컴퓨터학부 20172642 신형담

1-1. 제출 소스코드

1. opt.c : Optimal Algorithm 구현 코드
2. fifo.c : FIFO Algorithm 구현 코드
3. lru.c : LRU Algorithm 구현 코드
4. sc.c : Second-Chance Algorithm 구현 코드

각각의 소스코드를 gcc를 통해 컴파일 하여 실행 시 인자로 input 파일을 주면 결과를 출력한다.

1-2. 구현개요

OS의 가상 메모리 관리 기법인 페이지 교체 기법 4가지(OPT, FIFO, LRU, Second-Chance)를 구현하여 page frame 수와 page reference string 정보가 담긴 파일을 입력으로 받아 동작 과정을 시뮬레이션 하는 프로그램을 구현한다. 각각의 알고리즘을 각각의 프로그램으로 작성하였다.

1-3. 구현과정

- main()

```
int main(int argc, char* argv[]) {
    if(argc != 2) {
        fprintf(stderr, "check your input.\n");
        exit(1);
    }

    char* file_name = argv[1];
    FILE* fp;

    if((fp = fopen(file_name, "r")) == NULL) {
        fprintf(stderr, "fopen err %s\n", file_name);
        exit(1);
    }

    frame_num = atoi(fgets(buf, sizeof(buf), fp));
    fgets(buf, sizeof(buf), fp);
    times = count_times(buf);
    int frame[frame_num];
    memset(frame, -1, sizeof(frame));
    printf("Used method : OPT\n");
    printf("page reference string : %s\n", buf);
    printf("\tframe   ");
    for(int i = 1; i <= frame_num; i++) {
        printf("%d\t", i);
        if(i == frame_num)
            printf("page fault\n");
    }
    printf("time\n");
    opt(frame);
    printf("Number of page faults : %d times\n", opt_fault);
}
```

4개의 프로그램 모두 main 함수는 위와 같은 형식을 취한다. sc.c는 page frame배열을

second_chance()라는 함수에서 선언하는 것만이 다르다. fgets() 함수를 통해 파일을 읽고 전역변수인 frame_num에 프레임 수, times에 page reference의 수를 저장한다.

- count_times()

```
int count_times() {
    int cnt = 1;
    for(int i = 0; i < strlen(buf) - 1; i++) {
        if(buf[i] == ' ') {
            if(i == strlen(buf) - 2){
                continue;
            }
            cnt++;
        }
    }
    return cnt;
}
```

4개의 프로그램 모두 동일한 코드로 파일에서 읽어온 page reference string을 통해 총 몇번의 page reference가 등장하는지 리턴하하는 함수이다.

- opt.c

```
void set_page_ref(int page_ref[]) {
    char* tmp;
    for(int i = 0; i < times; i++) {
        if(i == 0)
            tmp = strtok(buf, " ");
        else
            tmp = strtok(NULL, " ");
        page_ref[i] = atoi(tmp);
    }

}
```

Optimal 방식은 page fault가 일어날 때 해당 page reference 다음에 등장 순번을 계산해야 한다. 이때 좀 더 간편한 작업을 위해 미리 page reference string을 토큰화 하여 int 형 배열에 저장하는 set_page_ref() 함수이다.

```
int next_offset(int frame[], int now, int f_index, int page_ref_arr[]) {
    for(int i = now + 1; i < times; i++) {
        if(frame[f_index] == page_ref_arr[i])
            return i;
    }
    return (times + 1);
}
```

page frame 배열과 page reference를 저장한 배열을 통해 해당 frame에 있는 page reference가 몇 번 뒤에 등장하는지 계산하는 next_offset() 함수이다.

```c
int page_fault(int frame[], int page_ref) {
    int flag = 1;
    for(int i = 0; i < frame_num; i++) {
        if(frame[i] == page_ref)
            flag = 0;
    }
    return flag;
}
```

