

운영체제 과제 #2

2-1. Multi-level queues with Real-time class

2-2. SRJF Scheduling

2-3. Round-Robin Scheduling

2-4. Evaluation

과제 내용 및 제출 방법

- 프로그램 작성: /home/ubuntu/hw2/os2.c (반드시 지정된 이름으로 작성)
 - (2-1) 멀티레벨큐 구조 구현
 - (2-2) SRJF 스케줄링 기법 구현
 - (2-3) 라운드 로빈 스케줄링 기법 구현
 - (2-4) FIFO, SRJF, Round-Robin에 대한 스케줄링 성능 분석
- LMS "과제2" 제출
 - 보고서와 소스코드 4개를 함께 압축 (zip) 해서 하나의 파일로 제출
 - 소스코드는 각 부분 과제 별로 모두 각각 제출 (os2-1.c ~ os2-4.c)
- 보고서 내용
 - 표지 포함 총 A4 20장 이하, PDF 형식으로 제출 (이외 형식은 10%p 감점)
 - 간략한 자율 진도표 포함 (for Iterative and incremental development)
 - 주석이 포함된 전체 코드와 각 주요한 부분에 대한 서술, test2.bin 을 이용한 수행 결과
 - JOTA "2021 운영체제 과제 2-1, 2-2, 2-3" 제출 결과 캡처 (분량에 포함되지 않음)
 - 과제 수행 시 어려웠던 점 및 해결 방안
- 기한: 5/17 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

(과제1과 다른 점)

입력되는 이진 프로세스 정보의 형태

- Process tuple: 프로세스 정보와 코드로 구성
 - 입력되는 이진 데이터는 N개의 process tuple 로 구성
 - 각 tuple의 크기는 코드의 크기에 따라 가변적

- 프로세스 정보 (고정 크기)

- PID: 프로세스 ID
 - 80~99인 경우, real-time process
- 도착시간: 프로세스가 실행을 시작한 시간
- 코드길이: 프로그램 코드의 길이 (바이트 단위, 짝수)

- 코드 (가변 크기)

- Code tuple 의 집합: 1 tuple = 2 Bytes
 - 예) 코드 길이가 10 Bytes인 경우, 5개의 tuple 로 구성
- Code tuple: 각 1 바이트 크기의 동작과 길이로 구성
 - 동작: 시스템에 요청하는 작업의 종류 (예. 00 -> CPU 작업, 01 -> IO 작업) <- IO 작업 없음
 - 길이: 해당 동작을 수행하는데 걸리는 시간
 - 예) 00 05 = CPU 작업을 5 만큼의 시간 동안 수행

```
typedef struct {  
    int pid;           //ID  
    int arrival_time;  //도착시간  
    int code_bytes;    //코드길이(바이트)  
} process;
```



(과제1과 다른 점)

IO 작업 없음, CS Overhead 10->5

- IO 작업 없음: 과제 1 과의 의존성, 난이도를 낮추기 위함
 - 과제 1을 다 수행하지 못했더라도,
1-2 과제에서 IO 처리를 제외한 부분까지는 수행했다면
과제 2를 진행할 수 있음
 - 가변 길이의 Code tuple 없음: fixed length = 2
 - 과제 1을 다 수행한 경우, IO 작업을 굳이 제거할 필요 없음
최종 코드 기반으로 과제 2 요구사항대로 수정 및 추가하면 됨
 - CPU 스케줄링 동작 습득에 집중하도록 수행 난이도를 낮춤
- Context Switching overhead 를 10 -> 5 clocks 로 낮춤
 - 스케줄링 기법들의 성능 비교를 할 때, CS의 영향을 낮추기 위함
- 기타 다른 사항들은 모두 과제 1의 동작을 기반으로 함



Iterative and incremental development

- 반복적, 점진적 프로그래밍 습관을 기르기 위함
- 도달하고자 하는 최종 목표까지 단계를 잘게 나누어 하나씩 검증, 달성하며 진행
 - 단계별로 백업 파일을 남기는 것도 좋은 습관 (Git 사용 시, Commit 의 단위가 됨)
- 아래 예시를 참고하여 각자 **자율적으로 진도표**를 간략하게 작성해서, **보고서에 반드시 첨부할 것**

