OS PROJECT REPORT

제출일		전공	컴퓨터소프트웨어 공학과
과목	운영체제	학번	20194014
담당교수	김 대 영 교수님	이름	신유승

목차

4.실행결과

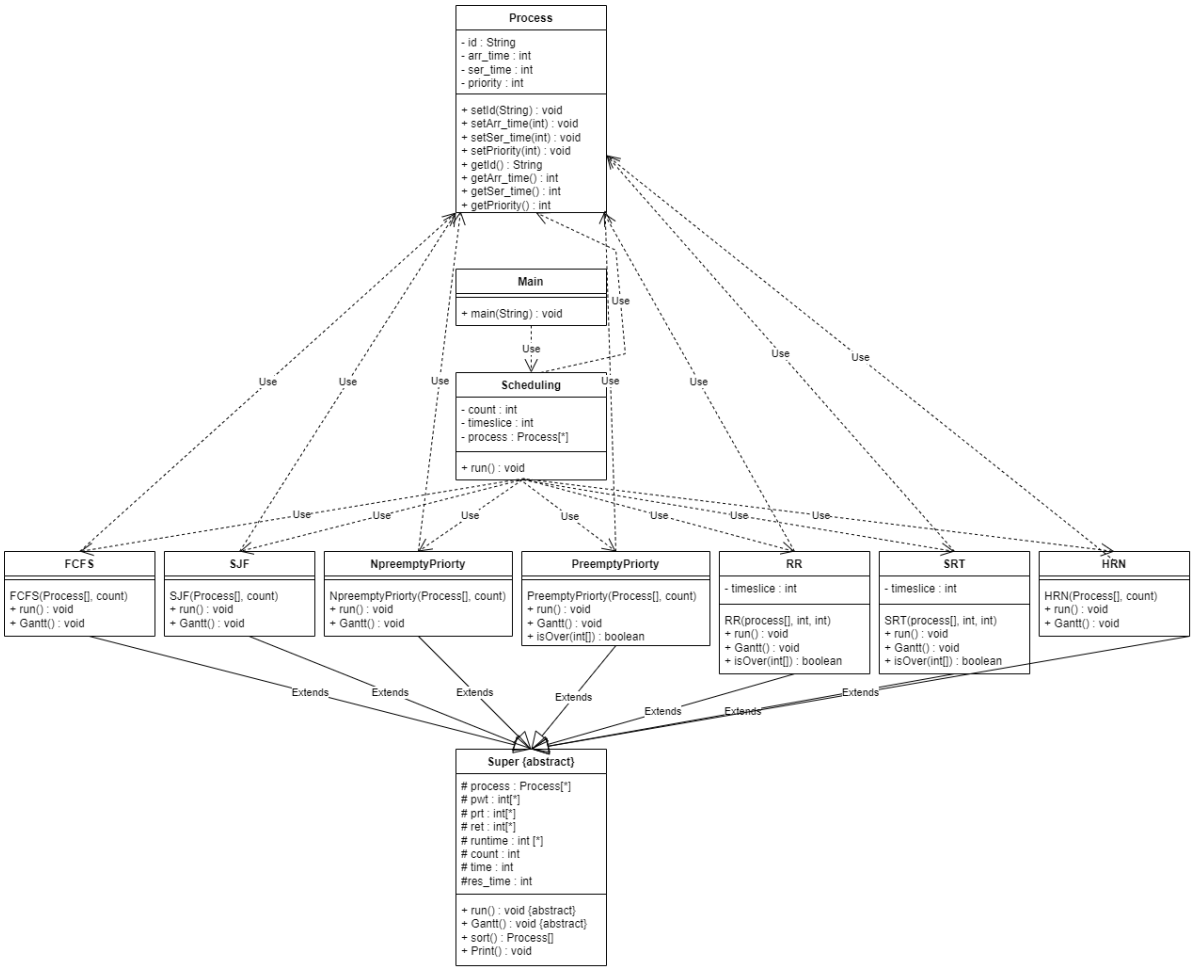
5.느낀점

1. 계획

1-1. 개발 언어

개발 언어는 자바로 정했다. C언어로 할까, 자바로 할까 고민을 했다. C언어로 하게 된다면 구조체로 프로세스를 구현해내야 하고, 자바는 클래스로 프로세스 하나를 객체로 보고 구현해내야 한다. 이번 과제는 객체 개념이 들어가는 것이 편리할 것 같아서 개발 언어는 자바로 구현하기로 했다.

1-2. 클래스 다이어그램



2. 코드

2-1. Main

```
public class Main {
    public static void main(String[] args) {
        Scheduling sch = new Scheduling();
        sch.run();
    }
}
```

2-2. Process

```
public class Process {
    private String id;
    private int arr_time;
    private int ser_time;
    private int priority;
    public void setId(String id){this.id = id;}
    public void setArr_time(int arr_time){this.arr_time = arr_time;}
    public void setSer_time(int ser_time){this.ser_time = ser_time;}
    public void setPriority(int priority){this.priority = priority;}
    public String getId(){return id;}
    public int getArr_time(){return arr_time;}
    public int getSer_time(){return ser_time;}
    public int getPriority(){return priority;}
}
```

2-3. Scheduling

```
import java.io.*;
public class Scheduling {
    int count;
    int timeSlice;
    Process process[];
    public void run(){

        String[]splitedStr=null;
        try {
            File file = new File("src/OS.txt");
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);
            String line = br.readLine();
            count = Integer.parseInt(line);
            process = new Process[count];
            for(int i = 0 ; i < count ; i++){
                process[i]=new Process();
                line = br.readLine();
                splitedStr = null;
                splitedStr = line.split(" ");
```

```
                for(int j = 0 ; j < splitedStr.length ; j++){
                    splitedStr[j] = splitedStr[j].trim();
                }

                process[i].setId(splitedStr[0]);
                process[i].setArr_time(Integer.parseInt(splitedStr[1]));
                process[i].setSer_time(Integer.parseInt(splitedStr[2]));
                process[i].setPriority(Integer.parseInt(splitedStr[3]));
            }
            timeSlice = Integer.parseInt(br.readLine());

            FCFS fcfs = new FCFS(process, count);
            fcfs.run();
            SJF sjf = new SJF(process,count);
            sjf.run();
            NpreemptyPriority npreePriorty = new NpreemptyPriority(process, count);
            npreePriorty.run();
            PreemptyPriority preemptyPriority = new PreemptyPriority(process, count);
            preemptyPriority.run();
            RR rr = new RR(process, count, timeSlice);
            rr.run();
            SRT srt = new SRT(process,count,timeSlice);
            srt.run();
            HRN hrn = new HRN(process, count);
            hrn.run();

            br.close();
            fr.close();
        }catch(FileNotFoundException e){
            System.out.println("파일이 존재하지 않습니다.");
        }catch(IOException e){
            System.out.println("파일 입출력 실패");
        }
    }
}
```

