

## Activity 8: Virtual Memory

ชื่อกลุ่ม

	ชื่อ - นามสกุล	รหัสนิสิต
1	Sakdipat Sukhaneskul	6633239021
2	Sadit Wongprayon	6633233121

### วัตถุประสงค์

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

### กิจกรรมในชั้นเรียน

ให้นิสิตศึกษาการทำงานของโปรแกรม pagefault\_noreplace.c ที่ให้ไว้ข้างล่าง  
โปรแกรมนี้จำลองการทำงานของ page fault และคำนวณอัตราการเกิด page fault แต่โปรแกรมนี้อย่างไม่ได้จัดการกรณีที่ไม่มี frame วางเหลือให้ใช้  
โปรแกรมนี้นี้เมื่อรันแล้วจะขอให้ผู้ใช้ป้อนข้อมูลสองอย่าง ได้แก่ จำนวน frame ที่มีให้ใช้ และ page reference string  
และถ้าตอนรันใส่ option -v จะพิมพ์รายละเอียดของการเกิด page fault ด้วย

ตัวอย่างการใช้งานโปรแกรม 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;
}

```

## สิ่งที่ต้องทำ

โปรแกรม pagefault\_assignment.c ที่ให้ข้างล่าง เป็นโปรแกรมที่ปรับปรุงมาจากโปรแกรม pagefault\_noreplace เพื่อให้สามารถจัดการกรณีที่ไม่มี frame ว่าง ด้วยการทำ page replacement โดยใช้ อัลกอริทึม First In First Out (FIFO) หรือ Least Recently Used (LRU) ซึ่งทั้งสองอัลกอริทึมมีการเก็บข้อมูล timestamp ของแต่ละ frame และเมื่อมีความจำเป็นจะต้องทำ page replacement ก็จะต้องเลือก frame ที่เก่าที่สุด (timestamp น้อยที่สุด) ความแตกต่างของสองอัลกอริทึมนี้อยู่ที่ FIFO จะอัปเดต timestamp เมื่อมีการนำ page ใหม่เข้ามาใน frame ตอนที่เกิด page fault เพียงครั้งเดียว แต่ LRU จะอัปเดต timestamp ทุกครั้งที่มีการเข้าถึงข้อมูล

เพื่อความง่าย โปรแกรมนี้ใช้เพียงอาร์เรย์ชื่อ frames ในการเก็บข้อมูล page\_number และ timestamp การค้นหา frame ที่ต้องการก็สามารถใช้การวนลูป

- 1) ให้นิสิตเติมโค้ดในส่วนที่มี comment ว่า Assignment 1.x เพื่อให้โปรแกรมใช้อัลกอริทึม FIFO
- 2) ให้นิสิตเติมโค้ดต่อจากโปรแกรมที่ได้ในข้อที่แล้ว ในส่วนที่มี comment ว่า Assignment 2 เพื่อให้โปรแกรมใช้อัลกอริทึม 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 simulate page faults and calculate 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;
}

```

### สิ่งที่ต้องส่งใน MyCourseVille

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

## 1. ไฟล์โปรแกรมที่แก้ไขแล้วสำหรับ pagefault\_assignment.c

```
#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 = 0; i < num_frames; i++)
        {
            if (frames[i].timestamp < min_timestamp)
            {
                oldest_frame = i;
                min_timestamp = frames[i].timestamp;
            }
        }

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

```

        page_table[frames[oldest_frame].page_number].valid = 0;

        // 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
}

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
            frames[page_table[page_number].frame].timestamp = page_references;
        }
        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;
}
```

2. capture หน้าจอผลลัพธ์ เมื่อรันโปรแกรมแบบ verbose และใช้ page replacement algorithm แบบ fifo และ lru

```
> ./pagefault_assignment -v

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%
```

```
> ./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%
```

### 3. อธิบายเปรียบเทียบผลลัพธ์ของ fifo และ lru

num\_frames: 3

page\_reference: 1 2 3 1 2 4 1

#### FIFO Algorithm

Page Faults: 5

Page Hits: 2

Page Fault Rate: 71.43%

#### LRU Algorithm

Page Faults: 4

Page Hits: 3

Page Fault Rate: 57.14%

ในกรณีนี้ LRU มีอัตราการเกิด Page Fault น้อยกว่า เนื่องจาก page 1 ในตอนสุดท้าย ไม่ได้ถูกแทนที่เหมือน FIFO เพราะมีการเรียกใช้ก่อนหน้า ทำให้ความสำคัญของ page number 1 เพิ่มขึ้นจนไม่โดนแทน