flag값을 통해 page fault의 발생여부를 리턴하는 page_fault() 함수이다. for문을 통해 히트가 발생하면 0, 발생하지 않으면 1을 리턴하게 된다.

```c
void opt(int frame[]) {
    int f_index = 0;
    int page_ref_arr[times];
    int offset[frame_num];
    set_page_ref(page_ref_arr);
    for(int i = 0; i < times; i++) {
        printf("%d\t\t", i + 1);

        if(page_fault(frame, page_ref_arr[i])) {
            opt_fault++;
            if(frame[f_index] == -1) {
                frame[f_index] = page_ref_arr[i];
                for(int j = 0; j < frame_num; j++) {
                    if(frame[j] == -1)
                        printf(" \t");
                    else
                        printf("%d\t", frame[j]);
                }
                printf("F\n");
                f_index++;
            }
            else {
                int target = 0;
                for(int j = 0; j < frame_num; j++) {
                    offset[j] = next_offset(frame, i, j, page_ref_arr);
                }
                for(int j = 0; j < frame_num - 1; j++) {
                    if(offset[target] < offset[j + 1])
                        target = j + 1;
                }
                frame[target] = page_ref_arr[i];
                for(int j = 0; j < frame_num; j++) {
                    if(frame[j] == -1)
                        printf(" \t");
                    else
                        printf("%d\t", frame[j]);
                }
                printf("F\n");
            }
        }
        else {
            for(int j = 0; j < frame_num; j++) {
                if(frame[j] == -1)
                    printf(" \t");
                else
                    printf("%d\t", frame[j]);
            }
            printf("\n");
        }
    }
}
```

opt() 함수는 page_fault() 함수를 통해 페이지 폴트 여부를 확인하고, opt_fault를 증가시키고, 원소 -1로 초기화된 frame배열을 통해 frame이 비어있을 땐 해당 페이지 레퍼런스를 저장하고 이를 frame 배열을 출력한다. 만약 frame이 가득 찼다면 frame 배열에 있

는 페이지 레퍼런스들의 다음 등장 순번을 저장하는 offset 배열에 next_offset() 함수를 통해 값을 저장하고 for문을 통해 이 값이 가장 큰 레퍼런스 원소의 인덱스를 계산하여 replacement를 수행하고 frame 배열을 출력한다. 만약 히트가 발생하면 해당 frame를 출력한다.

- fifo.c

```
int page_fault(int frame[], int page_ref) {
    int flag = 1;
    for(int i = 0; i < frame_num; i++) {
        if(frame[i] == page_ref)
            flag = 0;
    }
    return flag;
}
```

Optimal 방식과 동일한 page_fault() 함수를 사용한다.

```
59 void fifo(int frame[]) {
60     int f_index = 0;
61     char* page_ref;
62     for(int i = 0; i < times; i++) {
63         if(i == 0)
64             page_ref = strtok(buf, " ");
65         else
66             page_ref = strtok(NULL, " ");
67         printf("%d\t\t", i + 1);
68 
69         if(page_fault(frame, atoi(page_ref))) {
70             fifo_fault++;
71             frame[f_index] = atoi(page_ref);
72             for(int j = 0; j < frame_num; j++) {
73                 if(frame[j] == -1)
74                     printf(" \t");
75                 else
76                     printf("%d\t", frame[j]);
77             }
78             printf("F\n");
79             f_index = (f_index + 1) % frame_num;
80         }
81         else {
82             for(int j = 0; j < frame_num; j++) {
83                 if(frame[j] == -1)
84                     printf(" \t");
85                 else
86                     printf("%d\t", frame[j]);
87             }
88             printf("\n");
89         }
90     }
91 }
```

fifo() 함수는 strtok() 함수를 통해 buf 배열에서 페이지 레퍼런스를 읽고 이를 통해 page_fault() 함수가 참일 때 fifo_fault를 증가시키고 frame 배열 해당 인덱스에 페이지 레퍼런스를 저장한다. 어떤 조건도 없이 FIFO 방식으로만 돌아가면 되기 때문에 다음 인덱스 값을 (f_index + 1) % frame_num으로 해주면 정상적으로 FIFO 방식으로 페이지 레퍼런스가 저장된다. 이후 frame 배열을 출력한다. 만약 히트가 일어났다면 frame 배열을 출력한다.

