Activity 9: Virtual memory

Naron Chatjitakornkul 6530113921
Pongpak Manoret 6532126421
Pupipat Singkhorn 6532142421

Source Code

```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)
        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;
```

```c
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): ");
```

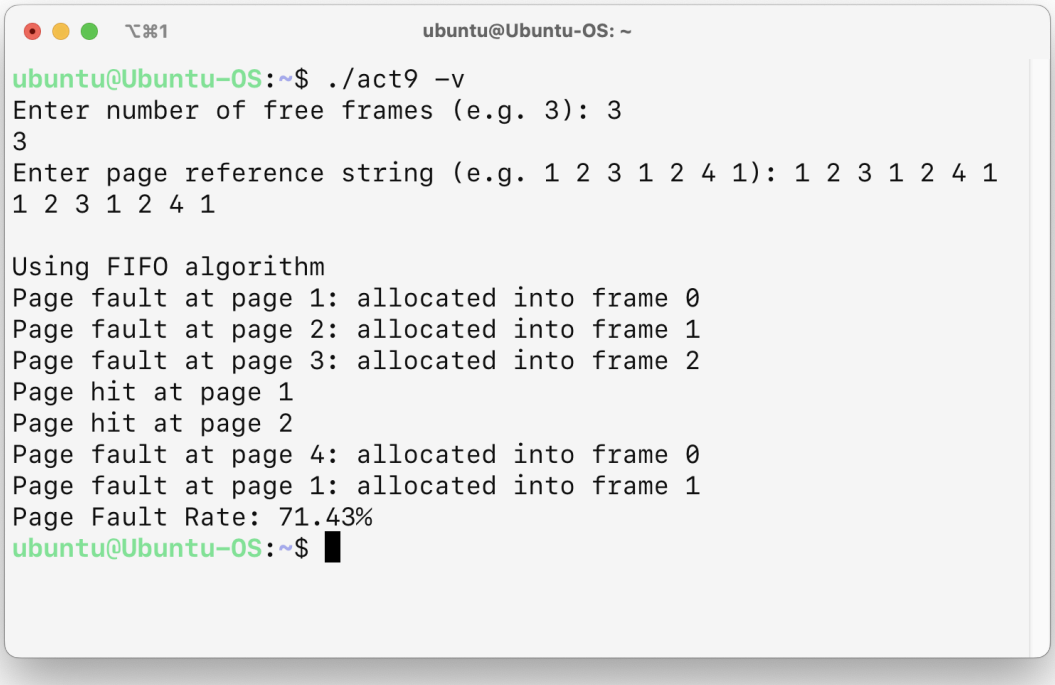
```
    fgets(buf, sizeof(buf), stdin);
    num_frames = atoi(buf);
    printf("%d\n", num_frames);

    // 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;
}
```

