보고서

학과 : 컴퓨터공학과
학번: 21720903
이름 : 조태식

목차

개요

운영체제설계란 수업을 들으며 cpu가 프로세스들을 처리하는 다양한 방법을 배우며, 어떠한 프로세스가 순서와 실행시간 우선순위를 가지고 복잡하게 들어오더라도 그때에 맞게 상황에 맞게 가장 효과적으로 처리하는 것을 보고 나도 한번 만들어봐야겠다. 매커니즘을 이해해보자, 그리고 나만의 cpu 스케쥴링 기법을 만들어 보자 이러한 생각에 프로젝트를 진행하게 되었습니다.

알고리즘 및 구현

대표적으로 FCFS기법은, 들어온 프로세스 순서대로 정보를 처리하는 방법이다. 코드를 보면서 부가 설명을 하겠다. 그전에 전역변수로 여러 가지 기법을 코딩하는데 있어 필요한 것들을 선언하였다.

```
int process_number0[50] = { 0, }; //프로세스 개수
int burst_time0[50] = { 0, };       //버스트타임
int waiting_time0[50] = { 0, };     //대기시간그리고 실행시작시간을 계산하는데 사용
int arrive_time0[50] = { 0, };      //도착시간
int prioirty1[50] = { 0, }; //우선순위
int order[50] = { 0, };     //순서함수 하지만 프1이 무조건 먼저되게
int priority1[50] = { 0, };
int priority0[50] = { 0, }; //우선순위 정렬
int who[50] = { 0, };       //우선순위 순서 정렬
int multi[50] = { 0, };     //우선순위*버스트시간
int multi1[50] = { 0, };    //multi 순서정렬
```

이렇게 선언을 하였다. 그다음은 FCFS.SJF,PS,RR,태식쓰 이렇게 5가지 기법이있다.
choice란 변수를 선언하고 입력받아,
printf("스케쥴링 기법을 고르세요:\n1.fcfs기법\n2.sjf기법\n3.ps기법\n4.RR기법\n5.태식쓰 기법\n");
기법을 고르는 식으로 코딩을 하였다.

여기서 1번을 고르게 되면 FCFS로 가게된다.
printf("fcfs기법선택함. 몇개의 프로세스를 입력하실건가?"); 처음에 몇 개를 입력받을지 숫자를 받고,

```
for (int i = 0; i < number; i++) {
        int burst_time = 0;
        int process_number = 0;
        scanf("%d", &process_number);
        process_number0[i] = process_number;
        scanf("%d", &burst_time);
        burst_time0[i] = burst_time;
}
```

FOR문을 사용하여서 프로세스 숫자와 버스트 타임을 배열에 넣는다.
그다음 int full_time = 0; 선언을 하여서 for (int i = 0; i < number; i++) {
full_time = full_time + burst_time0[i];
} 이렇게 총 전체실행시간을 구한다.
int awaiting_time = 0;를 선언한다. 이 변수는 훗날 평균 프로세스 실행시간을 구할 때 사용이 된다. 그다음 각 프로세스별 실행시작시간을 구한다. 만약 P1 24 P2 3 P3 3 이런식으로 입력을 받는다면, 이론상 P1 0~24 P2 24~27 P3 27~30이렇게 된다.

```
for (int i = 0; i < number; i++) {
        printf("프로세스번호%d  ", process_number0[i]);
        int waiting_time = 0;
        for (int j = 0; j < i; j++) {
                waiting_time = waiting_time + burst_time0[j];
        }
        waiting_time0[i] = waiting_time;
        awaiting_time = awaiting_time + waiting_time;
        printf("%d\n", waiting_time);

}
```

이렇게 FOR문을 통하여 각각의 실행시작시간을 waiting_time0[i]에 저장을 한다. 그리고 waiting_time를 awaiting_time에 더해줌으로써 시간을 축적해간다.
그 다음 printf("평균 프로세스 실행시작시간:%d", awaiting_time / number);로 평균 프로세스 실행시작시간을 구한다.
칸트차르를 그리기 위해서 for (int i = 0; i < full_time; i++) {

```
        printf("%-2d ",i);
}
```

선언을 해준다.

```
for (int i = 0; i < number; i++) {
        for (int j = 0; j < burst_time0[i]; j++) {
                printf("%-2d ", process_number0[i]);
        }
}
```

이렇게 선언을 해줌으로써 각각 프로세스의 버스트타임시간만큼 프로세스 넘버가 나타나진다.