- lru.c

```
typedef struct NODE* node_ptr;
typedef struct NODE {
    node_ptr uplink;
    node_ptr downlink;
    int data;
} NODE;

node_ptr top = NULL;
node_ptr bottom = NULL;
```

이중 연결리스트를 위한 자료구조들의 선언이다. NODE 구조체는 위 아래 원소들에 대한 포인터와 페이지 레퍼런스를 저장할 data 변수를 멤버로 갖는다.

```
145 int page_fault(int page_ref) {
146     int flag = 1;
147     node_ptr tmp = (node_ptr)malloc(sizeof(NODE));
148     node_ptr pos = (node_ptr)malloc(sizeof(NODE));
149 
150     tmp = top;
151     pos = top;
152 
153     while(pos != NULL) {
154         if(pos->data == page_ref) {
155             tmp = pos;
156             flag = 0;
157             if((tmp->uplink) && (tmp->downlink)) {
158                 tmp->uplink->downlink = tmp->downlink;
159                 tmp->downlink->uplink = tmp->uplink;
160                 tmp->downlink = top;
161                 tmp->uplink = top->uplink;
162                 top->uplink = tmp;
163                 top = tmp;
164             }
165             else if(!(tmp->downlink) && (tmp->uplink)) {
166                 tmp->uplink->downlink = tmp->downlink;
167                 bottom = tmp->uplink;
168                 tmp->uplink = top->uplink;
169                 top->uplink = tmp;
170                 tmp->downlink = top;
171                 top = tmp;
172             }
173         }
174         pos = pos->downlink;
175     }
176     return flag;
177 }
```

두개의 node_ptr 변수 tmp, pos를 통해 데이터가 일치한다면(히트) 노드가 중간에 있을 경우와 bottom에 있을 경우를 구분하여 해당 노드가 top에 위치하도록 연결리스트를 조정하고 다음 노드를 탐색해 나가면서 일치하는 페이지 레퍼런스가 없다면 1을 리턴하고, 있다면 0을 리턴하게된다.

```c
void lru(int frame[]) {
    int dlist_num = 0;
    int f_index = 0;
    char* page_ref;
    for(int i = 0; i < times; i++) {
        if(i == 0)
            page_ref =  strtok(buf, " ");
        else
            page_ref = strtok(NULL, " ");
        printf("%d\t\t", i + 1);

        if(page_fault(atoi(page_ref))) {
            lru_fault++;
            if(frame_num == 1) {
                node_ptr tmp = (node_ptr)malloc(sizeof(NODE));
                tmp->downlink = top;
                tmp->uplink = NULL;
                tmp->data = atoi(page_ref);
                top = tmp;
                bottom = tmp;
            }
            else if(dlist_num < frame_num) {
                node_ptr tmp = (node_ptr)malloc(sizeof(NODE));
                dlist_num++;
                frame[f_index] = atoi(page_ref);
                tmp->downlink = top;
                tmp->uplink = NULL;

                if(top)
                    top->uplink = tmp;
                tmp->data = atoi(page_ref);
                top = tmp;

                if(!bottom)
                    bottom = tmp;
            }
            else {
                node_ptr tmp = (node_ptr)malloc(sizeof(NODE));
                tmp = bottom;

                for(int j = 0; j < frame_num; j++) {
                    if(frame[j] == tmp->data) {
                        frame[j] = atoi(page_ref);
                        break;
                    }
                }

                tmp->uplink->downlink = tmp->downlink;
                bottom = tmp->uplink;
                tmp->data = atoi(page_ref);
                tmp->downlink = top;
                tmp->uplink = top -> uplink;
                top->uplink = tmp;
                top = tmp;
            }

            f_index++;
            for(int j = 0; j < frame_num; j++) {
                if(frame[j] == -1)
                    printf(" \t");
                else
                    printf("%d\t", frame[j]);
            }
            printf("F\n");
        }
        else {
            for(int j = 0; j < frame_num; j++) {
                if(frame[j] == -1)
```

```
136                     printf(" \t");
137                 else
138                     printf("%d\t", frame[j]);
139             }
140             printf("\n");
141         }
142     }
143 }
```

