

Computer Graphics (UCS505)

Project on LRU Page Replacement Algorithm

Submitted By

Harish Pariyar 102103011

Devansh Gupta 102103028

3CO1

B.E. Third Year – COE

Submitted To:

Ms. Anupam Garg



Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Sr. No.	Description	Page No.
1.	Introduction to Project	3-4
2.	Computer Graphics concepts used	56
3.	User Defined Functions	7
4.	Code	8-16
5.	Output/ Screen shots	17-21

1.0 Introduction

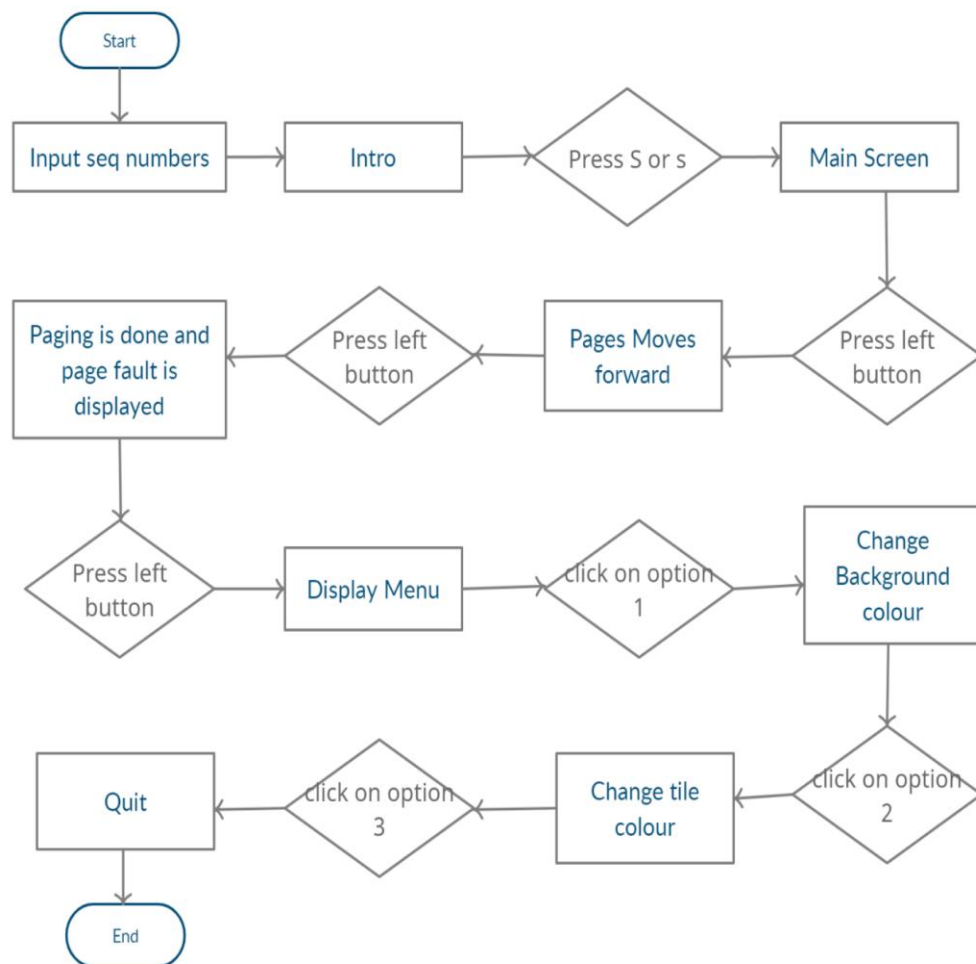
In the realm of computer science and operating systems, memory management plays a pivotal role in optimizing system performance, particularly in scenarios where the available memory is insufficient to accommodate all active processes. Among various memory management techniques, the Least Recently Used (LRU) page replacement algorithm stands out as a fundamental approach for managing memory efficiently, especially in systems utilizing virtual memory.