다음은 choice==2번을 선택하게 되면 SJF기법으로 가게된다. SJF은 처음 프로세스를 받고 실

행시킨다음 훗날 들어오는 프로세스들을 버스트 시간이 짧은 순으로 실행시키는 것을 의미한다. 즉 P1 0 7 P2 2 4 P3 4 1 P4 5 4이런식으로 입력을 받으면 순서는 P1->P3->P2->P4가 된다. FCFS기법과 마찬가지로 프로세스이름 도착시간 실행시간을 입력을 받고,
process_number0[i] = process_number;
order[i] = process_number;
arrive_time0[i] = arrive_time;
burst_time0[i] = burst_time; 에 저장을 해둔다. 여기서 order[i]는 훗날 우선순위에 따른 프로세스 실행순서를 표현한 배열이다. FCFS기법처럼 전체실행시간을 표현하고,
order[0] = process_number0[0]; //당연히 처음 실행되는 프로세스는 처음 입력받은 프로세스0번이 될 것이다.

```
int temp = 0;
for (int i = 1; i < number; i++) {        //0번은 이미 확정이 되엇으므로
    for (int j = 1; j < i; j++) {
        if (burst_time0[j] > burst_time0[j + 1]) {
            temp = order[burst_time0[j]];
            order[burst_time0[j]] = order[burst_time0[j + 1]];
            order[burst_time0[j + 1]] = temp;
        }
    }
}
```

for문을 통하여 버스트 타임에 맞게 0번을 제외하고 순서를 정리해준다. order[i]는 1234

```
temp = 0;
for (int i = 1; i < number; i++) {
    for (int j = 1; j < i; j++) {
        if (burst_time0[j] > burst_time0[i]) {
            temp = order[i];
            order[i] = order[j];
            order[j] = temp;
        }
    }
} 다음은 순서정리된 order[i]를 다시 버스트 시간에 따른 오름차순으로정렬을 한다 여기서 order[i] 1324가 된다.

int r[100] = { 0, };
int time = 0;
int q = 0;
```

이 r[100]이란 배열은 훗날 시작시간과 기다리는 시간을 표현해준다.

```
for (int i = 0; i < number; i++) {
    if (i == 0) {       //처음 시작하는 프로세스는 당연히 0이니깐
        printf("%d의 시작시간은 0입니다.", order[i]);
```

```
                        r[i] = 0;
                }
                else {
                        time = 0;
                        for (int j = 0; j < i; j++) {
                                time = time + burst_time0[order[j] - 1];
                        }
                        r[i] = time;
                        printf("\n%d의 시작시간은 %d입니다.", order[i], time);
                }
        }
```

이렇게 r[100]를 정의를 내려준다.

int z[100] = { 0, }; 이것은 기다리는 시간을 정의해준다. 기다리는 시간이라고 하면 그 프로세스가 실행되는시간에서 도착시간을 뺀 값이다.

```
for (int i = 0; i < number; i++) {
                if (i == 0) {       //당연히 첫 프로세스는 기다리는 것이 없다.
                        printf("%d의 기다리는 시간은 0입니다\n", order[i]);
                        z[i] = 0;
                }
                else {
                        printf("%d의 기다리는 시간은 %d입니다.\n", order[i], r[i] -
arrive_time0[order[i] - 1]);
                        z[i] = r[i] - arrive_time0[order[i] - 1];
                }
        }
```

평균적으로 프로세스가 기다리는 시간을 계산하기 위하여 double av = 0;

```
        for (int i = 0; i < number; i++) {
                av = av + z[i];
        }
```

선언해주고 av를 number로 나누어 줌으로써 평균기다리는 시간을 구한다.

