

REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 운영체제

학수번호 : SWE3004_43

과 제 명 : virtual memory management
(project-2)

학 과 : 통계학과

학 년 : 4

학 번 : 2017310426

이 름 : 김혜영

목차

1. 서론
2. MIN 기법
3. FIFO 기법
4. LRU 기법
5. LFU 기법
6. WS 기법
7. 결론

1. 서론

본 프로젝트의 전체적인 내용은 다양한 Virtual Memory Management 기법들을 구현하는 것이다. 그 종류로는 MIN, FIFO, LRU, LFU, WS Memory Management의 5가지이며, page reference string을 입력 받아 각 기법을 사용했을 경우의 memory residence set 변화과정과 page fault 발생 과정을 출력하도록 하는 프로그램을 구현하는 것이 그 목적에 해당한다. 프로젝트를 진행하는데 사용된 개발 플랫폼은 MS플랫폼이며 C언어로 작성된 소스코드를 Visual studio 2019에서 컴파일하여 수행하였다. page frame 할당량 및 window size 등은 입력 받은 값으로 수행되도록 하며 초기 할당된 page frame들은 모두 비어 있는 것으로 가정한다. 긴 가로 선으로 나누어지는 출력형태는 각각 위에서부터 시작하여 current time, current reference string(현재 참조하는 page#), 해당하는 시간의 memory residence set, 현재 시간의 page fault 발생여부에 해당하며 마지막에는 총 page fault 발생 횟수를 출력하도록 한다. Memory residence set에서 비어 있는 공간은 "X"가 출력되도록 설정하였으며 Page fault여부를 나타낼 때 page fault가 발생하면 F를, 그렇지 않으면 0이 출력되도록 하였다. Page의 번호는 0~15까지 가능하도록 정하며 Test의 편의를 위해 page refence string의 길이는 100이하로 가정한다. 처음으로 page reference가 시작되는 시간은 1에서부터 시작하는 것으로 가정한다. 또한 WS기법을 제외한 다른 모든 기법에서 page frame이 비어 있는 경우에는 가장 위쪽의 Page frame에서부터 채워지는 것을 원칙으로 한다. 각 기법 별로 하나의 함수들을 선언하여 사용하는데 각각의 함수들은 두 개의 인자를 가지게 되는데 page개수, page frame개수, window size, reference string길이의 내용을 담고 있는 arr1과 page reference string의 정보를 담고 있는 arr2라는 인자가 이에 해당한다.

2. MIN 기법

```
void MIN(int* arr1, int* arr2)
{
    int pf = arr1[1];
    int slength = arr1[3];
    int pageframe[15];
    memset(pageframe, -1, sizeof(pageframe));
    int pagefault = 1;
    int pftime[100] = { 0 };
    int gothrough = 1;
    int rset[16][100];
    int pfcount = 0;
```

위의 이미지는 함수 내부에서 필요한 변수들에 해당한다. pf 변수는 page frame 개수를 의미하고 slength 는 page reference string 길이이며 현 시점에서 page frame 내부의 상태를 표현하기 위한 pageframe 배열을 선언하여 -1 로 (page 번호가 0 부터 시작되기 때문에) 초기화해준다. Page fault 여부를 나타내는 변수 pagefault 와 Page fault 가 발생하는 시점을 나타내기 위한 pftime 배열을

선언한다. Pftime 이 0 인 경우에는 page fault 가 발생하지 않은 경우를 나타낸다. 또한 추후에 조건문 수행을 위한 gothrough 변수와 출력값을 나타내기 위한 rset 배열과 pfcount 변수(page fault 발생 횟수를 의미)를 선언한다.

```
17   for (int i = 0; i < slength; i++) {
18       for (int j = 0; j < pf; j++) {
19           if (pageframe[j] == -1) {
20               pageframe[j] = arr2[i];
21               gothrough = 0;
22               pftime[i] = i+1;
23               pagefault = 1;
24               break;
25           }
26           else if (pageframe[j] != -1 && arr2[i] == pageframe[j]) {
27               pagefault = 0;
28               break;
29           }
30       }
31   }
```

