Activity 9: Virtual Memory

ชื่อกลุ่ม LigmaBoy

	ชื่อ - นามสกุล	**รหัสนิสิต**
1	นายเนติภัทร โพธิพันธ์	6631331621
2	นายวรลภย์ ศรีชัยนนท์	6632200221
3	นายสิปปภาส ชวานนท์	6630333721

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจหลักการทำงานของ page fault และ page replacement
2. เพื่อให้นิสิตสามารถเปรียบเทียบการทำงานและคุณสมบัติของ page replacement algorithm แบบต่างๆ

กิจกรรมในชั้นเรียน

ให้นิสิตศึกษาการทำงานของโปรแกรม **pagefault_noreplece.c** ที่ให้ไว้ข้างล่าง
โปรแกรมนี้จำลองการทำงานของ page fault และคำนวณอัตราการเกิด page fault แต่โปรแกรมนี้ยังไม่ได้จัดการกรณีที่ไม่มี frame ว่างเหลือให้ใช้
โปรแกรมนี้เมื่อรันแล้วจะขอให้ผู้ใช้ป้อนข้อมูลสองอย่าง ได้แก่ จำนวน frame ที่มีให้ใช้ และ page reference string และถ้าตอนรันใส่ **option -v** จะพิมพ์รายละเอียดของการเกิด page fault ด้วย

ตัวอย่างการใช้โปรแกรม **pagefault_noreplace -v** เมื่อให้จำนวน frame = 3 และ page reference string = 1 2 3 1 2 4 1

```
Enter number of free frames (e.g. 3): 3
3
Enter page reference string (e.g. 1 2 3 1 2 4 1): 1 2 3 1 2 4 1
1 2 3 1 2 4 1

Page fault at page 1: allocated into frame 0
Page fault at page 2: allocated into frame 1
Page fault at page 3: allocated into frame 2
Page hit at page 1
Page hit at page 2
Page fault at page 4: No Free Frame!
Page hit at page 1
Page Fault Rate: 57.14%
```

pagefault_noreplace.c

```
// A program that simulates page faults and calculates page fault rate.
// Input: a list of page references (a series of page numbers, separated by a space).
// Output: page fault rate
// Option: -v   --> verbose mode: print the result of every page reference,
//              whether a page fault occurs, the involved page table entry, page number, and
frame number.

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#define PAGE_TABLE_SIZE 128
#define MAX_FRAMES 128

typedef struct PageTableEntry {
    uint16_t valid : 1;
    uint16_t frame : 15;
} PageTableEntry;

PageTableEntry page_table[PAGE_TABLE_SIZE];
int frames[MAX_FRAMES];
int num_frames, num_free_frames;


int get_free_frame(int page_number) {
    if (num_free_frames > 0) {
            // Get the first free frame
            for (int i = 0; i < num_frames; i++) {
                if (frames[i] == -1) {
                        frames[i] = page_number;
                        num_free_frames--;
                        return i;
                }
            }
    }
    else {
        return -1; // No free frame available
    }
}

int main(int argc, char *argv[]) {
    char buf[5];
    int page_faults = 0, page_references = 0;
    char page_reference_string[1024];
    int verbose = 0;

    // Parse command line arguments
    if (argc > 1 && strcmp(argv[1], "-v") == 0) {
        verbose = 1;
    }

    // Read in number of free frame
    printf("Enter number of free frames (e.g. 3): ");
    fgets(buf, sizeof(buf), stdin);
    num_frames = atoi(buf);
    printf("%d\n", num_frames);

    // Initialize frame list. -1 = free
    num_free_frames = num_frames;
    for (int i = 0; i < num_frames; i++) {
            frames[i] = -1;
    }

    // Read in page reference string
    printf("Enter page reference string (e.g. 1 2 3 1 2 4 1): ");
    fgets(page_reference_string, sizeof(page_reference_string), stdin);
```

```
    printf("%s\n", page_reference_string);

    // Initialize page table
    for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
        page_table[i].valid = 0;
        page_table[i].frame = 0;;
    }

    // Parse page reference string and simulate paging
    char *token = strtok(page_reference_string, " ");
    while (token != NULL) {
        int page_number = atoi(token);
        int frame_number;
        page_references++;

        // If page is not in memory, page fault occurs, try to get a free frame.
        if (page_table[page_number].valid == 0) {
                page_faults++;
                frame_number = get_free_frame(page_number);
                if (frame_number != -1) {
                        page_table[page_number].valid = 1;
                        page_table[page_number].frame = frame_number;
                        if (verbose) printf("Page fault at page %d: allocated into frame %d\n",
page_number, frame_number);
                }
                else {
                        if (verbose) printf("Page fault at page %d: No Free Frame!\n",
page_number);
                }
        }
        else {
                if (verbose) printf("Page hit at page %d\n", page_number);
        }
        token = strtok(NULL, " ");
    }

    // Calculate page fault rate
    float page_fault_rate = (float)page_faults / page_references * 100;
    printf("Page Fault Rate: %.2f%%\n", page_fault_rate);

    return 0;
}
```