그리고 위의 FCFS기법처럼 숫자로 칸트차트를 포현한다.

다음은 CHOICE==3이 되었을 때 우선순위 스케쥴링 기법이다

우선순위 스케쥴링 기법이란, 우선순위 순서대로 프로세스를 처리한다. 예를 들면

P1 10 3 P2 1 1 P3 2 4 P4 1 5 P5 5 2이렇게 입력을 받으면 우선순위가 작은대로 P2->P5->P1->P3->P4이렇게 처리가 된다.

위의 기법들과 마찬가지로 프로세스 번호와 실행시간을 받지만 추가로 우선순위도 입력을 받는다. 마찬가지로 전체 실행시간을 구한다.

```
double waiting_time = 0;
        int min = 1; 우선순위에서 젤 작은 값이 1이므로 1로 초기화를 해준다.
        int a = 0;
        for (int j = 0; j < number; j++) {
                for (int i = 0; i < number; i++) {
                        if (min == priority1[i]) { //우선순위가 min과 같다면
```

```
                                        who[a] = i + 1; //우선순위를 넣어준다
                                        break;
                                }
                        }
                        min++; //그리고 1을 증가시킨다.
                        a++;
                }
```

이렇게 되면 who배열은 2 5 1 3 4 가된다. 다음은 프로세스별 실행시간을 구해야한다. 다른 기법처럼 첫 우선순위가 1인 프로세스는 실행시간은 바로 시작되므로 0일 될것이고,

```
                for (int i = 0; i < number; i++) {
                        int time = 0;
                        if (b == 0) {
                                printf("%d의 실행시간은 0이다.\n", who[b]);
                                waiting_time0[b] = 0;
                        }
                        else {
                                for (int i = 0; i < b; i++) {
                                        time = time + burst_time0[who[i] - 1];
                                }
                                printf("%d의 실행시간은 %d이다.\n", who[b], time);
                                waiting_time0[b] = time;
                        }
                        b++;
                }
```

첫 프로세스가 아닌것들은 그 전것들의 버스트타임을 더해줌으로써 실행시작시간을 성립하고 그 성립된 시간을 waiting_time0배열에 저장한다. 그런다음 위에기법들과 같게 평균대기시간을 구하고 칸트차트를 그린다.

그다음은 4번째 기법인 RR기법의 알고리즘이다. RR기법은 프로젝트 순서대로 적당한 퀀텀을 지정해두고 퀀텀씩 돌아가면서 시행하는기법이다. 예를 들면P1 53 P2 17 P3 68 P4 24이렇게 들어온다면 P1 0~20 P2 20~37 P3 37~57 P4 57~77 P1 77~97 P3 97~117 P4 117~121 P1 121~134 P3 134~154 P3 154~162이렇게 된다. 다른기법들과 마찬가지로 입력을 받고 무엇보다 나눌 퀀텀이라는 것을 추가로 받는다. 너무작아도 안되고 너무 커서도 안된다. 그다음 전체실행시간을 구하고, int p = 0;

```
                int q = 0;
                int check = 0;
                int t[100][100] = { 0, };
                int a = 0;
                for (int i = 0; i < number; i++) {
                        t[i][a] = burst_time0[i];
                }
```

변수를 선언해줌과 동시에 t라는 이차원배열을 선언한다. 그리고 이 배열에 각각의 버스트타임을 53 17 68 24를 넣어준다. 이렇게 하는이유는 나중에 이 값에서 퀀텀을 뺀값

