운영체제 프로젝트1 결과보고서

수 강 과 목 : 운영체제
담 당 교 수 : 엄영익 교수님
학 과 : 소프트웨어학과
학 번 : 2020315798
이 름 : 최진우
제 출 일 : 2022년 10월 6일

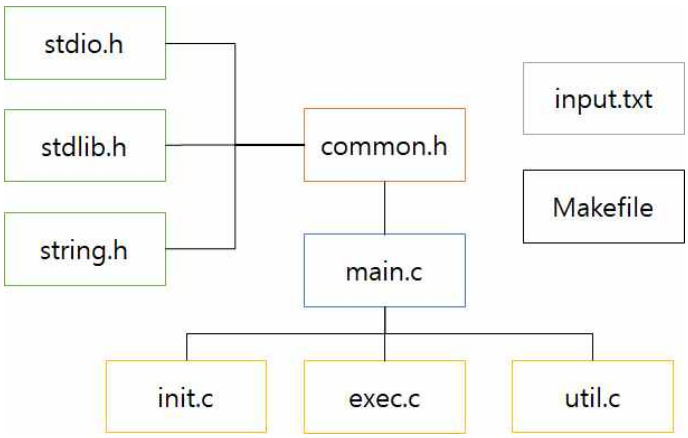
1. 프로젝트 개요

1) 목표 : MFQ 스케줄링 기법 구현
2) 세부 목표
    - 3개의 Ready queue를 갖는 MFQ 스케줄링 기법 구현
        · Queue 0 : Round Robin 스케줄링 기법 사용 (Time quantum 2)
        · Queue 1 : Round Robin 스케줄링 기법 사용 (Time quantum 4)
        · Queue 2 : SRTN 스케줄링 기법 사용

2. 프로젝트 개발 환경

1) 개발 언어 : C
2) 개발 환경 : Ubuntu 18.04.5 LTS (성균관대학교 인의예지 클러스터)
3) 컴파일러 : gcc 7.5.0
4) 사용 프로그램 및 목적
    · Putty : 컴파일 및 vi에디터
    · FileZilla : SFTP를 이용한 백업 및 테스트 케이스 전송
    · EditPlus5 : 코드 작성
5) 라이브러리 및 목적
    · stdio.h : 표준 출력 및 파일 접근
    · stdlib.h : 동적 할당
    · string.h : input txt 파일의 띄어쓰기 토큰화

3. 프로젝트 구조도



1) common.h

커스텀 라이브러리로써 stdio.h, stdlib.h, string.h를 include하고 여러 상수를 define 하고 있습니다. 또한 프로세스 구조체와 Ready queue 배열, 기타 변수들을 전역으로 선언하고 있습니다.

2) main.c

텍스트를 argv인자로 받고 이를 세팅하는 initPrcs(), initQueue()를 호출합니다. 스케줄링 로직을 실행하는 exec()를 호출하며 종료 전 quit()를 통해 메모리를 동적 할당 변수들을 free()합니다.

3) init.c

파일 포인터를 통해 접근하여 총 프로세스 수와 프로세스별 정보를 구조체 변수에 담습니다. 초기 프로세스를 저장하는 구조체 배열에 각 프로세스를 담는 로직이 정리되어 있습니다.

4) exec.c

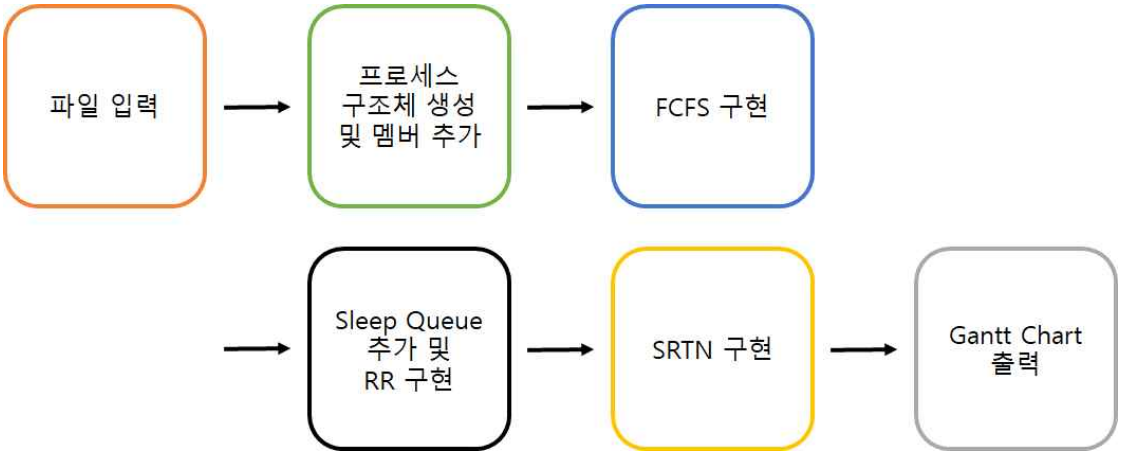
시간 증가에 따른 스케줄링 로직을 구현했습니다. 초기 프로세스 배열에서 시간별 ready queue 진입과 CPU burst, I/O burst, Preemption, Gantt chart 출력 등을 진행합니다.

