Activity 8: Memory management

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจหลักการทำงานของ address translation
2. เพื่อให้นิสิตสามารถเปรียบเทียบการทำงานและคุณสมบัติของ page table แบบต่างๆ

กิจกรรมในชั้นเรียน

ให้นิสิตศึกษาการทำงานของโปรแกรม paging_1level.c ที่ให้ข้างล่าง
โปรแกรมนี้จำลองการทำงานของ memory management แบบ paging โดยใช้ page table แบบง่าย โดยกำหนดให้

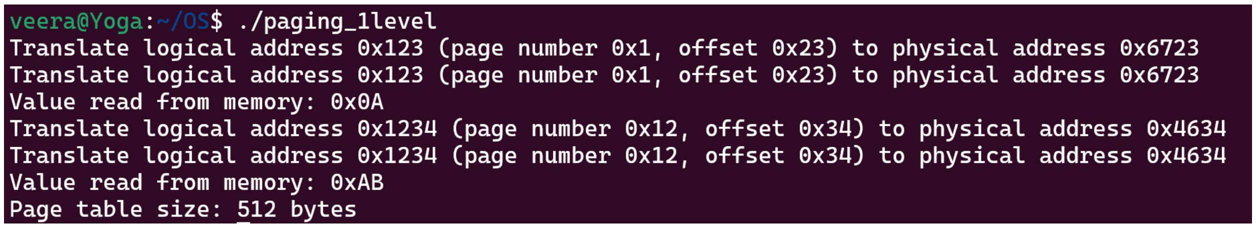
ขนาดของ physical address space = 2^{15} = 32,768 bytes
ขนาดของแต่ละ frame = 2^{8} = 256 bytes
จำนวน frame = 2^{7} = 128 frames
ขนาดของ physical address = 15 bit แบ่งเป็น frame no. 7 bit และ offset 8 bit

ขนาดของ logical address space = 2^{16} = 65,536 bytes
ขนาดของแต่ละ page = 2^{8} = 256 bytes
จำนวน page = 2^{8} = 256 pages
ขนาดของ logical address = 16 bit แบ่งเป็น page no. 8 bit และ offset 8 bit

paging_1level.c

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

#define FRAME_SIZE 256
#define FRAME_ENTRIES 128
#define PAGE_SIZE 256
#define PAGE_ENTRIES 256

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

PageTableEntry page_table[PAGE_ENTRIES];
uint8_t *physical_memory;

uint16_t translate_address(uint16_t logical_address) {
    uint8_t frame_number;
    uint8_t page_number = logical_address >> 8;

    if (page_table[page_number].present == 0) {
        // Page not present, allocate a frame for it.
        // For simplicity, just random a frame. Must fix this later.
        frame_number = rand() % FRAME_ENTRIES;

        page_table[page_number].present = 1;
        page_table[page_number].frame = frame_number;
    }

    uint16_t physical_address = (page_table[page_number].frame << 8) + (logical_address & 0xFF);

    printf("Translate logical address 0x%X (page number 0x%x, offset 0x%02x) to physical address 0x%X \n",
        logical_address, page_number, logical_address & 0xFF, physical_address);

    return physical_address;
}

void read_from_memory(uint16_t logical_address, uint8_t *value) {
    uint16_t physical_address = translate_address(logical_address);
    *value = physical_memory[physical_address];
}

void write_to_memory(uint16_t logical_address, uint8_t value) {
    uint16_t physical_address = translate_address(logical_address);
    physical_memory[physical_address] = value;
}

int main() {
    // Allocate physical memory
    physical_memory = calloc(PAGE_ENTRIES,  PAGE_SIZE);

    // Read and write to memory
    uint8_t value;
    write_to_memory(0x123, 0xA);
    read_from_memory(0x123, &value);
    printf("Value read from memory: 0x%02X\n", value);
    write_to_memory(0x1234, 0xAB);
    read_from_memory(0x1234, &value);
    printf("Value read from memory: 0x%02X\n", value);