을 비교해서 이 프로세스가 끝이 났는지 안끝났는지를 확인해주기 때문이다.

```
int big = 0;
for (int i = 0; i < number; i++) {
        if (big < burst_time0[i]) {
                big = burst_time0[i];
        }
```

}이 big이란 것은 위에서 말햇다시피 이차원배열t를 100 100 으로 해놧기 때문에 너무길어서 버스트타임중 젤 긴값을 저장해두는 용도이다.

```
for (int d = 0; d < number; d++) {
        a++;
        for (int i = 0; i < big / quantum + 1; i++) {
                if (t[i][a - 1] - quantum >= 0) {
                        t[i][a] = t[i][a - 1] - quantum;
                }
                else {
                        t[i][a] = 0;
                }
        }
```

} 그리고 big / quantum + 1를 통해서 가로배열의 개수를 지정해준다음 t이채원배열을 정의한다. 대략적으로

53 33 13 0

17 0

68 48 28 8

24 4 0 이렇게 된다. int z = 1; int x = 0;

```
int y = 0;
int z1 = 1; 변수 선언과 함께,
for (int y = 0; y < big / quantum + 1; y++) {
        for (int x = 0; x < number; x++) {
                if ((t[x][y] - t[x][y + 1]) != 0) {
                        waiting_time0[z1] = waiting_time0[z1 - 1] + t[x][y] - t[x][y + 1];

                        printf("%d 에서 %d까지는 %d가 사용\n", waiting_time0[z1 - 1], waiting_time0[z1], process_number0[x]);
                        z1++;
                }

        }
```

} 이 for문을 작성을 하는데 이 for문은 프로세스가 몇초부터 몇초까지 실행되는지를 퀀텀에 따라 나눈 것이다.

다음은 직접 만든 태식쓰 기법이다. 태식쓰기법은 어떻게 하면 실행시간과 우선순위를 가질수잇을지를 고민을하고 우선순위는 낮을수록 먼저되고 실행시간도 낮을수록 좋기에 두 개의 값을 곱해서

그에 따라 낮은 값부터 실행시켜주는 기법을 구상해보았다. 다른기법들처럼 입력을 받고,

```
for (int i = 0; i < number; i++) {
                        multi[i] =          burst_time0[i] * prioirty1[i];
                }를 통하여 버스트시간*우선순위를 곱한값을 정해준다.

for (int a = 0; a < number; a++) {
                        multi1[a] = multi[a];
                        printf("%d ", multi1[a]);
                } 이렇게 multi1을 multi에 값을 복사한다
for (int i = 0; i < number; i++) {
                        for (int j = 0; j < number-1; j++) {
                                if (multi1[j] > multi1[j + 1]) {
                                        temp = multi1[j];
                                        multi1[j] = multi1[j + 1];
                                        multi1[j + 1] = temp;
                                }
                        }
                } 그다음 multi1을 오름차순 정렬시킨다.
```

이러면 현재 multi1은 3 5 6 10 32가 저장된다. 다른기법들과 마찬가지로 전체실행시간을 출력하고,

```
for (int j = 0; j < number; j++) {
                        for (int i = 0; i < number; i++) {
                                if (multi1[z] == multi[i]) {
                                        order[z] = i + 1;
                                        z++;
                                }
                        }
                }이 for문을 통하여서 프로세스를 어떤게 먼저 실행되어야할지를 구한다.
```

왜냐하면 이 기법은 우선순위*버스트타임이 작은 순으로 실행되기 때문이다 이렇게 되면 P3->P2->P1->P4->P5이런 순이 된다.