단계	완료 여부
List 자료구조 파악: 예제 수행	O
list_for_each_entry()를 이용한 순회	O
list_for_each_entry_safe_reverse()...	O
과제 1-1 완료 (JOTA 확인)	O
Idle process 구현	
...	
과제 1-2 완료 (JOTA 확인)	
...	
과제 1-3 완료 (JOTA 확인)	

2-1. Multi-level queues



2-1. Multi-level queue 구조 구현

- Ready queue를 세 개의 계층적 스케줄링 클래스로 구분하고, 각각의 클래스별로 queue를 따로 관리하며, 큐에 대해 Priority-based scheduling 사용
 - Real-time > Normal > Idle class 순으로 높은 우선순위 부여 (높은 클래스부터 프로세스 선택)
- Real-time class
 - 실시간 처리를 필요로 하는 Real-time process 들을 관리 (PID: 80~99)
 - FIFO 스케줄링 사용
 - Preemption : Real-time process 가 로드되었을 때, normal 프로세스가 동작 중이라면, 즉각 스케줄링을 수행하여 RT process 가 수행되도록 함 (구현 시 유의: clock 증가 없이 즉각 변경)
 - 동작 중이던 Normal 프로세스는 Ready Q의 맨 앞에 삽입하여 다음 스케줄링 시 다시 선택되도록 함
- Normal class
 - Real-time, Idle class의 process 를 제외한 모든 다른 프로세스를 관리
 - FIFO, SRJF, Round Robin 등 여러 기법 중 하나를 선택, 적용할 수 있는 구조로 설계 (2-1: FIFO 사용)
- Idle class
 - 상위 classes 에 아무 프로세스가 없는 경우에만 선택되어,
 - (Idle process 만 존재하므로) idle process 가 스케줄링되어 실행됨

Output

- 과제 2 출력
 - Loading, Switching, Termination 정보 출력
 - Termination 시, 각 프로세스 별로 CPU utilization, waiting , response time 출력
 - Waiting time: ready 상태에서 스케줄링을 대기한 total time
 - Response time: Arrival 하고, 처음 스케줄링되기까지 시간
 - Final report: CPU utilization + Avg. waiting time + Avg. Response time
- 과제 2-1, 2-2, 2-3 까지는 JOTA 테스트를 하여야 하므로 위 출력 그대로 사용
- 2-4 에서는 필요한 metric 을 각자 자유롭게 추가해서 사용

test2.bin

- 5개의 프로세스 정보가 저장됨 (코드 길이는 모두 0x02)
 - 0번 프로세스: 도착시간=0, CPU 작업 2 clocks
 - 1번 프로세스: 도착시간=4, CPU 작업 170 clocks
 - 2번 프로세스: 도착시간=5, CPU 작업 80 clocks
 - 80번 프로세스: 도착시간=6, CPU 작업 10 clocks (real-time)
 - 81번 프로세스: 도착시간=40, CPU 작업 10 clocks (real-time, preemption)
- 짧은 operations 로 debugging 이 용이하도록 수정하였음
 - 유의할 점: 0번 프로세스가 종료된 2 clock 시점에 다른 프로세스들이 없으므로, IDLE process 가 context switching 되어 실행됨.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D
00000000	00	00	00	00	00	00	00	00	02	00	00	00	00	02
0000000E	01	00	00	00	04	00	00	00	02	00	00	00	00	AA
0000001C	02	00	00	00	05	00	00	00	02	00	00	00	00	50
0000002A	50	00	00	00	06	00	00	00	02	00	00	00	00	0A
00000038	51	00	00	00	28	00	00	00	02	00	00	00	00	0A

operation length

JOTA: 2-1

2021운영체제 과제 2-1

- LMS(구버전) 과제 2 참조

stdin 으로부터 Binary 형태의 프로세스 정보와 코드를 읽어들이, 아래와 같이 출력하시오.

2-1.

