

Homework 6

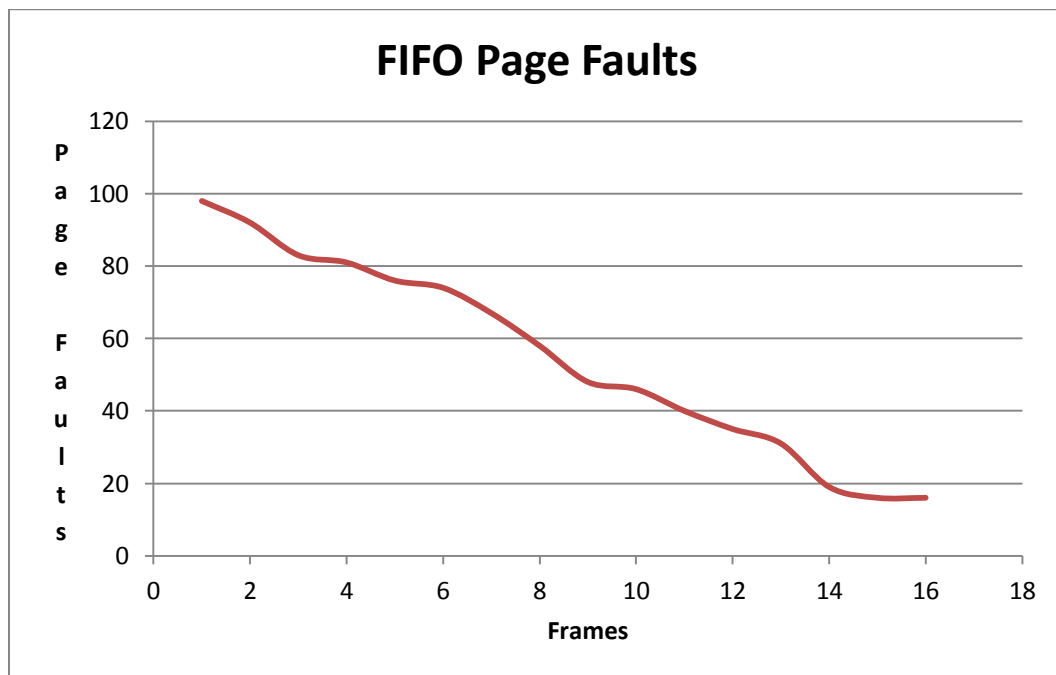
Hw6.c was built in windows. The program uses a struct called frame to keep three values, a bool value called isEmpty which identifies if the frame has a page in it, an int called page, which keeps the page number if there is a page allocated to this frame, and an int called count, which is used to select the frame used for replacement according to the FIFO(first in first out) algorithm or the LRU (last recently used) algorithm.

The program simply assigns pages as needed as long as it has free frames, once no frames are remaining a method called victimSelector() searches for the frame with the highest count and sends it to a method called pageSwap() which replaces the frames page with the needed page. If the page required is already contained in one of the frames than it is considered a hit, and now pageSwap occurs. The difference between how the two algorithms are executed is minor. FIFO increments the count of each frame every turn, and selects the frame with the highest count, thereby insuring the oldest is replaced. LRU also increments the count on every turn, however when a hit occurs that frames count is reset, insuring that the frame with the highest count is the one that was the least recently used.