우선 reference string 의 길이만큼 과정을 반복할 수 있도록 큰 for 문을 만들어주고 그 안에 pagefault 발생여부를 판단하기 위해서 page frame 개수만큼 검사를 반복하는 반복문을 넣어준다. 이어지는 조건문은 해당하는 page frame 이 비어 있는 경우에는 그 page frame 에 현재 참조하는 page 번호를 넣어주고 현재 이미지 이후에 해당하는 victim 을 선정하는 과정을 수행하지 않도록 하기 위해 gothrough 변수를 0 으로 만들어준다. 또한 page fault 가 발생했기 때문에 pagefault 변수는 1 로 만들어주고 pftime 을 현재 시간으로 만들어주며(이때 i+1 이 현재시간이 되는 이유는 시작 시간은 1 부터로 가정했으나 가장 바깥쪽의 큰 반복문에서 i 는 0 부터 시작하기 때문이다.) 반복문을 빠져나오기 위한 break 를 사용한다. 다음 Pageframe 이 비어있지 않은 경우에는 pageframe 에 들어있는 값이 현재 참조하는 page 번호와 같은 지 확인하고 같은 경우에 pagefault 가 발생하지 않는다는 의미로 0 을 pagefault 변수에 넣어주고 반복문을 빠져나오도록 한다.

```

32 if (pagefault == 1 && gothrough==1) {
33
34     pftime[i] = i+1;
35     int current = i;
36     int min[15][2] = { 0 };
37     for (int k = 0; k < pf; k++) {
38         while (pageframe[k] != arr2[current]) {
39             current++;
40             if (current > slength) break;
41         }
42         min[k][0] = pageframe[k];
43         min[k][1] = current; // 미래 인덱스값
44         current = i;
45     }
46     int temp[1][2];
47     for (int m = pf - 1; m > 0; m--) {
48         for (int n = 0; n < m; n++) {
49             if (min[n][1] < min[n+1][1]) {
50                 temp[0][1] = min[n+1][1];
51                 min[n+1][1] = min[n][1];
52                 min[n][1] = temp[0][1];
53                 temp[0][0] = min[n+1][0];
54                 min[n+1][0] = min[n][0];
55                 min[n][0] = temp[0][0];
56             }
57             else if (min[n][1] == min[n+1][1]) {
58                 if (min[n][0] > min[n+1][0]) {
59                     temp[0][1] = min[n+1][1];
60                     min[n+1][1] = min[n][1];
61                     min[n][1] = temp[0][1];
62                     temp[0][0] = min[n+1][0];
63                     min[n+1][0] = min[n][0];
64                     min[n][0] = temp[0][0];
65                 }
66             }
67         }
68     }

```

다음으로 page fault 가 발생하고 victim 선정이 필요한 경우에는 먼저 page fault 발생시점을 저장한다. 그리고 current 변수에 현재시점을 나타내기 위해 i 값을 넣어주고 미래 참조 위치에 따라 page frame 안의 page 번호를 sort 하기 위해 min 이라는 이차원 배열을 선언한다. 이때 이차원 배열로 선언하는 이유는 첫번째 열에는 page 번호를 저장하고 두번째 열에는 나중에 참조되는 위치를 저장하기 위함이다. 우선 모든 page frame 내부의 page 들에 대해서 현재시점 이후에 참조되는 위치를 찾기 위한 반복문을 수행한다. 그리고 이후에 한번도 참조되지 않는 경우는 current 값이 reference string 길이보다 커지는 순간 반복문을 빠져나오도록 설계한다. while 문을 빠져나와 얻게 되는 current 값은 미래에 참조되는 시점에 해당하는 값인데 이를 min 배열의 두번째 열에 저장한다. 그리고 min 배열의 첫번째 열에는 page 번호가 저장되도록 한다. 미래에 참조되는 시점을 기준으로 정렬을 하기 위해서 temp 배열을 두고 다음 반복문을 수행하는데 page frame 의 개수가 크지 않기 때문에 위의 정렬방법을 사용해도 괜찮을 것이라고 판단하여 수행하였고 미래에 참조되는 시점이 같게 나오는 경우, 즉, 두 page 모두가 앞으로 참조되지 않을 경우에는 page 번호가 작은 것이 victim 으로 선정되도록 규칙을 정하여 정렬기법의 조건으로 사용하였다.

```

69         int s = 0;
70         while (pageframe[s] != min[0][0]) {
71             s++;
72         }
73         pageframe[s] = arr2[i];
74     }
75     gothrough = 1;
76     pagefault = 1;
77     for (int p = 0; p < pf; p++) {
78         rset[p + 1][i] = pageframe[p];
79     }
80 }
81
82 }