2-4. Super

```
public abstract class Super {
    protected Process process[]; //프로세스 객체 배열
    protected int pwt[]; //각 프로세스 대기시간
    protected int prt[]; //각 프로세스 응답시간
    protected int ret[]; //각 프로세스 반환시간
    protected int runtime[]; //프로세스의 남은 실행시간
    protected int count; //프로세스 총 개수
    protected int time = 0; //시간
    protected int res_time=1; //응답시간
```

```
    public abstract void run(); //
    public abstract void Gantt();
    public Process[] sort(Process process[], int num) {
        if (num == 1) {//1 번은 도착시간 순으로 정렬
            Process tp;
            for (int i = 0; i < process.length; i++) {
                int min = process[i].getArr_time();
                int minindex = i;
                for (int j = i; j < process.length; j++) {
                    if (min > process[j].getArr_time()) {
                        min = process[j].getArr_time();
                        minindex = j;
                    }
                }
                if (i == minindex) continue;
                else {
                    tp = process[i];
                    process[i] = process[minindex];
                    process[minindex] = tp;
                }
            }
        } else if (num == 2) { //2. 우선순위 기준으로 정렬
            Process tp;
            for (int i = 0; i < process.length; i++) {
                int min = process[i].getPriority();
                int minindex = i;
                for (int j = i; j < process.length; j++) {
                    if (min > process[j].getPriority()) {
                        min = process[j].getPriority();
                        minindex = j;
                    }
                }
                if (i == minindex) continue;
                else {
                    tp = process[i];
                    process[i] = process[minindex];
                    process[minindex] = tp;
                }
            }
        }
        else if (num == 3) { //3. 서비스 시간 기준으로 정렬
            Process tp;
            for (int i = 0; i < process.length; i++) {
                int min = process[i].getSer_time();
                int minindex = i;
                for (int j = i; j < process.length; j++) {
                    if (min > process[j].getSer_time()) {
                        min = process[j].getSer_time();
                        minindex = j;
                    }
                }
```

```
                if (i == minindex) continue;
                else {
                    tp = process[i];
                    process[i] = process[minindex];
                    process[minindex] = tp;
                }
            }
        }
        return process;
    }
    public void Print(){
        double awt = 0;
        for(int i = 0 ; i < count ; i++) {
            System.out.println(process[i].getId() + "의 대기시간 = " + pwt[i]);
            awt+=pwt[i];
        }
        awt/=count;
        System.out.println("평균 대기시간(AWT) = " + awt);

        double art = 0;
        for(int i = 0 ; i < count ; i++) {
            System.out.println(process[i].getId() + "의 응답시간 = " + prt[i]);
            art+=prt[i];
        }
        art/=count;
        System.out.println("평균 응답시간(ART) = " + art);

        double att = 0;
        for(int i = 0 ; i < count ; i++) {
            System.out.println(process[i].getId() + "의 반환시간 = " + ret[i]);
            att+=ret[i];
        }
        att/=count;
        System.out.println("평균 반환시간(ATT) = " + att);
    }
}
```

2-5. FCFS

```
public class FCFS extends Super{
    FCFS(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
```

```
    }
    public void run(){
        System.out.println("FCFS 알고리즘 실행");
        process = sort(process,1);
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }
    public void Gantt(){
        for(int i = 0 ; i < count ; i++){
            pwt[i] = time - process[i].getArr_time();
            for(int j = 0 ; j < process[i].getSer_time() ; j++) {
                if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
                System.out.print(process[i].getId() + " ");
                time++;
            }
            System.out.print("| ");
            ret[i]=time - process[i].getArr_time();
        }
        System.out.println();
    }
}
```

2-6. SJF

```
public class SJF extends Super{
    SJF(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
    }

    public void run(){
        System.out.println("SJF 알고리즘 실행");
        process = sort(process,3);
        int min = process[0].getArr_time();
        int minindex = 0;
        for(int i = 0 ; i < count ; i++){
            if(min>process[i].getArr_time()){
                min = process[i].getArr_time();
                minindex = i;
            }
        }
        Process tp = process[minindex];
        for(int i = minindex ; i > 0 ; i --){
            process[i] = process[i-1];
```

```
        }
        process[0]=tp;
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }
    public void Gantt(){
        for(int i = 0 ; i < count ; i++){
            if(process[i].getArr_time()>time){
                int j;
                for(j = i ; j < count;j++){
                    if(process[j].getArr_time()<time)
                        break;
                }
                if(j == count && process[j-1].getArr_time()>time){
                    for(;time<process[i].getArr_time();time++)
                        System.out.print("0 ");
                }
                else {
                    Process tp = process[j];
                    for (int x = j; x > i; x--) {
                        process[x] = process[x - 1];
                    }
                    process[i] = tp;
                }
            }
            pwt[i] = time - process[i].getArr_time();
            for(int j = 0 ; j < process[i].getSer_time() ; j++) {
                if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
                System.out.print(process[i].getId() + " ");
                time++;
            }
            System.out.print("| ");
            ret[i]=time - process[i].getArr_time();
        }
        System.out.println();
    }
}
```

2-7. NpreemptyPriority(비선점형 우선순위 알고리즘)

```
public class NpreemptyPriority extends Super{
    NpreemptyPriority(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
```

```
    }

    public void run(){
        System.out.println("비선점 Priority 알고리즘 실행");
        process = sort(process,1);
        process = sort(process,2);
        int min = process[0].getArr_time();
        int minindex = 0;
        for(int i = 0 ; i < count ; i++){
            if(min>process[i].getArr_time()){
                min = process[i].getArr_time();
                minindex = i;
            }
        }
        Process tp = process[minindex];
        for(int i = minindex ; i > 0 ; i --){
            process[i] = process[i-1];
        }
        process[0]=tp;
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }

    public void Gantt(){
        for(int i = 0 ; i < count ; i++){
            if(process[i].getArr_time()>time){
                int j;
                for(j = i ; j < count;j++){
                    if(process[j].getArr_time()<time)
                        break;
                }
                Process tp = process[j];
                for (int x = j; x > i; x--) {
                    process[x] = process[x - 1];
                }
                process[i] = tp;
            }
            pwt[i] = time - process[i].getArr_time();
            for(int j = 0 ; j < process[i].getSer_time() ; j++) {
                if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
                System.out.print(process[i].getId() + " ");
                time++;
            }
            System.out.print("| ");
            ret[i]=time - process[i].getArr_time();
        }
        System.out.println();
    }
}
```

2-8. PreemptyPriority(선점형 우선순위 알고리즘)

```
public class PreemptyPriority extends Super{
    PreemptyPriority(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
    }
    public void run(){
        System.out.println("선점형 Priority 알고리즘 실행");
        process = sort(process, 1);
        process = sort(process,2);
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }
    public void Gantt(){
        int pindex = 0;
        int minarr = process[0].getArr_time();
        for(int i = 0 ; i < count ; i++){
            if(minarr>process[i].getArr_time()) {
                minarr = process[i].getArr_time();
                pindex = i;
            }
        }
        int j = 0;
        while(isOver(runtime)){
            int c=0;
            for(int i = 0 ; i < runtime[pindex];){
                System.out.print(process[pindex].getId() + " ");
                if(prt[pindex]==0)prt[pindex]=time+res_time-process[pindex].getArr_time();
                time++;
                for(int x = 0 ; x < count ; x++){
                    if(x == pindex) continue;
                    else{
                        if(process[x].getArr_time()<time && runtime[x]!=0)
                            pwt[x]++;
                    }
                }
                runtime[pindex]--;
                for(j = 0 ; j < pindex ; j++){
                    if(process[j].getArr_time() == time && runtime[j] != 0){
                        i = runtime[pindex];
                        break;
                    }
```

```
                }
                c++;
            }
            if(c>0)System.out.print("| ");
            if(runtime[pindex] == 0)ret[pindex] = time-process[pindex].getArr_time();
            if(pindex == j) {
                pindex++;
                j++;
            }
            else pindex = j;
        }
        System.out.println();
    }
    boolean isOver(int runtime[]){
        for(int i = 0 ; i < count ; i++){
            if(runtime[i] != 0){
                return true;
            }
        }
        return false;
    }
}
```