lru() 함수는 page_fault() 함수가 1일 리턴하면 lru_fault를 증가시키고 frame수가 1일 경우 해당 노드의 데이터 값만을 변경해주고 그 외의 경우 dlist_num 변수와 frame_num 변수의 비교를 통해 연결리스트가 꽉 찼는지 판단 후 그렇지 않다면 추가되는 노드가 탑에 위치하도록 한다. 이때 기존 노드의 존재 여부를 top과 bottom을 지정해야한다. 만약 연결리스트가 빈 공간이 없다면 bottom이 victim이 되고 링크의 조정을 해주는 방식으로 동작한다.

- sc.c

```
typedef struct Node {
    int data;
    int ref_bit;
} node;
```

페이지 프레임 배열의 원소 자료구조로 reference bit를 추가적으로 멤버로 갖는다.

```
int page_fault(node q[], int page_ref) {
    int flag = 1;
    for(int i = 0; i < frame_num; i++) {
        if(q[i].data == page_ref)
            flag = 0;
    }
    return flag;
}
```

페이지 폴트 판단 함수도 역시 동일하다.

```
60 void second_chance() {
61     node queue[frame_num];
62     int f_index = 0;
63     char* page_ref;
64     for(int i = 0; i < frame_num; i++) {
65         queue[i].data = -1;
66         queue[i].ref_bit = -1;
67     }
68     for(int i = 0; i < times; i++) {
69         if(i == 0)
70             page_ref = strtok(buf, " ");
71         else
72             page_ref = strtok(NULL, " ");
73         printf("%d\t\t", i + 1);
74 
75         if(page_fault(queue, atoi(page_ref))) {
76             sc_fault++;
77             // 큐가 비어있을 때
78             if(queue[f_index].data == -1) {
79                 queue[f_index].data = atoi(page_ref);
80                 queue[f_index].ref_bit = 0;
81                 for(int j = 0; j < frame_num; j++) {
82                     if(queue[j].data == -1)
83                         printf(" \t");
84                     else
85                         printf("%d\t", queue[j].data);
86                 }
87                 printf("F\n");
88                 f_index = (f_index + 1) % frame_num;
89             }
90             // 큐가 찼을 때
91              else {
92                 while(1) {
93                     if(queue[f_index].ref_bit == 1) {
94                         queue[f_index].ref_bit = 0;
95                         f_index = (f_index + 1) % frame_num;
96                     }
97                     else {
98                         queue[f_index].data = atoi(page_ref);
99                         f_index = (f_index + 1) % frame_num;
100                        break;
101                    }
102                }
103                for(int j = 0; j < frame_num; j++) {
104                    if(queue[j].data == -1)
105                        printf(" \t");
106                    else
107                        printf("%d\t", queue[j].data);
108                }
109                printf("F\n");
110            }
111        }
112        else {
113            for(int j = 0; j < frame_num; j++) {
114                if(queue[j].data == atoi(page_ref)) {
115                    queue[j].ref_bit = 1;
116                    break;
117                }
118            }
119            for(int j = 0; j < frame_num; j++) {
120                if(queue[j].data == -1)
121                    printf(" \t");
122                else
123                    printf("%d\t", queue[j].data);
int page_fault(node q[], int page_ref) {
    int flag = 1;
    for(int i = 0; i < frame_num; i++) {
        if(q[i].data == page_ref)
            flag = 0;
    }
    return flag;
}
```

second_chance() 함수는 기본적으로 FIFO 방식과 동일하다 다만 차이가 있다면 페이지

프레임 배열이 꽉 찼을 때 해당 인덱스의 원소의 ref_bit 값이 1인지를 판별하고 1이라면 이 값을 0으로 바꿔주고 다음 인덱스로 넘어간다. 또한 히트가 났을 경우 해당 원소의 ref_bit을 1로 지정하고 인덱스는 이동하지 않도록 한다.