```

위해서 구한 정렬 값을 이용하여 가장 나중에 참조되는 page 번호에 해당하는 min[0][0]값과 일치하는 pageframe 을 찾아서 그에 해당하는 page 번호를 현재 참조한 page 번호로 바꿔준다. 그리고 큰 조건문을 빠져나온뒤 gothrough 값과 pagefault 의 값을 원래대로 돌려주고 현재 page frame 에 들어있는 page 번호들을 출력 값을 위한 rset 배열에 저장한 뒤 바깥쪽 반복문의 위쪽으로 돌아가 반복 혹은 반복문을 빠져나온다.

```

88     printf("\n-----\n");
89     for (int i = 0; i < slength; i++) {
90         printf("%3d ", arr2[i]);
91     }
92     printf("\n-----\n");
93     for (int i = 1; i < pf+1; i++) {
94         for (int j = 0; j < slength; j++) {
95             if(rset[i][j] != -1) printf("%3d ", rset[i][j]);
96             else printf("%3c ", 'X');
97         }
98         printf("\n");
99     }
100     printf("\n-----\n");
101     for (int i = 0; i < slength; i++) {
102         if (ptime[i] != 0) {
103             printf("%3c ", 'F');
104             pfcunt++;
105         }
106         else {
107             printf("%3d ", ptime[i]);
108         }
109     }
110     printf("\n-----\n");
111     printf("Total Page fault : %d \n", pfcunt);
112 }

```

위의 내용은 출력 포맷에 해당하는 내용이다.

3. FIFO 기법

```

122     int timestamp[100];
123     memset(timestamp, -1, sizeof(timestamp));

```

위의 min기법과 동일한 변수들을 선언해주고 추가된 변수로는 최근 참조된 시점을 저장하기 위한 timestamp배열이 있다.

```

127     for (int i = 0; i < slength; i++) {
128         for (int j = 0; j < pf; j++) {
129             if (pageframe[j] == -1) {
130                 pageframe[j] = arr2[i];
131                 timestamp[i] = arr2[i];
132                 gothrough = 0;
133                 pftime[i] = i + 1;
134                 pagefault = 1;
135                 break;
136             }
137             else if (pageframe[j] != -1 && arr2[i] == pageframe[j]) {
138                 pagefault = 0;
139                 break;
140             }
141         }
142         if (pagefault == 1 && gothrough == 1) {
143             pftime[i] = i + 1;
144
145             int t = 0;
146             while(timestamp[t]==-1){
147                 t++;
148             }
149
150             for (int a = 0; a < pf; a++) {
151                 if (pageframe[a] == timestamp[t]) {
152                     pageframe[a] = arr2[i];
153                     timestamp[i] = arr2[i];
154                     timestamp[t] = -1;
155                 }
156             }
157             gothrough = 1;
158             pagefault = 1;
159             for (int p = 0; p < pf; p++) {
160                 rset[p + 1][i] = pageframe[p];
161             }
162         }
163     }

