목차

1. Implement SJF & SRTF Scheduling Algorithms

SJF (Core logic)

```
// 모든 프로세스가 완료될 때까지
while (completed != n) {
    int idx = -1;
    int mn = 10000000;
    // 현재 시간 이전에 도착한, 아직 완료되지 않은 프로세스 중에서
    // 가장 짧은 실행 시간을 가진 프로세스를 찾기
    for (int i = 0; i < n; i++) {
        // 현재 시간 이전에 도착했고, 아직 완료되지 않은 프로세스인지 확인
        if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
            // 해당 프로세스의 실행 시간이 현재까지 찾은 가장 짧은 실행 시간보다 짧은 경우
            if (p[i].burst_time < mn) {
                mn = p[i].burst_time; // 가장 짧은 실행 시간 업데이트
                idx = i; // 선택된 프로세스의 인덱스 업데이트
            }
            // 만약 실행 시간이 같다면, 더 빨리 도착한 프로세스를 선택
            if (p[i].burst_time == mn) {
                if (p[i].arrival_time < p[idx].arrival_time) {
                    mn = p[i].burst_time; // 가장 짧은 실행 시간 업데이트
                    idx = i; // 선택된 프로세스의 인덱스 업데이트
                }
            }
        }
    }
    if (idx != -1) {
        // 선택된 프로세스의 시작 시간, 완료 시간 등 계산
        p[idx].start_time = current_time;
        p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        // 총 시간들을 계산
        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;
        total_idle_time += p[idx].start_time - prev;

        // 프로세스 완료 표시
        is_completed[idx] = 1;
        completed++;
        current_time = p[idx].completion_time;
        prev = current_time;
    } else {
        current_time++; // 아직 실행할 프로세스가 없는 경우 시간을 증가
    }

}

// 전체 프로세스 중 가장 빠른 도착 시간과 가장 늦은 완료 시간을 찾는 부분
int min_arrival_time = 10000000; // 가장 빠른 도착 시간을 찾기 위한 초기 값 설정
int max_completion_time = -1; // 가장 늦은 완료 시간을 찾기 위한 초기 값 설정

for (int i = 0; i < n; i++) {
    // 모든 프로세스를 순회하며
    // 현재까지의 최소 도착 시간과 비교하여 더 작은 값을 min_arrival_time 에 저장
    min_arrival_time = min(min_arrival_time, p[i].arrival_time);

    // 현재까지의 최대 완료 시간과 비교하여 더 큰 값을 max_completion_time 에 저장
    max_completion_time = max(max_completion_time, p[i].completion_time);
}
```

SJF 스케줄링 방법에 따라 가장 짧은 실행 시간을 가진 프로세스를 우선적으로 스케줄링하고, 그에 따른 시작 시간, 완료 시간, 반환 시간, 대기 시간, 응답 시간을 구하는 로직을 구현하였다.

SRTF (Core logic)

```
// 모든 프로세스가 완료될 때까지
while (completed != n) {
    int idx = -1;
    int mn = 10000000;
    // 현재 시간 이전에 도착한, 아직 완료되지 않은 프로세스 중에서
```

```
    // 가장 짧은 남은 실행 시간을 가진 프로세스를 찾기
    for (int i = 0; i < n; i++) {
        // 현재 시간 이전에 도착했고, 아직 완료되지 않은 프로세스인지 확인
        if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
            // 해당 프로세스의 남은 실행 시간이 현재까지 찾은 가장 짧은 실행 시간보다 짧은 경우
            if (burst_remaining[i] < mn) {
                mn = burst_remaining[i]; // 가장 짧은 남은 실행 시간 업데이트
                idx = i; // 선택된 프로세스의 인덱스 업데이트
            }
            // 만약 남은 실행 시간이 같다면, 더 빨리 도착한 프로세스를 선택
            if (burst_remaining[i] == mn) {
                if (p[i].arrival_time < p[idx].arrival_time) {
                    mn = burst_remaining[i]; // 가장 짧은 남은 실행 시간 업데이트
                    idx = i; // 선택된 프로세스의 인덱스 업데이트
                }
            }
        }
    }

    if (idx != -1) {
        // 프로세스 실행 시작
        if (burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time; // 시작 시간 설정
            total_idle_time += p[idx].start_time - prev; // 유휴 시간 계산
        }
        burst_remaining[idx] -= 1; // 남은 실행 시간 감소
        current_time++; // 현재 시간 증가
        prev = current_time;

        // 프로세스 완료 처리
        if (burst_remaining[idx] == 0) {
            p[idx].completion_time = current_time; // 완료 시간 설정
            p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time; // 반환 시간
계산

            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time; // 대기 시간 계산
            p[idx].response_time = p[idx].start_time - p[idx].arrival_time; // 응답 시간 계산


            // 총 시간들을 계산
            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;

            is_completed[idx] = 1; // 완료 상태 설정
            completed++;
        }
    } else {
        current_time++; // 현재 시간 증가 (유휴 상태)
    }
}
```

