

Programming Assignment 4

ReadMe

Page replacement algorithms are an essential part of completing the illusion of infinite virtual memory to the user by masking the need for reading pages from the disk for every execution. In this assignment, you will implement two algorithms First In First Out (FIFO) and Least Recently Used (LRU) and measure their performance on three benchmark datasets that are actual traces of the disk operations performed when using real-world applications such as unzipping a file using bzip and compiling a C program using *gcc*. Each file will have two values, the first correspond to the address of the frame being referenced and the second corresponds to the type of operation (**R**ead or **W**rite). Your program must take in three arguments – the file name, the algorithm type and the number of frames in memory for the process. Your output must be the number of disk reads and disk writes performed and the contents of the frames at the end of the simulation.

Example usage:

```
./virtual_memory <trace file> <nframes> <fifo|lru>
```

Source Code:

```
//Name: Achyutha Santhoshi
```

```
//CWID: A20314248
```

```
//CS 5323 Operating Systems 2
```

```
//Programming Assignment 4
```

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

```
#include <stdlib.h>
```

```
#include<string.h>
```

```
#include <stdbool.h>
```

```
bool dirty;
```

```
struct Page
```

```
{
```

```
    unsigned long value;
```

```
    struct Page *next;
```

```
};
```

```
struct Page *head = NULL;
```

```
struct Page *current = NULL;
```

```
struct Page* search_list(unsigned long value, struct Page  
**previous)
```

```
{
```

```
    struct Page *ptr = head;
```

```
    struct Page *temp = NULL;
```

```
    while (ptr != NULL)
```

```
    {
```

```
        if (ptr->value == value)
```

```
        {
```

```
            if(previous)
```

```
                *previous = temp;
```

```
            return ptr;
```

```
        }
```

```
    else
```

```
    {
```

```
        temp = ptr;
```

```
        ptr = ptr->next;
```

```
    }
```

```
}
```

```
return NULL;
```

```
}
```

```
// create list a list
```

```
struct Page* create_list(unsigned long value)
```

```
{
```

```
    struct Page *ptr = (struct Page*) malloc(sizeof(struct  
Page));
```

```
    if (ptr == NULL)
```

```
    {
```

```
        fprintf(stderr, "Page creation failed!");
```

```
        return NULL;
```

```
    }
```

```
    ptr->value = value;
```

```
    ptr->next = NULL;
```

```
    head = current = ptr;
```

```
    return ptr;
```

```
}
```

```
struct Page* insert_front(unsigned long value)
```

```
{
```

```
    if(head == NULL)
```

```
    {
```

```
        return (create_list(value));
```

```
    }
```

```

    struct Page *ptr = (struct Page*)malloc(sizeof(struct
Page));
    if (ptr == NULL)
    {
        fprintf(stderr, "Page creation failed!");
        return NULL;
    }
    ptr->value = value;
    ptr->next = head;
    head = ptr;
    return ptr;
}

```

```

struct Page* insert_tail(unsigned long value)
{
    if(head == NULL)
    {
        return (create_list(value));
    }
}

```

```

    struct Page *ptr = (struct Page*)malloc(sizeof(struct
Page));
    if (ptr == NULL)
    {
        fprintf(stderr, "Page creation failed!");
    }
}

```

```
        return NULL;
    }
    ptr->value = value;
    ptr->next = NULL;
    current->next = ptr;
    current = ptr;
    return ptr;
}
```

```
int remove_list(unsigned long value)
{
    struct Page *delet = NULL;
    struct Page *previous = NULL;

    delet = search_list(value, &previous);

    if(delet == NULL)
        return -1;

    else
    {
        if(previous != NULL)
            previous->next = delet->next;
        if(delet == current)
            current = previous;
    }
}
```

```
        if (delet == head)
            head = delet->next;
    }
```

```
    free(delet);
    delet = NULL;
    return 0;
}
```

```
int fifo(int table_size, unsigned long page_request)
{
    struct Page *ptr = NULL;
    struct Page *previous = NULL;
    int count = 0;
    ptr = search_list(page_request, &previous);

    // page fault
    if (ptr == NULL)
    {
        ptr = head;
        // Track count of size
        while (ptr != NULL)
        {
            ptr = ptr->next;
            count++;
        }
    }
}
```

```
}
```

```
if (count == table_size)
```

```
{
```

```
    ptr = head;
```

```
    insert_tail(page_request);
```

```
    remove_list(ptr->value);
```

```
}
```

```
// Insert at back
```

```
else
```

```
{
```

```
    insert_tail(page_request);
```

```
}
```

```
return 1;
```

```
}
```

```
return 0;
```

```
}
```

```
int lru(int table_size, unsigned long page_request)
```

```
{
```

```
    struct Page *ptr = NULL;
```

```
    struct Page *previous = NULL;
```

```
ptr = search_list(page_request, &previous);
int count = 0;

if (ptr == NULL)
{
    ptr = head;
    while (ptr != NULL)
    {
        ptr = ptr->next;
        count++;
    }
    while (count >= table_size)
    {
        remove_list(current->value);
        count--;
    }
    insert_front(page_request);

    return 1;
}
else
{
    remove_list(page_request);
    insert_front(page_request);
}
```



```
    return 0;
}

int main(int argc, char *argv[])
{
    unsigned long bit;
    unsigned long one;
    unsigned long two;
    unsigned long three;
    int countn = 0;
    char rw;
    int size=10;
    int table_size = atoi(argv[2]);
    int page_request;
    int num_requests = 0;
    unsigned long frame[50];
    int i = 0;
    FILE *fp = fopen(argv[1], "r");
    char buff[size];
    char str[128];
    int c =0, r=0, w=0;
```

```

char *op;
while(!feof(fp))
{
    fscanf(fp,"%lx %c",&bit,&rw);
    //printf("%lx\n",bit);
    //printf ("%s\n", buff);
    //strcpy (str, buff);
    //strtok(str, " ");
    //printf("%s",str);
    //int buff1 = atoi(buff);
    //printf("%d\n", buff1);
    if(strcmp(argv[3],"fifo")==0){
        if(fifo(table_size,bit))
        {
            //printf("%lx\t", bit);
            if(countn==0)
            {
                one=bit;
                //printf("%lx this is one\n", one);

            }
            if(countn==1)
            {
                two=bit;
                //printf("%lx is two\n", two);
            }
        }
    }
}