```

우선 MIN기법과 동일하게 가장 바깥쪽 반복문을 넣어주고 page fault 발생 여부를 판단해주는 반복문을 동일하게 작성한다. 이때 MIN함수와 다른 점은 page frame이 비어 있는 경우 참조되는 page들의 번호가 timestamp배열의 해당시점을 인덱스로 하는 곳에 저장된다는 것이다. 그리고 이어지는 조건문에서 page frame이 가득 차 있지만 page fault가 발생하는 경우에 victim을 선정하는 방법에 대한 내용을 구현하도록 한다. 먼저 page fault 발생 시점을 저장해주고 while문을 이용하여 현재 page frame 내부에 있는 page들 중에 가장 먼저 참조된 page가 참조된 시점을 찾는다.(그 시점이 변수 t의 값에 해당한다.) 다음으로 이어지는 반복문에서 해당 page번호와 동일한 page값을 가지는 page frame을 찾아서 그 값을 현재 참조하는 page번호로 바꾸고 현재시점의 timestamp배열 값도 현재 참조하는 page번호로 저장해준다. 또한 사용된 t시점의 timestamp값은 다시 -1로 초기화해줘서 page들 별로 가장 최근에 참조된 위치를 timestamp배열을 통해 얻을 수 있도록 설정한다. gothrough값과 pagefault값을 다시 원래의 값으로 돌려주고 현재 시점의 page frame 내부의 상태를 rset배열에 저장하도록 하고 맨 위의 반복문으로 돌아간다. 위 이미지 이후에 이어지는 내용은 반복문 이후의 출력 포맷으로, MIN함수와 동일하다.

4. LRU 기법

```

203 | int timestamp[15] = { 0 };

```

LRU함수에서도 역시 위의 변수만 추가되었고 선언되는 나머지 변수들은 MIN함수의 경우와 동일하다. FIFO의 timestamp와 다른 것은 LRU함수의 timestamp의 인덱스는 각 page 번호에 해당하며 그 안에 참조되는 시점이 update되어 들어간다는 것이다. 결국 timestamp변수에는 그 page가 최근 참조된 시점이 저장된다.

```

207     for (int i = 0; i < slength; i++) {
208         timestamp[arr2[i]] = i + 1;
209         for (int j = 0; j < pf; j++) {
210             if (pageframe[j] == -1) {
211                 pageframe[j] = arr2[i];
212                 gothrough = 0;
213                 pftime[i] = i + 1;
214                 pagefault = 1;
215                 break;
216             }
217             else if (pageframe[j] != -1 && arr2[i] == pageframe[j]) {
218                 pagefault = 0;
219                 break;
220             }
221         }

```

먼저 위의 두 기법들과 동일한 큰 반복문이 만들어지고 현재 참조되는 page번호의 timestamp를 찍는 내용을 추가해준다. 이어지는 Page fault 발생여부를 판단하는 반복문과 조건문은 MIN기법과 동일하다.

```

222     if (pagefault == 1 && gothrough == 1) {
223         pftime[i] = i + 1;
224         int checkarr[15] = { 0 };
225         for (int k = 0; k < pf; k++) {
226             checkarr[k] = timestamp[pageframe[k]];
227         }
228         int temp;
229         for (int m = pf - 1; m > 0; m--) {
230             for (int n = 0; n < m; n++) {
231                 if (checkarr[n] > checkarr[n + 1]) {
232                     temp = checkarr[n + 1];
233                     checkarr[n + 1] = checkarr[n];
234                     checkarr[n] = temp;
235                 }
236             }
237         }
238         int t = 0;
239         while (checkarr[t] == 0) t++;
240         for (int s = 0; s < pf; s++) {
241             if (timestamp[pageframe[s]] == checkarr[t]) {
242                 pageframe[s] = arr2[i];
243                 break;
244             }
245         }
246         gothrough = 1;
247         pagefault = 1;
248         for (int p = 0; p < pf; p++) {
249             rset[p + 1][i] = pageframe[p];
250         }
251     }
252 }

```

Page fault가 발생하는 경우 동일하게 먼저 발생 시점을 저장해주고 현재 page frame에 들어있는 page들의 최근 참조 시점을 비교하여 정렬하기 위한 checkarr배열을 선언하고 반복문을 이용하여 timestamp배열에 저장되어 있는 값을 각 page에 대해서 checkarr값에 저장한다. 참조되는 시점이 동일한 경우는 없으므로 checkarr를 값이 작은 순으로 정렬하고 이때 page frame의 개수가 15가 아닌 경우 checkarr배열을 초기화할 때 사용한 0들이 앞쪽에 위치하게 되므로 첫번째 양수 값을 찾기 위해 while문을 사용해준다. Timestamp에 들어있는 값과 checkarr배열의 첫번째 양수 값과

일치하는 page가 들어있는 pageframe을 찾아 그 값을 현재 참조하는 page로 바꿔주고 break를 사용하여 반복문을 빠져나온다. gothrough값과 pagefault값을 다시 원래의 값으로 돌려주고 현재 시점의 page frame 내부의 상태를 rset배열에 저장하도록 하고 맨 위의 반복문으로 돌아간다. 위 이미지 이후에 이어지는 내용은 반복문 이후의 출력 포맷으로, MIN함수와 동일하다.