2-9. RR(Round Robin 알고리즘)

```
public class RR extends Super{
    private int timeslice;
    RR(Process process[], int count, int timeslice){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
        this.timeslice = timeslice;
    }

    public void run(){
        System.out.println("RR 알고리즘 실행");
        process = sort(process, 1);
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }

    public void Gantt(){
        int queue[] = new int[count];
        int queueoutindex = 0;
        int queueinindex = 0;
```

```
        int ncount=0;
        while(isOver(runtime)){
            int isIn = 0;
            if(ncount == 0){
                queue[queueinindex] = 0;
                queueinindex++;
            }
            for(int i = 0 ; i < timeslice && runtime[queue[queueoutindex]] != 0; i++){
                System.out.print(process[queue[queueoutindex]].getId() + " ");
                if(prt[queue[queueoutindex]] == 0)prt[queue[queueoutindex]] = time +
res_time - process[queue[queueoutindex]].getArr_time();
                time++;
                runtime[queue[queueoutindex]]--;
                for(int j = 0 ; j < count ; j++){
                    if(process[j].getArr_time() == time){
                        queue[queueinindex] = j;
                        queueinindex++;
                        if(queueinindex == count)queueinindex=0;
                    }
                }
                for(int j = 0 ; j < count ; j++){
                    if(j == queue[queueoutindex])continue;
                    else {
                        if (runtime[j] != 0 && process[j].getArr_time() < time) {
                            pwt[j]++;
                        }
                    }
                }
                isIn++;
            }
            if(isIn>0) {
                System.out.print("| ");
                if(runtime[queue[queueoutindex]] == 0)ret[queue[queueoutindex]]=time-
process[queue[queueoutindex]].getArr_time();
            }
            if(runtime[queue[queueoutindex]] != 0){
                queue[queueinindex] = queue[queueoutindex];
                queueinindex++;
            }
            if(queueinindex == count)queueinindex = 0;
            queueoutindex++;
            if(queueoutindex == count)queueoutindex = 0;
            ncount++;
        }
        System.out.println("");
    }
    boolean isOver(int runtime[]){
        for(int i = 0 ; i < count ; i++){
            if(runtime[i] != 0){
                return true;
            }
        }
```

```
        return false;
    }
}
```

2-10. SRT

```
public class SRT extends Super{
    int timeslice;
    SRT(Process process[], int count, int timeslice){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
        this.timeslice = timeslice;
    }

    public void run(){
        System.out.println("SRT 알고리즘 실행");
        process = sort(process, 1);
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }

    public void Gantt(){
        int pindex=0;
        int minarr = process[0].getArr_time();
        for(int i = 0 ; i < count ; i++){
            if(minarr>process[i].getArr_time()) {
                minarr = process[i].getArr_time();
                pindex = i;
            }
        }
        while(isOver(runtime)){
            int isIn=0;
            for(int j = 0; j < timeslice && runtime[pindex]!=0 ; j++){
                if(prt[pindex]==0)prt[pindex]=time+res_time-process[pindex].getArr_time();
                runtime[pindex]--;
                System.out.print(process[pindex].getId() + " ");
                time++;
                for(int x = 0 ; x < count ; x++){
                    if(pindex == x)continue;
                    else{
                        if(process[x].getArr_time()<time && runtime[x]!=0)
                            pwt[x]++;
                    }
                }
```

```
                isIn++;
            }
            if(isIn>0) {
                System.out.print("| ");
                if(runtime[pindex] == 0)ret[pindex]=time-process[pindex].getArr_time();
            }
            int oppindex = pindex;
            if(runtime[pindex] == 0) {
                oppindex = pindex;
                runtime[pindex] = 10000;
            }
            for(int i = 0 ; i < count ; i++){
                if(runtime[i]!= 0 && process[i].getArr_time()<=time &&
runtime[pindex]>runtime[i])
                    pindex = i;
            }
            if(runtime[oppindex] == 10000) runtime[oppindex] = 0;
        }
        System.out.println();
    }
    boolean isOver(int runtime[]){
        for(int i = 0 ; i < count ; i++){
            if(runtime[i] != 0){
                return true;
            }
        }
        return false;
    }
}
```

2-11. HRN

```
public class HRN extends Super{
    HRN(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
    }
    public void run(){
        System.out.println("HRN 알고리즘 실행");
        process = sort(process,1);
        for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
        Gantt();
        Print();
    }
    public void Gantt(){
```

```
        double priority[] = new double[count];
        for(int i = 0 ; i < count ; i++){
            for(int j = 0 ; j < process[i].getSer_time() ; j++) {
                if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
                System.out.print(process[i].getId() + " ");
                time++;
                for(int x = i ; x < count ; x++){
                    if(x == i) continue;
                    else{
                        pwt[x]++;
                    }
                }
            }
            System.out.print("| ");
            ret[i]=time - process[i].getArr_time();
            if(i != count-1) {
                double maxpriority = pwt[i + 1] / process[i + 1].getSer_time();
                int maxpriorityindex = i + 1;
                for (int j = i + 1; j < count; j++) {
                    priority[j] = (double)pwt[j] / (double)process[j].getSer_time();
                    if (maxpriority < priority[j] && process[j].getArr_time() < time) {
                        maxpriority = priority[j];
                        maxpriorityindex = j;
                    }
                }
                Process tp = process[maxpriorityindex];
                for (int j = maxpriorityindex; j > i + 1; j--) {
                    process[j] = process[j - 1];
                }
                process[i + 1] = tp;
            }
        }
        for(int i = 0 ; i < count ; i++){
            pwt[i] -= process[i].getArr_time();
        }
        System.out.println();
    }
}
```

3. 코드 설명

3-1. Main

Main 함수는 그저 Scheduling 객체를 생성하고 이 객체의 함수를 호출하는 역할을 한다. Main함수에서 중요 함수 호출을 할 수 있었지만, static과 관련하여 제약이 생길 수 있어 따로 Scheduling에서 중요 함수 호출을 하였다.

```
public static void main(String[] args) {
    Scheduling sch = new Scheduling();
    sch.run();
}
```

3-2. Process

각 프로세스의 정보를 저장하는 클래스이다. 프로세스에는 프로세스 아이디, 도착시간, 실행시간, 우선순위가 저장된다. 이 정보들은 private로 선언되어 보호받고 함수들로 이 정보들을 저장하고, 전달받을 수 있다.

```
public class Process {
    2개 사용 위치
    private String id;
    2개 사용 위치
    private int arr_time;
    2개 사용 위치
    private int ser_time;
    2개 사용 위치
    private int priority;
    1개 사용 위치
    public void setId(String id){this.id = id;}
    1개 사용 위치
    public void setArr_time(int arr_time){this.arr_time = arr_time;}
    1개 사용 위치
    public void setSer_time(int ser_time){this.ser_time = ser_time;}
    1개 사용 위치
    public void setPriority(int priority){this.priority = priority;}
    10개 사용 위치
    public String getId(){return id;}
    47개 사용 위치
    public int getArr_time(){return arr_time;}
    16개 사용 위치
    public int getSer_time(){return ser_time;}
    3개 사용 위치
    public int getPriority(){return priority;}
}
```

3-3. Scheduling

이 클래스는 파일에서 데이터들을 읽어오고 그 데이터를 저장하는 역할을 한다. 그리고 저장한 데이터를 가지고 알고리즘 클래스와 함수들을 호출하여 모든 알고리즘의 시작점의 역할도 한다. 코드를 보자.

```
int count;
3개 사용 위치
int timeSlice;
13개 사용 위치
Process process[];
```