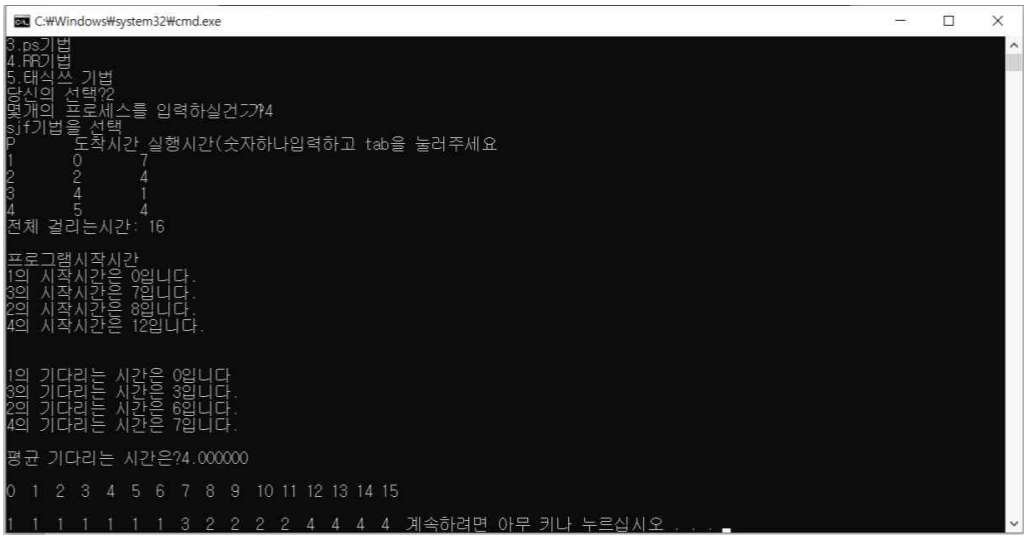
```
for (int i = 0; i < number; i++) {
                        if (i == 0) {
                                printf("\n%d는 시작시간은0입니다.", order[i]);
                                waiting_time0[i] = 0;
                        }
                        else {
                                wait = 0;
                                for (int j = 0; j < i; j++) {
                                        wait = wait + burst_time0[order[j] - 1];
                                }
                                printf("\n%d는 시작시간은%d입니다.", order[i], wait);
                                waiting_time0[i] = wait;
                        }
```

} 이 FOR문을 통해 프로세스별 실행시간을 그린다.

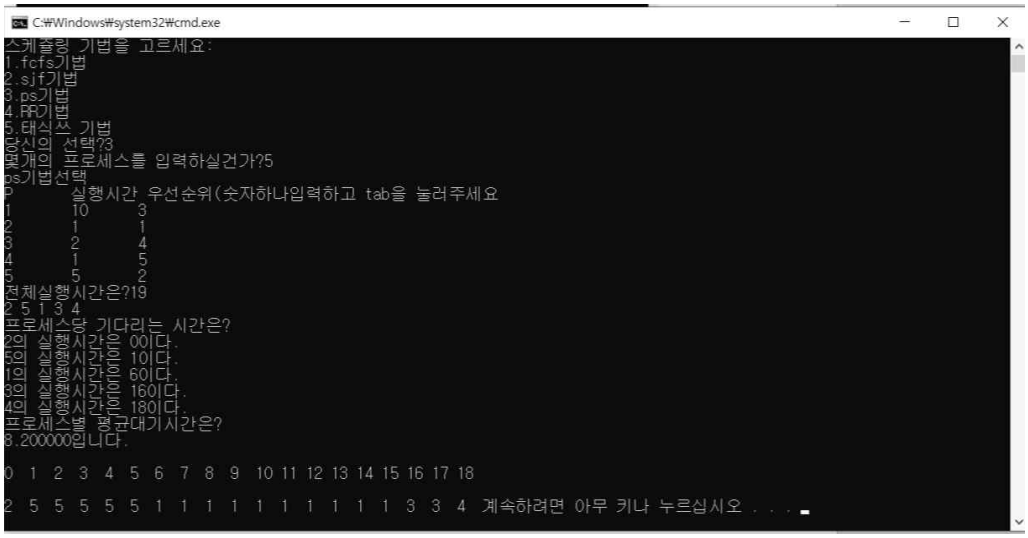
3.실험결과 분석

첫 번째 FCFS 결과이다. 이 알고리즘은 다 잘되지만 칸트차트를 표현함에 있어서 조금 어수룩하였다. 하지만 다양한 예시를 넣었을 때 알고리즘 자체는 잘 돌아갔다.