5. LFU 기법

```
295 | int count[15] = { 0 };
```

LFU기법에서 추가되는 변수는 현재시점까지 참조된 횟수를 저장하는 count배열이고 나머지는 MIN함수에서 선언된 변수들과 동일하다.

```
299 | for (int i = 0; i < slength; i++) {
300 |     count[arr2[i]]++;
301 |     for (int j = 0; j < pf; j++) {
302 |         if (pageframe[j] == -1) {
303 |             pageframe[j] = arr2[i];
304 |             gothrough = 0;
305 |             pftime[i] = i + 1;
306 |             pagefault = 1;
307 |             break;
308 |         }
309 |         else if (pageframe[j] != -1 && arr2[i] == pageframe[j]) {
310 |             pagefault = 0;
311 |             break;
312 |         }
313 |     }
```

큰 반복문 안에서 가장 먼저 실행되어야 하는 것은 참조되는 page를 인덱스로 하는 count배열 값을 1증가시켜 참조 횟수를 늘리는 것이다. 그리고 이어지는 Page fault 발생 여부를 판단해주는 반복문은 MIN함수의 내용과 동일하다.

```

314     if (pagefault == 1 && gothrough == 1) {
315         pftime[i] = i + 1;
316         int checkarr[15][2] = { 0 };
317         for (int k = 0; k < pf; k++) {
318             checkarr[k][0] = pageframe[k];
319             checkarr[k][1] = count[pageframe[k]];
320         }
321         int temp[1][2];
322         for (int m = pf-1; m > 0; m--) {
323             for (int n = 0; n < m; n++) {
324                 if (checkarr[n][1] > checkarr[n + 1][1]) {
325                     temp[0][1] = checkarr[n + 1][1];
326                     temp[0][0] = checkarr[n + 1][0];
327                     checkarr[n + 1][1] = checkarr[n][1];
328                     checkarr[n + 1][0] = checkarr[n][0];
329                     checkarr[n][1] = temp[0][1];
330                     checkarr[n][0] = temp[0][0];
331                 }
332                 else if (checkarr[n][1] == checkarr[n + 1][1]) {
333                     int startpoint = i-1;
334                     while (arr2[startpoint] != checkarr[n][0] && arr2[startpoint] != checkarr[n + 1][0]) {
335                         startpoint--;
336                     }
337                     if (arr2[startpoint] == checkarr[n][0]) {
338                         temp[0][1] = checkarr[n + 1][1];
339                         temp[0][0] = checkarr[n + 1][0];
340                         checkarr[n + 1][1] = checkarr[n][1];
341                         checkarr[n + 1][0] = checkarr[n][0];
342                         checkarr[n][1] = temp[0][1];
343                         checkarr[n][0] = temp[0][0];
344                     }
345                 }
346             }
347         }

```

Page frame이 차 있는 상태에서 page fault가 발생할 경우 먼저 page fault 발생시점을 저장해두고 참조 횟수에 따라 정렬을 하기 위한 checkarr라는 이차원배열을 선언한다. Page frame의 개수만큼 반복되는 반복문을 사용하여 checkarr의 첫번째 열에는 page frame안에 있는 page번호를 저장하고 두번째 열에는 그 page의 참조횟수를 저장한다. 참조횟수가 작은 순서대로 정렬되도록 하는 반복문을 시행하는데 이때 참조횟수가 같은 경우에는 LRU기법을 사용하여 Tie breaking을 해주도록 한다. 조건문의 else if 부분이 그에 해당하는 부분인데 먼저 현재시점보다 1작은 시점을 startpoint로 두고 while문을 사용하여 현 시점에서 가장 가까운 참조위치에 해당하는 page를 찾을 동안 startpoint를 1씩 줄인다. 그리고 현재시점과 비교하여 참조위치가 더 먼 page가 앞쪽에 정렬되도록 한다.

```

348
349     int s = 0;
350     while (pageframe[s] != checkarr[0][0]) {
351         s++;
352     }
353     pageframe[s] = arr2[i];
354 }
355 gothrough = 1;
356 pagefault = 1;
357 for (int p = 0; p < pf; p++) {
358     rset[p + 1][i] = pageframe[p];
359 }
360 }