우선 알고리즘 실행에 필요한 중요한 변수들을 클래스 필드로 선언하였다. count변수는 프로세스의 총 개수를 저장하고 timeSlice 변수는 말 그래도 타임슬라이스, 그리고 process 객체 배열에 프로세스의 정보들을 저장한다. Process 클래스는 후에 나온다.

```
String[]splitedStr=null;
try {
}catch(FileNotFoundException e){
    System.out.println("파일이 존재하지 않습니다.");
}catch(IOException e){
    System.out.println("파일 입출력 실패");
```

파일에는 한 줄에 프로세스 id, 프로세스 도착시간, 프로세스 실행시간, 프로세스 우선순위가 띄어쓰기를 기준으로 나뉘어져 있다. 그러므로 띄어쓰기를 기준으로 split 함수를 사용하여 저장할 수 있는 splitedStr 변수와 file 예외처리를 위한 try, catch문을 만들었다.

```
File file = new File( pathname: "src/OS.txt");
FileReader fr = new FileReader(file);
BufferedReader br = new BufferedReader(fr);
String line = br.readLine();
count = Integer.parseInt(line);
process = new Process[count];
```

파일 변수들을 생성하고, 알고리즘을 실행할 때 필요한 변수들을 선언해준다. 파일 맨 첫 줄에 프로세스의 개수가 저장되어 있으므로 우선 count 변수에 파일 데이터를 읽어 프로세스의 개수를 저장하고, process 객체 배열을 생성해준다.

```
for(int i = 0 ; i < count ; i++){
    process[i]=new Process();
    line = br.readLine();
    splitedStr = null;
    splitedStr = line.split( regex: " ");
    for(int j = 0 ; j < splitedStr.length ; j++){
        splitedStr[j] = splitedStr[j].trim();
    }
    process[i].setId(splitedStr[0]);
    process[i].setArr_time(Integer.parseInt(splitedStr[1]));
    process[i].setSer_time(Integer.parseInt(splitedStr[2]));
    process[i].setPriority(Integer.parseInt(splitedStr[3]));
}
timeSlice = Integer.parseInt(br.readLine());
```

반복문을 통하여 process 객체를 생성해주고, 파일에서 데이터들을 읽어 들여 process 객체 배열에 데이터들을 저장하는 과정이다. 파일 마지막 줄에 시간 할당량 정보가 저장되어 있으므로 마지막엔 timeSlice 변수에 데이터를 저장한다.

```
FCFS fcfs = new FCFS(process, count);
fcfs.run();
SJF sjf = new SJF(process,count);
sjf.run();
NpreemptyPriority npreePriorty = new NpreemptyPriority(process, count);
npreePriorty.run();
PreemptyPriority preemptyPriority = new PreemptyPriority(process, count);
preemptyPriority.run();
RR rr = new RR(process, count, timeSlice);
rr.run();
SRT srt = new SRT(process,count,timeSlice);
srt.run();
HRN hrn = new HRN(process, count);
hrn.run();

br.close();
fr.close();
```

알고리즘 클래스들과 함수들을 생성 및 호출해주고 파일을 닫아준다. 이렇게 Scheduling 클래스는 끝이 난다.

3-4. Super

Super 클래스는 추상클래스로 중복 코드를 줄이기 위해서 태어났다. 알고리즘 클래스들은 필드로 많은 정보를 저장하는데, 대부분 비슷한 정보들을 저장하기에 Super 추상클래스를 상속받아 필드를 사용하고, 함수를 구현하도록 만들었다.

```
protected Process process[]; //프로세스 객체 배열
26개 사용 위치
protected int pwt[]; //각 프로세스 대기시간
30개 사용 위치
protected int prt[]; //각 프로세스 응답시간
23개 사용 위치
protected int ret[]; //각 프로세스 반환시간
34개 사용 위치
protected int runtime[]; //프로세스의 남은 실행시간
49개 사용 위치
protected int count; //프로세스 총 개수
40개 사용 위치
protected int time = 0; //시간
7개 사용 위치
protected int res_time=1; //응답시간
```

프로세스 객체 배열, 각 프로세스 대기시간, 응답시간, 반환시간을 저장하는 배열들, 알고리즘을 진행하면서 프로세스의 남은 실행시간을 저장하는 배열, 프로세스 총 개수를 저장하는 변수, 시간을 뜻하는 time 변수, 응답시간을 저장하는 변수가 있다. 이 배열들과 변수들은 알고리즘 생성자에 의해 매번 비슷하게 초기화 된다.

```
public abstract void run(); /
7개 사용 위치  7개 구현
public abstract void Gantt();
```

알고리즘 클래스들이 Super 클래스를 상속받으면 구현해야하는 추상 함수들이다. run() 함수는 Gantt() 함수를 호출하고, Gantt() 함수에서는 간트차트를 그리면서 실직적으로 알고리즘이 실행되는 함수이다. 알고리즘이 실행되면서 각 프로세스의 대기시간, 응답시간, 반환시간을 배열에 저장하고 끝나면 run()함수로 돌아가 run()함수는 Print()라는 함수를 호출하여 배열에 저장된 정보들을 출력한다.

```java
public void Print(){
    double awt = 0;
    for(int i = 0 ; i < count ; i++) {
        System.out.println(process[i].getId() + "의 대기시간 = " + pwt[i]);
        awt+=pwt[i];
    }
    awt/=count;
    System.out.println("평균 대기시간(AWT) = " + awt);

    double art = 0;
    for(int i = 0 ; i < count ; i++) {
        System.out.println(process[i].getId() + "의 응답시간 = " + prt[i]);
        art+=prt[i];
    }
    art/=count;
    System.out.println("평균 응답시간(ART) = " + art);

    double att = 0;
    for(int i = 0 ; i < count ; i++) {
        System.out.println(process[i].getId() + "의 반환시간 = " + ret[i]);
        att+=ret[i];
    }
    att/=count;
    System.out.println("평균 반환시간(ATT) = " + att);
}
```

Print()함수이다. 각 프로세스의 대기, 응답, 반환시간을 다 출력하고 평균 대기, 응답, 반환시간까지 구하여 출력하는 함수인 것을 알 수 있다.

```java
public Process[] sort(Process process[], int num) {
    if (num == 1) {//1번은 도착시간 순으로 정렬
        Process tp;
        for (int i = 0; i < process.length; i++) {
            int min = process[i].getArr_time();
            int minindex = i;
            for (int j = i; j < process.length; j++) {
                if (min > process[j].getArr_time()) {
                    min = process[j].getArr_time();
                    minindex = j;
                }
            }
            if (i == minindex) continue;
            else {
                tp = process[i];
                process[i] = process[minindex];
                process[minindex] = tp;
            }
        }
```

각 알고리즘들은 서로 다른 방법으로 CPU스케줄링을 진행하는데 이 때 프로세스가 상황에 맞게 정렬이 되어 있으면 좋을 것 같아서 필요한 상황에 맞게 정렬을 해주는 sort() 함수를 작성하게 되었다. 만약 num이 1이라면 프로세스를 도착시간 순으로 정렬하고, 2라면 우선순위 순으로 정렬하고, 3이라면 서비스 시간 순으로 정렬하는 함수이다. 정렬하는 방법은 선택정렬을 사용하였다.