Multi-level queue 구조 구현

출력 형식

CPU clock 진행에 따라,

```
로드된 프로세스 정보: "%04d CPU: Loaded PID: %03d\tArrival: %03d\tCodesize: %03d\tPC: %03d\n"
프로세스 종료: %04d CPU: Process is terminated\tPID:%03d\tPC:%03d\n"
CPU 작업 전환: "%04d CPU: Switched\tfrom: %03d\tto: %03d\n"
모든 작업 종료 후, 각 프로세스별 정보 출력: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"
모든 작업 종료 후, 최종 리포트: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"
```

출력 예 (test2.bin 이용)

```
0000 CPU: Loaded PID: 000   Arrival: 000   Codesize: 002   PC: 000
0000 CPU: Loaded PID: 100   Arrival: 000   Codesize: 002   PC: 000
0002 CPU: Process is terminated PID:000 PC:001
0007 CPU: Switched  from: 000   to: 100
0007 CPU: Loaded PID: 001   Arrival: 004   Codesize: 002   PC: 000
0007 CPU: Loaded PID: 002   Arrival: 005   Codesize: 002   PC: 000
0007 CPU: Loaded PID: 000   Arrival: 006   Codesize: 002   PC: 000
```



JOTA: 2-1 with test2.bin

```
ubuntu@41983:~/hw2$ cat test2.bin | ./os2-1
0000 CPU: Loaded PID: 000      Arrival: 000      Codesize: 002      PC: 000
0000 CPU: Loaded PID: 100      Arrival: 000      Codesize: 002      PC: 000
0002 CPU: Process is terminated PID:000 PC:001
0007 CPU: Switched      from: 000      to: 100
0007 CPU: Loaded PID: 001      Arrival: 004      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 002      Arrival: 005      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 080      Arrival: 006      Codesize: 002      PC: 000
0012 CPU: Switched      from: 100      to: 080
0022 CPU: Process is terminated PID:080 PC:001
0027 CPU: Switched      from: 080      to: 001
0040 CPU: Loaded PID: 081      Arrival: 040      Codesize: 002      PC: 000
0045 CPU: Switched      from: 001      to: 081
0055 CPU: Process is terminated PID:081 PC:001
0060 CPU: Switched      from: 081      to: 001
0217 CPU: Process is terminated PID:001 PC:001
0222 CPU: Switched      from: 001      to: 002
0302 CPU: Process is terminated PID:002 PC:001
PID: 100      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
PID: 081      ARRIVAL: 040      CODESIZE: 002      WAITING: 005      RESPONSE: 005
PID: 080      ARRIVAL: 006      CODESIZE: 002      WAITING: 006      RESPONSE: 006
PID: 002      ARRIVAL: 005      CODESIZE: 002      WAITING: 217      RESPONSE: 217
PID: 001      ARRIVAL: 004      CODESIZE: 002      WAITING: 043      RESPONSE: 023
PID: 000      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
*** TOTAL CLOCKS: 0302 IDLE: 0030 UTIL: 90.07% WAIT: 54.20 RESPONSE: 50.20
```



2-2. SRJF Scheduling



2-2. SRJF Scheduling

- Normal class 스케줄링 기법을 Shortest Remaining Job First로 변경
 - Normal class queue의 프로세스 중,
 - 현재 진행 중인 CPU operation 을 비교하여,
 - 남은 작업 길이 (code length) 가 가장 짧은 프로세스를 선택
 - RT 프로세스에 의해 preemption 된 normal process 도 동일하게 처리
- 과제 2-2 출력
 - Loading, Switching, Termination 정보 출력
 - Termination 시, 각 프로세스 별로 CPU utilization, waiting , response time 출력
 - Waiting time: ready 상태에서 스케줄링을 대기한 total time
 - Response time: Arrival 하고, 처음 스케줄링되기까지 시간
 - Final report: CPU utilization + Avg. waiting time + Avg. Response time

JOTA: 2-2

2021운영체제 과제 2-2

- LMS(구버전) 과제 2 참조

stdin 으로부터 Binary 형태의 프로세스 정보와 코드를 읽어들이고, 아래와 같이 출력하시오.

2-2.

