ใบงานที่ 8

เรื่อง Main memory management

จัดทำโดย

นางสาวรัชนีกร เชื้อดี 65543206077-1

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ประจำภาคที่ 2 ปีการศึกษา 2566

ใบงานที่ 8

Main memory management

ขั้นตอนการทดลอง

1. ออกแบบโปรแกรม
2. อธิบายโค้ดโปรแกรมแยกเป็นส่วนๆ
3. อธิบายผลลัพธ์การทำงานของโปรแกรม
4. สรุปผลการทดลอง

โปรแกรมที่ 1

CPU Address Bus 24 Bit
Main Memory Page 256 byte offset = 2^8 = 8 Bit

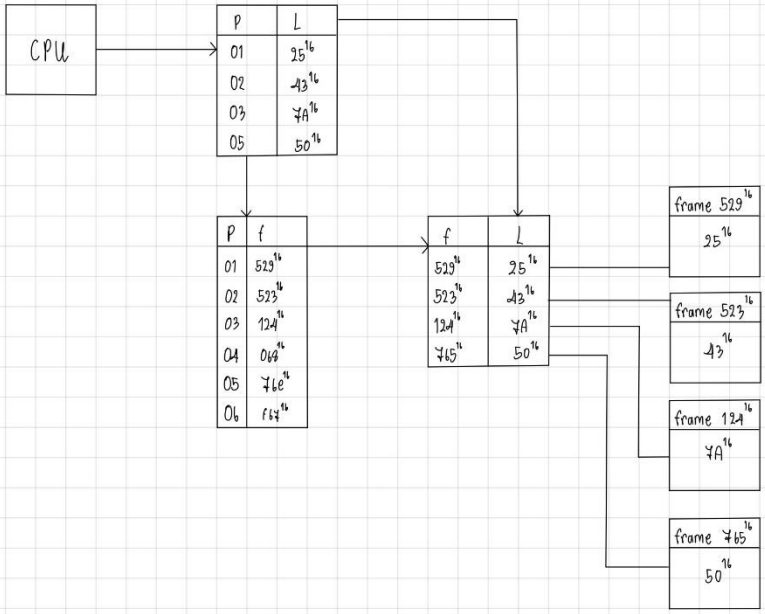
Logical Address

01	25H
02	43H
03	7AH
05	50H

Process page table

Page 1	000	0101	0010	1001	B
Page 2	000	0101	0010	0011	B
Page 3	000	0001	0010	0100	B
Page 4	000	1101	0110	1011	B
Page 5	000	0111	0110	1100	B
Page 6	000	1111	0110	0111	B

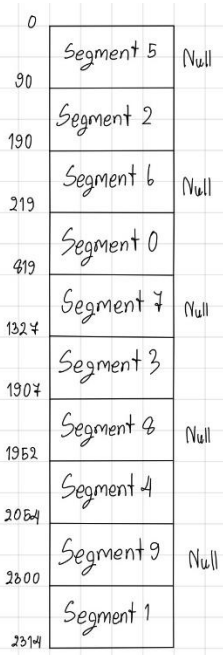
Page table



โปรแกรมที่ 2

Segment	Base	Length
0	219	600
1	2500	14
2	90	100
3	1327	580
4	1952	96

	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96
5	0	90
6	190	99
7	219	508
8	1907	45
9	2054	246



a.) 0,430 = 430 + 219 = 649
b.) 1,10 = 10 + 2300 = 2310
c.) 2, 500 = อ้างอิงค่าไม่ได้
d.) 3, 400 = 400 + 1327 = 1727
e.) 4, 112 = อ้างอิงค่าไม่ได้