5) Makefile

main.c, init.c, exec.c를 컴파일/링킹하여 실행파일을 만드는 코드가 정리되어 있습니다.

4. 프로젝트 설계 및 알고리즘

1) 설계 흐름도

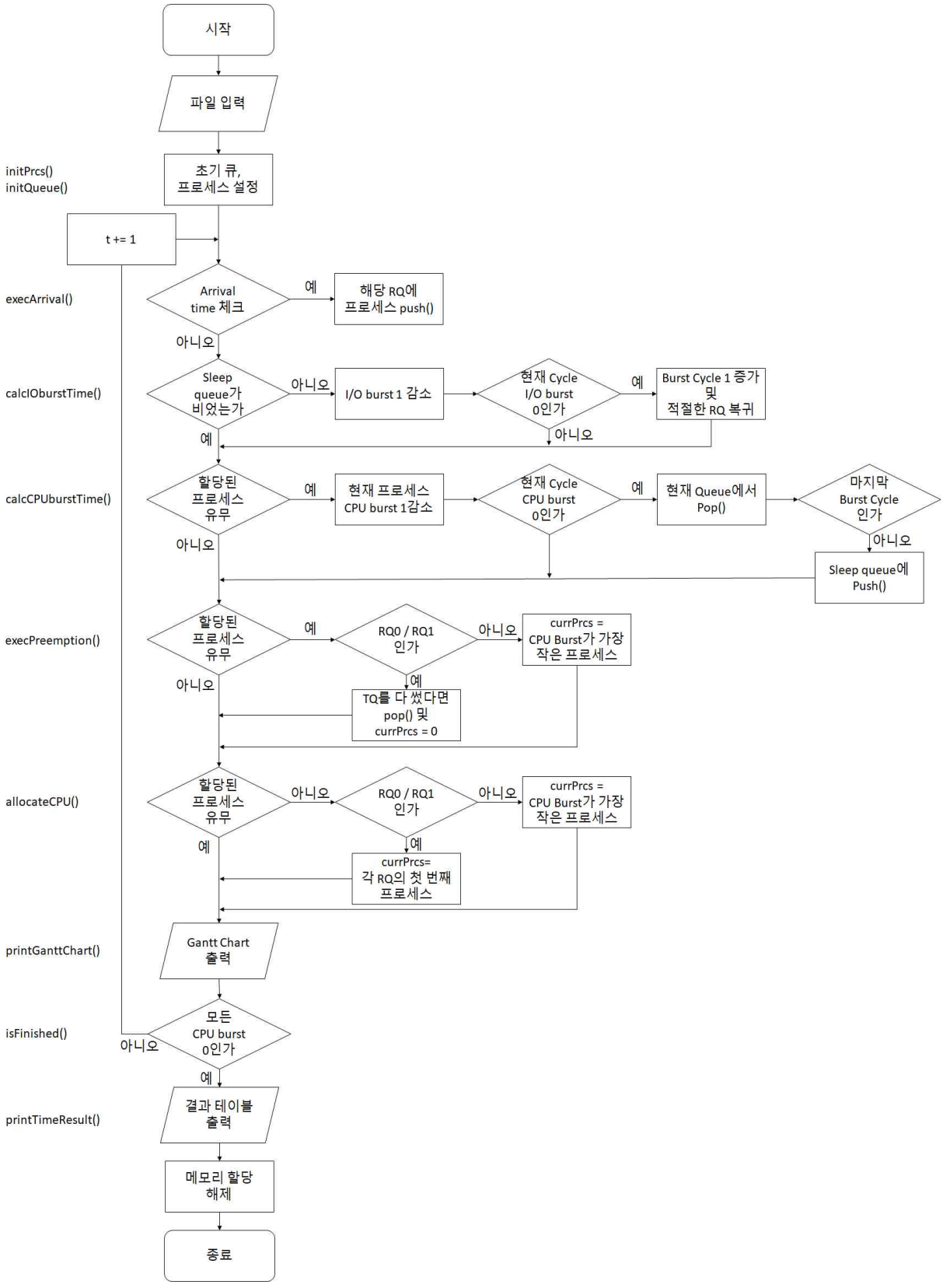


2) 설계/알고리즘 설명

우선 time quantum의 조건이 추가되는 RR 기법에 앞서 제일 간단한 FCFS 기법을 구현했습니다. FCFS 기법을 구현하는 과정에서는 CPU burst를 체크 기능의 함수를 작성했습니다. 1개의 FCFS ready queue 구현 이후 time quantum과 sleep queue추가를 통해 RR 기법을 구현했습니다. 이 과정에서는 I/O burst 체크 기능의 함수 및 preemption 기능의 함수를 작성했습니다. 이후 ready queue를 2개 더 추가하여 총 3개의 RR 기법의 MFQ를 구현했습니다. 단순히 ready queue 1개와 sleep queue 1개에서 이동하는 것이 아닌 다양한 queue에서의 이동을 확인하고 싶었습니다. 최종적으로 마지막 queue의 기법을 RR에서 SRTN으로 변경하며 preemtion의 조건을 변경해주는 것으로, RR 기법의 ready queue 2개와 SRTN 기법의 ready queue 1개로 이루어진 MFQ 스케줄링에 대한 구현을 완성했습니다.

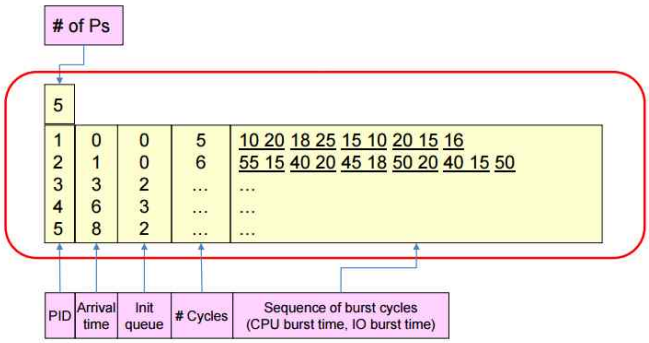
전반적인 알고리즘은 스케줄링 파일을 입력받아 프로세스 구조체 배열에 저장하고, 초기 세팅이 완료되었으면 모든 프로세스의 마지막 burst cycle의 CPU burst가 0이 되는 순간까지 무한루프를 돌며 MFQ 기법대로 스케줄링하는 것입니다. 무한루프를 도는 과정에서 해당 시간에 arrival time을 갖는 프로세스를 push하고, sleep queue에서 I/O burst를 끝낸 프로세스를 적당한 queue에 push하고, preemtion과 CPU 할당을 거칩니다. 