<u>สิ่งที่ต้องทำ</u>

โปรแกรม **pagefault_assignment.c** ที่ให้ข้างล่าง เป็นโปรแกรมที่ปรับปรุงมาจากโปรแกรม pagefault_noreplace เพื่อให้สามารถจัดการกรณีที่ไม่มี frame ว่าง ด้วยการทำ page replacement โดยใช้อัลกอริทึม First In First Out (FIFO) หรือ Least Recently Used (LRU) ซึ่งทั้งสองอัลกอริทึมมีการเก็บข้อมูล timestamp ของแต่ละ frame และเมื่อมีความจำเป็นจะต้องทำ page replacement ก็จะเลือก frame ที่เก่าที่สุด (timestamp น้อยที่สุด) ความแตกต่างของสองอัลกอริทึมนี้อยู่ที่ FIFO จะอัพเดต timestamp เมื่อมีการนำ page ใหม่เข้ามาใน frame ตอนที่เกิด page fault เพียงครั้งเดียว แต่ LRU จะอัพเดต timestamp ทุกครั้งที่มีการเข้าถึงข้อมูล

เพื่อความง่าย โปรแกรมนี้ใช้เพียงอาร์เรย์ชื่อ frames ในการเก็บข้อมูล **page_number** และ timestamp การค้นหา frame ที่ต้องการก็สามารถใช้การวนลูป

1) ให้นิสิตเติมโค้ดในส่วนที่มี comment ว่า Assignment 1.x เพื่อทำให้โปรแกรมใช้อัลกอริทึม FIFO
2) ให้นิสิตเติมโค้ดต่อจากโปรแกรมที่ได้ในข้อที่แล้ว ในส่วนที่มี comment ว่า Assignment 2 เพื่อทำให้โปรแกรมใช้อัลกอริทึม LRU