3-5. FCFS

```
public class FCFS extends Super{
    1개 사용 위치
    FCFS(Process process[], int count){
        this.process = process;
        this.count = count;
        pwt = new int[count];
        ret = new int[count];
        prt = new int[count];
        runtime = new int[count];
        for(int i = 0 ; i < count ; i++)pwt[i]=0;
        for(int i = 0 ; i < count ; i++)ret[i]=0;
        for(int i = 0 ; i < count ; i++)prt[i]=0;
    }
```

Super 클래스를 상속받고, 생성자에서 알고리즘에 필요한 배열과 변수들을 초기화한다. 모든 알고리즘 클래스는 똑같이 Super 클래스를 상속받고, 생성자에서 필요한 작업을 처리한다. 그러므로 앞으로의 알고리즘 코드 설명에서 이 과정은 특별한 부분이 없다면 생략한다.

```
public void run(){
    System.out.println("FCFS 알고리즘 실행");
    process = sort(process, num: 1);
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

run()함수이다. FCFS는 먼저 도착한 프로세스가 먼저, 끝까지 실행되는 알고리즘이므로 1번(도착시간 순)으로 정렬하였다. 정렬된 상태에서 runtime 배열에 프로세스들의 서비스 시간을 저장한 후에 Gantt()함수를 호출한다.

```
public void Gantt(){
    for(int i = 0 ; i < count ; i++){
        pwt[i] = time - process[i].getArr_time();
        for(int j = 0 ; j < process[i].getSer_time() ; j++) {
            if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
            System.out.print(process[i].getId() + " ");
            time++;
        }
        System.out.print("| ");
        ret[i]=time - process[i].getArr_time();
    }
    System.out.println();
}
```

정렬된 상태에서 반복문을 통해 각 프로세스의 진행 시간까지 진행해준다. 이 과정에서 각 프로세스의 대기시간을 배열에 저장했고, 안쪽에 있는 반복문이 현재 진행되고 있는 프로세스라고 생각하고 작성했다. 안쪽 반복문이 실행되면 시간이 지난다는 개념으로 time에 1초씩 추가해줬고, 현재 진행되고 있는 프로세스의 이름을 출력하여 간트차트를 출력했다. 한 프로세스의 작업이 끝나면 다음 프로세스로 넘어가는데 이것을 구분하기 위해 "| "를 출력했고, 반환시간까지 저장했다.

다시 run()함수로 넘어와 Print()함수를 통해 저장된 정보의 내용들을 출력한다.

3-6. SJF

```
public void run(){
    System.out.println("SJF 알고리즘 실행");
    process = sort(process, num: 3);
    int min = process[0].getArr_time();
    int minindex = 0;
    for(int i = 0 ; i < count ; i++){
        if(min>process[i].getArr_time()){
            min = process[i].getArr_time();
            minindex = i;
        }
    }
    Process tp = process[minindex];
    for(int i = minindex ; i > 0 ; i --){
        process[i] = process[i-1];
    }
    process[0]=tp;
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

SJF의 run()함수이다. FCFS와 좀 다른 부분들을 볼 수 있다. SJF는 가장 적은 서비스시간을 가진 프로세스를 먼저 실행하는 알고리즘이다. 그렇기에 서비스 시간 순으로 정렬을 하였다. 하지만 문제가 발생한다. 만약 맨 첫번째에 있는 프로세스, 즉 가장 적은 서비스 시간을 가진 프로세스가 만약 도착을 안했다는 문제이다. 이를 해결하기 위해 우선 서비스 시간 순으로 정렬을 하고 처음에 도착한 프로세스를 객체 배열 맨 앞에 배치했다. 그리고 Gantt()함수를 호출하여 알고리즘을 실행한다.

```
for(int i = 0 ; i < count ; i++){
    if(process[i].getArr_time()>time){
        int j;
        for(j = i ; j < count;j++){
            if(process[j].getArr_time()<time)
                break;
        }
        Process tp = process[j];
        for (int x = j; x > i; x--) {
            process[x] = process[x - 1];
        }
        process[i] = tp;
    }
    pwt[i] = time - process[i].getArr_time();
    for(int j = 0 ; j < process[i].getSer_time() ; j++) {
        if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
        System.out.print(process[i].getId() + " ");
        time++;
    }
    System.out.print("| ");
    ret[i]=time - process[i].getArr_time();
}
System.out.println();
```

배열에는 이미 서비스시간 순으로 정렬되어 있기 때문에 차례대로 진행을 하면 되지만, 도착하지 않은 프로세스를 진행하면 문제가 생긴다. 그러므로 프로세스 도착 시간과 비교하며 적절한 프로세스를 고르도록 만들고, 프로세스가 실행될 때나 반환될 때 필요한 정보들을 배열에 저장한다.

3-7. NpreemptyPriority(비선점형 우선순위 알고리즘)

```
public void run(){
    System.out.println("비선점 Priority 알고리즘 실행");
    process = sort(process, num: 1);
    process = sort(process, num: 2);
    int min = process[0].getArr_time();
    int minindex = 0;
    for(int i = 0 ; i < count ; i++){
        if(min>process[i].getArr_time()){
            min = process[i].getArr_time();
            minindex = i;
        }
    }
    Process tp = process[minindex];
    for(int i = minindex ; i > 0 ; i --){
        process[i] = process[i-1];
    }
    process[0]=tp;
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

비선점형 우선순위 알고리즘은 우선순위에 따라 프로세스가 실행되고, 그 프로세스가 끝날 때까지 프로세스를 실행하는 것이다. 이 클래스의 run()함수는 sort()함수를 두 번 호출하는 것을 볼 수 있다. 그 이유는 같은 우선순위를 가진 프로세스 때문인데, 이 경우에는 먼저 도착한 프로세스가 실행되기에 도착한 시간 순으로 정렬한 후에 우선순위 순으로 정렬을 하였다. 그리고 SJF와 같이 프로세스 실행시간 때문에 비슷한 코드를 작성하여 제일 먼저 도착한 프로세스를 배열 맨 앞에 배치했다. 그리고 Gantt()함수를 호출한다

```
for(int i = 0 ; i < count ; i++){
    if(process[i].getArr_time()>time){
        int j;
        for(j = i ; j < count;j++){
            if(process[j].getArr_time()<time)
                break;
        }
        Process tp = process[j];
        for (int x = j; x > i; x--) {
            process[x] = process[x - 1];
        }
        process[i] = tp;
    }
```

Gantt()함수의 첫 부분이다. process 객체 배열에는 우선순위 순으로 정렬이 되어 있지만, 만약 현재 실행되어야 할 우선순위를 가진 프로세스가 아직 도착하지 않았다면 그 다음 프로세스가 먼저 실행되어야 한다. 그러기 위해서 배열 기준 현재 실행되어야 할 프로세스 뒤에 있는 프로세스 중에 우선순위가 가장 높고 도착 상태인 프로세스를 찾아내고 그 프로세스를 현재 실행해야 할 프로세스로 지정한다. 그 과정을 코딩으로 작성한 것이다.

```
pwt[i] = time - process[i].getArr_time();
for(int j = 0 ; j < process[i].getSer_time() ; j++) {
    if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
    System.out.print(process[i].getId() + " ");
    time++;
}
System.out.print("| ");
ret[i]=time - process[i].getArr_time();
```

그런 다음에 프로세스 대기시간, 프로세스 반환시간, 프로세스 응답시간 등을 배열에 저장하면서 프로세스를 진행시킨다. 그리고 Print()함수를 통하여 배열에 저장된 정보들을 출력해낸다.

3-8. PreemptyPriority(선점형 우선순위 알고리즘)

```
public void run(){
    System.out.println("선점형 Priority 알고리즘 실행");
    process = sort(process, num: 1);
    process = sort(process, num: 2);
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

선점형 우선순위 알고리즘도 비선점형 우선순위 알고리즘과 같이 도착한 시간 순서로 정렬하고 우선순위 순으로 정렬해준 뒤 Gantt()함수를 호출했다.

```
int pindex = 0;
int minarr = process[0].getArr_time();
for(int i = 0 ; i < count ; i++){
    if(minarr>process[i].getArr_time()) {
        minarr = process[i].getArr_time();
        pindex = i;
    }
}
```