이 과정에서 직전 시간 대비 CPU를 할당받은 프로세스가 변하였다면 Gantt chart에 파티션을 추가하여 작성합니다. 최종적으로 루프가 끝나면 전체 프로세스 별 turnaround time과 waiting time 및 각 시간 별 평균에 대한 정보를 출력하며 프로그램이 끝납니다.

3) 알고리즘 흐름도


시작
파일 입력
initPrcs()
initQueue()
초기 큐,
프로세스 설정
t += 1
execArrival()
Arrival
time 체크
예
해당 RQ에
프로세스 push()
아니오
calcIOburstTime()
Sleep
queue가
비었는가
아니오
I/O burst 1 감소
현재 Cycle
I/O burst
0인가
예
Burst Cycle 1 증가
및
적절한 RQ 복귀
아니오
예
calcCPUburstTime()
할당된
프로세스
유무
예
현재 프로세스
CPU burst 1감소
현재 Cycle
CPU burst
0인가
예
현재 Queue에서
Pop()
마지막
Burst Cycle
인가
아니오
아니오
Sleep queue에
Push()
execPreemption()
할당된
프로세스
유무
예
RQ0 / RQ1
인가
아니오
currPrcs =
CPU Burst가 가장
작은 프로세스
아니오
예
TQ를 다 썼다면
pop() 및
currPrcs = 0
allocateCPU()
할당된
프로세스
유무
아니오
RQ0 / RQ1
인가
아니오
currPrcs =
CPU Burst가 가장
작은 프로세스
예
예
currPrcs=
각 RQ의 첫 번째
프로세스
printGanttChart()
Gantt Chart
출력
isFinished()
모든
CPU burst
0인가
아니오
예
printTimeResult()
결과 테이블
출력
메모리 할당
해제
종료