    // Calculate page table size
    size_t page_table_size = PAGE_ENTRIES * sizeof(PageTableEntry);
    printf("Page table size: %lu bytes\n", page_table_size);

    return 0;
}
```

Output ของโปรแกรม



```
veera@Yoga:~/OS$ ./paging_1level
Translate logical address 0x123 (page number 0x1, offset 0x23) to physical address 0x6723
Translate logical address 0x123 (page number 0x1, offset 0x23) to physical address 0x6723
Value read from memory: 0x0A
Translate logical address 0x1234 (page number 0x12, offset 0x34) to physical address 0x4634
Translate logical address 0x1234 (page number 0x12, offset 0x34) to physical address 0x4634
Value read from memory: 0xAB
Page table size: 512 bytes
```

ให้นิสิตแก้ไขโปรแกรม paging_2level_dynamic.c ที่ให้ข้างล่างให้ทำงานได้อย่างถูกต้อง

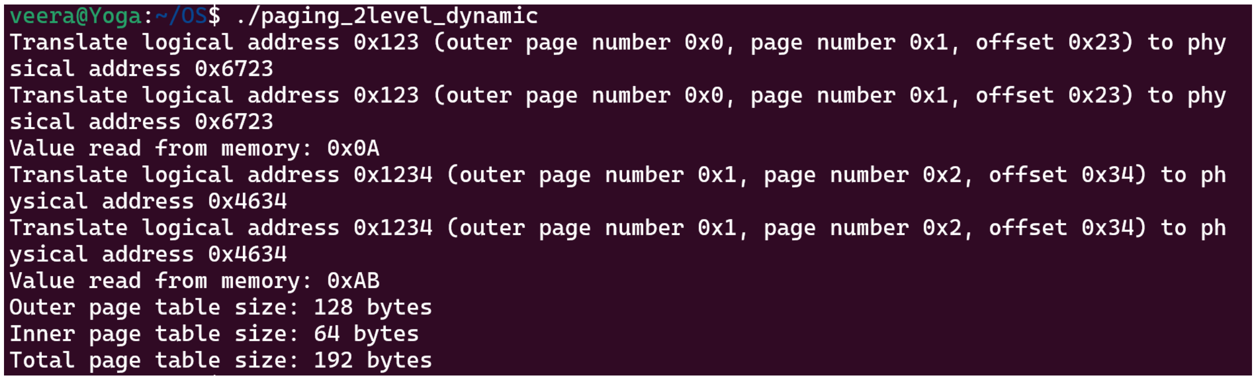
โปรแกรมนี้เป็นการปรับปรุงจากโปรแกรม paging_1level.c เพื่อให้ใช้ two-level page table ซึ่งแบ่ง page number ออกเป็นสองส่วนคือ p1 เป็น index ของ outer page table มีขนาด 4 bit (outer page table มี 16 entries) และ p2 เป็น index ของ page table มีขนาด 4 bit (page of page table แต่ละ page มี 16 entries)

โดยที่ outer page table จะถูก allocate แบบ static เมื่อโปรแกรมทำงาน แต่ inner page table จะถูก allocate แบบ dynamic เมื่อจำเป็นต้องใช้

paging_2level_dynamic.c

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

#define FRAME_SIZE 256
#define FRAME_ENTRIES 128
#define PAGE_SIZE 256
#define PAGE_ENTRIES 16
#define OUTER_PAGE_ENTRIES 16

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

PageTableEntry *page_table;
PageTableEntry *outer_page_table[OUTER_PAGE_ENTRIES];

uint8_t *physical_memory;

uint16_t translate_address(uint16_t logical_address) {

    // Assignment: complete following statements that get outer page number and page number from logical address
    uint8_t outer_page_number = ?
    uint8_t page_number = ?

    // Assignment: complete following statements that allocate inner page table
    if (outer_page_table[outer_page_number] == ?) {
        // Inner page table not present, allocate an inner page table for it
        outer_page_table[outer_page_number] = ?
    }

    if (outer_page_table[outer_page_number][page_number].present == 0) {
        // Page not present, allocate a frame for it
        // For simplicity, just random a frame. Must fix this later.
        uint16_t frame_number = rand() % FRAME_ENTRIES;

        // Assignment: complete following statements that fill in page table
        outer_page_table? = ?
        outer_page_table? = ?
    }

    // Assignment: complete following statement that constructs physical address from frame number and offset
    uint16_t physical_address = ?
    printf("Translate logical address 0x%X (outer page number 0x%X, page number 0x%X, offset 0x%X) to physical address 
0x%X\n",
        logical_address, outer_page_number, page_number, logical_address & 0xFF, physical_address);

    return physical_address;
}

void read_from_memory(uint16_t logical_address, uint8_t *value) {
    uint16_t physical_address = translate_address(logical_address);
    *value = physical_memory[physical_address];
}

void write_to_memory(uint16_t logical_address, uint8_t value) {
    uint16_t physical_address = translate_address(logical_address);
    physical_memory[physical_address] = value;
}

int main() {
    // Allocate physical memory
    physical_memory = calloc(PAGE_ENTRIES, PAGE_SIZE);

    // Read and write to memory
    uint8_t value;
    write_to_memory(0x123, 0xA);
    read_from_memory(0x123, &value);
    printf("Value read from memory: 0x%02X\n", value);
    write_to_memory(0x1234, 0xAB);
    read_from_memory(0x1234, &value);
    printf("Value read from memory: 0x%02X\n", value);


   // Calculate total size of outer page table and inner page tables
    size_t page_table_size = 0;
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++) {
        if (outer_page_table[i] != NULL) {
            page_table_size += PAGE_ENTRIES * sizeof(PageTableEntry);
        }
    }

    printf("Outer page table size: %zu bytes\n", sizeof(outer_page_table));
    printf("Inner page table size: %zu bytes\n", page_table_size);
    printf("Total page table size: %zu bytes\n", sizeof(outer_page_table)+page_table_size);

    return(0);
}
```