อธิบาย Code

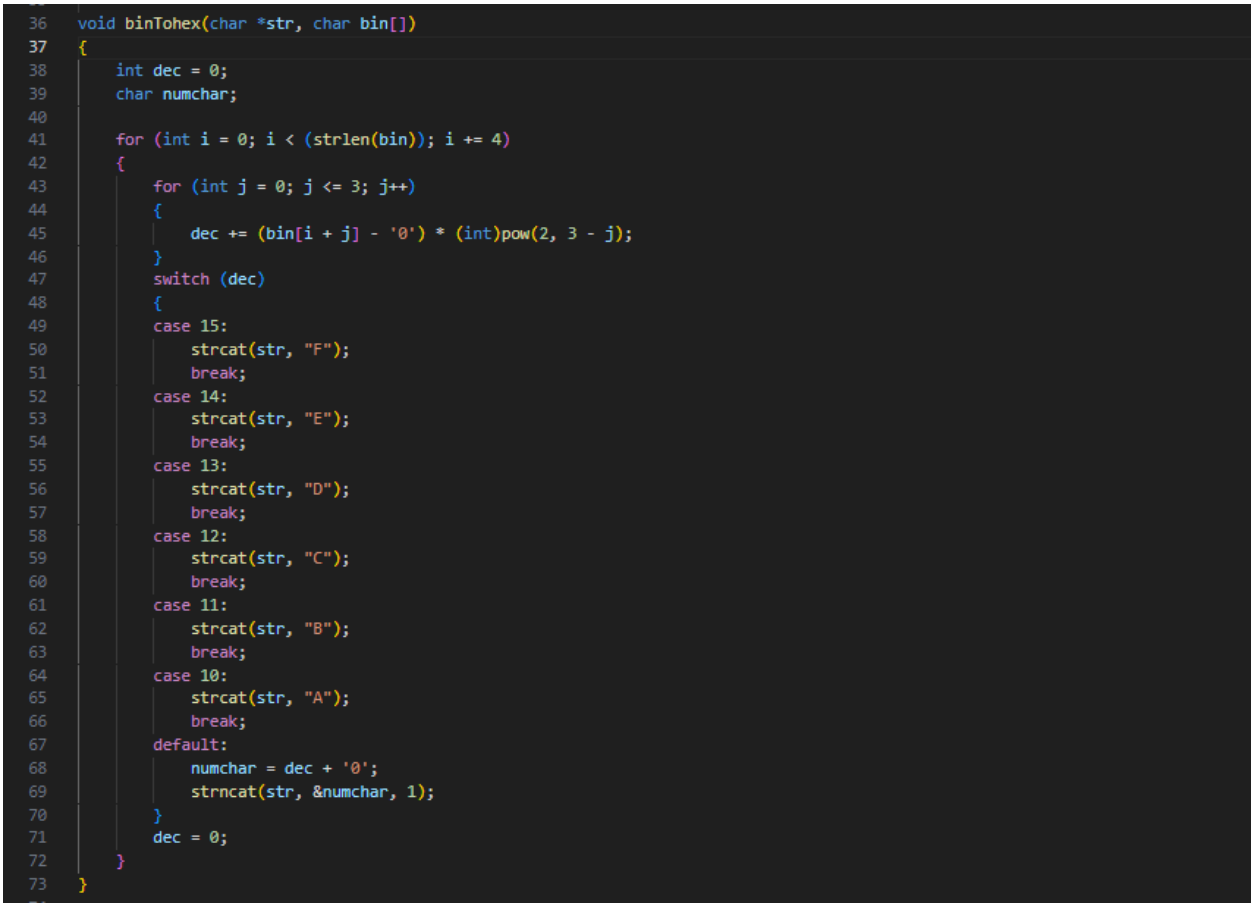
```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#define NUM_P 6
#define NUM_LOGI 4

struct Logical
{
    int page_No;        // เก็บหมายเลขหน้าที่ต้องการอ้างอิง
    char offset[100];   // เก็บค่า offset หรือตำแหน่งภายในหน้านั้นๆ
};

struct ProcessPage
{
    int page_No;        // เก็บหมายเลขหน้าที่ต้องการอ้างอิง
    char offset[100];   // เก็บค่า offset หรือตำแหน่งภายในหน้านั้นๆ
};
```

เป็นการกำหนดโครงสร้างข้อมูลสองอย่างคือ Logical และ ProcessPage:

1. Logical มีสมาชิกดังนี้:
    - page_No: เป็นตัวแปรชนิด int ที่ใช้เก็บหมายเลขหน้าที่ต้องการอ้างอิง
    - offset: เป็นอาร์เรย์ของ char ที่มีขนาด 100 ช่องเก็บเครื่องหมายหรือตำแหน่งภายในหน้านั้นๆ ในรูปของสตริง
2. ProcessPage มีสมาชิกดังนี้:
    - page_No: เป็นตัวแปรชนิด int ที่ใช้เก็บหมายเลขหน้าที่ต้องการอ้างอิง
    - offset: เป็นอาร์เรย์ของ char ที่มีขนาด 100 ช่องเก็บเครื่องหมายหรือตำแหน่งภายในหน้านั้นๆ ในรูปของสตริง

```
20  char str[100] = {'\0'}; // แก้ str เป็น char array และกำหนดค่าเริ่มต้นเป็น '\0'
21  int memoryPage = 8;
22  int systemBus = 24;
23  struct ProcessPage PageTable[NUM_P + 1] = {{},
24                                             {1, "0000010100101001"},
25                                             {2, "0000010100100011"},
26                                             {3, "0000000100100100"},
27                                             {4, "0000110101101011"},
28                                             {5, "0000011101101100"},
29                                             {6, "0000111101100111"}};
30  struct Logical logi[NUM_LOGI + 1] = {{},
31                                       {1, "25"},
32                                       {2, "43"},
33                                       {3, "7"},
34                                       {5, "50"}};
35
```

1. char str[100] = {'\0'};: กำหนดตัวแปรชนิดอาร์เรย์ของ char ชื่อ str ขนาด 100 ช่อง และกำหนดค่าเริ่มต้นให้ทุกช่องเป็น '\0' เพื่อให้เป็นสตริงว่าง ซึ่งสามารถใช้เก็บข้อมูลเพิ่มเติมในภายหลังได้.
2. int memoryPage = 8;: กำหนดตัวแปรชื่อ memoryPage และกำหนดค่าเริ่มต้นให้เท่ากับ 8 ซึ่งเป็นจำนวนหน้าหน่วยความจำต่อหน้า.
3. int systemBus = 24;: กำหนดตัวแปรชื่อ systemBus และกำหนดค่าเริ่มต้นให้เท่ากับ 24 ซึ่งเป็นจำนวนบิตของส่วนของระบบสื่อสาร.
4. struct ProcessPage PageTable[NUM_P + 1] = {...};: กำหนดอาร์เรย์ของโครงสร้าง ProcessPage ชื่อ PageTable ที่มีขนาดเท่ากับ NUM_P + 1 โดยกำหนดค่าเริ่มต้นให้กับแต่ละสมาชิกของอาร์เรย์ โดยใช้ลำดับการกำหนดจาก {}.
5. struct Logical logi[NUM_LOGI + 1] = {...};: กำหนดอาร์เรย์ของโครงสร้าง Logical ชื่อ logi ที่มีขนาดเท่ากับ NUM_LOGI + 1 โดยกำหนดค่าเริ่มต้นให้กับแต่ละสมาชิกของอาร์เรย์ โดยใช้ลำดับการกำหนดจาก {}.



```c
void binTohex(char *str, char bin[])
{
    int dec = 0;
    char numchar;

    for (int i = 0; i < (strlen(bin)); i += 4)
    {
        for (int j = 0; j <= 3; j++)
        {
            dec += (bin[i + j] - '0') * (int)pow(2, 3 - j);
        }
        switch (dec)
        {
        case 15:
            strcat(str, "F");
            break;
        case 14:
            strcat(str, "E");
            break;
        case 13:
            strcat(str, "D");
            break;
        case 12:
            strcat(str, "C");
            break;
        case 11:
            strcat(str, "B");
            break;
        case 10:
            strcat(str, "A");
            break;
        default:
            numchar = dec + '0';
            strncat(str, &numchar, 1);
        }
        dec = 0;
    }
}
```