SRTF 스케줄링 방법에 따라 현재 시간에서 가장 짧은 남은 실행 시간을 가진 프로세스를 우선적으로 스케줄링하고, 그에 따른 시작 시간, 완료 시간, 반환 시간, 대기 시간, 응답 시간을 구하는 로직을 구현하였다.

2. Test it for the “SJF_Scheduling_Notes” in the attachment. (Test results)

FCFS (Baseline)

```
                    [Problem 1~2]
#P      AT      BT      ST      CT      TAT     WT      RT

1       3       1       13      14      11      10      10
```

```
2       1       4       6       10      9       5       5

3       4       2       14      16      12      10      10

4       0       6       0       6       6       0       0

5       2       3       10      13      11      8       8


Average Turnaround Time = 9.80
Average Waiting Time = 6.60
Average Response Time = 6.60
CPU Utilization = 100.00%
Throughput = 0.31 process/unit time
```

[Problem 3]

```
#P      AT      BT      ST      CT      TAT     WT      RT

1       0       7       0       7       7       0       0

2       1       5       7       12      11      6       6

3       2       3       12      15      13      10      10

4       3       1       15      16      13      12      12

5       4       2       16      18      14      12      12

6       5       1       18      19      14      13      13


Average Turnaround Time = 12.00
Average Waiting Time = 8.83
Average Response Time = 8.83
CPU Utilization = 100.00%
Throughput = 0.32 process/unit time
```

[Problem 4]

```
#P      AT      BT      ST      CT      TAT     WT      RT

1       0       9       0       9       9       0       0

2       1       4       9       13      12      8       8

3       2       9       13      22      20      11      11


Average Turnaround Time = 13.67
Average Waiting Time = 6.33
Average Response Time = 6.33
```

```
CPU Utilization = 100.00%
Throughput = 0.14 process/unit time
```

[Problem 5]

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	20	0	20	20	0	0
2	15	25	20	45	30	5	5
3	30	10	45	55	25	15	15
4	45	15	55	70	25	10	10

```
Average Turnaround Time = 25.00
Average Waiting Time = 7.50
Average Response Time = 7.50
CPU Utilization = 100.00%
Throughput = 0.06 process/unit time
```

SJF

[Problem 1~2]

#P	AT	BT	ST	CT	TAT	WT	RT
1	3	1	6	7	4	3	3
2	1	4	12	16	15	11	11
3	4	2	7	9	5	3	3
4	0	6	0	6	6	0	0
5	2	3	9	12	10	7	7

```
Average Turnaround Time = 8.00
Average Waiting Time = 4.80
Average Response Time = 4.80
CPU Utilization = 100.00%
Throughput = 0.31 process/unit time
```

[Problem 3]

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	7	0	7	7	0	0
2	1	5	14	19	18	13	13
3	2	3	11	14	12	9	9
4	3	1	7	8	5	4	4
5	4	2	9	11	7	5	5
6	5	1	8	9	4	3	3

Average Turnaround Time = 8.83
Average Waiting Time = 5.67
Average Response Time = 5.67
CPU Utilization = 100.00%
Throughput = 0.32 process/unit time

[Problem 4]