Output ของโปรแกรม



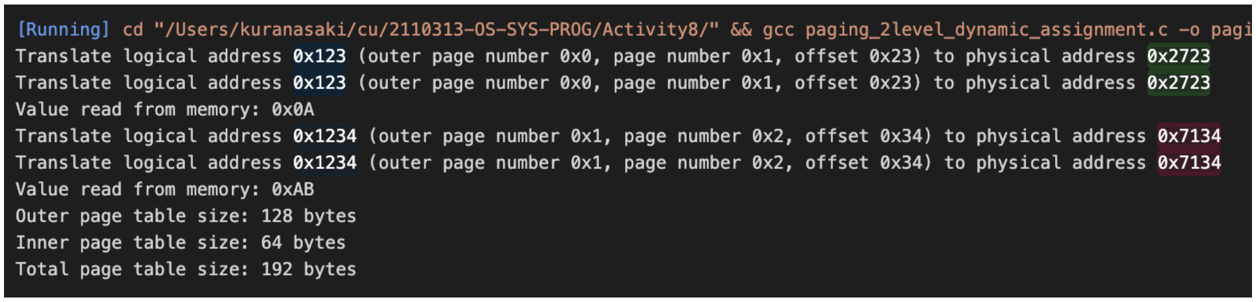
```
veera@Yoga:~/OS$ ./paging_2level_dynamic
Translate logical address 0x123 (outer page number 0x0, page number 0x1, offset 0x23) to phy
sical address 0x6723
Translate logical address 0x123 (outer page number 0x0, page number 0x1, offset 0x23) to phy
sical address 0x6723
Value read from memory: 0x0A
Translate logical address 0x1234 (outer page number 0x1, page number 0x2, offset 0x34) to ph
ysical address 0x4634
Translate logical address 0x1234 (outer page number 0x1, page number 0x2, offset 0x34) to ph
ysical address 0x4634
Value read from memory: 0xAB
Outer page table size: 128 bytes
Inner page table size: 64 bytes
Total page table size: 192 bytes
```