ฟังก์ชัน binTohex นี้มีหน้าที่แปลงสตริงที่เก็บค่าในรูปแบบ binary (ฐาน 2) เป็นสตริงที่เก็บค่าในรูปแบบ hexadecimal (ฐาน 16) โดยทำการแปลงไปทีละ 4 บิต

การทำงานของฟังก์ชัน

1. int dec = 0;: สร้างตัวแปร dec เพื่อเก็บค่าที่ได้จากการแปลง binary เป็น decimal (ฐาน 10).
2. char numchar;: สร้างตัวแปร numchar เพื่อใช้เก็บค่าตัวอักษรที่ได้จากการแปลง decimal เป็น character สำหรับการเก็บค่า hexadecimal.
3. for (int i = 0; i < strlen(bin); i += 4) { ... }: ใช้ลูปเพื่อแบ่งสตริง bin ออกเป็นส่วนๆ ที่มีขนาด 4 บิตต่อรอบ โดยเริ่มต้นที่ index 0 และเพิ่มขึ้นทีละ 4 บิตในแต่ละรอบ.
4. for (int j = 0; j <= 3; j++) { ... }: ในแต่ละรอบของลูปนี้จะทำการแปลง 4 บิตของ binary เป็นค่า decimal โดยนำค่าของแต่ละบิตมาคูณกับ 2 ยกกำลัง (3 - j) เพื่อนำมาบวกกันเพื่อให้ได้ค่า decimal ของส่วนนั้นๆ ซึ่งจะเก็บไว้ใน dec.

5. switch (dec) { ... }: ใช้ switch เพื่อตรวจสอบค่า dec หลังจากที่ได้ค่า decimal จากการแปลง binary เป็น decimal โดยใช้ลำดับเงื่อนไขเพื่อกำหนดค่า hexadecimal ที่เป็นไปได้ตามค่า decimal ที่ได้รับ ถ้าไม่อยู่ในเงื่อนไขที่กำหนดไว้ จะใช้ default เพื่อเก็บค่า decimal ที่ได้ในรูปของ character.
6. strcat(str, "...") และ strncat(str, &numchar, 1): ใช้ strcat และ strncat เพื่อเพิ่มค่า hexadecimal ที่ได้เข้าไปในสตริง str ซึ่งเป็นสตริงผลลัพธ์ของการแปลง.
7. dec = 0;: กำหนดค่า dec เป็น 0 เพื่อเตรียมสำหรับการแปลงส่วนถัดไปของ binary ในลูปนั้นๆ ใหม่.

```
75  void hexTobin(char *str, char hex[])
76  {
77      for (int i = 0; i < strlen(hex); ++i)
78      {
79          switch (hex[i])
80          {
81          case '0':
82              strcat(str, "0000");
83              break;
84          case '1':
85              strcat(str, "0001");
86              break;
87          case '2':
88              strcat(str, "0010");
89              break;
90          case '3':
91              strcat(str, "0011");
92              break;
93          case '4':
94              strcat(str, "0100");
95              break;
96          case '5':
97              strcat(str, "0101");
98              break;
99          case '6':
100             strcat(str, "0110");
101             break;
102         case '7':
103             strcat(str, "0111");
104             break;
105         case '8':
106             strcat(str, "1000");
107             break;
108         case '9':
109             strcat(str, "1001");
110             break;
111         case 'A':
112             strcat(str, "1010");
113             break;
114         case 'B':
115             strcat(str, "1011");
116             break;
117         case 'C':
118             strcat(str, "1100");
119             break;
120         case 'D':
121             strcat(str, "1101");
122             break;
123         case 'E':
124             strcat(str, "1110");
125             break;
126         case 'F':
127             strcat(str, "1111");
128             break;
129         }
130     }
131 }
```

ฟังก์ชัน hexTobin นี้มีหน้าที่แปลงสตริงที่เก็บค่าในรูปแบบ hexadecimal (ฐาน 16) เป็นสตริงที่เก็บค่าในรูปแบบ binary (ฐาน 2) โดยทำการแปลงตัวอักษรที่แทนค่า hexadecimal ให้เป็นค่า binary ตามตำแหน่งของแต่ละตัวอักษร ดังนี้:

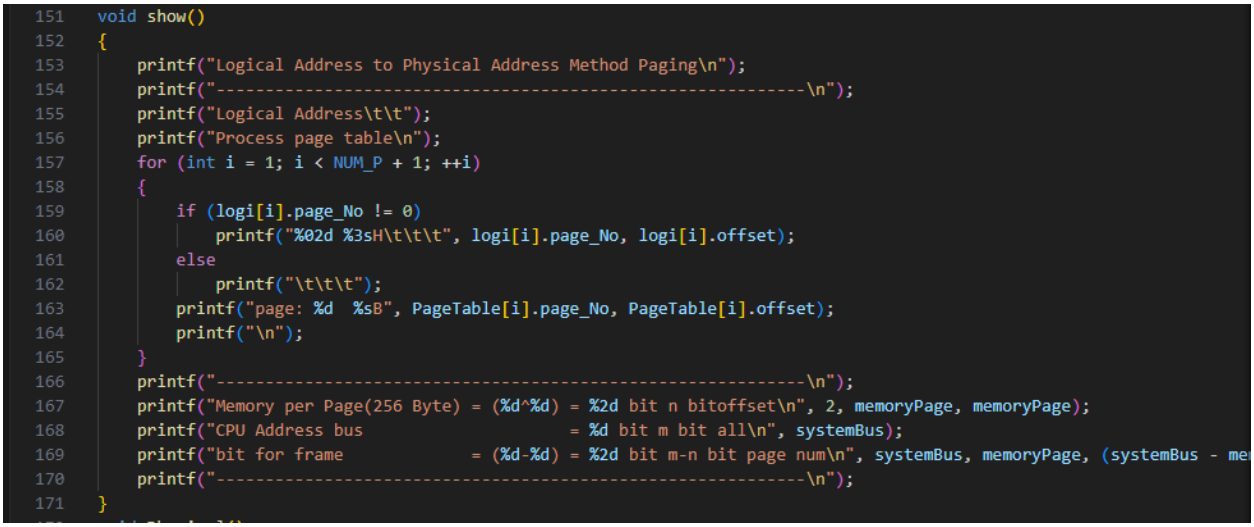
1. ในลูป for เราจะวนลูปผ่านทุกตัวอักษรในสตริง hex เพื่อทำการแปลงค่า hexadecimal เป็น binary.
2. ในแต่ละรอบของลูป switch, เราใช้ switch เพื่อตรวจสอบค่าของแต่ละตัวอักษรในสตริง hex.
3. โดยเราทำการเพิ่มค่า binary ที่แปลงได้เข้าไปในสตริง str โดยใช้ strcat.
4. เราใช้เงื่อนไข case ในการตรวจสอบค่าของแต่ละตัวอักษร และทำการเพิ่มค่า binary ลงในสตริง str ตามลำดับของค่า hexadecimal.
5. หลังจากแปลงทุกตัวอักษรเสร็จสิ้น เราจะได้สตริง str ที่เก็บค่า binary ที่แปลงจากค่า hexadecimal ทั้งหมดในสตริง hex.

```
133  const char *Convert(char *star, char number[], const char *base) // แก้ base เป็น const char*
134  {
135      strcpy(str, "");
136      if (strcmp(base, "2") == 0)
137      {
138          hexTobin(str, number);
139      }
140      else if (strcmp(base, "16") == 0)
141      {
142          binTohex(str, number);
143      }
144      else
145      {
146          return "error!!!";
147      }
148      return str;
149  }
```

ฟังก์ชัน Convert นี้มีหน้าที่แปลงค่าตัวเลขที่เก็บในรูปแบบ binary หรือ hexadecimal ไปยังรูปแบบอื่น ๆ โดยสามารถระบุรูปแบบที่ต้องการแปลงได้ผ่านพารามิเตอร์ base.

การทำงานของฟังก์ชัน

1. เริ่มต้นด้วยการล้างค่าของสตริง str โดยใช้ strcpy เพื่อเตรียมพร้อมสำหรับการเก็บผลลัพธ์ใหม่.
2. ใช้เงื่อนไข if-else เพื่อตรวจสอบว่า base เป็น "2" หรือ "16" หรือไม่ โดยใช้ strcmp เพื่อเปรียบเทียบสตริง.
3. ถ้า base เป็น "2" ก็จะเรียกใช้ฟังก์ชัน hexTobin เพื่อแปลงค่า hexadecimal เป็น binary และเก็บผลลัพธ์ใน str.
4. ถ้า base เป็น "16" ก็จะเรียกใช้ฟังก์ชัน binTohex เพื่อแปลงค่า binary เป็น hexadecimal และเก็บผลลัพธ์ใน str.
5. หาก base ไม่ใช่ "2" หรือ "16" จะส่งคืนสตริง "error!!!" เพื่อแสดงว่ามีข้อผิดพลาดในการระบุรูปแบบ.
6. สุดท้ายจะส่งคืนสตริง str ซึ่งเป็นผลลัพธ์หลังจากการแปลงค่า.



```
151  void show()
152  {
153      printf("Logical Address to Physical Address Method Paging\n");
154      printf("------------------------------------------------------------\n");
155      printf("Logical Address\t\t");
156      printf("Process page table\n");
157      for (int i = 1; i < NUM_P + 1; ++i)
158      {
159          if (logi[i].page_No != 0)
160              printf("%02d %3sH\t\t\t", logi[i].page_No, logi[i].offset);
161          else
162              printf("\t\t\t");
163          printf("page: %d  %sB", PageTable[i].page_No, PageTable[i].offset);
164          printf("\n");
165      }
166      printf("------------------------------------------------------------\n");
167      printf("Memory per Page(256 Byte) = (%d^%d) = %2d bit n bitoffset\n", 2, memoryPage, memoryPage);
168      printf("CPU Address bus                   = %d bit m bit all\n", systemBus);
169      printf("bit for frame             = (%d-%d) = %2d bit m-n bit page num\n", systemBus, memoryPage, (systemBus - me
170      printf("------------------------------------------------------------\n");
171  }
```

ฟังก์ชัน show มีหน้าที่แสดงข้อมูลเกี่ยวกับการจัดการหน้า (paging) และข้อมูลที่เกี่ยวข้องกับการแปลงที่อยู่ตำแหน่ง (address) จากที่อยู่ตำแหน่งตรรกะ (logical address) เป็นที่อยู่ตำแหน่งก่อนหน้า (physical address) ตามหลักการของการจัดการหน้า (paging).