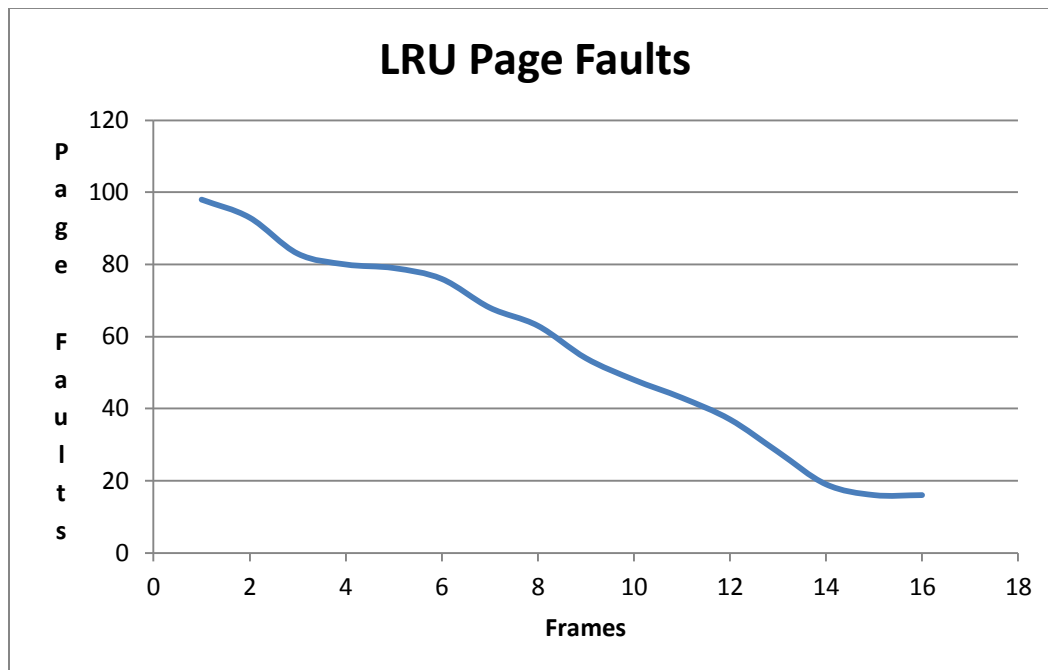
Graph Discussion

The graphs on the following page of the two algorithms are very similar, the actual numbers tended to be within 2 or 3 page faults of each other but the lead varied between the 2 algorithms around the 5 to 15 page fault range. However, I believe that 100 sample values is not a great enough pool to show which algorithm is better.

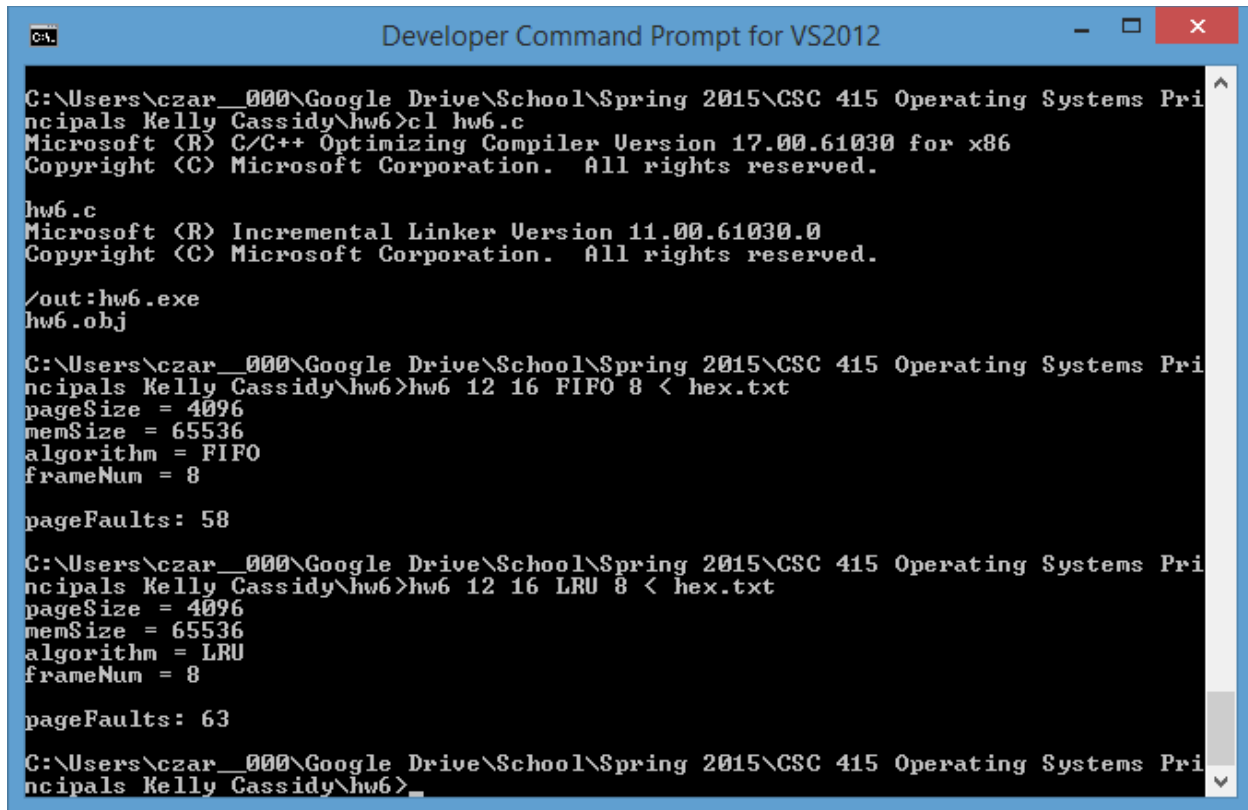
The FIFO algorithm generates the following graph. Belady's anomaly did not occur with my particular choice of hex values.