นิสิตสามารถทดสอบความถูกต้องของโปรแกรม เช่น ให้ free frame = 3 และ page reference = 1 2 3 1 2 4 1 หรือตามตัวอย่างในสไลด์ (7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1) และเทียบผลลัพธ์

```
// A program to simulates page faults and calculates page fault rate.
// Input: a list of page references (a series of page numbers, separated by a space).
// Output: page fault rate
// Options:
//   -v        --> verbose mode: print the result of every page reference
//   -a <alg>  --> choose algorithm: fifo (default) or lru

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>

#define PAGE_TABLE_SIZE 128
#define MAX_FRAMES 128

typedef struct PageTableEntry {
    uint16_t valid : 1;
    uint16_t frame : 15;
} PageTableEntry;

typedef struct OccupiedFrameEntry {
        int page_number;
        int timestamp;
} FrameEntry;

PageTableEntry page_table[PAGE_TABLE_SIZE];
FrameEntry frames[MAX_FRAMES];
int num_frames, num_free_frames;

int get_free_frame(int page_number, int timestamp) {
    if (num_free_frames > 0) {
        // Get the first free frame
        for (int i = 0; i < num_frames; i++) {
            if (frames[i].page_number == -1) {
                    // Assignment 1.1
                    // Update frames[i], and num_free_frames



                    return i;
            }
        }
    }
    // If no free frame, select one of occupied frames using the chosen algorithm
    else { // all frames are occupied
        int oldest_frame = 0;
        int min_timestamp = frames[0].timestamp;

        // Assignment 1.2
        // Find the oldest frame that is to be replaced



        // Assignment 1.3
        // invalidate the replaced page in the page table (valid=0)



        // Assignment 1.4
        // assign page number and timestamp to the selected frame (frames[oldest_frame])
```

```
        return oldest_frame;
    }
    return -1;  // Should never reach here
}

void print_usage(const char* program_name) {
    printf("Usage: %s [-v] [-a alg]\n", program_name);
    printf("Options:\n");
    printf("  -v        Enable verbose mode\n");
    printf("  -a alg    Choose algorithm: fifo (default) or lru\n");
}

int main(int argc, char *argv[]) {
    char buf[5];
    int page_faults = 0, page_references = 0;
    char page_reference_string[1024];
    int verbose = 0;
    int use_lru = 0;  // Default to FIFO
    int opt;

    // Parse command line arguments
    while ((opt = getopt(argc, argv, "va:")) != -1) {
        switch (opt) {
            case 'v':
                verbose = 1;
                break;
            case 'a':
                if (strcmp(optarg, "lru") == 0) {
                    use_lru = 1;
                } else if (strcmp(optarg, "fifo") == 0) {
                    use_lru = 0;
                } else {
                    fprintf(stderr, "Invalid algorithm: %s\n", optarg);
                    print_usage(argv[0]);
                    return 1;
                }
                break;
            default:
                print_usage(argv[0]);
                return 1;
        }
    }

    // Read in number of free frames
    printf("Enter number of free frames (e.g. 3): ");
    fgets(buf, sizeof(buf), stdin);
    num_frames = atoi(buf);
    printf("%d\n", num_frames);

    // Initialize frame list. page_number = -1 = free
    num_free_frames = num_frames;
    for (int i = 0; i < num_frames; i++) {
            frames[i].page_number = -1;
    }

    // Read in page reference string
    printf("Enter page reference string (e.g. 1 2 3 1 2 4 1): ");
    fgets(page_reference_string, sizeof(page_reference_string), stdin);
    printf("%s\n", page_reference_string);

    // Initialize page table
    for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
        page_table[i].valid = 0;
        page_table[i].frame = 0;
    }

    printf("Using %s algorithm\n", use_lru ? "LRU" : "FIFO");

    // Parse page reference string and simulate paging
```

```
    char *token = strtok(page_reference_string, " ");
    while (token != NULL) {
        int page_number = atoi(token);
        int frame_number;
        page_references++;

        // If page is not in memory, page fault occurs, try to get a free frame.
        if (page_table[page_number].valid == 0) {
                page_faults++;
                frame_number = get_free_frame(page_number, page_references); // use
page_references as timestamp
                if (frame_number != -1) {
                        page_table[page_number].valid = 1;
                        page_table[page_number].frame = frame_number;
                        if (verbose) printf("Page fault at page %d: allocated into frame %d\n",
page_number, frame_number);
                }
                else {
                        if (verbose) printf("Page fault at page %d: No Free Frame!\n",
page_number);
                }
        }
        else {
                // For LRU, update timestamp on page hits
                if (use_lru) {
                // Assignment 2
                // Update timestamp of the referenced page in the frames list



                }
                if (verbose) printf("Page hit at page %d\n", page_number);
        }
        token = strtok(NULL, " ");
    }

    // Calculate page fault rate
    float page_fault_rate = (float)page_faults / page_references * 100;
    printf("Page Fault Rate: %.2f%%\n", page_fault_rate);

    return 0;
}
```

<u>สิ่งที่ต้องส่งใน MyCourseVille</u>

1. ไฟล์โปรแกรมที่แก้ไขแล้วสำหรับ pagefault_assignment.c
2. capture หน้าจอผลลัพธ์ เมื่อรันโปรแกรมแบบ verbose และใช้ page replacement algorithm แบบ fifo และ lru
3. อธิบายเปรียบเทียบผลลัพธ์ของ fifo และ lru

