]=a[i]; j=(j+1)%no; count++;
			for(k=0;k<no;k++) printf("%d\t",frame[k]);
		}
		printf("\n");
	}
	printf("Number of Page Fault is %d\n",count);
}
```

**Screenshot of the implementation:**



```
milan@milan-018BSCIT019:~$ nano fifo.c
milan@milan-018BSCIT019:~$ gcc fifo.c -o fifo
milan@milan-018BSCIT019:~$ ./fifo
ENTER THE NUMBER OF PAGES: 8
ENTER THE PAGE NUMBER: 1 3 2 4 3 1 5 2
ENTER THE NUMBER OF FRAMES: 3
ref string      page frames
        1       1       -1      -1
        3       1       3       -1
        2       1       3       2
        4       4       3       2
        3               HIT
        1       4       1       2
        5       4       1       5
        2       2       1       5
Number of Page Fault is 7
```

2. **Optimal Page Replacement Algorithm**

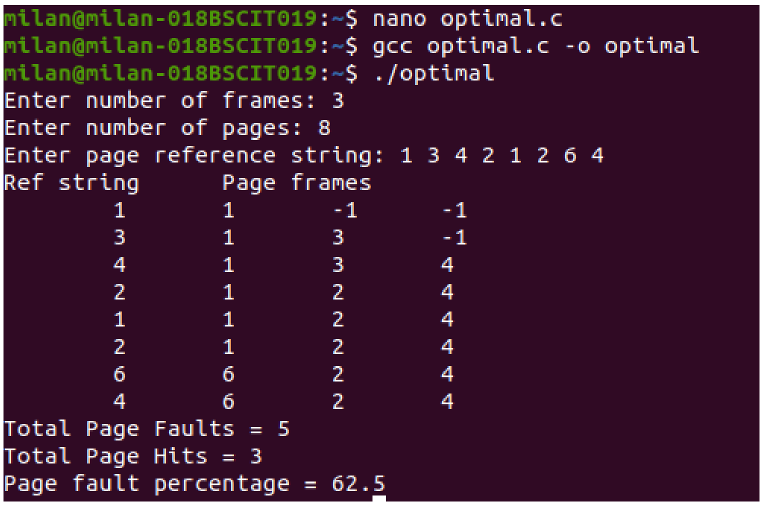
**Idea:** In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Source code**:

```
#include<stdio.h>
int main(void){
	int nf, np, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults
= 0;
	printf("Enter number of frames: "); scanf("%d", &nf);
	printf("Enter number of pages: "); scanf("%d", &np);
	printf("Enter page reference string: ");
	for(i = 0; i < np; ++i) scanf("%d", &pages[i]);
	for(i = 0; i < nf; ++i) frames[i] = -1;
	printf("Ref string\tPage frames");
	for(i = 0; i < np; ++i){
		printf("\n\t%d\t", pages[i]);
		flag1=flag2=0;
		for(j=0; j<nf; ++j)
```

```
                        if(frames[j]==pages[i]) {flag1=flag2=1; break;}
                if(flag1 == 0)
                        for(j = 0; j < nf; ++j)
                                if(frames[j] == -1){
                                        faults++; frames[j]=pages[i]; flag2=1; break;
                                }
                if(flag2 == 0){
                        flag3 =0;
                        for(j = 0; j < nf; ++j){
                                temp[j]=-1;
                                for(k = i+1; k<np; ++k)
                                        if(frames[j]==pages[k]){
                                                temp[j] = k; break;
                                        }
                        }
                        for(j = 0; j < nf; ++j)
                                if(temp[j] == -1){
                                        pos=j; flag3=1; break;
                                }
                        if(flag3 ==0){
                                max=temp[0]; pos=0;
                                for(j = 1; j < nf; ++j)
                                        if(temp[j] > max){
                                                max=temp[j]; pos=j;  }
                        }
                        frames[pos]=pages[i]; faults++;
                }
                for(j = 0; j < nf; ++j) printf("%d\t", frames[j]);
        }
        printf("\nTotal Page Faults = %d\n", faults);
        printf("Total Page Hits = %d\n",np-faults);
        fflush(stdin);
        printf("Page fault percentage = %g\n",(faults*100.0)/np);
}
```

**Screenshot of the implementation**