여기서는 pindex라는 변수를 사용하여 실행해야 할 프로세스 배열 인덱스를 지정해줬다. 우선 pindex를 제일 먼저 도착한 프로세스를 가리키도록 설정을 해주었다.

```
int j = 0;
```
j란 변수도 선언해주었는데 반복문에 필요한 변수로 생각을 하고 선언했다.

```
while(isOver(runtime)){
```
맨 처음 반복문인데, 조건문 안에 isOver이라는 함수가 들어가 있는 것을 볼 수 있다. 이는 따로 만들어준 함수로 코드를 보자면,

```
boolean isOver(int runtime[]){
    for(int i = 0 ; i < count ; i++){
        if(runtime[i] != 0){
            return true;
        }
    }
    return false;
}
```
매개변수로 받은 배열(남은 실행시간)이 모두 0이라면 false, 단 하나라도 0이 아니라면 true를 반환하는 함수이다. 이를 반복문으로 사용하여 남은 실행시간이 있는지 없는지를 확인할 수 있다. 다시 본론으로 들어가서

```
int isIn=0;
for(int i = 0 ; i < runtime[pindex];){
    System.out.print(process[pindex].getId() + " ");
    if(prt[pindex]==0)prt[pindex]=time+res_time-process[pindex].getArr_time();
    time++;
    for(int x = 0 ; x < count ; x++){
        if(x == pindex) continue;
        else{
            if(process[x].getArr_time()<time && runtime[x]!=0)
                pwt[x]++;
        }
    }
    runtime[pindex]--;
    for(j = 0 ; j < pindex ; j++){
        if(process[j].getArr_time() == time && runtime[j] != 0){
            i = runtime[pindex];
            break;
        }
    }
    isIn++;
}
```
우선 isIn이라는 변수를 설정해줬다. 이 변수는 그 다음 반복문 안에 들어갔는지 들어가지 않았는지 확인시켜주는 변수이다. 그리고 반복문을 시작하여 현재 pindex가 가리키는

프로세스(실행되어야 할 프로세스)를 실행시간만큼 실행해줬다. 그 과정에서 다른 프로세스들의 대기시간, 현재 프로세스의 응답시간까지 저장해줬다. 그 다음에 pindex를 설정해준다. 배열에는 우선순위를 기준으로 정렬되어 있는데, 현재 실행되는 프로세스의 앞에 있고, 그 프로세스가 도착한 상태라면 우선 i를 이용해 현재 진행되는 반복문을 벗어나게 만들었다.

```
if(isIn>0)System.out.print("| ");
if(runtime[pindex] == 0)ret[pindex] = time-process[pindex].getArr_time();
if(pindex == j) {
    pindex++;
    j++;
}
else pindex = j;
```

만약 isIn변수가 0보다 크다면 반복문을 한번이라도 들어갔다 나와서 프로세스가 바뀐 과정이므로 |를 출력하게 만들었다. 그리고 프로세스의 반환시간을 저장해주고 pindex를 설정해줬다. 이렇게 한다면 원하는 결과를 얻을 수 있고 Print()함수를 호출하여 배열에 저장된 정보들을 출력하고 끝낸다.

3-9. RR(Round Robin 알고리즘)

라운드로빈 알고리즘은 시간 할당량(타임 슬라이스) 개념이 들어가므로 timeslice 필드를 선언해주고 생성자에서 이 필드를 초기화하도록 만들었다.

```
private int timeslice;
1개 사용 위치
RR(Process process[], int count, int timeslice){
    this.process = process;
    this.count = count;
    pwt = new int[count];
    ret = new int[count];
    prt = new int[count];
    runtime = new int[count];
    for(int i = 0 ; i < count ; i++)pwt[i]=0;
    for(int i = 0 ; i < count ; i++)ret[i]=0;
    for(int i = 0 ; i < count ; i++)prt[i]=0;
    this.timeslice = timeslice;
}
```

그리고 run함수를 실행시켰고 이 알고리즘에서는 도착시간을 기준으로 정렬했다.

```
System.out.println("RR 알고리즘 실행");
process = sort(process,  num: 1);
for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
Gantt();
Print();
```

```
int queue[] = new int[count];
int queueoutindex = 0;
int queueinindex = 0;
int ncount=0;
```

Gantt()함수의 첫 부분이다. 이 알고리즘에서는 큐를 사용하여 구현했다. 그렇기에 큐 개념을 사용할 queue 배열과 queueoutindex(큐에서 데이터를 꺼낼 때 쓰는 인덱스) 변수와 queueinindex(큐에 데이터를 넣을 때 쓰는 인덱스) 변수를 선언하고 ncount 변수를 선언했다. ncount변수는 반복문을 몇 번 돌렸는지 알 수 있는 변수이다.

```
while(isOver(runtime)){
```

반복문의 첫 시작은 선점형 우선순위 알고리즘과 같이 isOver함수를 사용하여 시작한다.

```
int isIn = 0;
if(ncount == 0){
    queue[queueinindex] = 0;
    queueinindex++;
}
```

맨 처음 시작할 때에는 당연히 큐에 아무것도 들어 있지 않고, ncout가 0일 것이다. 프로세스 객체 배열은 도착시간 기준으로 정렬되어 있고, 맨 처음에 가장 먼저 도착한 프로세스가 저장되어 있으므로 0번째 인덱스부터 시작을 해야한다. 그러므로 queue의 맨 첫 번째에 0을 넣어준다.

```
for(int i = 0 ; i < timeslice && runtime[queue[queueoutindex]] != 0; i++){
```

그 다음 반복문인데, 타임슬라이스만큼 반복하거나 현재 큐에서 실행되어야 할 프로세스가 실행 시간이 0이 될 때까지 실행되도록 만들었다.

```
System.out.print(process[queue[queueoutindex]].getId() + " ");
if(prt[queue[queueoutindex]] == 0)prt[queue[queueoutindex]] = time + res_time - process[queue[queueoutindex]].getArr_time();
time++;
runtime[queue[queueoutindex]]--;
for(int j = 0 ; j < count ; j++){
    if(process[j].getArr_time() == time){
        queue[queueinindex] = j;
        queueinindex++;
        if(queueinindex == count)queueinindex=0;
    }
}
```

이전에도 그래왔지만 time 변수를 1씩 늘려가며 시간이 지나가는 것으로 생각했다. 만약 프로세스 도착시간과 time 변수가 같다면 그 프로세스는 도착했다는 뜻이고 그러므로 그 프로세스를 큐에 넣는 과정이 위 코드이다. 프로세스의 응답시간도 배열에 넣어주었다.

```
for(int j = 0 ; j < count ; j++){
    if(j == queue[queueoutindex])continue;
    else {
        if (runtime[j] != 0 && process[j].getArr_time() < time) {
            pwt[j]++;
        }
    }
}
isIn++;
```

프로세스의 대기시간을 저장하는 코드이다.

```
if(isIn>0) {
    System.out.print("| ");
    if(runtime[queue[queueoutindex]] == 0)ret[queue[queueoutindex]]=time-process[queue[queueoutindex]].getArr_time();
}
if(runtime[queue[queueoutindex]] != 0){
    queue[queueinindex] = queue[queueoutindex];
    queueinindex++;
}
if(queueinindex == count)queueinindex = 0;
queueoutindex++;
if(queueoutindex == count)queueoutindex = 0;
ncount++;
```

프로세스의 반환시간 및 실행했던 프로세스를 다시 큐에 넣는 과정을 구현한 코드이다.