จะใส่สิ่งที่ต้องส่งโดยเพิ่มลงในไฟล์นี้ หรือส่งเป็นไฟล์แยกต่างหากก็ได้

Solution Code

ไฟล์ pagefault_assignment.c

```c
// A program to simulates page faults and calculates page fault rate.
// Input: a list of page references (a series of page numbers, separated by a space).
// Output: page fault rate
// Options:
//   -v        --> verbose mode: print the result of every page reference
//   -a <alg>  --> choose algorithm: fifo (default) or lru

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>

#define PAGE_TABLE_SIZE 128
#define MAX_FRAMES 128

typedef struct PageTableEntry {
    uint16_t valid : 1;
    uint16_t frame : 15;
} PageTableEntry;

typedef struct OccupiedFrameEntry {
        int page_number;
        int timestamp;
} FrameEntry;

PageTableEntry page_table[PAGE_TABLE_SIZE];
FrameEntry frames[MAX_FRAMES];
int num_frames, num_free_frames;
```

```
int get_free_frame(int page_number, int timestamp) {
    if (num_free_frames > 0) {
        // Get the first free frame
        for (int i = 0; i < num_frames; i++) {
            if (frames[i].page_number == -1) {
                    // Assignment 1.1
                    // Update frames[i], and num_free_frames
                    frames[i].page_number = page_number;
                    frames[i].timestamp = timestamp;
                    num_free_frames--;
                    return i;
            }
        }
    }
    // If no free frame, select one of occupied frames using the chosen algorithm
    else { // all frames are occupied
        int oldest_frame = 0;
        int min_timestamp = frames[0].timestamp;

        // Assignment 1.2
        // Find the oldest frame that is to be replaced
        for (int i = 1; i < num_frames; i++) {
            if (frames[i].timestamp < min_timestamp) {
                min_timestamp = frames[i].timestamp;
                oldest_frame = i;
            }
        }

        // Assignment 1.3
        // invalidate the replaced page in the page table (valid=0)
        for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
            if (page_table[i].valid && page_table[i].frame == oldest_frame) {
                page_table[i].valid = 0;
                break;
            }
        }

        // Assignment 1.4
        // assign page number and timestamp to the selected frame (frames[oldest_frame])
        frames[oldest_frame].page_number = page_number;
        frames[oldest_frame].timestamp = timestamp;

        return oldest_frame;
    }
    return -1;  // Should never reach here
}
```

```c
void print_usage(const char* program_name) {
    printf("Usage: %s [-v] [-a alg]\n", program_name);
    printf("Options:\n");
    printf("  -v        Enable verbose mode\n");
    printf("  -a alg    Choose algorithm: fifo (default) or lru\n");
}

int main(int argc, char *argv[]) {
    char buf[5];
    int page_faults = 0, page_references = 0;
    char page_reference_string[1024];
    int verbose = 0;
    int use_lru = 0;  // Default to FIFO
    int opt;

    // Parse command line arguments
    while ((opt = getopt(argc, argv, "va:")) != -1) {
        switch (opt) {
            case 'v':
                verbose = 1;
                break;
            case 'a':
                if (strcmp(optarg, "lru") == 0) {
                    use_lru = 1;
                } else if (strcmp(optarg, "fifo") == 0) {
                    use_lru = 0;
                } else {
                    fprintf(stderr, "Invalid algorithm: %s\n", optarg);
                    print_usage(argv[0]);
                    return 1;
                }
                break;
            default:
                print_usage(argv[0]);
                return 1;
        }
    }
```

```
    // Read in number of free frames
    printf("Enter number of free frames (e.g. 3): ");
    fgets(buf, sizeof(buf), stdin);
    num_frames = atoi(buf);
    printf("%d\n", num_frames);

    // Initialize frame list. page_number = -1 = free
    num_free_frames = num_frames;
    for (int i = 0; i < num_frames; i++) {
            frames[i].page_number = -1;
    }

    // Read in page reference string
    printf("Enter page reference string (e.g. 1 2 3 1 2 4 1): ");
    fgets(page_reference_string, sizeof(page_reference_string), stdin);
    printf("%s\n", page_reference_string);

    // Initialize page table
    for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
        page_table[i].valid = 0;
        page_table[i].frame = 0;
    }

    printf("Using %s algorithm\n", use_lru ? "LRU" : "FIFO");

    // Parse page reference string and simulate paging
    char *token = strtok(page_reference_string, " ");
    while (token != NULL) {
        int page_number = atoi(token);
        int frame_number;
        page_references++;

        // If page is not in memory, page fault occurs, try to get a free frame.
        if (page_table[page_number].valid == 0) {
            page_faults++;
            frame_number = get_free_frame(page_number, page_references); // use
page_references as timestamp
            if (frame_number != -1) {
                page_table[page_number].valid = 1;
                page_table[page_number].frame = frame_number;
                if (verbose) printf("Page fault at page %d: allocated into frame %d\n",
page_number, frame_number);
            }
            else
                fprintf(stderr, "Page fault at page %d: No Free Frame!\n", page_number);
        }
```

```
        else {
            // For LRU, update timestamp on page hits
            if (use_lru) {
                // Assignment 2
                // Update timestamp of the referenced page in the frames list
                for (int i = 0; i < num_frames; i++) {
                    if (frames[i].page_number == page_number) {
                        frames[i].timestamp = page_references;
                        break;
                    }
                }
            }
            if (verbose) printf("Page hit at page %d\n", page_number);
        }
        token = strtok(NULL, " ");
    }

    // Calculate page fault rate
    float page_fault_rate = (float)page_faults / page_references * 100;
    printf("Page Fault Rate: %.2f%%\n", page_fault_rate);

    return 0;
}
```

