



ใบงานที่ 9

เรื่อง Page Replacement

จัดทำโดย

นางสาวรัชนิกร เชื้อดี 65543206077-1

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

ใบงานนี้เป็นส่วนหนึ่งของรายวิชา ระบบปฏิบัติการ

หลักสูตรวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลธัญบุรี

ประจำภาคที่ 2 ปีการศึกษา 2566

## ใบงานที่ 9

### Page Replacement

#### ขั้นตอนการทดลอง

1. จงแสดงวิธีการแทนที่หน่วยความจำและออกแบบโปรแกรมจำลองการแทนที่หน่วยความจำ
2. จงเขียนโปรแกรมจำลองการทำงานของ Algorithm ของ Page Replacement

Consider the following page-reference string for one process:

**2,1,2,3,7,6,2,3,4,2,1,5,6,3,2,3,6,2,1.**

How many page faults would occur for the below replacement algorithms, assuming a total 4 frames available for the process. Assume pure demand paging (all frames are initially empty), so the first page faults will all cost one fault each.

Refer  
ence  
string

2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

1. FIFO

2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

2	2	2	2	2	6	6	6	6	6	6	5	5	5	5	5	5	5	1
	1	1	1	1	1	2	2	2	2	2	2	6	6	6	6	6	6	6
			3	3	3	3	3	4	4	4	4	4	3	3	3	3	3	3
				7	7	7	7	7	7	1	1	1	1	2	2	2	2	2
F	F		F	F	F	F		F		F	F	F	F	F				F

Page faults  
= 13

2. LRU

2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

2	2	1	1	1	2	3	7	6	6	3	4	2	1	5	5	5	5	3
	1	2	2	2	3	7	6	2	3	4	2	1	5	6	6	2	3	6
			3	3	7	6	2	3	4	2	1	5	6	3	2	3	6	2
				7	6	2	3	4	2	1	5	6	3	2	3	6	2	1
F	F		F	F	F			F		F	F	F	F	F				F

Page faults  
= 12

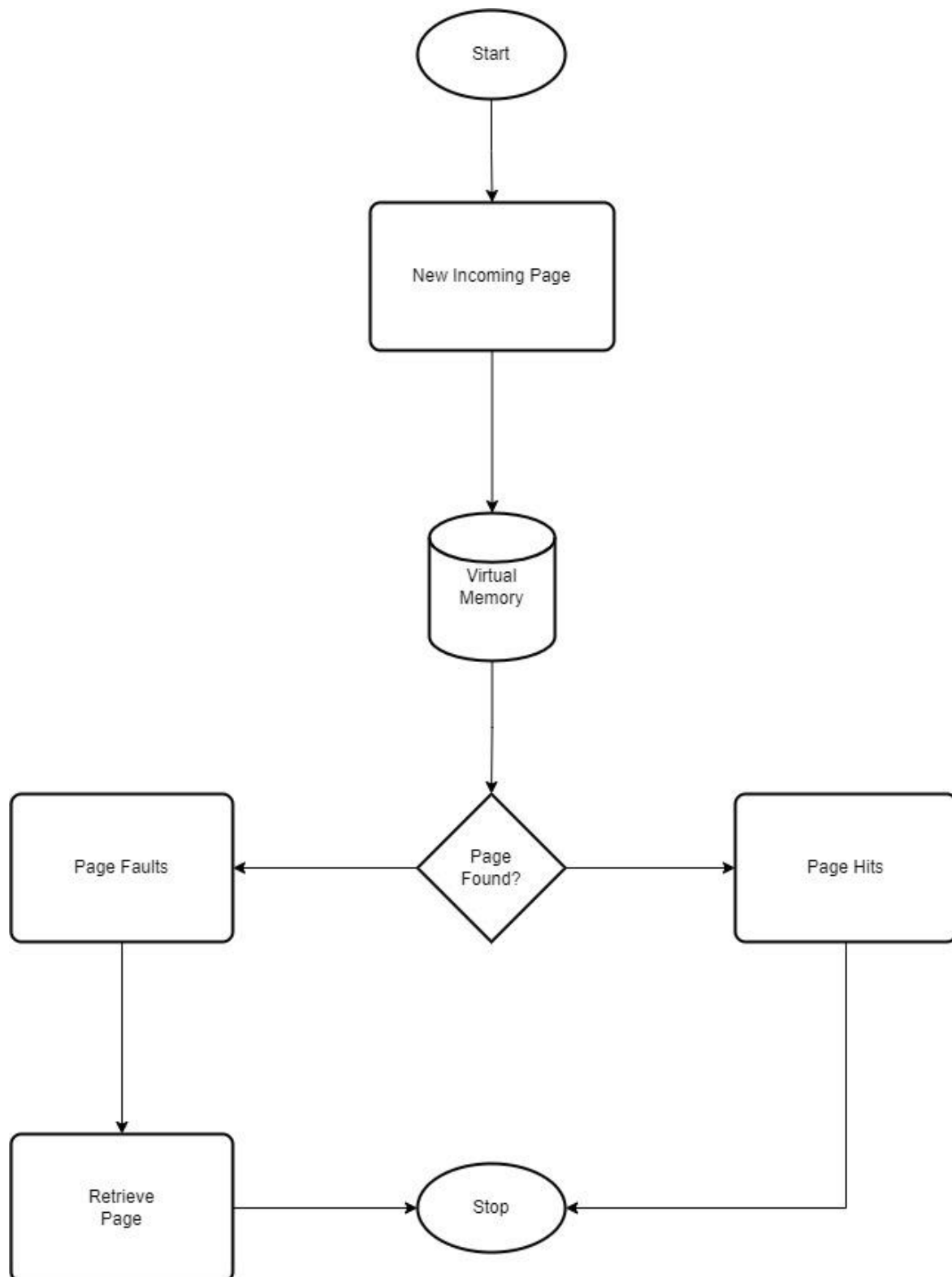
3. OPT

2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

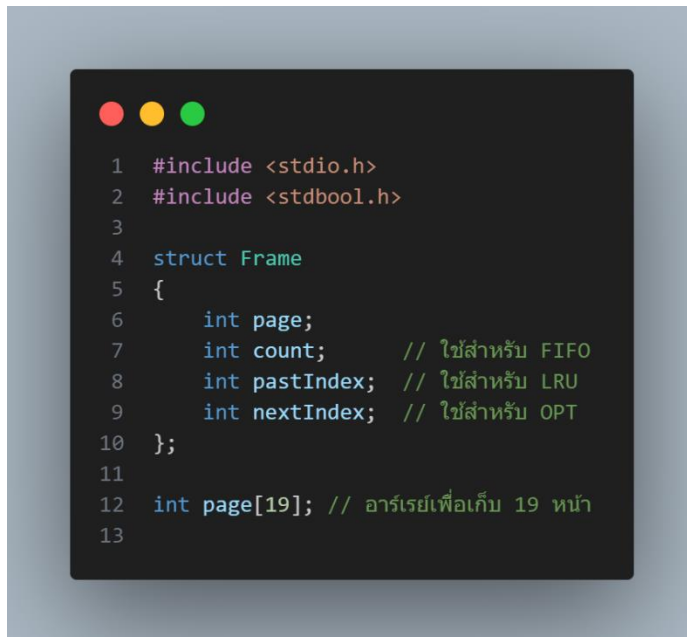
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
			3	3	3	3	3	4	4	4	5	5	3	3	3	3	3	3
				7	6	6	6	6	6	6	6	6	6	6	6	6	6	6
F	F		F	F	F			F			F		F					