5. 프로젝트 결과 정책

1) input.txt



출처 : 2022-2 운영체제 과제안내서

2) 콘솔 결과

· PID와 시간의 길이가 가변적이기에 세로로 정리
· 프로세스 별 CPU 할당 시간을 기록한 Gantt chart 출력
· 각 프로세스 별 Turnaround time, Waiting time 및 전체 프로세스 평균 Turnaround time, Waiting time을 정리한 Result Table 출력
· Preemption 되어 Ready queue가 바뀌어도 CPU 할당이 이어질 경우 파티션 생략
· 모든 프로세스가 Ready queue에 있지 않은 경우 Not Allocated 출력(오른쪽 사진 42~45초)
· 하단 Result Table의 평균 시간은 소수점 2째 자리까지 반올림

· 분석 포인트
- 상단 input.txt의 CPU burst 총합과 하단 Result Table에서 각 프로세스 별 BT(=TT-WT)와의 일치 여부
- Gantt chart에서 프로세스별 최종 기록 시간에서 input.txt의 Arrival time을 뺀 값과 Result Table에서 해당 프로세스의 Turnaround Time의 일치 여부

```
4
1 0 0 1 7
2 1 0 4 2 5 8 10 1 3 5
3 3 1 3 1 4 5 3 8
4 6 2 1 10
```

```
+-------+--------+
|   GanttChart    |
+-------+--------+
|  PID  |  Time  |
+--------  0     |
|   1   |        |
+--------  2     |
|   2   |        |
+--------  4     |
|   1   |        |
+--------  8     |
|   3   |        |
+--------  9     |
|   2   |        |
+--------  15    |
|   3   |        |
+--------  20    |
|   1   |        |
+--------  21    |
|   2   |        |
+--------  23    |
|   3   |        |
+--------  31    |
|   4   |        |
+--------  41    |
|   2   |        |
+--------  42    |
| Not Allocated  |
+--------  45    |
|   2   |        |
+--------  50    |
+----------------+
+----------------------+
|     Result Table     |
+----------------------+
|  PID  |  TT  |  WT   |
+----------------------+
|   1   |  21  |  14   |
+----------------------+
|   2   |  49  |  33   |
+----------------------+
|   3   |  28  |  14   |
+----------------------+
|   4   |  35  |  25   |
+----------------------+
| Average TT:  33.25   |
+----------------------+
| Average WT:  21.50   |
+----------------------+
```

6. 프로젝트 결과 케이스 분석

(각 프로세스 별 TT, WT와 전체 프로세스 평균 TT, WT는 생략했습니다.)

1) I/O burst가 없는 가장 간단한 경우

```
4
1 0 0 1 5
2 1 0 1 1
3 6 2 1 5
4 7 2 1 2
```

```
+--------+---------+
|   GanttChart     |
+--------+---------+
|  PID   |  Time   |
+--------+    0    |
|   1    |         |
+--------+    2    |
|   2    |         |
+--------+    3    |
|   1    |         |
+--------+    6    |
|   3    |         |
+--------+    7    |
|   4    |         |
+--------+    9    |
|   3    |         |
+--------+    13   |
+------------------+
```

프로세스 1이 2초간 Time quantum을 모두 사용할 경우 Ready queue 1로 Preemption 됩니다. 프로세스 2는 자신의 CPU burst를 모두 사용하고 종료됩니다. 바로 프로세스 1이 Ready queue 1에서 남은 3초 동안 진행됩니다. 6초에 종료되는 순간 프로세스 3이 Ready queue 2에서 진입합니다. 7초가 되면서 프로세스 4가 진입하는데 CPU burst가 더 작으므로 프로세스 4가 진행되고 종료 후 다시 프로세스 3으로 돌아갑니다.