Output

FIFO



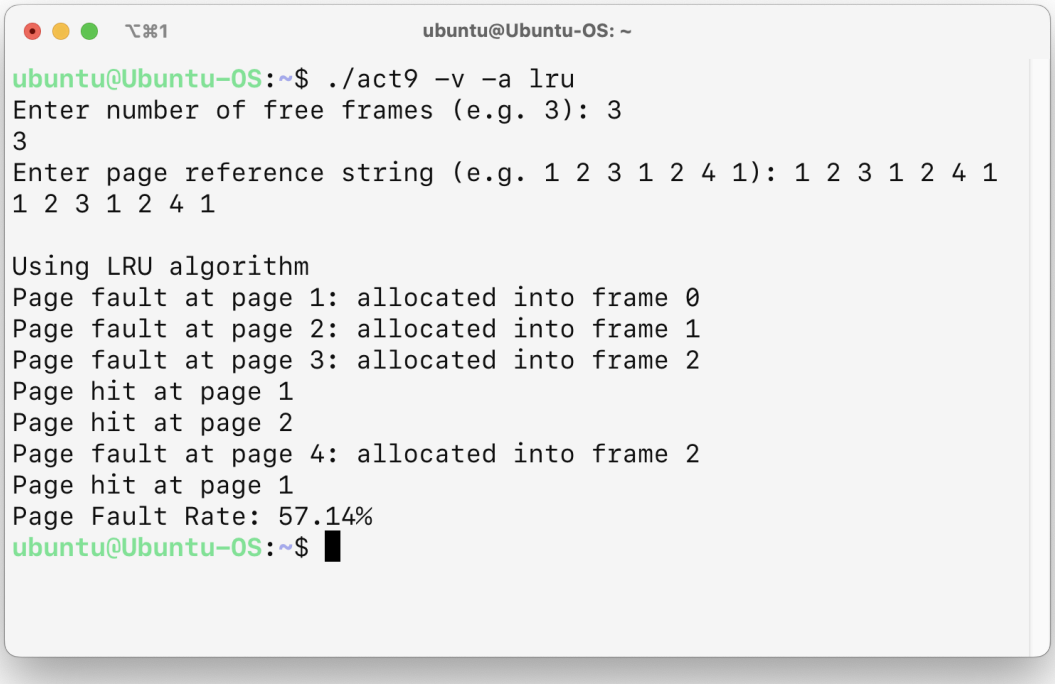
อัตราการเกิด Page Fault: 71.43%

กระบวนการทำงาน:

- โหลดหน้าเพจ 1, 2 และ 3 เข้าไปในเฟรมที่มี
- เมื่อมีการเข้าถึงหน้าเพจ 4 ซึ่งยังไม่มีในเฟรม ต้องแทนที่หน้าเพจที่เก่าที่สุด (1)
- เมื่อมีการเข้าถึงหน้าเพจ 1 อีกครั้ง เกิด Page Fault เพราะถูกลบไปก่อนหน้า

ข้อสังเกตสำคัญ: อัลกอริทึม FIFO แทนที่หน้าเพจโดยดูจากลำดับเวลาการเข้ามาแรกสุด โดยไม่คำนึงว่าหน้าเพจนั้นจะถูกใช้ล่าสุดหรือไม่

LRU



อัตราการเกิด Page Fault: 57.14%
กระบวนการทำงาน:

- โหลดหน้าเพจ 1, 2 และ 3 เข้าไปในเฟรมที่มี
- เมื่อมีการเข้าถึงหน้าเพจ 4 ซึ่งยังไม่มีในเฟรม ต้องแทนที่หน้าเพจที่ไม่ได้ถูกใช้งานนานที่สุด (3)
- เมื่อมีการเข้าถึงหน้าเพจ 1 อีกครั้ง ไม่มี Page Fault เพราะหน้านี้ยังอยู่ในหน่วยความจำ

ข้อสังเกตสำคัญ: อัลกอริทึม LRU เลือกแทนที่หน้าเพจที่ถูกใช้งานล่าสุดน้อยที่สุด ทำให้ลดจำนวน Page Fault ได้ดีกว่า

ความแตกต่างหลัก

	FIFO	LRU
อัตราการเกิด Page Fault	สูงกว่า (71.43%)	ต่ำกว่า (57.14%)
วิธีการแทนที่หน้าเพจ	แทนที่เพจที่เข้ามาก่อนสุด	แทนที่เพจที่ไม่ได้ถูกใช้ล่าสุดนานที่สุด
การอัปเดต Timestamp	อัปเดตเฉพาะตอนเพิ่มเพจใหม่	อัปเดตทุกครั้งที่มีการเข้าถึงเพจ
ประสิทธิภาพในทางปฏิบัติ	อาจแทนที่หน้าเพจที่ยังถูกใช้งานอยู่เร็วเกินไป	ปรับตัวได้ดีกับการใช้งานจริง

ข้อสรุป:

- FIFO ง่ายต่อการใช้งานแต่มีประสิทธิภาพต่ำกว่า เนื่องจากแทนที่หน้าเพจโดยไม่คำนึงถึงการใช้งานล่าสุด
- LRU มีประสิทธิภาพดีกว่า เพราะสามารถ รักษาหน้าเพจที่ถูกใช้งานบ่อยไว้ในหน่วยความจำ ส่งผลให้มีอัตรา Page Fault ต่ำกว่า
- ดังนั้น LRU เป็นอัลกอริทึมที่นิยมใช้ในระบบปฏิบัติการจริง เนื่องจากช่วยลดการเข้าถึงดิสก์และทำให้การทำงานของระบบมีประสิทธิภาพมากขึ้น