Page faults  
= 8

## Flowchart



## อธิบาย Code



### โครงสร้างข้อมูล Frame:

- page: เก็บหมายเลขหน้าที่อยู่ในหน่วยความจำ
- count: ใช้สำหรับวิธีการจำลอง FIFO (First-In-First-Out) ซึ่งบ่งบอกถึงการเข้าถึงหน้าล่าสุดของหน่วยความจำ
- pastIndex: ใช้สำหรับวิธีการจำลอง LRU (Least Recently Used) ซึ่งบ่งบอกถึงการเข้าถึงหน้าที่ใช้ครั้งสุดท้ายของหน่วยความจำ
- nextIndex: ใช้สำหรับวิธีการจำลอง OPT (Optimal) ซึ่งบ่งบอกถึงการเข้าถึงหน้าที่จะใช้ในอนาคต

### อาร์เรย์ page:

- เป็นอาร์เรย์ที่เก็บหน้า (page) ซึ่งเป็นตัวเลข
- มีขนาดเท่ากับ 19

```

1 // โปรโตไทป์ของฟังก์ชัน
2 bool isValueInArray(int val, struct Frame frame[], int size);
3 int max(struct Frame frame[], int size, int key);
4 int min(struct Frame frame[], int size, int key);
5 int searchNextIndex(int Nframe, int nowIndex);
6 int searchPastIndex(int Nframe, int nowIndex);
7 int FIFO();
8 int LRU();
9 int OPT();

```

**bool isValueInArray(int val, struct Frame frame[], int size);**

- ฟังก์ชันที่รับค่าเลขเต็ม (int val) และอาร์เรย์ของโครงสร้าง Frame (struct Frame frame[]) รวมทั้งขนาดของอาร์เรย์ (int size) เพื่อตรวจสอบว่าค่า val มีอยู่ในอาร์เรย์ frame หรือไม่ และจะส่งค่าความจริง (true/false) กลับเมื่อตรวจพบหรือไม่พบค่า val ในอาร์เรย์ frame ตามลำดับ

**int max(struct Frame frame[], int size, int key);**

- ฟังก์ชันที่รับอาร์เรย์ของโครงสร้าง Frame (struct Frame frame[]) รวมทั้งขนาดของอาร์เรย์ (int size) และค่าเลขเต็ม (int key) เพื่อค้นหาค่าสูงสุดในอาร์เรย์ frame โดยดูจากค่า key ที่กำหนด

**int min(struct Frame frame[], int size, int key);**

- ฟังก์ชันที่รับอาร์เรย์ของโครงสร้าง Frame (struct Frame frame[]) รวมทั้งขนาดของอาร์เรย์ (int size) และค่าเลขเต็ม (int key) เพื่อค้นหาค่าต่ำสุดในอาร์เรย์ frame โดยดูจากค่า key ที่กำหนด

**int searchNextIndex(int Nframe, int nowIndex);**

- ฟังก์ชันที่รับจำนวนเต็มบวก Nframe และ nowIndex เพื่อค้นหาดัชนี (index) ถัดไปของหน้าในหน่วยความจำ โดยใช้วิธีการจำลอง OPT

**int searchPastIndex(int Nframe, int nowIndex);**

- ฟังก์ชันที่รับจำนวนเต็มบวก Nframe และ nowIndex เพื่อค้นหาดัชนี (index) ที่ใช้ครั้งสุดท้ายของหน้าในหน่วยความจำ โดยใช้วิธีการจำลอง LRU

**int FIFO();**

- ฟังก์ชันที่ไม่รับพารามิเตอร์ ใช้ในการจำลองวิธี FIFO (First-In-First-Out) ในการจัดการหน้าในหน่วยความจำ

int LRU();

- ฟังก์ชันที่ไม่รับพารามิเตอร์ ใช้ในการจำลองวิธี LRU (Least Recently Used) ในการจัดการหน้าในหน่วยความจำ

int OPT();

- ฟังก์ชันที่ไม่รับพารามิเตอร์ ใช้ในการจำลองวิธี OPT (Optimal) ในการจัดการหน้าในหน่วยความจำ

```

1  int main()
2  {
3      int key;
4      printf("\n===== Page Replacement Program =====\n");
5      printf("Please input 19 pages : ");
6      for (int i = 0; i < 19; i++)
7      {
8          scanf("%d", &page[i]);
9      }
10     printf("\nYour input : ");
11     for (int i = 0; i < 19; i++)
12     {
13         printf("%d ", page[i]);
14     }
15     while (key != 4)
16     {
17         printf("\n\n===== Please select a menu =====\n");
18         printf("1. FIFO\n2. LRU\n3. OPT\n4. Exit Program\n");
19         printf("Please input a key to continue : ");
20         scanf("%d", &key);
21         switch (key)
22         {
23             case 1:
24                 FIFO();
25                 break;
26             case 2:
27                 LRU();
28                 break;
29             case 3:
30                 OPT();
31                 break;
32         }
33     }
34     return 0;
35 }

```

เป็นโปรแกรมหลัก (main program) ที่ใช้ในการจำลองการทำงานของวิธีการจัดการหน้า (page replacement algorithms) ต่าง ๆ ในหน่วยความจำของคอมพิวเตอร์ โดยโปรแกรมจะทำงานตามขั้นตอนต่อไปนี้:

1. แสดงข้อความต้อนรับและรับค่าหน้า 19 หน้าจากผู้ใช้ผ่านทางคีย์บอร์ด
2. แสดงหน้าที่ผู้ใช้ป้อนเข้ามา
3. เข้าสู่ลูป while เพื่อรอรับค่า key จากผู้ใช้และดำเนินการตามค่า key ที่ผู้ใช้ป้อน
  - ถ้า key เท่ากับ 1: เรียกใช้ฟังก์ชัน FIFO() เพื่อจำลองวิธี FIFO
  - ถ้า key เท่ากับ 2: เรียกใช้ฟังก์ชัน LRU() เพื่อจำลองวิธี LRU
  - ถ้า key เท่ากับ 3: เรียกใช้ฟังก์ชัน OPT() เพื่อจำลองวิธี OPT