The LRU algorithm generates the following graph. LRU is immune to Belady's anomaly.



Screen cap of hw6.c compiling and executing with LRU and FIFO algorithms (8 frames)



```
C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw6>cl hw6.c
Microsoft (R) C/C++ Optimizing Compiler Version 17.00.61030 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

hw6.c
Microsoft (R) Incremental Linker Version 11.00.61030.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:hw6.exe
hw6.obj

C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw6>hw6 12 16 FIFO 8 < hex.txt
pageSize = 4096
memSize = 65536
algorithm = FIFO
frameNum = 8

pageFaults: 58

C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw6>hw6 12 16 LRU 8 < hex.txt
pageSize = 4096
memSize = 65536
algorithm = LRU
frameNum = 8

pageFaults: 63

C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw6>
```

Contents of hex.txt

7b44
398c
5fc6
4cf6
bba2
34c8
e8e3
9f59
1eb8
83f3
00f0

2ad1

3176

d54a

27ad

2e98

f845

b914

c0a0

57ef

94bc

e598

579a

06f5

3d32

bd3e

4049

d17d

6556

9247

575f

3213

5122

fed1

34b7

f69a

03b0

3da0

1bf8

4c86

81bf

51dc

6488

c6fb

4623

c575

2773

f62b

dad6

6127

adaa

39d6

b438

97b5

19dc

289a

9820

b26f

5ab2

ff6b

c4ab

8c61

2d7a

01af

b7d6

9e3a

1cb9

1b68

c1a0

543f

e766

4c42

589a

0efd

69e9

3cfb

0cc9

f8eb

86d6

2722

5a9b

ec66

af2d

5b1c

e7c2

d300

64c7

d3c7

5e96

c329

4ae0

e415

6f0a

226a

3fc6

a574

2426

5da4

92f7

5da7

Source code for hw6.c

```
/******\
| Program : hw6.c                                |
| Problem : Write a program that simulates FIFO(first in first out) |
|           and LRU (least recently used) page-replacement algorithms. |
| Purpose : Demonstrate paging algorithms FIFO and LRU                |
| Author  : Artem Tolstov                                             |
| Date   : 4/21/2015                                                  |
\*****/
#include <stdio.h>
#include <string.h>
#include <math.h>

//allows the use of 'int bool' type with true and false
typedef int bool;
#define false 0
#define true 1

void pageSwap(struct frame *currentFrame, int address);
int victimSelector(struct frame allocatedFrames[], int framesNum);

struct frame
{
    bool isEmpty;
    int page;
    int count;
};

int main(int argc, char *argv[])
{
    //Declarations
    int pageSize, memSize, framesNum, i, emptyLeft, nextValue, pageNumber, pageFaults,
    address, breakcounter;
    struct frame *allocatedFrames;
    char string[6];
    int victim;
    bool pageExists = false;

    //Initializations
    pageSize = pow(2, atoi(argv[1]));
    memSize = pow(2, atoi(argv[2]));
    framesNum = atoi(argv[4]);
    allocatedFrames = (struct frame*) malloc(sizeof(struct frame) * framesNum);
    emptyLeft = framesNum;
    pageFaults = 0;
    breakcounter = 0;

    //Makes sure there are exactly 3 commandline arguments
    if (argc != 5 || (atoi(argv[4]) < 1))
    {
        printf("proprs usage: %s int int (FIFO or LRU) (int > 0)", argv[0] );
    }
    else
    {
        //sets all .isEmpty in allocatedFrames[] as true
        for(i = 0; i < framesNum; i++)
        {

```

```

        allocatedFrames[i].isEmpty = true;
        allocatedFrames[i].count = 0;
        allocatedFrames[i].page = 'null';
    }

    //debug to see what is contained in our variables