1-4. 실행결과

- opt.c

```
shin@shin-virtual-machine:~/shin/os/hw5$ gcc opt.c
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input
Used method : OPT
page reference string : 2 3 2 1 5 2 4 5 3 2 5 2

        frame   1       2       3       page fault
time
1               2                       F
2               2       3               F
3               2       3
4               2       3       1       F
5               2       3       5       F
6               2       3       5
7               4       3       5       F
8               4       3       5
9               4       3       5
10              2       3       5       F
11              2       3       5
12              2       3       5
Number of page faults : 6 times
```

과제 명세에 있는 예시와 같은 page frame와 page reference string을 담은 파일로 실험한 결과이다. 같은 결과를 출력했음을 알 수 있다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : OPT
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       0       3       F
7               2       0       3
8               2       4       3       F
9               2       4       3
10              2       4       3
11              2       0       3       F
12              2       0       3
13              2       0       3
14              2       0       1       F
15              2       0       1
16              2       0       1
17              2       0       1
18              7       0       1       F
19              7       0       1
20              7       0       1
Number of page faults : 9 times
```

강의자료에 있는 예시로 실행한 결과이다. Optimal Algorithm의 동작방식대로 출력되었음을 확인했다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : OPT
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       4       page fault
time
1               7                               F
2               7       0                       F
3               7       0       1               F
4               7       0       1       2       F
5               7       0       1       2
6               3       0       1       2       F
7               3       0       1       2
8               3       0       4       2       F
9               3       0       4       2
10              3       0       4       2
11              3       0       4       2
12              3       0       4       2
13              3       0       4       2
14              1       0       4       2       F
15              1       0       4       2
16              1       0       4       2
17              1       0       4       2
18              1       0       7       2       F
19              1       0       7       2
20              1       0       7       2
Number of page faults : 8 times
```

앞과 같은 page reference string에 page frame을 4로 변경하여 실행하였다. 역시 정상적인 결과를 출력했다.

- fifo.c

```
shin@shin-virtual-machine:~/shin/os/hw5$ gcc fifo.c
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input
Used method : FIFO
page reference string : 2 3 2 1 5 2 4 5 3 2 5 2

        frame   1       2       3       page fault
time
1               2                       F
2               2       3               F
3               2       3
4               2       3       1       F
5               5       3       1       F
6               5       2       1       F
7               5       2       4       F
8               5       2       4
9               3       2       4       F
10              3       2       4
11              3       5       4       F
12              3       5       2       F
Number of page faults : 9 times
```

과제 명세에 있는 예시와 같은 page frame와 page reference string을 담은 파일로 실험한 결과이다. 같은 결과를 출력했음을 알 수 있다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ vi input.txt
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : FIFO
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       3       1       F
7               2       3       0       F
8               4       3       0       F
9               4       2       0       F
10              4       2       3       F
11              0       2       3       F
12              0       2       3
13              0       2       3
14              0       1       3       F
15              0       1       2       F
16              0       1       2
17              0       1       2
18              7       1       2       F
19              7       0       2       F
20              7       0       1       F
Number of page faults : 15 times
```

강의자료에 있는 예시로 실행한 결과이다. FIFO Algorithm의 동작방식대로 출력되었음을 확인했다. page reference string이 길어지면서 Optimal 방식과의 성능차이가 극도로 심해짐을 확인할 수 있었다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : FIFO
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       4       page fault
time
1               7                               F
2               7       0                       F
3               7       0       1               F
4               7       0       1       2       F
5               7       0       1       2
6               3       0       1       2       F
7               3       0       1       2
8               3       4       1       2       F
9               3       4       1       2
10              3       4       1       2
11              3       4       0       2       F
12              3       4       0       2
13              3       4       0       2
14              3       4       0       1       F
15              2       4       0       1       F
16              2       4       0       1
17              2       4       0       1
18              2       7       0       1       F
19              2       7       0       1
20              2       7       0       1
Number of page faults : 10 times
```