2) I/O burst가 없는 가장 간단한 경우 + 어떤 프로세스도 할당받지 못한 경우

```
4
1 0 0 1 5
2 1 0 1 1
3 7 2 1 5
4 8 2 1 2
```

```
+--------+---------+
|   GanttChart     |
+--------+---------+
|  PID   |  Time   |
+--------+    0    |
|   1    |         |
+--------+    2    |
|   2    |         |
+--------+    3    |
|   1    |         |
+--------+    6    |
| Not Allocated    |
+--------+    7    |
|   3    |         |
+--------+    8    |
|   4    |         |
+--------+    10   |
|   3    |         |
+--------+    14   |
+------------------+
```

위의 1)의 경우에서 프로세스 3, 4의 Arrival time을 1씩 증가시켜 6~7초 사이에 어떤 프로세스도 할당받지 못하는 경우입니다. 오른쪽 사진처럼 Not Allocated가 출력됩니다.

3) I/O burst가 있는 경우 + SRTN Preemption이 없는 경우

```
3
1 0 0 1 3
2 1 0 2 3 5 9
3 2 2 2 9 4 15
```

GanttChart	
PID	Time
	0
1	
	2
2	
	4
1	
	5
2	
	6
3	
	15
2	
	21
3	
	25
2	
	28
3	
	39

0~5초에 RQ0에서 프로세스 1, 2의 Preemption이 일어납니다. 5초부터 프로세스 2는 RQ1에서 CPU burst 중 남은 1초만 사용하고 SQ에 진입합니다. 프로세스 3이 RQ2에서 실행됩니다. 그동안 프로세스 2는 SQ에서 규칙대로 RQ0에 진입합니다. 여기서 TQ를 모두 사용하고 RQ1로 Preemption 됩니다. 하지만 이 시점(17초)에 프로세스 3은 SQ에 있는 상태이므로 프로세스 2가 계속 CPU를 사용합니다. RQ1의 TQ를 사용한 프로세스 2는 21초에 RQ2로 Preemption됩니다. 19초에 SQ에서 벗어나 규칙에 따라 RQ1로 진입한 프로세스 3이 CPU를 이어받습니다. 프로세스 3도 TQ를 모두 사용하고 25초에 RQ2로 Preemption 됩니다. RQ2에서 두 프로세스가 SRTN 기법에 따라 CPU burst를 비교합니다. 프로세스 2는 3초, 프로세스 3은 11초가 남았으므로 프로세스 2가 CPU를 할당받습니다.

4) I/O burst가 있는 경우 + SRTN Preemption이 있는 경우

```
3
1 0 0 1 3
2 1 0 2 3 5 9
3 2 2 2 9 4 6
```

GanttChart	
PID	Time
	0
1	
	2
2	
	4
1	
	5
2	
	6
3	
	15
2	
	21
3	
	27
2	
	30

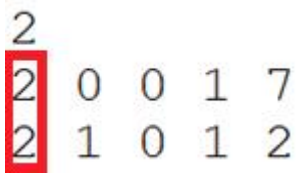
위의 3)의 경우에서 25초에 프로세스 3이 프로세스 2보다 남은 CPU burst가 작은 경우입니다. 프로세스 3이 RQ1에서 TQ를 사용하고 RQ2에서도 CPU를 할당받고 27초까지 사용하고 종료됩니다.

부록1. 프로젝트 오류 대응 - 미대응 시 Seg fault 등 오류 발생

1) 파일이 없는 경우

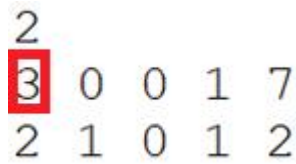
```
2020315798@swui:/home/2020315798/2022-2/OS$ ./main sample7.
txt
Failed to open input file.
```

2) 프로세스 ID가 중복된 경우




```
2020315798@swui:/home/2020315798/2022-2/OS$ ./main sample6.
txt
Invalid input file - Duplicated process Id
```

3) 프로세스 ID가 프로세스 개수보다 큰 경우




```
2020315798@swui:/home/2020315798/2022-2/OS$ ./main sample6.
txt
Invalid input file - Invalid process Id
```

4) 프로세스 개수와 프로세스 정보가 다른 경우


```
2020315798@swui:/home/2020315798/2022-2/OS$ ./main sample6.
txt
Invalid input file - Process count mismatch
```

5) Ready queue 번호가 전체 Ready queue 개수보다 큰 경우


```
2020315798@swui:/home/2020315798/2022-2/OS$ ./main sample6.
txt
Invalid input file - Maximum queue exceeded
```

6) 메모리 할당 오류