Implementation Details

Assignment 1.1. Update frame[i] and num_free_frames

- ตรวจสอบว่ามี free frame หรือไม่ (ใน assignment 1.1. เป็นกรณีที่มี free frame)
- ลูปตั้งแต่ frame แรกไปยัง frame สุดท้าย ภายในแต่ละลูปให้ตรวจสอบว่า frame ปัจจุบันนั้นว่างหรือไม่ (ถ้า frame นั้นว่างจะมี **page_number** เท่ากับ -1)
- เมื่อ frame นั้นว่าง ให้ป้อนค่า **page_number** และ **time_number** ลงไปใน frame ปัจจุบัน พร้อมทั้งลดจำนวน free frame ลงไป 1
- return ค่าหมายเลข frame ที่กรอกค่าเข้าไป และสิ้นสุดการทำงาน

```
int get_free_frame(int page_number, int timestamp) {
    if (num_free_frames > 0) {
        // Get the first free frame
        for (int i = 0; i < num_frames; i++) {
            if (frames[i].page_number == -1) {
                    // Assignment 1.1
                    // Update frames[i], and num_free_frames
                    frames[i].page_number = page_number;
                    frames[i].timestamp = timestamp;
                    num_free_frames--;
                    return i;
            }
        }
    }
```

Assignment 1.2. Find the oldest frame that is to be replaced

- ตรวจสอบว่ามี free frame หรือไม่ (ใน assignment 1.2. เป็นกรณีที่ไม่มี free frame)
- ลูปตั้งแต่ frame แรกไปยัง frame สุดท้าย หาหมายเลข frame ที่มี timestamp น้อยที่สุด แล้วเก็บในตัวแปร **oldest_frame**

```
// If no free frame, select one of occupied frames using the chosen algorithm
else { // all frames are occupied
    int oldest_frame = 0;
    int min_timestamp = frames[0].timestamp;

    // Assignment 1.2
    // Find the oldest frame that is to be replaced
    for (int i = 1; i < num_frames; i++) {
        if (frames[i].timestamp < min_timestamp) {
            min_timestamp = frames[i].timestamp;
            oldest_frame = i;
        }
    }
```

Assignment 1.3. Invalidate the replaced page in the page table (valid = 0)