The primary objective of the LRU algorithm is to determine which page to replace in memory when a new page needs to be brought in. Unlike simpler algorithms like First In First Out (FIFO), which replace the oldest page, LRU replaces the page that has not been accessed for the longest time. This approach is based on the principle of temporal locality, which suggests that recently accessed pages are more likely to be accessed again in the near future.

The LRU algorithm maintains a record of the order in which pages are accessed. When a page needs to be replaced, the algorithm selects the page that was least recently used, ensuring that the pages most likely to be accessed again remain in memory. This selection process often involves keeping track of page access timestamps or using specialized data structures such as linked lists or caches to efficiently determine the least recently used page.

Despite its effectiveness in capturing temporal locality and minimizing page faults, implementing the LRU algorithm can pose challenges in terms of computational overhead and memory requirements. Efficient data structures and algorithms are crucial for maintaining the access order of pages, especially in systems with large memory sizes and high access rates.

In conclusion, the Least Recently Used (LRU) page replacement algorithm is a key component of modern memory management systems, offering a balance between simplicity and effectiveness in maximizing system performance by prioritizing the retention of frequently accessed pages in memory.



1.1 Flow Diagram

2.0 Computer Graphics concepts used

1. **glutInit()** : glutInit is used to initialize the GLUT library. Usage: void glutInit (int *argc, char **argv);
2. **glutInitDisplayMode()** : glutInitDisplayMode sets the initial display mode. Usage: void glutInitDisplayMode (unsigned int mode);
3. **glutCreateWindow()** : glutCreateWindow creates a top-level window. Usage: int glutCreateWindow (char *name); Name-ASCII character string for use as window name.
4. **glutDisplayFunc()** : glutDisplayFunc sets the display callback for the current window. Usage: void glutDisplayFunc (void(*func)(void));
5. **glutMainLoop()** : glutMainLoop enters the GLUT event processing loop.
6. **glMatrixMode()** : The two most important matrices are the model-view and projection matrix. At many times, the state includes values for both of these matrices, which are initially set to identity matrices. There is only a single set of functions that can be applied to any type of matrix. Select the matrix to which the operations apply by first set in the matrix mode, a variable that is set to one type of matrix and is also part of the state.
7. **glutInitWindowPosition, glutInitWindowSize** **glutInitWindowPosition** and **glutInitWindowSize** set the initial window position and size respectively.
8. **glutCreateMenu** : glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the current menu is set to the newly created menu
9. **glTranslatef(GLfloat X, GLfloat Y, GLfloat Z)** : glTranslate produces a

translation by x y z. If the matrix mode is either GL_MODEL_VIEW or GL_PROJECTION, all objects drawn after a call to glTranslate.

10. glScalef(GLfloat x, GLfloat y, GLfloat z) : The glScalef function produces a general scaling along the x, y, and z axes. The three arguments indicate the desired scale factors along each of the three axes. The resulting matrix appears in the following image.

11. glPushMatrix() : glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stack contains one matrix, an identity matrix. It is an error to push a full matrix stack or pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

12. glPopMatrix() : glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stack contains one matrix, an identity matrix. It is an error to push a full matrix stack or pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

13. glutKeyboardFunc(key); GLUT provides function glutKeyboardFunc sets the keyboard callback for the current window

14. glLoadIdentity(void) : glLoadIdentity() function ensures that each time when we enter the projection mode, the matrix will be reset to identity matrix, so that the new viewing parameters are not combined with the previous one.

15. glutIdleFunc(idle)

glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received
glClearColor(red, green, blue, alpha)

3.0 User Defined Functions

1. **void output(int x,int y,char *string,int j) :** Drawtext function
2. **void front_page() :** Function for front page
3. **int lru():** Function to draw fish type 3
4. **void tile(float x, float y, char n):** Function to draw tile.
5. **void draw_flag():** Function to draw flag
6. **void setup_request() :** Function to set up request
7. **void setup_pages() :** Function to setup pages
8. **void mouse(int btn,int state,int x, int y):** Function to show mouse behaviour
9. **void handle_bg_colour(int action) :** Function to handle background colour \
- 10.**void handle_tile_colour(int action) :** Function to handle tile colour
11. **void init() :** Used to set Matrix Mode and clipping coordinates and set background color.
12. **void display1() :** Function to display front page
13. **void key(unsigned char key,int x,int y) :** Function to assign function to key
14. **void menu(int action) :** Function to create menu