    printf("pageSize = %d\nmemSize = %d\n", pageSize, memSize);
    printf("algorithm = %s\n", argv[3]);
    printf("frameNum = %d\n", framesNum);

    //Decide if using LRU or FIFO algorithm
    if (strcmp (argv[3], "FIFO") == 0)
    {
        //printf("string is FIFO\n");
        //1.Ask and receive input
        while(1)
        {
            //printf("\nEnter hex values: ");
            fgets(string, 6, stdin);
            if (strcmp(string, "\n") == 0)
            {
                break;
            }
            else
            {
                sscanf(string, "%x", &address);
            }
            if(address > memSize - 1)
            {
                printf("Requested address greater than %d.\n", memSize
- 1);
                break;
            }
            //printf("Got %d\n", address); //Debugging purposes

            //2.Figure out which page input belongs in and set as
pageNumber
            pageNumber = address/pageSize;
            //printf("pageNumber = %d\n", pageNumber);

            //3.Check if page number already exists in frames
            pageExists = false;
            for(i = 0; i < framesNum && pageExists == false; i++)
            {
                //if page exists set pageExists to true
                if(pageNumber == allocatedFrames[i].page)
                {
                    pageExists = true;
                    //printf("Page %d already in
allocatedFrames[%d].\n", pageNumber, i); //for debugging purposes
                }
            }
            if (pageExists == false)
            {
                pageFaults++; //Keep track of pageFaults
            }
        }
    }
}

```

```

//4.If pageExists == false choose victim frame
if (pageExists == false)
{
    //choose victim frame
    //only executes if empty frames left
    if(emptyLeft > 0)
    {
        //printf("looking for empty frame.\n");
        for(i = 0; i < framesNum; i++)
        {
            if(allocatedFrames[i].isEmpty == true)
            {
                victim = i;
                emptyLeft--;
                break;
            }
            //printf("part outside break
executes\n");
        }
        //chooses victim when no more empty frames left
        else
        {
            victim = victimSelector(allocatedFrames,
framesNum);

            //printf("victim selected: %d\n", victim);
            //printf("victim.page: %d\n",
allocatedFrames[victim]);

            //use pageSwap() to replace victim with pageNumber
            pageSwap(&allocatedFrames[victim], pageNumber);
        }
        //5.Increment count of every frame in array, only matters when
new page is swapped in, hence inside if statement
        for(i = 0; i < framesNum; i++)
        {
            allocatedFrames[i].count++;
        }
        //6.Display statues of pageFaults & frames
        //printf("\npageFaults: %d\n", pageFaults);
        /*
        for(i = 0; i < framesNum; i++)
        {
            printf("Frame: %d Page: %d isEmpty: %d ", i,
allocatedFrames[i].page, allocatedFrames[i].isEmpty);
            printf("allocatedFrames[%d].count: %d\n", i,
allocatedFrames[i].count);
        }
        */
        printf("\npageFaults: %d\n", pageFaults);
    } //end of fifo if statement
    else if (strcmp(argv[3], "LRU") == 0)
    {
        //printf("string is LRU\n");//1.Ask and receive input
        while(1)