- เมื่อเราได้ frame ที่เก่าที่สุดมาแล้ว เราจะทำการแทนที่ page เก่าก่อน (เราจะ replace ค่าใน Assignment 1.4.)
- เราจำเป็นที่จะต้องให้ valid = 0 เพื่อบ่งบอกว่า page นั้นถูกแทนที่ด้วย page ใหม่ไปแล้ว CPU ไม่สามารถเรียกใช้งาน page เก่าได้เพราะมันถูกอันใหม่แทนที่ทับไปแล้ว

```
// Assignment 1.3
// invalidate the replaced page in the page table (valid=0)
for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
    if (page_table[i].valid && page_table[i].frame == oldest_frame) {
        page_table[i].valid = 0;
        break;
    }
}
```

Assignment 1.4. Assign page number and timestamp to the selected frame (frames[oldest_frame])

- แทนที่ค่า **page_number** และ **timestamp** เป็นค่าใหม่
- return ค่าหมายเลข frame อันใหม่ที่แทนค่าเข้าไป และสิ้นสุดการทำงาน

```
// Assignment 1.4
// assign page number and timestamp to the selected frame (frames[oldest_frame])
frames[oldest_frame].page_number = page_number;
frames[oldest_frame].timestamp = timestamp;
return oldest_frame;
```

Assignment 2. Update timestamp of the referenced page in the frames list

- ในกรณีที่ไม่มีการแทนที่ค่า frame ใหม่เนื่องจาก frame ที่ต้องการเรียกอยู่ใน memory อยู่แล้ว (Page Hit) ให้อัปเดตค่า timestamp เพื่อทราบว่าเราใช้งาน frame นั้นๆ ล่าสุดเมื่อไหร่
- เมื่อมีการใช้คำสั่ง verbose (-v) ให้แสดงรายละเอียดของกระบวนการด้วย

```
// For LRU, update timestamp on page hits
if (use_lru) {
    // Assignment 2
    // Update timestamp of the referenced page in the frames list
    for (int i = 0; i < num_frames; i++) {
        if (frames[i].page_number == page_number) {
            frames[i].timestamp = page_references;
            break;
        }
    }
}
if (verbose) printf("Page hit at page %d\n", page_number);
```

Result

เตรียมพร้อมก่อนการรันโปรแกรม

- Compile โปรแกรมด้วยคำสั่ง gcc page_fault_assignment.c -o page_fault_assignment
- รันไฟล์ด้วยคำสั่ง ./page_fault_assignment
- ต่อท้ายคำสั่งด้วย -v เพื่อแสดงรายละเอียดของกระบวนการทั้งหมด (verbose)
- ต่อท้ายคำสั่งด้วย -a และ page replacement algorithm ที่ต้องการใช้ (หากไม่ระบุจะใช้งาน FIFO)

รันโปรแกรมแบบ verbose และใช้ page replacement algorithm แบบ FIFO

- กำหนดให้จำนวน free frames เท่ากับ 3 frames
- กำหนดให้ลำดับการเรียกใช้งาน page คือ 1 2 3 1 2 4 1
- เมื่อเกิด page fault จะแทนที่ page ที่เก่าที่สุด (page ที่ถูกสร้างขึ้นมานานที่สุด)

Page ที่เรียกใช้งาน	Memory ปัจจุบัน	เหตุการณ์ที่เกิดขึ้น
1	[1]	เกิด page fault (page 1 ยังไม่เคยอยู่ใน RAM)
2	[1,2]	เกิด page fault (page 2 ยังไม่เคยอยู่ใน RAM)
3	[1,2,3]	เกิด page fault (page 3 ยังไม่เคยอยู่ใน RAM)
1	[1,2,3]	เกิด page hit (เรียกใช้งาน page 1 จาก frame 0)
2	[1,2,3]	เกิด page hit (เรียกใช้งาน page 2 จาก frame 1)
4	[4,2,3]	เกิด page fault (แทนที่ page 0 ที่เก่าที่สุด)
1	[4,1,3]	เกิด page fault (แทนที่ page 1 ที่เก่าที่สุด)

```
reisenx@reisenx-VirtualBox:~/Documents/GitHub/2110313-OS-SYS-PROG/Activity 09/program$ gcc pagefault_assignment.c -o pagefault_assignment
reisenx@reisenx-VirtualBox:~/Documents/GitHub/2110313-OS-SYS-PROG/Activity 09/program$ ./pagefault_assignment -v -a fifo
Enter number of free frames (e.g. 3): 3
3
Enter page reference string (e.g. 1 2 3 1 2 4 1): 1 2 3 1 2 4 1
1 2 3 1 2 4 1

Using FIFO algorithm
Page fault at page 1: allocated into frame 0
Page fault at page 2: allocated into frame 1
Page fault at page 3: allocated into frame 2
Page hit at page 1
Page hit at page 2
Page fault at page 4: allocated into frame 0
Page fault at page 1: allocated into frame 1
Page Fault Rate: 71.43%
```