```
milan@milan-018BSCIT019:~$ nano optimal.c
milan@milan-018BSCIT019:~$ gcc optimal.c -o optimal
milan@milan-018BSCIT019:~$ ./optimal
Enter number of frames: 3
Enter number of pages: 8
Enter page reference string: 1 3 4 2 1 2 6 4
Ref string      Page frames
        1       1       -1      -1
        3       1       3       -1
        4       1       3       4
        2       1       2       4
        1       1       2       4
        2       1       2       4
        6       6       2       4
        4       6       2       4
Total Page Faults = 5
Total Page Hits = 3
Page fault percentage = 62.5
```

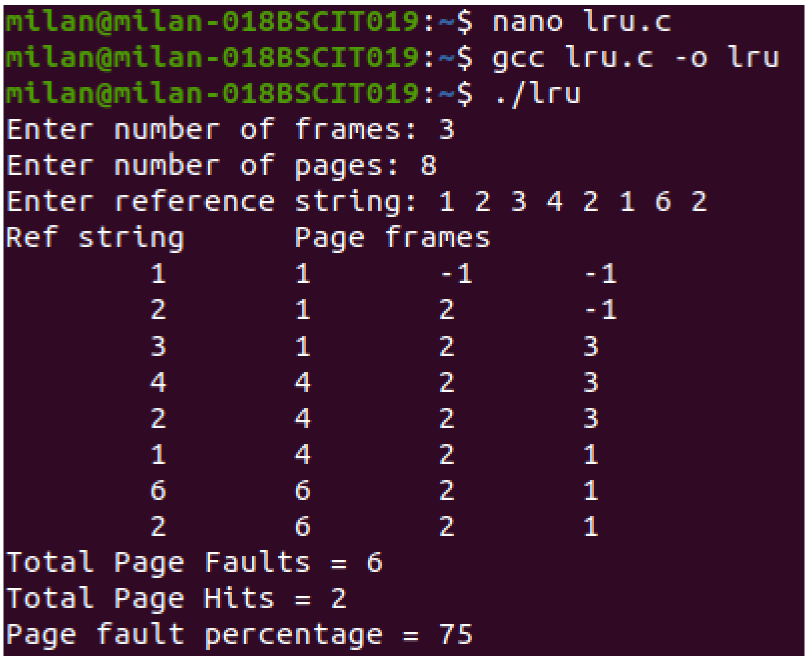
## 3. Least recently used algorithm

**Idea:** In this algorithm, when a page fault occurs, the page that has been unused for the longest time is thrown out .

**<u>Source code:</u>**

```
#include<stdio.h>
int findLRU(int time[], int n){
	int i, min=time[0], pos = 0;
	for(i = 1; i < n; ++i)
		if(time[i] < min) {min = time[i]; pos = i;}
	return pos;
}
int main(void){
	int nf, np, frames[10], pages[30], c=0, time[10], flag1, flag2, i, j, pos, faults = 0;
	printf("Enter number of frames: "); scanf("%d", &nf);
	printf("Enter number of pages: "); scanf("%d", &np);
	printf("Enter reference string: ");
	for(i = 0; i < np; ++i) scanf("%d", &pages[i]);
	for(i = 0; i < nf; ++i) frames[i] = -1;
	printf("Ref string\tPage frames");
	for(i = 0; i < np; ++i){
		printf("\n\t%d\t", pages[i]);
		flag1 = flag2 = 0;
		for(j = 0; j < nf; ++j)
			if(frames[j] == pages[i]){
				time[j]=++c; flag1=flag2=1; break;
			}
		if(flag1 == 0)
			for(j = 0; j < nf; ++j)
				if(frames[j] == -1){
					faults++; frames[j]=pages[i];
					time[j]=++c; flag2=1; break;
				}
		if(flag2 == 0){
			pos=findLRU(time, nf); faults++;
			frames[pos]=pages[i]; time[pos]=++c;
		}
		for(j=0; j<nf; ++j) printf("%d\t", frames[j]);
	}
	printf("\nTotal Page Faults = %d\n", faults);
	printf("Total Page Hits = %d\n",np-faults);
	fflush(stdin);
	printf("Page fault percentage = %g\n",(faults*100.0)/np);
}
```

**Screenshot of implementation:**