```

정렬된 배열의 맨 첫번째 page번호와 동일한 값을 pageframe에서 찾고 그 pageframe값을 현재 참조되는 page번호로 바꿔준다. gothrough값과 pagefault값을 다시 원래의 값으로 돌려주고 현재 시점의 page frame 내부의 상태를 rset배열에 저장하도록 하고 맨 위의 반복문으로 돌아간다. 위 이미지 이후에 이어지는 내용은 반복문 이후의 출력 포맷으로, MIN함수와 동일하다.

6. WS 기법

```
392 void WS(int* arr1, int* arr2) {
393     int pnum = arr1[0];
394     int wsize = arr1[2];
395     int slength = arr1[3];
396     int pageframe[15];
397     memset(pageframe, -1, sizeof(pageframe));
398     int pftime[100] = { 0 };
399     int memorystate[15] = { 0 };
400     int rset[16][100] = { 0 };
401     int pfcount = 0;
402 }
```

WS기법을 수행하기 위한 WS함수에는 page개수를 나타내는 pnum변수, window size를 나타내는 wsize변수, reference string 길이를 나타내는 slength변수와 현 시점의 memory residence set상태를 나타내 주는 데에 사용하는 배열 pageframe, page fault가 발생하는 시점을 저장하는 pftime배열, window size내에 들어있는 page번호 정보를 나타내는데 사용되는 memorystate배열, 출력포맷에 사용되는 rset배열과 pfcount변수가 선언된다.

```
403     for (int i = 0; i < slength; i++) {
404         if (i >= wsize) {
405             for (int m = 0; m < 15; m++) {
406                 memorystate[m] = 0;
407                 pageframe[m] = -1;
408             }
409             for (int j = 0; j < wsize; j++) {
410                 memorystate[arr2[i - 1 - j]] = 1;
411             }
412             for (int k = 0; k < 15; k++) {
413                 if (memorystate[k] != 0) {
414                     pageframe[k] = k;
415                 }
416             }
417             if (memorystate[arr2[i]] == 0) {
418                 pageframe[arr2[i]] = arr2[i];
419                 pftime[i] = i + 1;
420                 memorystate[arr2[i]] = 1;
421             }
422             for (int p = 0; p < pnum; p++) {
423                 rset[p + 1][i] = pageframe[p];
424             }
425         }
426     }
427 }
```

우선 reference string의 길이만큼 반복하는 반복문을 넣어주고 window size보다 현시점이 큰 경우에(i 가 0부터 시작되지만 $i=0$ 일 때 시점은 1이기 때문에 $i \geq wsize$ 이다.) 시행하게되는 조건문을 넣어주는데 이에 대한 설명은 그 외의 경우에 동작을 설명한 후에 다시 설명하도록 한다. 시작지점에서 residence set은 비어 있는 것으로 가정했기 때문에 $i < wsize$ 인 경우에는 단지 page fault를 발생시키고 채우는 과정만 진행하면 된다. 따라서 pageframe배열의 page번호 인덱스로 가서 그에 해당하는 값을 참조하는 page번호로 바꿔주고 memorystate의 동일 인덱스에 해당하는 값도 1로 바꿔줘서 memory residence set이 채워지도록 한다. 그리고 그 후에는 memory residence set상