앞과 같은 page reference string에 page frame을 4로 변경하여 실행한 결과이다. 페이지 frame 수가 늘어나니 나름대로 나름대로 성능이 다시 괜찮아졌다. FIFO 방식은 구현이 쉬우나 자체적으로 page fault를 해결할 수 있는 능력은 상당히 부족하다는 것을 느꼈다.

- lru.c

```
shin@shin-virtual-machine:~/shin/os/hw5$ gcc lru.c
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input
Used method : LRU
page reference string : 2 3 2 1 5 2 4 5 3 2 5 2

        frame   1       2       3       page fault
time
1               2                       F
2               2       3               F
3               2       3
4               2       3       1       F
5               2       5       1       F
6               2       5       1
7               2       5       4       F
8               2       5       4
9               3       5       4       F
10              3       5       2       F
11              3       5       2
12              3       5       2
Number of page faults : 7 times
```

과제 명세에 있는 예시와 같은 page frame와 page reference string을 담은 파일로 실험한 결과이다. 같은 결과를 출력했음을 알 수 있다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : LRU
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       0       3       F
7               2       0       3
8               4       0       3       F
9               4       0       2       F
10              4       3       2       F
11              0       3       2       F
12              0       3       2
13              0       3       2
14              1       3       2       F
15              1       3       2
16              1       0       2       F
17              1       0       2
18              1       0       7       F
19              1       0       7
20              1       0       7
Number of page faults : 12 times
```

강의자료에 있는 예시로 실행한 결과이다. LRU Algorithm의 동작방식대로 출력되었음을 확인했다. 확실히 이론상 최적인 Optimal 방식에 비하면 3번의 page fault가 더 발생했지만 FIFO 방식보다는 개선된 결과를 확인했다.