- ถ้า key เท่ากับ 4: ออกจากโปรแกรม

```

1 // ฟังก์ชันสำหรับการแทนที่หน้าโดยวิธี FIFO
2 int FIFO()
3 {
4     int n = 0;
5     int pf = 0;
6     printf("Please input the number of frames : ");
7     scanf("%d", &n);
8     struct Frame frame[n]; // อาร์เรย์ของเฟรม
9     for (int e = 0; e < n; e++)
10    {
11        frame[e].page = -1;
12        frame[e].count = 0;
13    }
14    for (int i = 0; i < 19; i++)
15    {
16        if (!isValueInArray(page[i], frame, n)) // ตรวจสอบค่าซ้ำ
17        {
18            if (isValueInArray(-1, frame, n)) // ตรวจสอบว่ามีเฟรมว่างหรือไม่
19            {
20                for (int e = 0; e < n; e++)
21                {
22                    if (frame[e].page == -1)
23                    {
24                        frame[e].page = page[i];
25                        frame[e].count++;
26                        pf++;
27                        break;
28                    }
29                    else
30                    {
31                        frame[e].count++;
32                    }
33                }
34            }
35            else
36            {
37                for (int e = 0; e < n; e++)
38                {
39                    if (max(frame, n, 1) != e)
40                    { // เพิ่ม count สำหรับหน้าที่ไม่ได้เข้าถึง
41                        frame[e].count++;
42                    }
43                }
44                frame[max(frame, n, 1)].page = page[i];
45                frame[max(frame, n, 1)].count = 1;
46                pf++;
47            }
48        }
49        else
50        { // กรณีหน้าอ้างอิงซ้ำ อัปเดต count สำหรับเฟรมทั้งหมด
51            for (int e = 0; e < n; e++)
52            {
53                frame[e].count++;
54            }
55        }
56        printf("\nInput page : %d\n", page[i]);
57        printf("Frame Content :\n");
58        for (int e = 0; e < n; e++)
59        {
60            if (frame[e].page == -1)
61            {
62                printf("Frame %d : [ ]\n", e + 1);
63            }
64            else
65            {
66                printf("Frame %d : [%d]\n", e + 1, frame[e].page);
67            }
68        }
69        printf("=====\n");
70        printf("Page fault (FIFO) : %d\n", pf);
71        printf("=====\n");
72    }
73    return 0;
74 }

```

ฟังก์ชัน FIFO() นี้ใช้ในการจำลองการแทนที่หน้าโดยใช้วิธี FIFO (First-In-First-Out) ในการจัดการหน้าในหน่วยความจำ โดยมีขั้นตอนการทำงานดังนี้:

1. รับค่าจำนวนเฟรม (frames) จากผู้ใช้และสร้างอาร์เรย์ของโครงสร้าง Frame โดยให้แต่ละเฟรมมีหน้า (page) เป็น -1 และ count เป็น 0
2. วนลูปผ่านหน้าทั้ง 19 หน้าที่ใช้ป้อนเข้ามา
3. ตรวจสอบว่าหน้าปัจจุบันที่อ้างถึง (page[i]) ไม่มีอยู่ในเฟรมหรือไม่
  - ถ้าไม่มี: ตรวจสอบว่ามีเฟรมว่างหรือไม่ หากมีให้ใส่หน้านั้นลงในเฟรมว่าง และเพิ่มจำนวน page fault (pf) ขึ้น หากไม่มีเฟรมว่างให้แทนที่หน้าที่มี count มากที่สุดด้วยหน้าปัจจุบัน
  - ถ้ามี: เพิ่ม count ของทุกเฟรมและอัปเดตเฟรมที่มีหน้าเดียวกับหน้าปัจจุบันเป็น 1
4. แสดงผลลัพธ์ของการแทนที่หน้าและจำนวน page fault ในแต่ละรอบ

```

1 // ฟังก์ชันสำหรับการแทนที่หน้าโดยใช้วิธี LRU
2 int LRU()
3 {
4     int n = 0;
5     int pf = 0;
6     printf("Please input the number of frames : ");
7     scanf("%d", &n);
8     struct Frame frame[n]; // อาร์เรย์ของเฟรม
9     for (int e = 0; e < n; e++)
10    {
11        frame[e].page = -1;
12        frame[e].pastIndex = 0;
13    }
14    for (int i = 0; i < 19; i++)
15    {
16        if (isValueInArray(page[i], frame, n)) // ตรวจสอบค่าซ้ำ
17        {
18            if (isValueInArray(-1, frame, n)) // ตรวจสอบว่ามีเฟรมว่างหรือไม่
19            {
20                for (int e = 0; e < n; e++)
21                {
22                    if (frame[e].page == -1)
23                    {
24                        frame[e].page = page[i];
25                        frame[e].pastIndex = i;
26                        pf++;
27                        break;
28                    }
29                    else
30                    {
31                        frame[e].pastIndex = searchPastIndex(frame[e].page, i);
32                    }
33                }
34            }
35            else
36            {
37                frame[min(frame, n, 2)].page = page[i];
38                frame[min(frame, n, 2)].pastIndex = searchPastIndex(frame[min(frame, n, 2)].page, i);
39                pf++;
40            }
41        }
42        else
43        { // กรณีหน้าอ้างอิงซ้ำ อัปเดต pastIndex สำหรับเฟรมทั้งหมด
44            for (int e = 0; e < n; e++)
45            {
46                frame[e].pastIndex = searchPastIndex(frame[e].page, i);
47            }
48        }
49        printf("\nInput page : %d\n", page[i]);
50        printf("Frame Content : \n");
51        for (int e = 0; e < n; e++)
52        {
53            if (frame[e].page == -1)
54            {
55                printf("Frame %d : [ ]\n", e + 1);
56            }
57            else
58            {
59                printf("Frame %d : [%d]\n", e + 1, frame[e].page);
60            }
61        }
62        printf("=====\n");
63        printf("Page fault (LRU) : %d\n", pf);
64        printf("=====\n");
65    }
66    return 0;
67 }
68

```

ฟังก์ชัน LRU() นี้ใช้ในการจำลองการแทนที่หน้าโดยใช้วิธี LRU (Least Recently Used) ในการจัดการหน้าในหน่วยความจำ โดยมีขั้นตอนการทำงานดังนี้:

1. รับค่าจำนวนเฟรม (frames) จากผู้ใช้และสร้างอาร์เรย์ของโครงสร้าง Frame โดยให้แต่ละเฟรมมีหน้า (page) เป็น -1 และ pastIndex เป็น 0
2. วาดรูปผ่านหน้าทั้ง 19 หน้าที่ใช้ป้อนเข้ามา
3. ตรวจสอบว่าหน้าปัจจุบันที่อ้างถึง (page[i]) ไม่มีอยู่ในเฟรมหรือไม่
  - ถ้าไม่มี: ตรวจสอบว่ามีเฟรมว่างหรือไม่ หากมีให้ใส่หน้านั้นลงในเฟรมว่าง และเพิ่มจำนวน page fault (pf) ขึ้น หากไม่มีเฟรมว่างให้แทนที่หน้าที่ไม่ได้ใช้เป็นเวลานานที่สุดด้วยหน้าปัจจุบัน
  - ถ้ามี: แทนที่หน้าที่ไม่ได้ใช้มานานที่สุดด้วยหน้าปัจจุบัน และอัปเดต pastIndex
4. แสดงผลลัพธ์ของการแทนที่หน้าและจำนวน page fault ในแต่ละรอบ

```

1 // ฟังก์ชันสำหรับการนับหน้าในไฟล์ OPT
2 int OPT()
3 {
4     int n = 0;
5     int pf = 0;
6     printf("Please input the number of frames : ");
7     scanf("%d", &n);
8     struct Frame frame[n]; // จัดสรรหน่วยความจำ
9     for (int e = 0; e < n; e++)
10     {
11         frame[e].page = -1;
12         frame[e].nextIndex = 0;
13     }
14     for (int i = 0; i < 19; i++)
15     {
16         if (!isValueInArray(page[i], frame, n)) // ตรวจสอบค่า
17         {
18             if (isValueInArray(-1, frame, n)) // ตรวจสอบว่ามีเฟรมว่างหรือไม่
19             {
20                 for (int e = 0; e < n; e++)
21                 {
22                     if (frame[e].page == -1)
23                     {
24                         frame[e].page = page[i];
25                         frame[e].nextIndex = searchNextIndex(frame[e].page, i);
26                         pf++;
27                         break;
28                     }
29                     else
30                     {
31                         frame[e].nextIndex = searchNextIndex(frame[e].page, i);
32                     }
33                 }
34             }
35             else
36             {
37                 if (frame[min(frame, n, 1)].nextIndex == 0)
38                 {
39                     frame[min(frame, n, 1)].page = page[i];
40                     frame[min(frame, n, 1)].nextIndex = searchNextIndex(frame[min(frame, n, 1)].page, i);
41                 }
42                 else
43                 {
44                     frame[max(frame, n, 2)].page = page[i];
45                     frame[max(frame, n, 2)].nextIndex = searchNextIndex(frame[max(frame, n, 2)].page, i);
46                 }
47                 pf++;
48             }
49         }
50         else
51         { // กรณีมีหน้าว่างไม่พอ มีค่า nextIndex สำหรับเฟรมที่แทน
52             for (int e = 0; e < n; e++)
53             {
54                 frame[e].nextIndex = searchNextIndex(frame[e].page, i);
55             }
56             printf("\nInput page : %d\n", page[i]);
57             printf("Frame Content :\n");
58             for (int e = 0; e < n; e++)
59             {
60                 if (frame[e].page == -1)
61                 {
62                     printf("Frame %d : [ ]\n", e + 1);
63                 }
64                 else
65                 {
66                     printf("Frame %d : [%d]\n", e + 1, frame[e].page);
67                 }
68             }
69             printf("=====\n");
70             printf("Page Fault (OPT) : %d\n", pf);
71             printf("=====\n");
72         }
73     }
74     return 0;
75 }
76
77 // ฟังก์ชันสำหรับตรวจสอบการเข้าถึงในอาร์เรย์ของเฟรม
78 bool isValueInArray(int val, struct Frame frame[], int size)
79 {
80     for (int i = 0; i < size; i++)
81     {
82         if (frame[i].page == val)
83             return true;
84     }
85     return false;
86 }
87
88 // ฟังก์ชันสำหรับหาค่าดัชนีของเฟรมที่มีค่า count หรือ nextIndex สูงสุด
89 int max(struct Frame frame[], int size, int key)
90 {
91     int index = 0;
92     if (key == 1) // หาค่าสำหรับ FIFO
93     {
94         int max = frame[index].count;
95         for (int i = 0; i < size; i++)
96         {
97             if (frame[i].count > max)
98             {
99                 max = frame[i].count;
100                 index = i;
101             }
102         }
103     }
104     else if (key == 2) // หาค่าสำหรับ OPT
105     {
106         int max = frame[index].nextIndex;
107         for (int i = 0; i < size; i++)
108         {
109             if (frame[i].nextIndex > max)
110             {
111                 max = frame[i].nextIndex;
112                 index = i;
113             }
114         }
115     }
116     return index;
117 }

```

ฟังก์ชัน OPT() นี้ใช้ในการจำลองการแทนที่หน้าโดยใช้วิธี OPT (Optimal) ในการจัดการหน้าในหน่วยความจำ โดยมีขั้นตอนการทำงานดังนี้:

1. รับค่าจำนวนเฟรม (frames) จากผู้ใช้และสร้างอาร์เรย์ของโครงสร้าง Frame โดยให้แต่ละเฟรมมีหน้า (page) เป็น -1 และ nextIndex เป็น 0
2. วาดรูปผ่านหน้าทั้ง 19 หน้าที่ใช้ป้อนเข้ามา
3. ตรวจสอบว่าหน้าปัจจุบันที่อ้างถึง (page[i]) ไม่มีอยู่ในเฟรมหรือไม่
  - ถ้าไม่มี: ตรวจสอบว่ามีเฟรมว่างหรือไม่ หากมีให้ใส่หน้านั้นลงในเฟรมว่าง และเพิ่มจำนวน page fault (pf) ขึ้น หากไม่มีเฟรมว่างให้แทนที่หน้าที่มีค่า nextIndex มากที่สุดด้วยหน้าปัจจุบัน
  - ถ้ามี: หากหน้าใหม่ไม่ถูกอ้างถึงในอนาคตอีกต่อไปให้แทนที่หน้าที่ไม่ถูกอ้างถึงในอนาคตที่ใกล้ที่สุด หากหน้าใหม่ยังมียังมีโอกาสถูกอ้างถึงอีกในอนาคตให้แทนที่หน้าที่ไม่ถูกอ้างถึงในอนาคตที่ไกลที่สุด
4. แสดงผลลัพธ์ของการแทนที่หน้าและจำนวน page fault ในแต่ละรอบ