4.0 Code

```
#include<stdlib.h>
#include<glut.h>
#include<stdio.h>
#include<string.h>
#include<math.h>
#define SCENE 10
int request[9] = { 0 }, counter[3] = { 0 }, pages[3] = { 0 }, fault[9] = { 0 }, colour[9] = {
1,1,1,1,1,1,1,1,1 }, pagecolour[3] = { 1,1,1 };
float assign[9] = { -5.5,-5.5,-5.5,-5.5,-5.5,-5.5,-5.5,-5.5,-5.5 };
int dest = 0, showresult = 0;
int step = -1, startani = 0, faults = 0;
char res[] = "No. of page faults = ";
float bgcolor[][3] = { { 1,0,0},{0,1,0},{0,0,1} };
int bgpointer = 0;
float tilecolor[][3] = { { 1,1,0},{1,0.7,0.7},{0,1,1} };
int tilepointer = 0;
void* fonts[] =
{
GLUT_BITMAP_9_BY_15,
GLUT_BITMAP_TIMES_ROMAN_10,
GLUT_BITMAP_TIMES_ROMAN_24,
GLUT_BITMAP_HELVETICA_12,
GLUT_BITMAP_HELVETICA_10,
GLUT_BITMAP_HELVETICA_18,
};
void output(int x, int y, const char* string, int j)
{
    int len, i;
    glColor3f(1.0f, 0.0f, 0.0f);
    glRasterPos2f(x, y);
    len = (int)strlen(string);
    for (i = 0; i < len; i++)
```



```

        glutBitmapCharacter(fonts[j], string[i]);
    }
void init()
{
    glColor3f(0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 600);
    glMatrixMode(GL_MODELVIEW);
}
int getLRU()
{
    if (counter[0] >= counter[1] && counter[0] >= counter[2]) return 0;
    if (counter[1] >= counter[0] && counter[1] >= counter[2]) return 1;
    return 2;
}
void tile(float x, float y, char n)
{
    float size = 20;
    glBegin(GL_POLYGON);
    glVertex2f(x - size, y - size);
    glVertex2f(x + size, y - size);
    glVertex2f(x + size, y + size);
    glVertex2f(x - size, y + size);
    glEnd();

    glColor3f(0, 0, 0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(x - size, y - size);
    glVertex2f(x + size, y - size);
    glVertex2f(x + size, y + size);
    glVertex2f(x - size, y + size);
    glEnd();
    glRasterPos2f(x - size / 2, y);

```

```

        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, n);
    }
    void draw_flag()
    {
        glColor3fv(bgcolor[bgpointer]);
        glBegin(GL_POLYGON);
        glVertex2f(-10, -10);
        glVertex2f(10, -10);
        glVertex2f(10, 10);
        glVertex2f(-10, 10);
        glEnd();
    }
    void setup_request()
    {
        int i;
        glColor3fv(bgcolor[bgpointer]);
        glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        glVertex2f(700, 0);
        glVertex2f(700, 100);
        glVertex2f(0, 100);
        glEnd();
        glPushMatrix();
        for (i = 0; i < 9; i++)
        {
            if (fault[i])
            {
                glPushMatrix();
                glTranslatef(70 * i, -385, 0);
                draw_flag();
                glPopMatrix();
            }
        }
    }

```

```

for (i = 0; i < 9; i++)
{
    glColor3fv(tilecolor[tilepointer]);
    glTranslatef(70, 0, 0);
    glPushMatrix();
    if (assign[i] > -4.5)
    {
        glTranslatef(70 * step - 70 * i, 0, 0);
    }
    glTranslatef(0, -250 - 45 * assign[i], 0);
    if ((int)assign[i] == dest && assign[i] >= 0)
        glColor3f(1, 0.3, 0.3);
    else
        glColor3fv(tilecolor[tilepointer]);
    tile(10, 10, request[i] + '0');
    glPopMatrix();
}
glPopMatrix();
}

