Activity 7: Memory management

ชื่อกลุ่ม

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

วัตถุประสงค์

1. เพื่อให้นิสิตเข้าใจหลักการทำงานของ address translation
2. เพื่อให้นิสิตสามารถเปรียบเทียบการทำงานและคุณสมบัติของ page table แบบต่างๆ

กิจกรรมในชั้นเรียน

ให้นิสิตศึกษาการทำงานของโปรแกรม paging_1level.c ที่ให้ข้างล่าง
โปรแกรมนี้จำลองการทำงานของ memory management แบบ paging โดยใช้ page table แบบง่ายๆ 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;
}

// Print the current state of the page table
void print_page_table() {
    printf("Page Table State:\n");
    printf("Page Number | Present | Frame Number\n");
    printf("------------------------------------\n");

    for (int i = 0; i < PAGE_ENTRIES; i++) {
        printf("    0x%02X    |    %d    |    0x%04X\n",
               i, page_table[i].present, page_table[i].frame);
    }
}

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);
    write_to_memory(0xFF12, 0xC);
    read_from_memory(0xFF12, &value);
    printf("Value read from memory: 0x%02X\n", value);

    // Print the page table state
    print_page_table();

    // 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

```
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
Translate logical address 0xFF12 (page number 0xff, offset 0x12) to physical address 0x6912
Translate logical address 0xFF12 (page number 0xff, offset 0x12) to physical address 0x6912
Value read from memory: 0x0C
Page Table State:
Page Number | Present | Frame Number
------------------------------------
    0x00    |    0    |    0x0000
    0x01    |    1    |    0x0067
    0x02    |    0    |    0x0000
    0x03    |    0    |    0x0000
    0x04    |    0    |    0x0000
    0x05    |    0    |    0x0000
    0x06    |    0    |    0x0000
    0x07    |    0    |    0x0000
    0x08    |    0    |    0x0000
    0x09    |    0    |    0x0000
    0x0A    |    0    |    0x0000
    0x0B    |    0    |    0x0000
    0x0C    |    0    |    0x0000
    0x0D    |    0    |    0x0000
    0x0E    |    0    |    0x0000
    0x0F    |    0    |    0x0000
    0x10    |    0    |    0x0000
    0x11    |    0    |    0x0000
    0x12    |    1    |    0x0046
    0x13    |    0    |    0x0000
    0x14    |    0    |    0x0000
```

.
.
.

```
    0xFA    |    0    |    0x0000
    0xFB    |    0    |    0x0000
    0xFC    |    0    |    0x0000
    0xFD    |    0    |    0x0000
    0xFE    |    0    |    0x0000
    0xFF    |    1    |    0x0069
Page table size: 512 bytes
```

เนื่องจาก page table แบบนี้ใช้เนื้อที่หน่วยความจำเปลืองมาก จึงได้มีความพยายามปรับปรุงเป็นโปรแกรม paging_2level.c ดังนี้

- ใช้ 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 inner page table มีขนาด 4 bit (page of page table แต่ละ page มี 16 entries)
- outer page table จะถูก allocate แบบ static เมื่อโปรแกรมทำงาน แต่ inner page table จะถูก allocate แบบ dynamic เมื่อจำเป็นต้องใช้
- เพิ่มการเก็บข้อมูลของ frame ที่ถูก allocate ไปแล้วใน array ชื่อ frame_allocated ซึ่งเก็บค่า 0 เมื่อ frame ยังว่าง และ 1 เมื่อ frame ถูก allocate แล้ว และมีการเช็คค่านี้เพื่อไม่ให้เกิดการ allocate ซ้ำ

- ฟังก์ชั่น print_page_tables() พิมพ์ outer page table และ inner page table แต่ละตารางแยกกัน

paging_2level.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;
uint8_t frame_allocated[FRAME_ENTRIES]; // 0 = free, 1 = allocated

uint16_t translate_address(uint16_t logical_address) {

    // Assignment: get outer page number and page number from logical address
    uint8_t outer_page_number = ?;
    uint8_t page_number = ?;