```

```

    }
    if(countn==2)
    {
        three=bit;
        //printf("%lx is three\n", three);
    }
    countn++;
    if(countn==3)
    {
        countn =0;
    }
    if(rw=='R')
    {
        dirty =0;
        r++;
    }
    if(rw=='W')
    {
        dirty = 1;
        w++;
    }
}
}

```

```

if(strcmp(argv[3],"lru")==0){
if(lru(table_size,bit))

```

```

{
    //printf("%lx\t", bit);
    if(countn==0)
    {
        one=bit;
        //printf("%lx this is one\n", one);

    }
    if(countn==1)
    {
        two=bit;
        //printf("%lx is two\n", two);
    }
    if(countn==2)
    {
        three=bit;
        //printf("%lx is three\n", three);
    }
    countn++;
    if(countn==3)
    {
        countn =0;
    }
    if(rw=='R')
    {

```

```

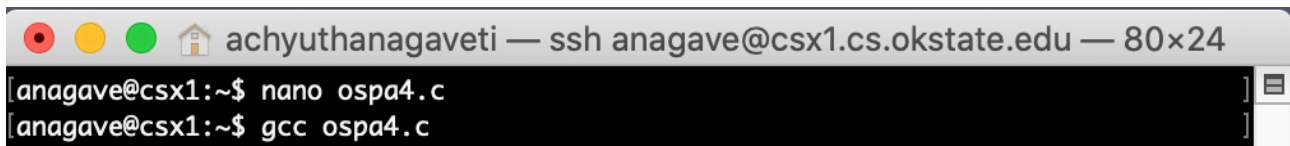
        r++;
    }
    if(rw=='W')
    {
        w++;
    }
}
}

printf("Contents of page frames\n");
printf("%lx\t%lx\t%lx",one, two, three);
//printf("%d",countn);
//printf("%d",(char)frame[i-1]);
//printf("%s")
printf("\nNumber of Reads: %d\n",r);
printf("Number of Writes: %d",w);
return 0;

}

```

Execution:



```
achyuthanagaveti — ssh anagave@csx1.cs.okstate.edu — 80x24
[anagave@csx1:~$ nano ospa4.c
[anagave@csx1:~$ gcc ospa4.c
```

Output:

```
[anagave@csx1:~$ ./a.out gcc.txt 3 fifo
Contents of page frames
2f8773e0      249db050      3d729358
Number of Reads: 637260
Number of Writes: 87755
```

```
[anagave@csx1:~$ ./a.out gcc.txt 3 lru
Contents of page frames
2f8773e0      249db050      3d729358
Number of Reads: 632018
Number of Writes: 89344
```

```
[anagave@csx1:~$ ./a.out bzip.txt 3 fifo
Contents of page frames
6645ba0 5fe5180 6645b98
Number of Reads: 474273
Number of Writes: 72143
```

```
[anagave@csx1:~$ ./a.out bzip.txt 3 lru
Contents of page frames
6645b98 6645ba0 5fe5180
Number of Reads: 470629
Number of Writes: 73976
```

Steps to Compile and Execute:

Step 1: Created a new file using nano with .c extension

Step 2: Programmed with all requirements(Explained in Next Section)

Step 3: 3.1 Using Command gcc ospa4.c to compile the program

3.2 Using Command `./a.out filename <no.of frames> <fifo/lru>` to execute

Step 4: Displays the output content of page frames and No of reads and writes

Explanation of Source Code:

- Firstly, I have created a linked list structure for frame and value.
- Then, I have created methods for it to perform
- In Main, 3 arguments are taken such as filename, no of frames, page replacement algorithm either fifo or Lru.
- Then to print the last frame contents and also to print number of reads and writes