การทำงานของฟังก์ชันนี้

1. พิมพ์ข้อความ "Logical Address to Physical Address Method Paging" เพื่อแสดงหัวข้อของข้อมูลที่จะแสดง.
2. พิมพ์ข้อความแบบตารางเพื่อแสดงข้อมูล logical address และ process page table.
3. ใช้ลูป for เพื่อวนลูปผ่านตาราง process page table และแสดงข้อมูลในรูปแบบตาราง โดยตรวจสอบว่าหมายเลขหน้า (page_No) ของแต่ละรายการไม่เป็น 0 หรือไม่ ถ้าไม่ใช่ก็จะแสดงข้อมูลของหน้านั้นๆ พร้อมกับ offset ของหน้า และหมายเลขหน้า.
4. พิมพ์ข้อมูลเกี่ยวกับ memory per page, CPU address bus และ bit for frame เพื่ออธิบายถึงการกำหนดขนาดของหน้า (page size) และขนาดของระบบเมมโมรี่ (memory) ที่ใช้.
5. สุดท้ายพิมพ์เส้นขั้น (separator line) เพื่อแสดงสิ้นสุดของข้อมูลที่แสดง.

```
172  void Physical()
173  {
174      for (int i = 1; i < NUM_LOGI + 1; ++i)
175      {
176          for (int j = 1; j < NUM_P + 1; ++j)
177          {
178              if (logi[i].page_No == PageTable[j].page_No)
179              {
180                  printf("Logical\t%sH\t\t%024s B\n", logi[i].offset, Convert(str, logi[i].offset, "2"));
181                  printf("\t%02d\t=\tPage No.%d\n", logi[i].page_No, logi[i].page_No);
182                  printf("\t%2s\t=\tPage off.%s\n", logi[i].offset, logi[i].offset);
183                  printf("-------------------------------------------------------------\n");
184                  printf("Physical\t\tf = %s B\n", PageTable[j].offset);
185                  printf("\t\t\t%7sFrame No. | Page off.\n", "");
186                  printf("Physical Address\t%s | %s B\n", PageTable[j].offset, Convert(str, logi[i].offset, "2"));
187                  printf("\t\t\t%11s %s | %s H\n", "", Convert(str, PageTable[j].offset, "16"), logi[i].offset);
188                  printf("-------------------------------------------------------------\n");
189                  break;
190              }
191          }
192      }
193  }
```

ฟังก์ชัน Physical มีหน้าที่แปลงที่อยู่ตำแหน่ง (logical address) เป็นที่อยู่ตำแหน่งก่อนหน้า (physical address) โดยใช้ข้อมูลจากตาราง process page table และ logical address.

การทำงานของฟังก์ชันนี้

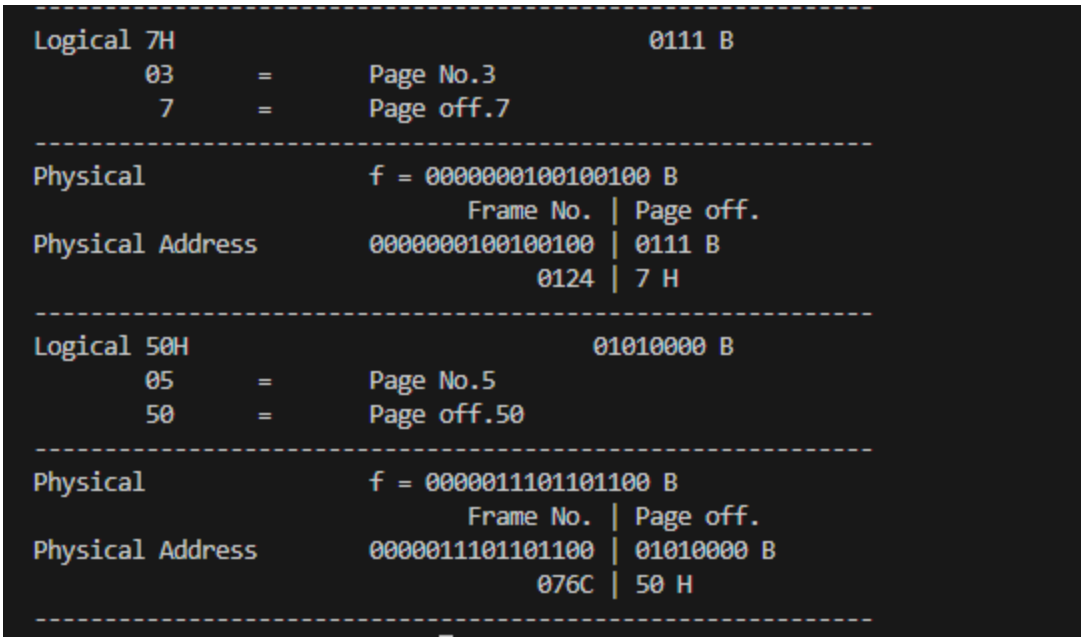
1. ใช้ลูป for เพื่อวนลูปผ่านตารางของ logical address (logi) และตารางของ process page table (PageTable).
2. ในลูปภายในมีลูปอีกทีซึ่งใช้เพื่อหาหน้าใน process page table ที่มีหมายเลขหน้า (page_No) ตรงกับหมายเลขหน้าของ logical address ที่กำลังพิจารณาอยู่.
3. ถ้าหากพบว่าหมายเลขหน้าของ logical address ตรงกับหมายเลขหน้าใน process page table จะทำการแปลงที่อยู่ตำแหน่ง (logical address) ให้เป็นที่อยู่ตำแหน่งก่อนหน้า (physical address) และแสดงข้อมูลที่เกี่ยวข้อง.
4. ใช้ฟังก์ชัน Convert เพื่อแปลงที่อยู่ตำแหน่งให้เป็นรูปแบบที่ต้องการ (binary หรือ hexadecimal) และแสดงผลลัพธ์ทั้งสองรูปแบบ.
5. สุดท้ายพิมพ์เส้นขั้น (separator line) เพื่อแสดงสิ้นสุดของข้อมูลที่แสดง.

```
195   int main()
196   {
197       show();
198       Physical();
199       return 0;
200   }
```