โดยฟังก์ชัน `isValueInArray()` ใช้ในการตรวจสอบว่าหน้าที่กำหนดอยู่ในอาร์เรย์ของเฟรมหรือไม่ และฟังก์ชัน `max()` ใช้ในการค้นหาดัชนีของเฟรมที่มีค่า nextIndex สูงสุด เพื่อใช้ในการแทนที่หน้าในกรณีของวิธีการจัดการหน้า OPT โดยในฟังก์ชัน `max()` นั้นมีการตรวจสอบค่า key เพื่อให้ใช้สำหรับการค้นหาดัชนีของเฟรมที่มีค่า count หรือ nextIndex สูงสุดตามลักษณะของ key ที่กำหนดโดยผู้เรียกใช้ฟังก์ชัน

```

1 // ฟังก์ชันสำหรับค้นหาดัชนีของเฟรมที่มีค่า nextIndex หรือ pastIndex ต่ำสุด
2 int min(struct Frame frame[], int size, int key)
3 {
4     int index = 0;
5     if (key == 1) // ใช้สำหรับ OPT
6     {
7         int min = frame[index].nextIndex;
8         for (int i = 0; i < size; i++)
9         {
10             if (frame[i].nextIndex < min)
11             {
12                 min = frame[i].nextIndex;
13                 index = i;
14             }
15         }
16     }
17     else if (key == 2) // ใช้สำหรับ LRU
18     {
19         int min = frame[index].pastIndex;
20         for (int i = 0; i < size; i++)
21         {
22             if (frame[i].pastIndex < min)
23             {
24                 min = frame[i].pastIndex;
25                 index = i;
26             }
27         }
28     }
29     return index;
30 }

```

ฟังก์ชัน min() นี้ใช้ในการค้นหาดัชนีของเฟรมที่มีค่า nextIndex หรือ pastIndex ต่ำสุดโดยมีลักษณะการทำงานดังนี้:

1. รับอาร์เรย์ของโครงสร้าง Frame และขนาดของอาร์เรย์นั้น เป็นข้อมูลเข้าเพื่อทำการค้นหาดัชนี
2. ตรวจสอบค่า key เพื่อกำหนดว่าจะใช้ในการค้นหาดัชนีของเฟรมที่มีค่า nextIndex หรือ pastIndex ต่ำสุด
3. วนลูปผ่านอาร์เรย์ของเฟรม เพื่อค้นหาดัชนีที่มีค่า nextIndex หรือ pastIndex ต่ำสุด
  - หากค่า key เท่ากับ 1 (ใช้สำหรับ OPT) จะเปรียบเทียบค่า nextIndex และเลือกดัชนีที่มีค่าน้อยที่สุด
  - หากค่า key เท่ากับ 2 (ใช้สำหรับ LRU) จะเปรียบเทียบค่า pastIndex และเลือกดัชนีที่มีค่าน้อยที่สุด
4. คืนค่าดัชนีของเฟรมที่มีค่า nextIndex หรือ pastIndex ต่ำสุดที่พบในการวนลูป

ฟังก์ชัน min() นี้ถูกออกแบบมาเพื่อใช้ในการค้นหาดัชนีของเฟรมที่มีค่า nextIndex หรือ pastIndex ที่น้อยที่สุดเพื่อใช้ในการแทนที่หน้าในกรณีของวิธีการจัดการหน้า OPT และ LRU ตามลำดับ



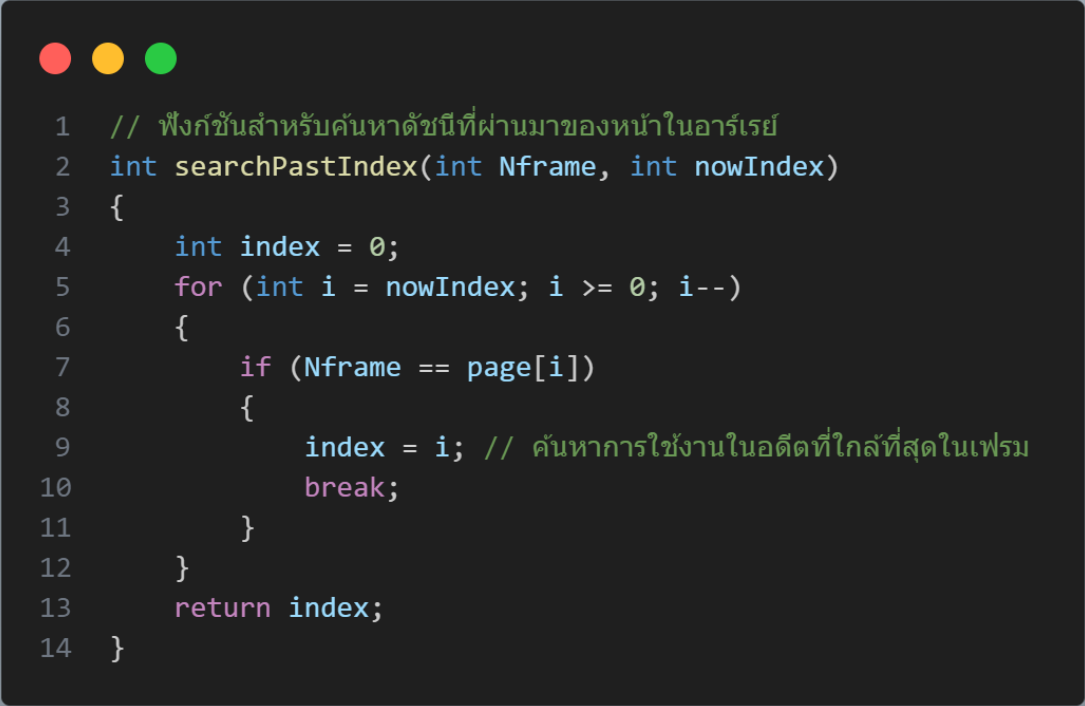


```
1  // ฟังก์ชันสำหรับค้นหาดัชนีถัดไปของหน้าในอาร์เรย์
2  int searchNextIndex(int Nframe, int nowIndex)
3  {
4      int index = 0;
5      for (int i = nowIndex + 1; i < 19; i++)
6      {
7          if (Nframe == page[i])
8          {
9              index = i; // หากเข้าถึงในอนาคต
10             break;
11         }
12     }
13     return index;
14 }
```

ฟังก์ชัน `searchNextIndex()` นี้ใช้ในการค้นหาดัชนีถัดไปของหน้าในอาร์เรย์โดยมีลักษณะการทำงานดังนี้:

1. รับค่าหน้าที่ต้องการค้นหา (Nframe) และดัชนีปัจจุบัน (nowIndex) เป็นข้อมูลเข้า
2. วนลูปผ่านหน้าที่ตามหลังดัชนีปัจจุบันเพื่อค้นหาหน้าต่อไปที่มีค่าเท่ากับ Nframe
3. หากพบหน้าที่มีค่าเท่ากับ Nframe ในอนาคต จะคืนค่าดัชนีของหน้านั้นออกมา
4. หากไม่พบหน้าที่มีค่าเท่ากับ Nframe ในอนาคต จะคืนค่า 0 แสดงว่าไม่มีหน้าที่มีค่าเท่ากับ Nframe ในอนาคต

ฟังก์ชัน `searchNextIndex()` นี้ถูกออกแบบมาเพื่อใช้ในการค้นหาดัชนีถัดไปของหน้าที่มีค่าเท่ากับ Nframe ในอาร์เรย์ของหน้า โดยการคืนค่าดัชนีที่พบในอนาคตหรือคืนค่า 0 หากไม่พบหน้าที่มีค่าเท่ากับ Nframe ในอนาคต



```

1  // ฟังก์ชันสำหรับค้นหาคำที่ผ่านของหน้าในอาร์เรย์
2  int searchPastIndex(int Nframe, int nowIndex)
3  {
4      int index = 0;
5      for (int i = nowIndex; i >= 0; i--)
6      {
7          if (Nframe == page[i])
8          {
9              index = i; // ค้นหาการใช้งานในอดีตที่ใกล้ที่สุดในเฟรม
10             break;
11         }
12     }
13     return index;
14 }

```

ฟังก์ชัน `searchPastIndex()` นี้ใช้ในการค้นหาคำที่ผ่านของหน้าในอาร์เรย์โดยมีลักษณะการทำงานดังนี้:

1. รับค่าหน้าที่ต้องการค้นหา (Nframe) และดัชนีปัจจุบัน (nowIndex) เป็นข้อมูลเข้า
2. วนลูปผ่านหน้าที่อยู่ก่อนหน้าดัชนีปัจจุบันเพื่อค้นหาหน้าที่มีค่าเท่ากับ Nframe
3. หากพบหน้าที่มีค่าเท่ากับ Nframe ในอดีต จะคืนค่าดัชนีของหน้านั้นออกมา
4. หากไม่พบหน้าที่มีค่าเท่ากับ Nframe ในอดีต จะคืนค่า 0 แสดงว่าไม่มีหน้าที่มีค่าเท่ากับ Nframe ในอดีต

ฟังก์ชัน `searchPastIndex()` นี้ถูกออกแบบมาเพื่อใช้ในการค้นหาคำที่ผ่านของหน้าที่มีค่าเท่ากับ Nframe ในอาร์เรย์ของหน้า โดยการคืนค่าดัชนีที่พบในอดีตหรือคืนค่า 0 หากไม่พบหน้าที่มีค่าเท่ากับ Nframe ในอดีต

ผลลัพธ์

FIFO

```
===== Page Replacement Program =====
Please input 19 pages : 2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

Your input : 2 1 2 3 7 6 2 3 4 2 1 5 6 3 2 3 6 2 1

===== Please select a menu =====
1. FIFO
2. LRU
3. OPT
4. Exit Program
Please input a key to continue : 1
Please input the number of frames : 4

Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [ ]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (FIFO) : 1
=====

Input page : 1
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (FIFO) : 2
=====
```

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [ ]

Frame 4 : [ ]

=====

Page fault (FIFO) : 2

=====

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [ ]

=====

Page fault (FIFO) : 3

=====

Input page : 7

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (FIFO) : 4

=====

Input page : 6

Frame Content :

Frame 1 : [6]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (FIFO) : 5

=====

Input page : 2

Frame Content :

Frame 1 : [6]

Frame 2 : [2]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (FIFO) : 6

=====

Input page : 3

Frame Content :

Frame 1 : [6]

Frame 2 : [2]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (FIFO) : 6

=====

```
Input page : 4
Frame Content :
Frame 1 : [6]
Frame 2 : [2]
Frame 3 : [4]
Frame 4 : [7]
=====
Page fault (FIFO) : 7
=====

Input page : 2
Frame Content :
Frame 1 : [6]
Frame 2 : [2]
Frame 3 : [4]
Frame 4 : [7]
=====
Page fault (FIFO) : 7
=====

Input page : 1
Frame Content :
Frame 1 : [6]
Frame 2 : [2]
Frame 3 : [4]
Frame 4 : [1]
=====
Page fault (FIFO) : 8
=====
```

Input page : 5

Frame Content :

Frame 1 : [5]

Frame 2 : [2]

Frame 3 : [4]

Frame 4 : [1]

=====

Page fault (FIFO) : 9

=====

Input page : 6

Frame Content :

Frame 1 : [5]

Frame 2 : [6]

Frame 3 : [4]

Frame 4 : [1]

=====

Page fault (FIFO) : 10

=====

Input page : 3

Frame Content :

Frame 1 : [5]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [1]

=====

Page fault (FIFO) : 11

=====

```
Input page : 2
Frame Content :
Frame 1 : [5]
Frame 2 : [6]
Frame 3 : [3]
Frame 4 : [2]
=====
Page fault (FIFO) : 12
=====

Input page : 3
Frame Content :
Frame 1 : [5]
Frame 2 : [6]
Frame 3 : [3]
Frame 4 : [2]
=====
Page fault (FIFO) : 12
=====

Input page : 6
Frame Content :
Frame 1 : [5]
Frame 2 : [6]
Frame 3 : [3]
Frame 4 : [2]
=====
Page fault (FIFO) : 12
=====
```



```

Input page : 2
Frame Content :
Frame 1 : [5]
Frame 2 : [6]
Frame 3 : [3]
Frame 4 : [2]
=====
Page fault (FIFO) : 12
=====

Input page : 1
Frame Content :
Frame 1 : [1]
Frame 2 : [6]
Frame 3 : [3]
Frame 4 : [2]
=====
Page fault (FIFO) : 13
=====

```

ผลลัพธ์ของวิธีการจัดการหน้าโดยใช้วิธี FIFO ในการแทนที่หน้ามีดังนี้:

1. หน้าแรกที่ยังถึงคือหน้า 2 ซึ่งไม่มีเฟรมใดเก็บข้อมูลอยู่ จึงต้องมี page fault และใส่หน้า 2 เข้าไปในเฟรมตำแหน่งแรก
2. หลังจากนั้นเมื่ออ้างถึงหน้า 1 ในรอบถัดไป หน้า 1 จะถูกใส่ลงในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
3. ในการอ้างถึงหน้า 2 ซ้ำ เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว ไม่เกิด page fault แต่จำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น
4. เมื่ออ้างถึงหน้า 3 จะต้องนำหน้า 3 เข้าไปใส่ในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
5. เมื่ออ้างถึงหน้า 7 จะต้องนำหน้า 7 เข้าไปใส่ในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
6. เมื่ออ้างถึงหน้า 6 จะต้องนำหน้า 6 เข้าไปใส่ในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
7. เมื่ออ้างถึงหน้า 2 ซ้ำ จะไม่เกิด page fault เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น
8. เมื่ออ้างถึงหน้า 3 ซ้ำ จะไม่เกิด page fault เนื่องจากหน้า 3 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น
9. เมื่ออ้างถึงหน้า 4 จะต้องนำหน้า 4 เข้าไปใส่ในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
10. หลังจากนั้นจะเห็นว่าหน้าที่อ้างถึงต่อมามีในเฟรมทั้งหมดแล้ว จึงไม่เกิด page fault ต่อมา

จากผลลัพธ์ด้านบน เมื่อใช้วิธี FIFO ในการจัดการหน้า จำนวน page fault รวมทั้งหมดคือ 13 รอบ โดยเฟรมมีขนาดเท่ากับ 4 หน้า ซึ่งเป็นผลลัพธ์ที่ต้องตามวิธีการทำงานของวิธี FIFO คือ หน้าที่ยังถึงใหม่จะถูกเพิ่มเข้าไปในเฟรมตำแหน่งที่ว่าง (หากมี) และหน้าที่ยังถึงน้อยที่สุดจะถูกนำออกจากเฟรม เมื่อเฟรมเต็ม ณ จุดนั้นๆ การเพิ่มหน้าใหม่เข้าไปในเฟรมจะทำให้เกิด page fault โดยที่หน้าที่ถูกนำออกจากเฟรมแล้วจะไม่ถูกใช้อีกต่อไป จนกว่าจะมีการอ้างถึงใหม่เข้ามาในอนาคต

## LRU

```
Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [ ]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (LRU) : 1
=====

Input page : 1
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (LRU) : 2
=====

Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (LRU) : 2
=====
```

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [ ]

=====

Page fault (LRU) : 3

=====

Input page : 7

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (LRU) : 4

=====

Input page : 6

Frame Content :

Frame 1 : [2]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (LRU) : 5

=====

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (LRU) : 5

=====

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (LRU) : 5

=====

Input page : 4

Frame Content :

Frame 1 : [2]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [4]

=====

Page fault (LRU) : 6

=====

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [6]

Frame 3 : [3]

Frame 4 : [4]

=====

Page fault (LRU) : 6

=====

Input page : 1

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [4]

=====

Page fault (LRU) : 7

=====

Input page : 5

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [5]

Frame 4 : [4]

=====

Page fault (LRU) : 8

=====

Input page : 6

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [5]

Frame 4 : [6]

=====

Page fault (LRU) : 9

=====

Input page : 3

Frame Content :

Frame 1 : [3]

Frame 2 : [1]

Frame 3 : [5]

Frame 4 : [6]

=====

Page fault (LRU) : 10

=====

Input page : 2

Frame Content :

Frame 1 : [3]

Frame 2 : [2]

Frame 3 : [5]

Frame 4 : [6]

=====

Page fault (LRU) : 11

=====

```

Input page : 3
Frame Content :
Frame 1 : [3]
Frame 2 : [2]
Frame 3 : [5]
Frame 4 : [6]
=====
Page fault (LRU) : 11
=====

Input page : 6
Frame Content :
Frame 1 : [3]
Frame 2 : [2]
Frame 3 : [5]
Frame 4 : [6]
=====
Page fault (LRU) : 11
=====

Input page : 2
Frame Content :
Frame 1 : [3]
Frame 2 : [2]
Frame 3 : [5]
Frame 4 : [6]
=====
Page fault (LRU) : 11
=====

```

```

Input page : 1
Frame Content :
Frame 1 : [3]
Frame 2 : [2]
Frame 3 : [1]
Frame 4 : [6]
=====
Page fault (LRU) : 12
=====

```

ผลลัพธ์ที่ได้จากการใช้วิธี LRU (Least Recently Used) ในการจัดการหน้ามีดังนี้:

1. เมื่อมีการอ้างถึงหน้า 2 ครั้งแรก ซึ่งหน้านี้อาจไม่มีในเฟรม จึงเกิด page fault 2 ครั้ง และต้องนำหน้า 2 และ 1 เข้าไปในเฟรม
2. เมื่อมีการอ้างถึงหน้า 2 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น

3. เมื่อมีการอ้างถึงหน้า 3 จะต้องนำหน้า 3 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
4. เมื่อมีการอ้างถึงหน้า 7 จะต้องนำหน้า 7 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
5. เมื่อมีการอ้างถึงหน้า 6 จะต้องนำหน้า 6 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
6. เมื่อมีการอ้างถึงหน้า 2 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น
7. เมื่อมีการอ้างถึงหน้า 3 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 3 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น
5. เมื่อมีการอ้างถึงหน้า 4 จะต้องนำหน้า 4 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
6. หลังจากนั้นจะเห็นว่าหน้าที่อ้างถึงต่อมามีในเฟรมทั้งหมดแล้ว จึงไม่เกิด page fault ต่อมา

จากผลลัพธ์ด้านบน เมื่อใช้วิธี LRU ในการจัดการหน้า จำนวน page fault รวมทั้งหมดคือ 12 รอบ โดยที่เฟรมมีขนาดเท่ากับ 4 หน้า ซึ่งเป็นผลลัพธ์ที่ต้องดูตามวิธีการทำงานของวิธี LRU คือ หน้าที่ไม่ถูกอ้างถึงนานที่สุดจะถูกนำออกจากเฟรมเมื่อมีการอ้างถึงใหม่เข้ามาและจำนวนเฟรมที่ใช้งานไม่เพิ่มขึ้น ณ จุดนั้นๆ การเพิ่มหน้าใหม่เข้าไปในเฟรมจะทำให้เกิด page fault โดยที่หน้าที่ถูกนำออกจากเฟรมแล้วจะไม่ถูกใช้อีกต่อไป จนกว่าจะมีการอ้างถึงใหม่เข้ามาในอนาคต



OPT

```
Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [ ]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (OPT) : 1
=====

Input page : 1
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (OPT) : 2
=====

Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [ ]
Frame 4 : [ ]
=====
Page fault (OPT) : 2
=====
```

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [ ]

=====

Page fault (OPT) : 3

=====

Input page : 7

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [7]

=====

Page fault (OPT) : 4

=====

Input page : 6

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [6]

=====

Page fault (OPT) : 5

=====

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [6]

=====

Page fault (OPT) : 5

=====

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [6]