3-10. SRT

```
public void run(){
    System.out.println("SRT 알고리즘 실행");
    process = sort(process,  num: 1);
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

SRT 알고리즘에서도 도착시간 순으로 프로세스를 정렬했다. 그리고 Gantt()함수를 호출하고 Print()함수를 호출했다.

```
int pindex=0;
int minarr = process[0].getArr_time();
for(int i = 0 ; i < count ; i++){
    if(minarr>process[i].getArr_time()) {
        minarr = process[i].getArr_time();
        pindex = i;
    }
}
```

Gantt()함수의 시작 부분이다. 이 알고리즘에서도 pindex로 프로세스를 가리키도록 만들

고 pindex가 제일 먼저 도착한 프로세스를 가리키도록 하였다.

```
while(isOver(runtime)){
    int isIn=0;
    for(int j = 0; j < timeslice && runtime[pindex]!=0 ; j++){
        if(prt[pindex]==0)prt[pindex]=time+res_time-process[pindex].getArr_time();
        runtime[pindex]--;
        System.out.print(process[pindex].getId() + " ");
        time++;
        for(int x = 0 ; x < count ; x++){
            if(pindex == x)continue;
            else{
                if(process[x].getArr_time()<time && runtime[x]!=0)
                    pwt[x]++;
            }
        }
```

우선 반복문을 isOver()함수를 사용해 시작하고, pindex가 가리키는 프로세스를 먼저 실행하도록 만들었다. 이 과정에서 응답시간과 대기시간을 저장하도록 만들었다.

```
if(isIn>0) {
    System.out.print("| ");
    if(runtime[pindex] == 0)ret[pindex]=time-process[pindex].getArr_time();
}
```

만약 위의 반복문을 한 번이라도 들어갔다면 isIn은 0보다 클 것이고 이를 사용해 프로세스의 타임 슬라이스가 끝났거나 프로세스의 실행 시간이 끝났다는 것을 알 수 있다. 그리고 다음 실행해야 할 프로세스를 pindex로 가리켜야 한다.

```
int oppindex = pindex;
if(runtime[pindex] == 0) {
    oppindex = pindex;
    runtime[pindex] = 10000;
}
for(int i = 0 ; i < count ; i++){
    if(runtime[i]!= 0 && process[i].getArr_time()<=time && runtime[pindex]>runtime[i])
        pindex = i;
}
if(runtime[oppindex] == 10000) runtime[oppindex] = 0;
```

oppindex를 선언해주었는데 이 변수 이름의 뜻은 option pindex로 현재 실행된 pindex를 임의로 저장하는 변수이다. 이 변수가 왜 필요하냐면, 현재 준비 중인 프로세스들이 있다. 그 프로세스들 중 가장 짧은 실행 시간을 가진 프로세스를 골라 실행해야 하는데 그 프로세스를 찾는 과정에서 현재 pindex가 가리키는, 즉 실행을 하고 나온 프로세스가 만약 실행 시간이 끝났다면 그 프로세스의 남은 실행 시간은 0일 것이고, 배열에 0이 섞여 있다면 가장 작은 숫자는 0이 될 것이다. 그러므로 oppindex에 pindex를 임의로 저장하고

pindex가 가리키는 프로세스의 실행 시간을 약간의 큰 숫자인 10000으로 설정을 해둔다. 그 다음에 가장 짧은 실행 시간을 찾아준다면 0이 아닌 실행 시간을 가진 프로세스를 pindex가 가리킬 것이다. 10000으로 실행 시간을 설정했던 프로세스를 oppindex 변수를 사용해 다시 0으로 바꿔 주면 pindex 설정이 끝난다.

그리고 문제가 하나 더 있는데 만약 실행 도중에 같은 실행 시간을 가진 프로세스 중 하나를 선택해야 하는 경우가 나온다. 이 경우를 생각해보면, 우선 SRT 알고리즘은 RR알고리즘을 기반으로 하지만 짧은 실행시간을 가진 프로세스를 선택하는 알고리즘이다. 결국 큐 개념을 사용한다는 것인데, 이 코드에서는 알고리즘을 실행하기 전에 프로세스를 도착시간 순으로 프로세스를 정렬했다. 그렇단 뜻은 배열 기준 앞에 있는 프로세스가 큐에 먼저 들어갔다는 뜻이다. 위 pindex를 설정하는 조건문 코드 중에서

`runtime[pindex]>runtime[i]`가 있는데 이를 통해 같은 실행 시간을 가지더라도 앞에 있는 프로세스를 선택하도록 만들 수 있다. 결국 큐에 먼저 들어간 프로세스를 선택한다는 것이다. 같은 실행시간을 가진 프로세스 중 선택해야 하는 문제는 이렇게 해결했다.

3-11. HRN

```
public void run(){
    System.out.println("HRN 알고리즘 실행");
    process = sort(process, num: 1);
    for(int i = 0 ; i < count ; i++)runtime[i]=process[i].getSer_time();
    Gantt();
    Print();
}
```

HRN알고리즘에서도 프로세스를 도착시간 순으로 정렬했다.

```
double priority[] = new double[count];
for(int i = 0 ; i < count ; i++){
    for(int j = 0 ; j < process[i].getSer_time() ; j++) {
        if(prt[i]==0)prt[i]=time+res_time-process[i].getArr_time();
        System.out.print(process[i].getId() + " ");
        time++;
        for(int x = i ; x < count ; x++){
            if(x == i) continue;
            else{
                pwt[x]++;
            }
        }
    }
```

Gantt()함수의 첫 부분이다. double형태의 priority 배열을 설정했는데 이 곳에 프로세스의 새로운 우선순위를 저장할 것이다. HRN의 우선순위는 다음과 같이 정한다.

$$\text{우선순위} = \frac{\text{대기 시간} + \text{CPU 사용 시간}}{\text{CPU 사용 시간}}$$

이를 계산한 우선순위를 저장하기 위한 배열이다. 다시 그 위 사진으로 돌아와 코드를 설명하자면, 반복문을 통해 실행해야하는 프로세스를 실행하는 과정이다.

```
System.out.print("| ");
ret[i]=time - process[i].getArr_time();
if(i != count-1) {
    double maxpriority = pwt[i + 1] / process[i + 1].getSer_time();
    int maxpriorityindex = i + 1;
    for (int j = i + 1; j < count; j++) {
        priority[j] = (double)pwt[j] / (double)process[j].getSer_time();
        if (maxpriority < priority[j] && process[j].getArr_time() < time) {
            maxpriority = priority[j];
            maxpriorityindex = j;
        }
    }
    Process tp = process[maxpriorityindex];
    for (int j = maxpriorityindex; j > i + 1; j--) {
        process[j] = process[j - 1];
    }
    process[i + 1] = tp;
}
```

이 알고리즘은 runtime 배열을 사용하지 않고 process 객체 배열의 인덱스만 사용하여 실행하기 때문에 process 객체 배열을 항상 갱신하며 진행하였다. 가장 높은 우선순위를 가진 프로세스를 앞으로 빼주면서 실행하는 과정이다. 위의 계산 과정이 간단한 것을 볼 수 있는데 계산식을 정리해보면 "우선순위 = 대기시간 / CPU 사용시간 + 1"이다. 모든 프로세스의 우선순위에 1을 빼주면 결국 "대기시간 / CPU 사용시간"이 우선순위를 결정하는 값이다. 그래서 이것만 계산을 해주었다. 실행해야 할 프로세스를 선택해주는 과정이었다.