ฟังก์ชัน main เริ่มต้นด้วยการเรียกใช้ฟังก์ชัน show เพื่อแสดงข้อมูลเกี่ยวกับตาราง process page table และข้อมูลที่เกี่ยวข้องกับการจัดการหน่วยความจำของระบบ จากนั้นเรียกใช้ฟังก์ชัน Physical เพื่อทำการแปลงที่อยู่ตำแหน่ง (logical address) เป็นที่อยู่ตำแหน่งก่อนหน้า (physical address) และแสดงผลลัพธ์ สุดท้ายคืนค่า 0 เพื่อแสดงว่าโปรแกรมทำงานสมบูรณ์แล้วและจบการทำงานโดยปกติ.

ผลลัพธ์โปรแกรมที่ 1

```
Logical Address to Physical Address Method Paging
------------------------------------------------------------
Logical Address         Process page table
01  25H                 page: 1  0000010100101001B
02  43H                 page: 2  0000010100100011B
03   7H                 page: 3  0000000100100100B
05  50H                 page: 4  0000110101101011B
                        page: 5  0000011101101100B
-1 BBBBBBBBBBBBBBBBBBBBBBBBH                    page: 6  0000111101100111B
------------------------------------------------------------
Memory per Page(256 Byte) = (2^8) =  8 bit n bitoffset
CPU Address bus                   = 24 bit m bit all
bit for frame             = (24-8) = 16 bit m-n bit page num
------------------------------------------------------------
Logical 25H                             00100101 B
        01      =       Page No.1
        25      =       Page off.25
------------------------------------------------------------
Physical                f = 0000010100101001 B
                                Frame No. | Page off.
Physical Address        0000010100101001 | 00100101 B
                                    0529 | 25 H
------------------------------------------------------------
Logical 43H                             01000011 B
        02      =       Page No.2
        43      =       Page off.43
------------------------------------------------------------
Physical                f = 0000010100100011 B
                                Frame No. | Page off.
Physical Address        0000010100100011 | 01000011 B
                                    0523 | 43 H
------------------------------------------------------------
```



สิ่งแรกที่แสดงคือตาราง Process page table ซึ่งระบุหมายเลขหน้า (Page Number) และตำแหน่งภายในหน้า (Page offset) ของแต่ละหน้าที่ใช้ในการแปลงที่อยู่ตำแหน่ง โดยมีการแสดงข้อมูลของที่อยู่ตำแหน่ง Logical Address และข้อมูลที่เกี่ยวข้องกับหน่วยความจำ เช่น Memory per Page, CPU Address bus, และ bit for frame ด้วย

ต่อมาคือการแสดงผลลัพธ์ของการแปลงที่อยู่ตำแหน่ง Logical Address เป็น Physical Address สำหรับแต่ละที่อยู่ตำแหน่ง โดยแสดง Page Number, Page offset และ Frame Number พร้อมกับที่อยู่ตำแหน่งในรูปแบบ Binary (B) และ Hexadecimal (H) โดยใช้ฟังก์ชัน Physical() ในการดำเนินการนี้

สุดท้ายจะได้ผลลัพธ์ที่แสดงที่อยู่ตำแหน่ง Physical Address ที่เป็นไปได้สำหรับแต่ละที่อยู่ตำแหน่ง Logical Address ซึ่งอธิบายถึงข้อมูลเกี่ยวกับหน่วยความจำที่ใช้ และวิธีการแปลงที่อยู่ตำแหน่งในระบบ Paging อย่างชัดเจน

โปรแกรมที่ 2

```c
#include <stdio.h>
#define NUM_T 5
#define NUM_L 5

struct Logical
{
    int segment;
    int request;
};

struct Segment_Table
{
    int segment;
    int base;
    int length;
};

Segment_Table SegmentTable[NUM_T]={{0,219,600},
                                   {1,2300,14},
                                   {2,90,100},
                                   {3,1327,580},
                                   {4,1952,96}};


Logical logical[NUM_L]={{0,430},
                        {1,10},
                        {2,500},
                        {3,400},
                        {4,112}};
```