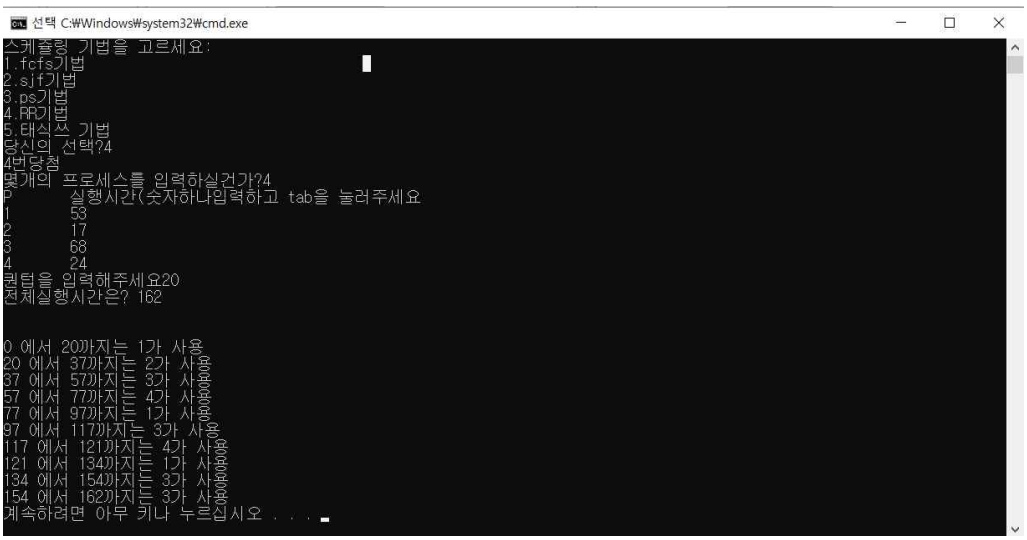
두 번째 SJF기법이다. 이 알고리즘도 첫 번째 기법처럼 칸트차트가 어색하지만 다양한 예시를 넣엇을 때 알고리즘 자체는 역시 잘 돌아갔다.



세 번째 기법인 PS기법이다.
이 알고리즘도 다른 기법들처럼 칸트차트가 어색하지만 다양한 예시를 넣엇을 때 알고리즘 자체는 잘 돌아갔다.



네 번째 기법인 RR기법이다. 역시나 알고리즘 자체는 잘 돌아갔지만 조금더 업그레이드 시킨 칸트차트를 시도해보았다.



마지막 기법인 태식쓰기법이다. 나의 이름을 따왔다. 처음에는 도착시간까지 고려를 하여서 좀더 융통성있는 기법을 만들려고했으나, 여러번 시도 끝에 도착시간까지는 차마 고려를 하지 못하였다.

이번 보고서를 작성하면서 다양한 CPU기법에 돌아가는 방법에 대하 알게 되었고, 같은 CPU 스케쥴링 기법이 있더라도 어떻게 배열을 만들고 동적이든 정적이든지에 따라서 스택을 잡아먹는 양도 달랐다. 처음에는 코드를 만지고 쉰지 1년이 넘어서 까마득해서 무조건 main함수 안에 변수를 다 선언하다가 스택이 커지고 실행이 안되는 경우도 허다하였지만 그런점들을 조금은 개선을 하였다. 공부를 하면서 느낀거지만 malloc를 이용햇으면 어땟을까, c말고도 다른

언어로 햇으면 어땟을까, stack과 que를 이용했으면 어땟을까라는 생각을 하게 되었다. 그래서 코드를 좀더 공부한 후에 혼자서 조금더 나은 cpu 스케쥴링을 코딩해볼까 한다.