```
for(int i = 0 ; i < count ; i++){
    pwt[i] -= process[i].getArr_time();
}
```

마지막으로 프로세스 대기시간을 저장해주고 끝낸다.

4. 실행결과

파일

```
5
P1 0 10 3
P2 1 28 2
P3 2 6 4
P4 3 4 1
P5 4 14 2
2
```

실행결과

FCFS 알고리즘 실행

P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 | P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 | P3 P3 P3 P3 P3 P3 | P4 P4 P4 P4 | P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 |

P1의 대기시간 = 0

P2의 대기시간 = 9

P3의 대기시간 = 36

P4의 대기시간 = 41

P5의 대기시간 = 44

평균 대기시간(AWT) = 26.0

P1의 응답시간 = 1

P2의 응답시간 = 10

P3의 응답시간 = 37

P4의 응답시간 = 42

P5의 응답시간 = 45

평균 응답시간(ART) = 27.0

P1의 반환시간 = 10

P2의 반환시간 = 37

P3의 반환시간 = 42

P4의 반환시간 = 45

P5의 반환시간 = 58

평균 반환시간(ATT) = 38.4

SJF 알고리즘 실행

P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 | P4 P4 P4 P4 | P3 P3 P3 P3 P3 P3 | P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 | P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 |

P1의 대기시간 = 0

P4의 대기시간 = 7

P3의 대기시간 = 12

P5의 대기시간 = 16

P2의 대기시간 = 33

평균 대기시간(AWT) = 13.6

P1의 응답시간 = 1

P4의 응답시간 = 8

P3의 응답시간 = 13

P5의 응답시간 = 17

P2의 응답시간 = 34

평균 응답시간(ART) = 14.6

P1의 반환시간 = 10

P4의 반환시간 = 11

P3의 반환시간 = 18

P5의 반환시간 = 30

P2의 반환시간 = 61

평균 반환시간(ATT) = 26.0

비선점 Priority 알고리즘 실행

P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 | P4 P4 P4 P4 | P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 | P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 | P3 P3 P3 P3 P3 P3 |

P1의 대기시간 = 0

P4의 대기시간 = 7

P2의 대기시간 = 13

P5의 대기시간 = 38

P3의 대기시간 = 54

평균 대기시간(AWT) = 22.4

P1의 응답시간 = 1

P4의 응답시간 = 8

P2의 응답시간 = 14

P5의 응답시간 = 39

P3의 응답시간 = 55

평균 응답시간(ART) = 23.4

P1의 반환시간 = 10

P4의 반환시간 = 11

P2의 반환시간 = 41

P5의 반환시간 = 52

P3의 반환시간 = 60

평균 반환시간(ATT) = 34.8

선점형 Priority 알고리즘 실행

P1 | P2 P2 | P4 P4 P4 P4 | P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 | P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 | P1 P1 P1 P1 P1 P1 P1 P1 P1 | P3 P3 P3 P3 P3 P3 |

P4의 대기시간 = 0

P2의 대기시간 = 4

P5의 대기시간 = 29

P1의 대기시간 = 46

P3의 대기시간 = 54

평균 대기시간(AWT) = 26.6

P4의 응답시간 = 1

P2의 응답시간 = 1

P5의 응답시간 = 30

P1의 응답시간 = 1

P3의 응답시간 = 55

평균 응답시간(ART) = 17.6

P4의 반환시간 = 4

P2의 반환시간 = 32

P5의 반환시간 = 43

P1의 반환시간 = 56

P3의 반환시간 = 60

평균 반환시간(ATT) = 39.0

RR 알고리즘 실행

P1 P1 | P2 P2 | P3 P3 | P1 P1 | P4 P4 | P5 P5 | P2 P2 | P3 P3 | P1 P1 | P4 P4 | P5 P5 | P2 P2 | P3 P3 | P1 P1 | P5 P5 | P2 P2 | P1 P1 | P5 P5 | P2 P2 | P5 P5 | P2 P2 | P5 P5 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 |

P1의 대기시간 = 24

P2의 대기시간 = 33

P3의 대기시간 = 18

P4의 대기시간 = 13

P5의 대기시간 = 30

평균 대기시간(AWT) = 23.6

P1의 응답시간 = 1

P2의 응답시간 = 2

P3의 응답시간 = 3

P4의 응답시간 = 6

P5의 응답시간 = 7

평균 응답시간(ART) = 3.8

P1의 반환시간 = 34

P2의 반환시간 = 61

P3의 반환시간 = 24

P4의 반환시간 = 17

P5의 반환시간 = 44

평균 반환시간(ATT) = 36.0

SRT 알고리즘 실행

P1 P1 | P3 P3 | P3 P3 | P3 P3 | P4 P4 | P4 P4 | P1 P1 | P1 P1 | P1 P1 | P1 P1 | P5 P5 | P5 P5 | P5 P5 | P5 P5 | P5 P5 | P5 P5 | P5 P5 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2

P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 | P2 P2 |

P1의 대기시간 = 10

P2의 대기시간 = 33

P3의 대기시간 = 0

P4의 대기시간 = 5

P5의 대기시간 = 16

평균 대기시간(AWT) = 12.8

P1의 응답시간 = 1

P2의 응답시간 = 34

P3의 응답시간 = 1

P4의 응답시간 = 6

P5의 응답시간 = 17

평균 응답시간(ART) = 11.8

P1의 반환시간 = 20

P2의 반환시간 = 61

P3의 반환시간 = 6

P4의 반환시간 = 9

P5의 반환시간 = 30

평균 반환시간(ATT) = 25.2

HRN 알고리즘 실행

P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 | P4 P4 P4 P4 | P3 P3 P3 P3 P3 P3 | P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 | P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 |

P1의 대기시간 = 0

P4의 대기시간 = 7

P3의 대기시간 = 12

P5의 대기시간 = 16

P2의 대기시간 = 33

평균 대기시간(AWT) = 13.6

P1의 응답시간 = 1

P4의 응답시간 = 8

P3의 응답시간 = 13

P5의 응답시간 = 17

P2의 응답시간 = 34

평균 응답시간(ART) = 14.6

P1의 반환시간 = 10

P4의 반환시간 = 11

P3의 반환시간 = 18

P5의 반환시간 = 30

P2의 반환시간 = 61

평균 반환시간(ATT) = 26.0

5. 느낀점

이번 프로젝트를 통해서 CPU 스케줄링에 대해서 다시 공부하게 되었다. 동시에 자바를 다시 한번 공부하게 되는 계기가 되었다. 자바는 "언제 한 번 다시 공부하겠다." 라고 생각을 매번 하지만 시간이 여의치 않아서 실행에 옮기지 못했다. 파일 입출력, 코드를 줄이기 위한 추상클래스와 상속을 공부하면서 어려움이 있었지만 자바를 다시 공부할 수 있다는 생각에 소홀히 하지 않았다. 또 스케줄링을 하면서 간트차트를 그릴 줄은 알지만 이게 정답이 맞는건지 조차 몰랐던 상황에서 이를 또 어떻게 구현해야 할지 정말 막막한 심정이었다. 하지만 며칠 동안 계속 쳐다보고 다시 한 번 검색해보고 책 한 번 더 보고를 반복하다 보니 어떻게 구현을 하게 되었다. 구현하면서 생기는 에러에 뭔가 종속되면서 따라가는 느낌이 들었지만 "그게 뭐 어떤가"라는 생각도 들었다. 구현을 다 한 친구들과 답도 맞춰보면서 코딩할 때의 쾌감을 다시 느낄 수 있었다. 이번 프로젝트는 쉬웠던 적이 없는 것 같다. 그 이유는 내가 공부를 하지 않은 탓일 것이다. 앞으로는 코딩 공부를 꾸준히 해주고 남은 학기 동안 운영체제 공부도 쉽게 생각하지 않고 열심히 할 예정이다.