**Activity 7: Memory management**

ชื่อกลุ่ม

|  |  |  |
| --- | --- | --- |
|  | ชื่อ - นามสกุล | รหัสนิสิต |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

**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 = 215 = 32,768 bytes

ขนาดของแต่ละ frame = 28 = 256 bytes

จำนวน frame = 27 = 128 frames

ขนาดของ physical address = 15 bit แบ่งเป็น frame no. 7 bit และ offset 8 bit

ขนาดของ logical address space = 216 = 65,536 bytes

ขนาดของแต่ละ page = 28 = 256 bytes

จำนวน page = 28 = 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

A screenshot of a computer

Description automatically generated

.

.

.

A white background with black and white clouds

Description automatically generated with medium confidence

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