1. โครงสร้าง Logical:
    - มีสองสมาชิกคือ segment และ request ซึ่งแทนหมายเลขเซ็กเมนต์ (segment) และหมายเลขคำขอ (request) ตามลำดับ
2. โครงสร้าง Segment_Table:
    - มีสามสมาชิกคือ segment, base, และ length ซึ่งแทนหมายเลขเซ็กเมนต์ (segment), ที่อยู่เบส (base address), และความยาว (length) ของแต่ละเซ็กเมนต์ตามลำดับ

หลังจากนิยามโครงสร้างแล้ว โค้ดกำหนดค่าของอาร์เรย์ของโครงสร้างเหล่านี้ดังนี้:

- SegmentTable[NUM_T] เป็นอาร์เรย์ของโครงสร้าง Segment_Table ที่กำหนดค่าให้มีสมาชิก 5 ตัว แทนข้อมูลเกี่ยวกับเซ็กเมนต์
- logical[NUM_L] เป็นอาร์เรย์ของโครงสร้าง Logical ที่กำหนดค่าให้มีสมาชิก 5 ตัว แทนคำขอต่าง ๆ ที่เข้ามาในระบบโดยแต่ละคำขอประกอบด้วยหมายเลขเซ็กเมนต์และหมายเลขคำขอตามลำดับ

```
31  void Showtable(){
32      printf("Consider the following segment table:\n");
33      printf("Segment\t Base\t Length\n");
34      for (int i = 0; i < NUM_T; ++i){
35          printf("%2d\t%5d\t%5d\n", SegmentTable[i].segment, SegmentTable[i].base, SegmentTable[i].length);
36      }
37  }
```

ฟังก์ชัน Showtable() มีหน้าที่แสดงข้อมูลในตารางของเซ็กเมนต์ (segment table) ที่ถูกนิยามไว้ โดยแสดงข้อมูลเซ็กเมนต์แต่ละตัวที่มีอยู่ในอาร์เรย์ SegmentTable ผ่านการวนลูปด้วยค่า index i จาก 0 ไปจนถึง NUM_T - 1 ซึ่ง NUM_T เป็นค่าคงที่ที่นิยามไว้ก่อนหน้านี้ เพื่อแสดงจำนวนของเซ็กเมนต์ทั้งหมดในตาราง

ภายในลูป for ฟังก์ชันจะใช้ printf() เพื่อแสดงข้อมูลของแต่ละเซ็กเมนต์ที่อยู่ในตาราง โดยมีรูปแบบเป็นตารางที่ประกอบด้วยคอลัมน์ทั้งสามของ segment, base, และ length โดยแต่ละคอลัมน์มีข้อมูลของเซ็กเมนต์แต่ละตัวที่อยู่ในอาร์เรย์ SegmentTable ซึ่งได้ดึงข้อมูลมาแสดงผ่านการเข้าถึงสมาชิกของโครงสร้าง Segment_Table ด้วยการใช้สมาชิกของตัวแปร SegmentTable ด้วยการใช้ดอต (.) ในการเข้าถึงข้อมูลแต่ละส่วนของโครงสร้าง ดังนี้:

- SegmentTable[i].segment เข้าถึงค่าของสมาชิก segment ของเซ็กเมนต์ที่ i
- SegmentTable[i].base เข้าถึงค่าของสมาชิก base ของเซ็กเมนต์ที่ i
- SegmentTable[i].length เข้าถึงค่าของสมาชิก length ของเซ็กเมนต์ที่ i

```
39  int main(){
40      int physical;
41      Showtable();
42      printf("\nWhat are the physical address for the following logical addresses?\n");
43      for (int i = 0; i < NUM_L; ++i){
44          printf("(%c)segment %2d, request %4d  -> ", 97 + i, logical[i].segment, logical[i].request);
45          if(logical[i].request < SegmentTable[i].length){
46              physical = SegmentTable[i].base + logical[i].request;
47              printf("%4d +%4d = %4d", SegmentTable[i].base, logical[i].request, physical);
48          }else{
49              printf("Cannot be referenced!!!");
50          }
51          printf("\n");
52      }
53      return 0;
54  }
```

ฟังก์ชัน main() มีหน้าที่เรียกใช้ฟังก์ชัน Showtable() เพื่อแสดงตารางของเซ็กเมนต์ก่อน และจากนั้นพิมพ์ข้อความ "What are the physical address for the following logical addresses?" ตามด้วยการวนลูปเพื่อหาที่อยู่ทางกายภาพสำหรับที่อยู่ตรรกะ (logical address) ที่กำหนดไว้ในอาร์เรย์ logical

ภายในลูป for ฟังก์ชันจะพิมพ์ข้อมูลของแต่ละที่อยู่ตรรกะที่อยู่ในอาร์เรย์ logical ซึ่งมีรูปแบบการแสดงผลเป็น segment และ request พร้อมกับตำแหน่งที่อยู่ของข้อมูลใน logical โดยใช้การวนลูปด้วย index i จาก 0 ไปจนถึง NUM_L - 1 ซึ่ง NUM_L เป็นค่าคงที่ที่นิยามไว้ก่อนหน้านี้