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	9	0	9	9	0	0
2	1	4	9	13	12	8	8
3	2	9	13	22	20	11	11

Average Turnaround Time = 13.67
Average Waiting Time = 6.33
Average Response Time = 6.33
CPU Utilization = 100.00%
Throughput = 0.14 process/unit time

```
                        [Problem 5]
#P      AT      BT      ST      CT      TAT     WT      RT

1       0       20      0       20      20      0       0

2       15      25      20      45      30      5       5

3       30      10      45      55      25      15      15

4       45      15      55      70      25      10      10


Average Turnaround Time = 25.00
Average Waiting Time = 7.50
Average Response Time = 7.50
CPU Utilization = 100.00%
Throughput = 0.06 process/unit time
```

SJF 스케줄링 방식을 적용한 결과에서는, 몇 가지 케이스에 대해 FCFS에 비해 평균 대기 시간과 응답 시간이 상대적으로 낮은 것을 확인할 수 있었다. 이는 SJF가 각 프로세스의 실행 시간이 짧은 순으로 스케줄링하기 때문에 발생하는 효과로 추정된다.

SRTF

```
                        [Problem 1~2]
#P      AT      BT      ST      CT      TAT     WT      RT

1       3       1       3       4       1       0       0

2       1       4       1       6       5       1       0

3       4       2       6       8       4       2       2

4       0       6       0       16      16      10      0

5       2       3       8       11      9       6       6


Average Turnaround Time = 7.00
Average Waiting Time = 3.80
Average Response Time = 1.60
CPU Utilization = 100.00%
Throughput = 0.31 process/unit time
                        [Problem 3]
#P      AT      BT      ST      CT      TAT     WT      RT
```

1	0	7	0	19	19	12	0
2	1	5	1	13	12	7	0
3	2	3	2	6	4	1	0
4	3	1	3	4	1	0	0
5	4	2	7	9	5	3	3
6	5	1	6	7	2	1	1

Average Turnaround Time = 7.17
Average Waiting Time = 4.00
Average Response Time = 0.67
CPU Utilization = 100.00%
Throughput = 0.32 process/unit time

[Problem 4]

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	9	0	13	13	4	0
2	1	4	1	5	4	0	0
3	2	9	13	22	20	11	11

Average Turnaround Time = 12.33
Average Waiting Time = 5.00
Average Response Time = 3.67
CPU Utilization = 100.00%
Throughput = 0.14 process/unit time

[Problem 5]

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	20	0	20	20	0	0
2	15	25	20	55	40	15	5
3	30	10	30	40	10	0	0
4	45	15	55	70	25	10	10

Average Turnaround Time = 23.75
Average Waiting Time = 6.25
Average Response Time = 3.75

```
CPU Utilization = 100.00%
Throughput = 0.06 process/unit time
```

FCFS에 비해 평균 대기 시간과 응답 시간이 상당히 단축된 것을 확인 할 수 있었다. SRTF는 남은 실행 시간이 가장 짧은 프로세스를 우선 처리하기 때문에 발생하는 효과로 보인다. 또한 SJF의 지표보다도 더 좋은 결과를 확인할 수 있었는데, 이는 SJF는 각 프로세스의 전체 실행 시간을 기준으로 스케줄링을 하는 반면, SRTF는 현재 시점에서 남은 실행 시간이 가장 짧은 프로세스를 선택한다. 이 차이로 인해 SRTF는 더 동적으로 반응하며, 프로세스의 실행 시간이 변경되거나 새로운 프로세스가 도착할 때 즉각적으로 대응한 결과로 분석된다.

3. Impressions

이번 과제를 통해 SJF 및 SRTF 스케줄링 알고리즘을 구현하고 분석하며, 운영 체제에서의 프로세스 관리와 스케줄링의 복잡성 및 중요성을 더 깊이 이해할 수 있었고, SJF 스케줄링을 적용함으로써 프로세스의 실행 시간이 짧은 순서대로 처리하는 방식의 장단점을 실제로 체험하고, 이를 통해 대기 시간과 응답 시간이 어떻게 달라지는지 관찰할 수 있었다.