태를 나타내는 pageframe배열을 이용해 그 상태를 rset에 저장한다. 다음으로는 앞에서 언급한 $i \geq wsize$ 의 경우에는 먼저 모든 memorystate배열의 값과 pageframe 값을 원래의 값으로 초기화 시켜준다. 그리고 현재시점보다 window size만큼 전에 해당하는 시점에서부터 참조되는 page들에 대한 memorystate배열 값을 1로 두는 반복문을 수행한다. 이후에 memorystate가 1로 변경된 page들에 대한 pageframe배열 값을 해당 page번호가 되도록 작동하는 반복문을 한 번 더 수행한 뒤 전의 경우와 동일한($i < wsize$ 의 경우) 수행을 한다.

```


438 for (int i = 1; i < pnun + 1; i++) {
439     for (int j = 0; j < slength; j++) {
440         if (rset[i][j] != -1) printf("%3d ", rset[i][j]);
441         else printf("%3c ", 'X');
442     }
443     printf("\n");
444 }
445 printf("-----\n");

```

그 다음으로 이어지는 내용은 출력 포맷에 관한 것인데 memory residence set상태를 나타내는 곳에서 위와 같이 모든 page를 출력하기 위해 pnun만큼의 반복을 하는 반복문이 사용되었다는 것 외에는 MIN함수와 동일하다.

7. 결론

위의 소스코드를 아래와 같은 Input값을 입력하여 실행해보았다.

 Microsoft Visual Studio 디버그 콘솔

```

6 3 3 15
0 1 2 3 2 3 4 5 4 1 3 4 3 4 5

```

Output값:

```

MIN:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-----
0  1  2  3  2  3  4  5  4  1  3  4  3  4  5
-----
0  0  0  3  3  3  3  5  5  5  5  5  5  5  5
X  1  1  1  1  1  1  1  1  1  3  3  3  3  3
X  X  2  2  2  2  4  4  4  4  4  4  4  4  4
-----
F  F  F  F  0  0  F  F  0  0  F  0  0  0  0
-----
Total Page fault : 7

```

FIFO:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
0	0	0	3	3	3	3	3	3	1	1	1	1	1	5
X	1	1	1	1	1	4	4	4	4	3	3	3	3	3
X	X	2	2	2	2	2	5	5	5	5	4	4	4	4
F	F	F	F	0	0	F	F	0	F	F	F	0	0	F

Total Page fault : 10

LRU:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
0	0	0	3	3	3	3	3	3	1	1	1	1	1	5
X	1	1	1	1	1	4	4	4	4	4	4	4	4	4
X	X	2	2	2	2	2	5	5	5	3	3	3	3	3
F	F	F	F	0	0	F	F	0	F	F	0	0	0	F

Total Page fault : 9

LFU:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
0	0	0	3	3	3	3	3	3	3	3	3	3	3	3
X	1	1	1	1	1	4	5	4	4	4	4	4	4	4
X	X	2	2	2	2	2	2	2	1	1	1	1	1	5
F	F	F	F	0	0	F	F	F	F	0	0	0	0	F

Total Page fault : 9

WS:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
0	0	0	0	X	X	X	X	X	X	X	X	X	X	X
X	1	1	1	1	X	X	X	X	1	1	1	1	X	X
X	X	2	2	2	2	2	2	X	X	X	X	X	X	X
X	X	X	3	3	3	3	3	3	X	3	3	3	3	3
X	X	X	X	X	X	4	4	4	4	4	4	4	4	4
X	X	X	X	X	X	X	5	5	5	5	X	X	X	5
F	F	F	F	0	0	F	F	0	F	F	0	0	0	F

Total Page fault : 9

이와 같은 결과 값을 출력하게 되는데 각 기법에 맞게 정상적으로 출력되는 것을 알 수 있다. 위의 입력 값으로 수행해본 결과 Page fault 발생 횟수 값을 비교해봤을 때 그렇게 큰 차이는 느낄 수 없었지만 MIN기법이 강의 내용에서 배운 대로 실제상황에서 구현하는 것은 불가능하지만 가장 이상적인 효과를 낼 것으로 기대할 수 있는 기법인 만큼 위의 TEST에서 가장 적은 수의

Page fault를 발생시킨다는 것을 볼 수 있었다. 이 프로젝트를 진행하면서 virtual memory management에서 사용되는 다양한 기법들을 실제로 구현해보고 다양한 input을 넣어 test해볼 수 있었다. 각 기법들이 작동하는 방식과 세부적인 내용들에 대해 더욱 폭넓고 깊게 이해할 수 있었던 유익한 시간이었다.