    // Assignment: allocate inner page table
    if (outer_page_table? == ?) {
        // Inner page table not present, allocate an inner page table for it
        outer_page_table? = ?
                        printf("Allocated inner page table for outer page %d\n", 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;
        do {
            frame_number = rand() % FRAME_ENTRIES;
        } while (frame_allocated[frame_number]); // Keep trying until we find a free frame

        // Assignment: mark frame as allocated
        frame_allocated? = ?;

        // Assignment: fill in page table
        outer_page_table? = ?;
        outer_page_table? = ?;
    }

    // Assignment: construct 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;
}

// Print the current state of the page table
void print_page_tables() {
    printf("Outer Page Table:\n");
    printf("Outer Page | Inner Page Table\n");
    printf("---------------------------------------\n");

    // Print the outer page table state
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++) {
        printf("   0x%02X    | %s\n",
               i,
               outer_page_table[i] != NULL ? "address of inner page table for this entry (see below)" : "                -");
    }

    // Print the inner page tables (only for allocated tables)
    printf("\nInner Page Tables (only allocated tables):\n");
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++) {
        if (outer_page_table[i] != NULL) {
            printf("\n--- Inner Page Table for Outer Page 0x%02X ---\n", i);
            printf("Inner Page | Present | Frame Number\n");
            printf("-----------------------------------\n");

            for (int j = 0; j < PAGE_ENTRIES; j++) {
                printf("   0x%02X    |    %d    |    0x%04X\n",
                       j,
                       outer_page_table[i][j].present,
                       outer_page_table[i][j].frame);
            }
        }
    }
}

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, 0xB);
    read_from_memory(0x1234, &value);
    printf("Value read from memory: 0x%02X\n", value);
    write_to_memory(0xFF12, 0xC);
    read_from_memory(0xFF12, &value);
    printf("Value read from memory: 0x%02X\n", value);

   // Print page table
   print_page_tables();

   // 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);
}
```

สิ่งที่ต้องทำ

ให้นิสิตแก้ไขโปรแกรม paging_2level.c **ให้ทำงานได้อย่างถูกต้องตามที่กำหนด**

สิ่งที่ต้องส่งใน MyCourseVille

1. **ไฟล์โปรแกรมที่แก้ไขแล้ว**
2. capture **หน้าจอผลลัพธ์**

จะใส่สิ่งที่ต้องส่งโดยเพิ่มลงในไฟล์นี้ หรือส่งเป็นไฟล์แยกต่างหากก็ได้

```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;
uint8_t frame_allocated[FRAME_ENTRIES]; // 0 = free, 1 = allocated

uint16_t translate_address(uint16_t logical_address)
{

   // Assignment: 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 & 0xF;

   // Assignment: 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] = calloc(PAGE_ENTRIES, sizeof(PageTableEntry));
       printf("Allocated inner page table for outer page %d\n", 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;
        do
        {
            frame_number = rand() % FRAME_ENTRIES;
        } while (frame_allocated[frame_number]); // Keep trying until we find a free frame

        // Assignment: mark frame as allocated
        frame_allocated[frame_number] = 1;

        // Assignment: 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: construct 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;
}

// Print the current state of the page table
void print_page_tables()
{
    printf("Outer Page Table:\n");
    printf("Outer Page | Inner Page Table\n");
    printf("---------------------------------------\n");

    // Print the outer page table state
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++)
    {
        printf("   0x%02X    | %s\n",
               i,
               outer_page_table[i] != NULL ? "address of inner page table for this entry (see below)" : "
-");
    }

    // Print the inner page tables (only for allocated tables)
    printf("\nInner Page Tables (only allocated tables):\n");
    for (int i = 0; i < OUTER_PAGE_ENTRIES; i++)
    {
        if (outer_page_table[i] != NULL)
        {
            printf("\n--- Inner Page Table for Outer Page 0x%02X ---\n", i);
            printf("Inner Page | Present | Frame Number\n");
            printf("-----------------------------------\n");
```