```

```

void drawText(const char* string, float x, float y, float z)
{
    const char* c;
    glRasterPos3f(x, y, z);
    for (c = string; *c != '\0'; c++)
    {
        if (*c == '\n')
            glRasterPos3f(x, y - 0.05, z);
        else
            glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *c);
    }
}

```

```

    }
}

void setup_pages()
{
    glPushMatrix();
    tile(0, 0, pages[0] == 0 ? '' : pages[0] + '0');
    glTranslatef(0, -45, 0);
    tile(0, 0, pages[1] == 0 ? '' : pages[1] + '0');
    glTranslatef(0, -45, 0);
    tile(0, 0, pages[2] == 0 ? '' : pages[2] + '0');
    glPopMatrix();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPushMatrix();
    glTranslatef(120 + 70 * step, 200, 0);
    setup_pages();
    glPopMatrix();
    glColor3f(1, 1, 0);
    glPushMatrix();
    glTranslatef(40, 440, 0);
    setup_request();
    glPopMatrix();
    glEnd();
    if (showresult && step == 8)
    {
        glColor3f(0, 0, 0);
        res[21] = faults + '0';
        drawText(res, 50, 20, 0);
    }
    drawText("LRU Page Replacement Algorithm", 250, 550, 0);
    glFlush();
}

```

```

        glutSwapBuffers();
    }

    void idle()
    {
        if (!startani)
            return;
        if (dest > assign[step])
            assign[step] += 0.01;
        if (dest <= assign[step])
        {
            if (fault[step])
                pages[dest] = request[step];
            startani = 0;
            dest = -10;
            showresult = 1;
        }
        display();
    }
}

```

```

void mouse(int btn, int state, int x, int y)
{
    int n, i, j;
    if (startani == 1)
        return;
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN && step < 8)
    {
        if (step < 9)
            step++;
        for (i = 0; i < 3; i++)
        {
            if (request[step] == pages[i])
            {

```

```

        fault[step] = 0;
        counter[i] = 0;
        for (j = 0; j < 3; j++)
            if (j != i) counter[j]++;
        dest = -5;

        startani = 1;
        colour[step] = 0;
        glutPostRedisplay();
        return;
    }
}

n = getLRU();
counter[n] = 0;
for (i = 0; i < 3; i++)
    if (i != n)
        counter[i]++;

dest = n;
startani = 1;
fault[step] = 1;
faults++;
}
glutPostRedisplay();
}
void handle_bg_colour(int action)
{
    bgpointer = action;
    glutPostRedisplay();
}

void handle_tile_colour(int action)
{
    tilepointer = action;

```

```

        glutPostRedisplay();
    }
    void menu(int action)
    {
        if (action == 0)
            exit(0);
    }
    void addMenu()
    {
        int submenu1, submenu2;
        submenu1 = glutCreateMenu(handle_bg_colour);
        glutAddMenuEntry("Red", 0);
        glutAddMenuEntry("Green", 1);

        glutAddMenuEntry("Blue", 2);

        submenu2 = glutCreateMenu(handle_tile_colour);
        glutAddMenuEntry("Yellow", 0);
        glutAddMenuEntry("Light Red", 1);
        glutAddMenuEntry("Light Blue", 2);
        glutCreateMenu(menu);
        glutAddSubMenu("Background Colour", submenu1);
        glutAddSubMenu("Tile Colour", submenu2);
        glutAddMenuEntry("Quit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
    }
    void display1()
    {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        front_page();
        glFlush();
        glutSwapBuffers();
    }
    void key(unsigned char key, int x, int y)