```
milan@milan-018BSCIT019:~$ nano lru.c
milan@milan-018BSCIT019:~$ gcc lru.c -o lru
milan@milan-018BSCIT019:~$ ./lru
Enter number of frames: 3
Enter number of pages: 8
Enter reference string: 1 2 3 4 2 1 6 2
Ref string      Page frames
        1       1       -1      -1
        2       1       2       -1
        3       1       2       3
        4       4       2       3
        2       4       2       3
        1       4       2       1
        6       6       2       1
        2       6       2       1
Total Page Faults = 6
Total Page Hits = 2
Page fault percentage = 75
```

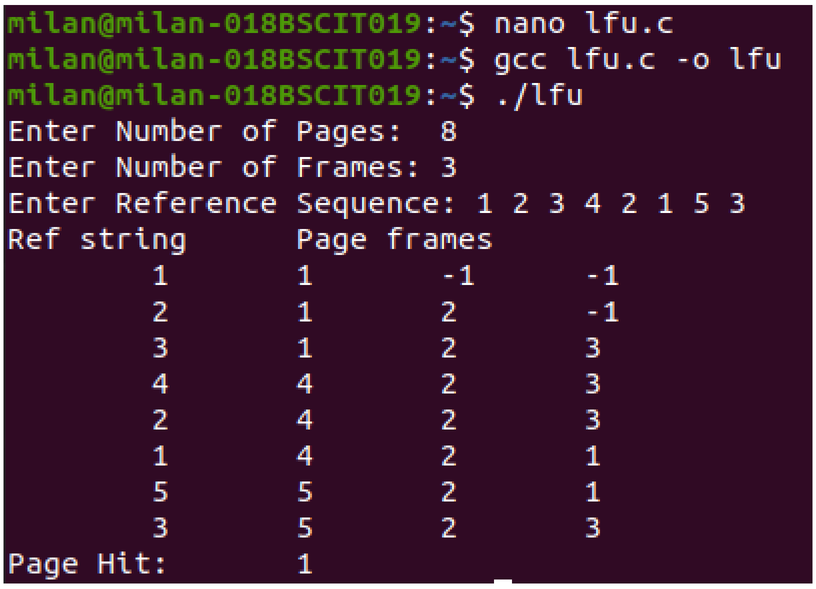
4. **Least Frequently Used (LFU)**

**Idea**: **Least Frequently Used** (**LFU**) is a type of cache algorithm used to manage memory within a computer. The standard characteristics of this method involve the system keeping track of the number of times a block is referenced in memory. When the cache is full and requires more room the system will purge the item with the lowest reference frequency.

**Source code:**

```
#include<stdio.h>
int main(void){
        int tf, tp, hit = 0, pages[25], frame[10], a[25], time[25], m, n, page, flag, k, min, temp;
        printf("Enter Number of Pages:\t"); scanf("%d", &tp);
        printf("Enter Number of Frames:\t"); scanf("%d", &tf);
        for(m = 0; m < tf; m++) frame[m] = -1;
        for(m = 0; m < 25; m++) a[m] = 0;
        printf("Enter Reference Sequence: ");
        for(m = 0; m < tp; m++) scanf("%d", &pages[m]);
        printf("Ref string\tPage frames");
        for(m = 0; m < tp; m++){
                printf("\n\t%d\t", pages[m]);
                a[pages[m]]++; time[pages[m]] = m;
                flag = 1; k = frame[0];
                for(n = 0; n < tf; n++){
                        if(frame[n] == -1 || frame[n] == pages[m]){
                                if(frame[n] != -1)  hit++;
                                flag = 0; frame[n] = pages[m]; break;
                        }
                        if(a[k]>a[frame[n]]) k=frame[n];
                }
                if(flag){
                        min = 25;
                        for(n = 0; n < tf; n++)
                                if(a[frame[n]]==a[k] && time[frame[n]]< min){
                                        temp = n; min = time[frame[n]];
                                }
                        a[frame[temp]]=0; frame[temp]=pages[m];
                }
                for(n=0; n<tf; n++) printf("%d\t", frame[n]);
        }
        printf("\nPage Hit:\t%d\n", hit);
}
```

**Screenshot of implementation:**



```
milan@milan-018BSCIT019:~$ nano lfu.c
milan@milan-018BSCIT019:~$ gcc lfu.c -o lfu
milan@milan-018BSCIT019:~$ ./lfu
Enter Number of Pages:  8
Enter Number of Frames: 3
Enter Reference Sequence: 1 2 3 4 2 1 5 3
Ref string      Page frames
        1       1       -1      -1
        2       1       2       -1
        3       1       2       3
        4       4       2       3
        2       4       2       3
        1       4       2       1
        5       5       2       1
        3       5       2       3
Page Hit:       1
```

**Final comparison:**

From all of the above comparisons we can say that the best solution is given by the **Optimal page** replacement algorithm and the least best solution is given by the **First In First Out** page replacement algorithm.

**Conclusion:**

Hence, we can conclude that the page replacement algorithm can be successfully implemented by using C language.