```
            for (int j = 0; j < PAGE_ENTRIES; j++)
            {
                printf("   0x%02X    |    %d    |    0x%04X\n",
                       j,
                       outer_page_table[i][j].present,
                       outer_page_table[i][j].frame);
            }
        }
    }
}

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, 0xB);
    read_from_memory(0x1234, &value);
    printf("Value read from memory: 0x%02X\n", value);
    write_to_memory(0xFF12, 0xC);
    read_from_memory(0xFF12, &value);
    printf("Value read from memory: 0x%02X\n", value);

    // Print page table
    print_page_tables();

    // 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);
}
```

```
Allocated inner page table for outer page 0
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
Allocated inner page table for outer page 1
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: 0x0B
Allocated inner page table for outer page 15
Translate logical address 0xFF12 (outer page number 0xF, page number 0xF, offset 0x12) to physical address 0x5912
Translate logical address 0xFF12 (outer page number 0xF, page number 0xF, offset 0x12) to physical address 0x5912
Value read from memory: 0x0C
Outer Page Table:
Outer Page | Inner Page Table
----------------------------------------
   0x00    | address of inner page table for this entry (see below)
   0x01    | address of inner page table for this entry (see below)
   0x02    |                   -
   0x03    |                   -
   0x04    |                   -
   0x05    |                   -
   0x06    |                   -
   0x07    |                   -
   0x08    |                   -
   0x09    |                   -
   0x0A    |                   -
   0x0B    |                   -
   0x0C    |                   -
   0x0D    |                   -
   0x0E    |                   -
   0x0F    | address of inner page table for this entry (see below)

Inner Page Tables (only allocated tables):

--- Inner Page Table for Outer Page 0x00 ---
Inner Page | Present | Frame Number
------------------------------------
   0x00    |    0    |    0x0000
   0x01    |    1    |    0x0027
   0x02    |    0    |    0x0000
   0x03    |    0    |    0x0000
   0x04    |    0    |    0x0000
   0x05    |    0    |    0x0000
   0x06    |    0    |    0x0000
   0x07    |    0    |    0x0000
   0x08    |    0    |    0x0000
   0x09    |    0    |    0x0000
   0x0A    |    0    |    0x0000
   0x0B    |    0    |    0x0000
   0x0C    |    0    |    0x0000
   0x0D    |    0    |    0x0000
   0x0E    |    0    |    0x0000
   0x0F    |    0    |    0x0000
```

```
--- Inner Page Table for Outer Page 0x01 ---
Inner Page | Present | Frame Number
-----------------------------------
   0x00    |    0    |    0x0000
   0x01    |    0    |    0x0000
   0x02    |    1    |    0x0071
   0x03    |    0    |    0x0000
   0x04    |    0    |    0x0000
   0x05    |    0    |    0x0000
   0x06    |    0    |    0x0000
   0x07    |    0    |    0x0000
   0x08    |    0    |    0x0000
   0x09    |    0    |    0x0000
   0x0A    |    0    |    0x0000
   0x0B    |    0    |    0x0000
   0x0C    |    0    |    0x0000
   0x0D    |    0    |    0x0000
   0x0E    |    0    |    0x0000
   0x0F    |    0    |    0x0000

--- Inner Page Table for Outer Page 0x0F ---
Inner Page | Present | Frame Number
-----------------------------------
   0x00    |    0    |    0x0000
   0x01    |    0    |    0x0000
   0x02    |    0    |    0x0000
   0x03    |    0    |    0x0000
   0x04    |    0    |    0x0000
   0x05    |    0    |    0x0000
   0x06    |    0    |    0x0000
   0x07    |    0    |    0x0000
   0x08    |    0    |    0x0000
   0x09    |    0    |    0x0000
   0x0A    |    0    |    0x0000
   0x0B    |    0    |    0x0000
   0x0C    |    0    |    0x0000
   0x0D    |    0    |    0x0000
   0x0E    |    0    |    0x0000
   0x0F    |    1    |    0x0059
Outer page table size: 128 bytes
Inner page table size: 96 bytes
Total page table size: 224 bytes
```