ให้นิสิตส่ง

1. ไฟล์โปรแกรมที่แก้ไขแล้ว

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define FRAME_SIZE 256
#define FRAME_ENTRIES 128
#define PAGE_SIZE 256
#define PAGE_ENTRIES 16
#define OUTER_PAGE_ENTRIES 16

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

PageTableEntry *page_table;
PageTableEntry *outer_page_table[OUTER_PAGE_ENTRIES];

uint8_t *physical_memory;

uint16_t translate_address(uint16_t logical_address) {

    // Assignment: complete following statements that get outer page number and page
number from logical address
    uint8_t outer_page_number = logical_address >> 12;
    uint8_t page_number = (logical_address >> 8 ) & 0x0F;
    // Assignment: complete following statements that allocate inner page table
    if (outer_page_table[outer_page_number] == NULL) {
        // Inner page table not present, allocate an inner page table for it
        outer_page_table[outer_page_number] = (PageTableEntry *)calloc(16,
sizeof(PageTableEntry));
    }

    if (outer_page_table[outer_page_number][page_number].present == 0) {
        // Page not present, allocate a frame for it
        // For simplicity, just random a frame. Must fix this later.
        uint16_t frame_number = rand() % FRAME_ENTRIES;

        // Assignment: complete following statements that fill in page table
        outer_page_table[outer_page_number][page_number].present = 1;
        outer_page_table[outer_page_number][page_number].frame = frame_number;
    }

    // Assignment: complete following statement that constructs physical address from
frame number and offset
    uint16_t physical_address =
(outer_page_table[outer_page_number][page_number].frame << 8) + (logical_address &
0xFF);

    printf("Translate logical address 0x%X (outer page number 0x%X, page number 0x%X,
offset 0x%X) to physical address 0x%X\n",
        logical_address, outer_page_number, page_number, logical_address & 0xFF,
physical_address);

    return physical_address;
}

void read_from_memory(uint16_t logical_address, uint8_t *value) {
    uint16_t physical_address = translate_address(logical_address);
    *value = physical_memory[physical_address];
}

void write_to_memory(uint16_t logical_address, uint8_t value) {
    uint16_t physical_address = translate_address(logical_address);
    physical_memory[physical_address] = value;
}

int main() {
    // Allocate physical memory
    physical_memory = calloc(PAGE_ENTRIES, PAGE_SIZE);

    // Read and write to memory
    uint8_t value;
    write_to_memory(0x123, 0xA);
    read_from_memory(0x123, &value);
    printf("Value read from memory: 0x%02X\n", value);
    write_to_memory(0x1234, 0xAB);
    read_from_memory(0x1234, &value);
    printf("Value read from memory: 0x%02X\n", value);


   // Calculate total size of outer page table and inner page tables
    size_t page_table_size = 0;
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++) {
        if (outer_page_table[i] != NULL) {
            page_table_size += PAGE_ENTRIES * sizeof(PageTableEntry);
        }
    }

    printf("Outer page table size: %zu bytes\n", sizeof(outer_page_table));
    printf("Inner page table size: %zu bytes\n", page_table_size);
    printf("Total page table size: %zu bytes\n",
sizeof(outer_page_table)+page_table_size);

    return(0);
}
```

2. capture หน้าจอผลลัพธ์



```
[Running] cd "/Users/kuranasaki/cu/2110313-OS-SYS-PROG/Activity8/" && gcc paging_2level_dynamic_assignment.c -o pagi
Translate logical address 0x123 (outer page number 0x0, page number 0x1, offset 0x23) to physical address 0x2723
Translate logical address 0x123 (outer page number 0x0, page number 0x1, offset 0x23) to physical address 0x2723
Value read from memory: 0x0A
Translate logical address 0x1234 (outer page number 0x1, page number 0x2, offset 0x34) to physical address 0x7134
Translate logical address 0x1234 (outer page number 0x1, page number 0x2, offset 0x34) to physical address 0x7134
Value read from memory: 0xAB
Outer page table size: 128 bytes
Inner page table size: 64 bytes
Total page table size: 192 bytes
```