```
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input.txt
Used method : LRU
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       4       page fault
time
1               7                               F
2               7       0                       F
3               7       0       1               F
4               7       0       1       2       F
5               7       0       1       2
6               3       0       1       2       F
7               3       0       1       2
8               3       0       4       2       F
9               3       0       4       2
10              3       0       4       2
11              3       0       4       2
12              3       0       4       2
13              3       0       4       2
14              3       0       1       2       F
15              3       0       1       2
16              3       0       1       2
17              3       0       1       2
18              7       0       1       2       F
19              7       0       1       2
20              7       0       1       2
Number of page faults : 8 times
```

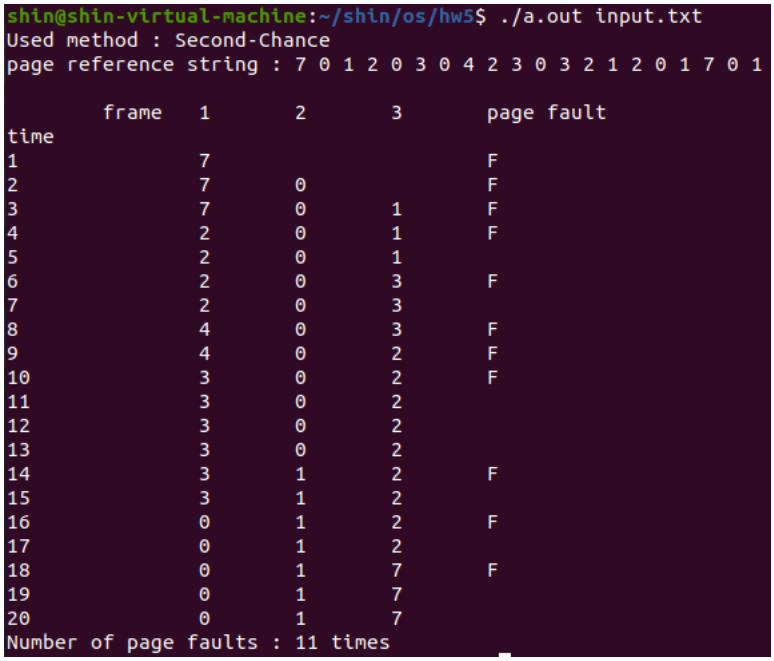
앞과 같은 page reference string에 page frame을 4로 변경하여 실행한 결과이다. 페이지 frame 수가 늘어나니 Optimal 방식과 같은 횟수의 page_fault가 발생했다. LRU 방식은 구현이 어렵고 오버헤드가 크지만 현실적으로 page fault를 해결할 수 있는 방식임을 느꼈다.

- sc.c

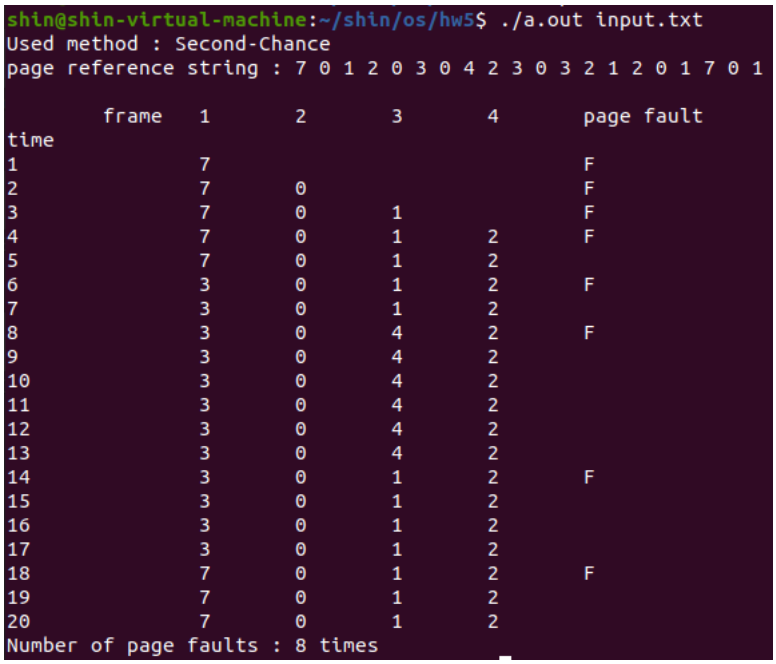
```
shin@shin-virtual-machine:~/shin/os/hw5$ gcc sc.c
shin@shin-virtual-machine:~/shin/os/hw5$ ./a.out input
Used method : Second-Chance
page reference string : 20 3 20 1 5 20 4 5 3 20 5 20

        frame   1       2       3       page fault
time
1               20                      F
2               20      3               F
3               20      3
4               20      3       1       F
5               20      5       1       F
6               20      5       1
7               20      5       4       F
8               20      5       4
9               20      5       3       F
10              20      5       3
11              20      5       3
12              20      5       3
Number of page faults : 6 times
```

과제 명세에 있는 예시와 같은 page frame와 page reference string을 담은 파일로 실험한 결과이다. 같은 결과를 출력했음을 알 수 있다.



강의자료에 있는 예시로 실행한 결과이다. Second-Chance Algorithm의 동작방식대로 출력되었음을 확인했다. 이론상 최적인 Optimal 방식에 비하면 2번의 page fault가 더 발생했지만 FIFO 방식보다는 개선된 결과를 확인했다. LRU 방식보다는 1번의 page fault가 더 발생했다. Second-Chance 방식은 구현이 간단하기 때문에 나름 사용할 만한 방식임을 느꼈다.



앞과 같은 page reference string에 page frame을 4로 변경하여 실행한 결과이다. 페이지 개수를 늘린 것 만으로 모든 방식의 page fault 횟수가 비슷해짐을 확인할 수 있었다. 역

시 가장 안 좋은 성능을 보인 방식은 FIFO 방식이었다.