ในการพิมพ์ข้อมูลของแต่ละที่อยู่ตรรกะ เงื่อนไขจะตรวจสอบว่าค่า request นั้นมีค่าน้อยกว่าความยาวของเซ็กเมนต์ที่กำหนดไว้หรือไม่ ถ้าใช่จะคำนวณหาที่อยู่ทางกายภาพ (physical address) โดยการเพิ่มค่า request ของเซ็กเมนต์นั้นกับ base address ของเซ็กเมนต์นั้น และจะนำค่าที่ได้ไปแสดงผล แต่ถ้าค่า request เกินความยาวของเซ็กเมนต์ จะแสดงข้อความ "Cannot be referenced!!!" แทน

ภายหลังจากการวนลูปเสร็จสิ้น ฟังก์ชัน main() จะส่งค่า 0 กลับเพื่อบ่งบอกว่าโปรแกรมทำงานสมบูรณ์แล้วและไม่มีข้อผิดพลาดในการทำงาน

ผลลัพธ์โปรแกรมที่ 2

```
Consider the following segment table:
Segment  Base    Length
 0        219     600
 1       2300      14
 2         90     100
 3       1327     580
 4       1952      96

What are the physical address for the following logical addresses?
(a)segment  0, request  430  ->  219 + 430 =  649
(b)segment  1, request   10  -> 2300 +  10 = 2310
(c)segment  2, request  500  -> Cannot be referenced!!!
(d)segment  3, request  400  -> 1327 + 400 = 1727
(e)segment  4, request  112  -> Cannot be referenced!!!
PS D:\เทอม 2 66\OS_Lab\lab8>
```

เป็นการคำนวณหาที่อยู่ทางกายภาพ (physical address) สำหรับที่อยู่ตรรกะ (logical address) ที่กำหนดไว้ในแต่ละข้อความ โดยใช้ข้อมูลจากตารางเซ็กเมนต์ที่กำหนดไว้ก่อนหน้านี้

(a) สำหรับที่อยู่ตรรกะที่กำหนดในข้อนี้คือ segment 0 และ request 430 โดยต้องการหาที่อยู่ทางกายภาพที่สอดคล้องกับ request นี้ เนื่องจาก request (430) ไม่เกินความยาวของ segment 0 (600) ดังนั้นที่อยู่ทางกายภาพจะเป็นผลบวกระหว่าง base address ของ segment 0 (219) กับ request (430) ซึ่งเท่ากับ 219 + 430 = 649

(b) สำหรับที่อยู่ตรรกะที่กำหนดในข้อนี้คือ segment 1 และ request 10 โดยต้องการหาที่อยู่ทางกายภาพที่สอดคล้องกับ request นี้ เนื่องจาก request (10) ไม่เกินความยาวของ segment 1 (14) ดังนั้นที่อยู่ทางกายภาพจะเป็นผลบวกระหว่าง base address ของ segment 1 (2300) กับ request (10) ซึ่งเท่ากับ 2300 + 10 = 2310

(c) สำหรับที่อยู่ตรรกะที่กำหนดในข้อนี้คือ segment 2 และ request 500 โดยต้องการหาที่อยู่ทางกายภาพที่สอดคล้องกับ request นี้ แต่ request (500) เกินความยาวของ segment 2 (100) ดังนั้นไม่สามารถอ้างอิงที่อยู่ทางกายภาพได้

(d) สำหรับที่อยู่ตรรกะที่กำหนดในข้อนี้คือ segment 3 และ request 400 โดยต้องการหาที่อยู่ทางกายภาพที่สอดคล้องกับ request นี้ เนื่องจาก request (400) ไม่เกินความยาวของ segment 3 (580) ดังนั้นที่อยู่ทางกายภาพจะเป็นผลบวกระหว่าง base address ของ segment 3 (1327) กับ request (400) ซึ่งเท่ากับ 1327 + 400 = 1727

(e) สำหรับที่อยู่ตรรกะที่กำหนดในข้อนี้คือ segment 4 และ request 112 โดยต้องการหาที่อยู่ทางกายภาพที่สอดคล้องกับ request นี้ แต่ request (112) เกินความยาวของ segment 4 (96) ดังนั้นไม่สามารถอ้างอิงที่อยู่ทางกายภาพได้ และจึงแสดงข้อความ "Cannot be referenced!!!" แทน

สรุปผลการทดลอง

1. โปรแกรมแรกใช้การจัดการข้อมูลแบบ Paging ซึ่งแบ่งหน้าหน่วยความจำออกเป็นชิ้นเล็ก ๆ ทำให้การจัดการข้อมูลที่อยู่ตรรกะและที่อยู่ทางกายภาพเป็นไปอย่างมีระเบียบ แต่ก็มีข้อจำกัดในการจัดการข้อมูลขนาดใหญ่หรือการจัดการเซ็กเมนต์ที่ใช้งานไม่ได้
2. โปรแกรมที่สองใช้เชิงโมเดล Segment-Based ซึ่งใช้การแยกแยะข้อมูลด้วย segment number และมีข้อได้เปรียบในการจัดการข้อมูลขนาดใหญ่ แต่ก็อาจมีความซับซ้อนในการจัดการข้อมูลที่อยู่ตรรกะและที่อยู่ทางกายภาพในกรณีที่มีเซ็กเมนต์มากมายและซับซ้อน

ดังนั้น การเลือกใช้โมเดลใดขึ้นอยู่กับความเหมาะสมและความต้องการของแต่ละโปรเจคและสถานการณ์การใช้งานที่แตกต่างกันไป

สื่อ / เอกสารอ้างอิง

อาจารย์ปิยพล ยืนยงสถาวร: เอกสารประกอบการสอน 8 การจัดการหน่วยความจำหลัก (Main Memory)