```

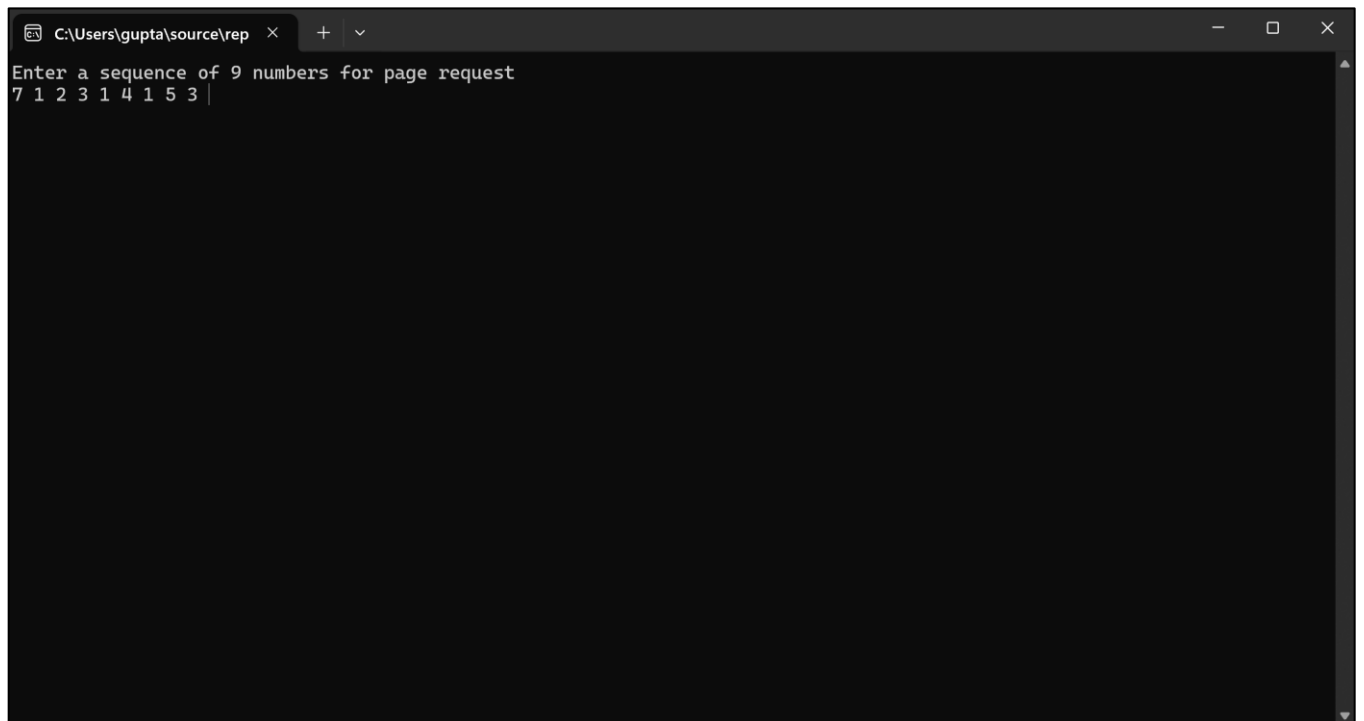
```

{
    switch (key)
    {
        case 'S':
        case 's':glutDisplayFunc(display);
            break;
        case 'Q':
        case 'q':exit(0);
    }
    glutPostRedisplay();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    int i;
    printf("Enter a sequence of 9 numbers for page request\n");
    for (i = 0; i < 9; i++)
        scanf_s("%d", &request[i]);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(1000, 800);
    glutCreateWindow("COH");

    glutKeyboardFunc(key);
    glutDisplayFunc(display1);
    glutMouseFunc(mouse);
    glutIdleFunc(idle);
    glClearColor(1, 1, 1, 1);
    init();
    addMenu();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 600);
    glMatrixMode(GL_MODELVIEW);
    glutMainLoop();
    return 0;
}

```


5.0 Input



A screenshot of a terminal window with a dark background. The window title bar shows the path "C:\Users\gupta\source\rep" and standard window controls. The terminal displays the prompt "Enter a sequence of 9 numbers for page request" followed by the input "7 1 2 3 1 4 1 5 3" and a cursor at the end.

```
C:\Users\gupta\source\rep x + v
Enter a sequence of 9 numbers for page request
7 1 2 3 1 4 1 5 3 |
```

5.1 Output