```

```

{
    /*
    for(i = 0; i < framesNum; i++)
    {
        printf("Frame: %d Page: %d isEmpty: %d\n", i,
allocatedFrames[i].page, allocatedFrames[i].isEmpty);
    }*/
    //printf("\nEnter hex values: ");
    fgets(string, 6, stdin);
    if (strcmp(string, "\n") == 0)
    {
        break;
    }
    else
    {
        sscanf(string, "%x", &address);
    }
    if(address > memSize - 1)
    {
        printf("Requested address greater than %d.\n", memSize
- 1);
        break;
    }
    //printf("Got %d\n", address); //Debugging purposes

    //2.Figure out which page input belongs in and set as
pageNumber
    pageNumber = address/pageSize;
    //printf("pageNumber = %d\n", pageNumber);

    //3.Check if page number already exists in frames
    pageExists = false;
    for(i = 0; i < framesNum && pageExists == false; i++)
    {
        //if page exists set pageExists to true
        if(pageNumber == allocatedFrames[i].page)
        {
            pageExists = true;
            //printf("Page %d already in
allocatedFrames[%d].\n", pageNumber, i); //for debugging purposes
            allocatedFrames[i].count = 0;
        }
    }
    if (pageExists == false)
    {
        pageFaults++; //Keep track of pageFaults
    }

    //4.If pageExists == false choose victim frame
    if (pageExists == false)
    {
        //choose victim frame
        //only executes if empty frames left
        if(emptyLeft > 0)
        {
            //printf("looking for empty frame.\n");
            for(i = 0; i < framesNum; i++)
            {

```

```

        if(allocatedFrames[i].isEmpty == true)
        {
            victim = i;
            emptyLeft--;
            break;
        }
        //printf("part outside break
executes\n");
    }
}
//chooses victim when no more empty frames left
else
{
    victim = victimSelector(allocatedFrames,
framesNum);

    //printf("victim selected: %d\n", victim);
    //printf("victim.page: %d\n",
allocatedFrames[victim]);
}
//use pageSwap() to replace victim with pageNumber
pageSwap(&allocatedFrames[victim], pageNumber);
}
//5.Increment count of every frame in array, only matters when
new page is swapped in, hence inside if statement
for(i = 0; i < framesNum; i++)
{
    allocatedFrames[i].count++;
}
//6.Display statues of pageFaults & frames
//printf("\npageFaults: %d\n", pageFaults);
/*
for(i = 0; i < framesNum; i++)
{
    printf("Frame: %d Page: %d isEmpty: %d ", i,
allocatedFrames[i].page, allocatedFrames[i].isEmpty);
    printf("allocatedFrames[%d].count: %d\n", i,
allocatedFrames[i].count);
}
*/
}printf("\npageFaults: %d\n", pageFaults);
}
else {
    printf("must select FIFO or LRU\n");
}
return 0;
}
}

void pageSwap(struct frame *currentFrame, int address)
{
    currentFrame->isEmpty = false;
    currentFrame->page = address;
    currentFrame->count = 0;
}

int victimSelector(struct frame allocatedFrames[], int framesNum)

```

```
{  
    int i;  
    int victim;  
    int count = 0;  
  
    for(i = 0; i < framesNum; i++)  
    {  
        if (count < allocatedFrames[i].count)  
        {  
            count = allocatedFrames[i].count;  
            victim = i;  
            //printf("VictimSelector count becomes %d\n", count);  
        }  
    }  
  
    return victim;  
}
```