=====

Page fault (OPT) : 5

=====

Input page : 4

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [4]

Frame 4 : [6]

=====

Page fault (OPT) : 6

=====

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [4]

Frame 4 : [6]

=====

Page fault (OPT) : 6

=====

Input page : 1

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [4]

Frame 4 : [6]

=====

Page fault (OPT) : 6

=====

Input page : 5

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [5]

Frame 4 : [6]

=====

Page fault (OPT) : 7

=====

Input page : 6

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [5]

Frame 4 : [6]

=====

Page fault (OPT) : 7

=====

Input page : 3

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [6]

=====

Page fault (OPT) : 8

=====

Input page : 2

Frame Content :

Frame 1 : [2]

Frame 2 : [1]

Frame 3 : [3]

Frame 4 : [6]

=====

Page fault (OPT) : 8

=====

```

Input page : 3
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [3]
Frame 4 : [6]
=====
Page fault (OPT) : 8
=====

Input page : 6
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [3]
Frame 4 : [6]
=====
Page fault (OPT) : 8
=====

Input page : 2
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [3]
Frame 4 : [6]
=====
Page fault (OPT) : 8
=====

```

```

Input page : 1
Frame Content :
Frame 1 : [2]
Frame 2 : [1]
Frame 3 : [3]
Frame 4 : [6]
=====
Page fault (OPT) : 8
=====

```

ผลลัพธ์ที่ได้จากการใช้วิธี OPT (Optimal Page Replacement) ในการจัดการหน้ามีดังนี้:

1. เมื่อมีการอ้างถึงหน้า 2 ครั้งแรก ซึ่งหน้านี้อาจไม่มีในเฟรม จึงเกิด page fault 2 ครั้ง และต้องนำหน้า 2 และ 1 เข้าไปในเฟรม
2. เมื่อมีการอ้างถึงหน้า 2 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างถึงไม่เพิ่มขึ้น

3. เมื่อมีการอ้างอิงหน้า 3 จะต้องนำหน้า 3 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
4. เมื่อมีการอ้างอิงหน้า 7 จะต้องนำหน้า 7 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
5. เมื่อมีการอ้างอิงหน้า 6 จะต้องนำหน้า 6 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
6. เมื่อมีการอ้างอิงหน้า 2 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 2 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างอิงถึงไม่เพิ่มขึ้น
7. เมื่อมีการอ้างอิงหน้า 3 อีกครั้งหนึ่ง จะไม่เกิด page fault เนื่องจากหน้า 3 อยู่ในเฟรมแล้ว และจำนวนเฟรมที่ถูกอ้างอิงถึงไม่เพิ่มขึ้น
8. เมื่อมีการอ้างอิงหน้า 4 จะต้องนำหน้า 4 เข้าไปในเฟรมตำแหน่งที่ว่าง ซึ่งเกิด page fault อีกครั้ง
9. หลังจากนั้นจะเห็นว่าหน้าที่อ้างอิงต่อมามีในเฟรมทั้งหมดแล้ว จึงไม่เกิด page fault ต่อมา

จากผลลัพธ์ด้านบน เมื่อใช้วิธี OPT ในการจัดการหน้า จำนวน page fault รวมทั้งหมดคือ 8 รอบ โดยที่เฟรมมีขนาดเท่ากับ 4 หน้า ซึ่งเป็นผลลัพธ์ที่ต้องตามวิธีการทำงานของวิธี OPT คือ หน้าที่ไม่ถูกอ้างอิงอีกต่อไปในอนาคตจะถูกนำออกจากเฟรมเมื่อมีการอ้างอิงใหม่เข้ามา ทำให้เฟรมมีการเปลี่ยนแปลงตามหน้าที่ถูกอ้างอิงในอนาคตที่จะมีความน่าจะเป็นที่มากที่สุดและจำนวนเฟรมที่ใช้งานไม่เพิ่มขึ้น ณ จุดนั้นๆ การเพิ่มหน้าใหม่เข้าไปในเฟรมจะทำให้เกิด page fault โดยที่หน้าที่ถูกนำออกจากเฟรมแล้วจะไม่ถูกใช้อีกต่อไป จนกว่าจะมีการอ้างอิงใหม่เข้ามาในอนาคต

### สรุปผลการทดลอง

การทดลอง page replacement ด้วยวิธีการ FIFO, LRU, และ OPT ให้ผลลัพธ์ดังนี้:

1. FIFO (First-In-First-Out):
  - FIFO ใช้หลักการแทนที่หน้าด้วยหน้าที่เข้ามาก่อนสุดในเฟรม โดยไม่คำนึงถึงการอ้างอิงหน้าในอดีต
  - ผลลัพธ์จะมีการแทนที่หน้าโดยตลอด แม้ว่าหน้านั้น ๆ จะมีการอ้างอิงหรือไม่ก็ตาม
  - จำนวนหน้าผิดพลาดจะขึ้นอยู่กับลำดับของหน้าที่ถูกอ้างอิงเข้ามา
2. LRU (Least Recently Used):
  - LRU ใช้หลักการแทนที่หน้าด้วยหน้าที่ไม่ถูกอ้างอิงนานที่สุดในอดีต
  - ผลลัพธ์จะมีการแทนที่หน้าที่ไม่ถูกอ้างอิงในอดีตนานที่สุด โดยให้ความสำคัญกับประวัติการอ้างอิงหน้าในอดีต
  - จำนวนหน้าผิดพลาดจะลดลงเมื่อหน้าที่ถูกอ้างอิงบ่อย ๆ จะไม่ถูกแทนที่ในเฟรม

### 3. OPT (Optimal Page Replacement):

- OPT ใช้หลักการแทนที่หน้าด้วยหน้าที่จะไม่ถูกอ้างอิงถึงนานที่สุดในอนาคต
- ผลลัพธ์ที่ได้จะเป็นการแทนที่หน้าโดยคาดการณ์อนาคตที่ดีที่สุด โดยพิจารณาจากลำดับของหน้าที่ถูกอ้างอิงถึงในอนาคต
- จำนวนหน้าผิดพลาดจะต่ำที่สุด เนื่องจากหน้าที่จะไม่ถูกอ้างอิงถึงในอนาคตจะไม่ถูกแทนที่ในเฟรม

ดังนั้น การเลือกใช้วิธีการแทนที่หน้าในการจัดการหน่วยความจำมีผลต่อประสิทธิภาพของระบบและจำนวนหน้าผิดพลาดที่เกิดขึ้นในแต่ละกรณี

### สื่อ / เอกสารอ้างอิง

อาจารย์ปิยพล ยืนยงสถาวร: เอกสารประกอบการสอน 9 หน่วยความจำเสมือน (Virtual Memory)