จากตารางจะพบว่าเกิด page fault จำนวน 5 ครั้ง จากการเรียกใช้งาน page ทั้งหมด 7 ครั้ง

$$page\ fault\ rate = \frac{5}{7} \times 100\% = 71.43\%$$

รันโปรแกรมแบบ verbose และใช้ page replacement algorithm แบบ LRU

- กำหนดให้จำนวน free frames เท่ากับ 3 frames
- กำหนดให้ลำดับการเรียกใช้งาน page คือ 1 2 3 1 2 4 1
- เมื่อเกิด page fault จะแทนที่ page ที่ไม่ได้ถูกเรียกใช้งานนานที่สุด

Page ที่เรียกใช้งาน	Memory ปัจจุบัน	เหตุการณ์ที่เกิดขึ้น
1	[1]	เกิด page fault (page 1 ยังไม่เคยอยู่ใน RAM)
2	[1,2]	เกิด page fault (page 2 ยังไม่เคยอยู่ใน RAM)
3	[1,2,3]	เกิด page fault (page 3 ยังไม่เคยอยู่ใน RAM)
1	[1,2,3]	เกิด page hit (เรียกใช้งาน page 1 จาก frame 0)
2	[1,2,3]	เกิด page hit (เรียกใช้งาน page 2 จาก frame 1)
4	[1,2,4]	เกิด page fault (แทนที่ page 3 ไม่ได้ถูกเรียกใช้นานที่สุด)
1	[1,2,4]	เกิด page hit (เรียกใช้งาน page 1 จาก frame 0)

```
reisenx@reisenx-VirtualBox:~/Documents/GitHub/2110313-OS-SYS-PROG/Activity 09/program$ ./pagefault_assignment -v -a lru
Enter number of free frames (e.g. 3): 3
3
Enter page reference string (e.g. 1 2 3 1 2 4 1): 1 2 3 1 2 4 1
1 2 3 1 2 4 1

Using LRU algorithm
Page fault at page 1: allocated into frame 0
Page fault at page 2: allocated into frame 1
Page fault at page 3: allocated into frame 2
Page hit at page 1
Page hit at page 2
Page fault at page 4: allocated into frame 2
Page hit at page 1
Page Fault Rate: 57.14%
reisenx@reisenx-VirtualBox:~/Documents/GitHub/2110313-OS-SYS-PROG/Activity 09/program$
```

จากตารางจะพบว่าเกิด page fault จำนวน 4 ครั้ง จากการเรียกใช้งาน page ทั้งหมด 7 ครั้ง

$$page\ fault\ rate = \frac{4}{7} \times 100\% = 57.14\%$$

สรุปความแตกต่างของผลลัพธ์

- ในช่วงเริ่มต้นการทำงาน ผลลัพธ์ของ FIFO และ LRU จะเหมือนกัน เนื่องจากว่า page ที่เรียกใช้งานยังไม่เคยถูกเก็บใน RAM
- แต่ถ้ามีการเก็บ page จนเต็มจำนวน frame ที่กำหนดให้แล้ว เมื่อมีการเรียกใช้งาน page ใหม่ๆ จะต้องมีการแทนที่ page เก่า ซึ่งนี่จะเป็นจุดที่ทำให้ผลลัพธ์ของ page replacement algorithm ทั้งสองแบบให้ผลลัพธ์แตกต่างกัน