SRJF 스케줄링 구현

출력 형식

CPU clock 진행에 따라,

```
로드된 프로세스 정보: "%04d CPU: Loaded PID: %03d\tArrival: %03d\tCodesize: %03d\tPC: %03d\n"
프로세스 종료: %04d CPU: Process is terminated\tPID:%03d\tPC:%03d\n"
CPU 작업 전환: "%04d CPU: Switched\tfrom: %03d\tto: %03d\n"
모든 작업 종료 후, 각 프로세스별 정보 출력: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"
모든 작업 종료 후, 최종 리포트: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"
```

출력 예 (test2.bin 이용)

```
0000 CPU: Loaded PID: 000   Arrival: 000   Codesize: 002   PC: 000
0000 CPU: Loaded PID: 100   Arrival: 000   Codesize: 002   PC: 000
0002 CPU: Process is terminated PID:000 PC:001
```



JOTA: 2-2 with test2.bin

```
ubuntu@41983:~/hw2$ cat test2.bin | ./os2-2
0000 CPU: Loaded PID: 000      Arrival: 000      Codesize: 002      PC: 000
0000 CPU: Loaded PID: 100      Arrival: 000      Codesize: 002      PC: 000
0002 CPU: Process is terminated PID:000 PC:001
0007 CPU: Switched      from: 000      to: 100
0007 CPU: Loaded PID: 001      Arrival: 004      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 002      Arrival: 005      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 080      Arrival: 006      Codesize: 002      PC: 000
0012 CPU: Switched      from: 100      to: 080
0022 CPU: Process is terminated PID:080 PC:001
0027 CPU: Switched      from: 080      to: 002
0040 CPU: Loaded PID: 081      Arrival: 040      Codesize: 002      PC: 000
0045 CPU: Switched      from: 002      to: 081
0055 CPU: Process is terminated PID:081 PC:001
0060 CPU: Switched      from: 081      to: 002
0127 CPU: Process is terminated PID:002 PC:001
0132 CPU: Switched      from: 002      to: 001
0302 CPU: Process is terminated PID:001 PC:001
PID: 100      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
PID: 081      ARRIVAL: 040      CODESIZE: 002      WAITING: 005      RESPONSE: 005
PID: 080      ARRIVAL: 006      CODESIZE: 002      WAITING: 006      RESPONSE: 006
PID: 002      ARRIVAL: 005      CODESIZE: 002      WAITING: 042      RESPONSE: 022
PID: 001      ARRIVAL: 004      CODESIZE: 002      WAITING: 128      RESPONSE: 128
PID: 000      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
*** TOTAL CLOCKS: 0302 IDLE: 0030 UTIL: 90.07% WAIT: 36.20 RESPONSE: 32.20
```



2-3. Round-Robin Scheduling



Round-Robin scheduling

- Normal class 스케줄링 기법을 Round-Robin scheduling (RR) 로 변경
 - Time-sharing 기반의 대표적 스케줄링 기법
 - 한 라운드가 시작될 때, 정책에 따라 각 프로세스에게 time slice (TS) 부여
 - 모든 프로세스에게 똑같이 50 clocks 씩 배분
 - (다른 예시) Priority 에 따라 다른 time slice를 배분: High=40, Normal=30, Low=20 TQs
 - Linux nice value는 normal class 내의 프로세스들에게 약간의 차등을 부여하기 위해, 이와 유사한 방식으로 동작함. Realtime class로 지정을 하면 훨씬 높은 우선순위를 갖게 됨. 그러나 starvation 은 없음
 - 과제 구현 내용 아님!
 - 라운드 중간에 로드된 프로세스에게도 동일한 TS 부여
 - 모든 normal class 의 프로세스들이 각자의 TS를 모두 소진하면 해당 라운드 종료
- 라운드 내에서는 각 프로세스들에 대해 FIFO 스케줄링 기법을 사용
 - 한 라운드 내에 Time slice 가 남은 프로세스들 사이에서 선택하는 방법이 필요함
 - Preemption 된 경우, running process를 ready queue의 맨 앞에 삽입
 - (다른 예시) 구현 내용 아님.
 - SRJF: 남은 프로세스들 사이에서 remaining time slice 가 가장 적은 프로세스를 선택
 - Priority-based: 우선 순위대로 스케줄링

JOTA: 2-3

2021운영체제 과제 2-3

- LMS(구버전) 과제 2 참조

stdin 으로부터 Binary 형태의 프로세스 정보와 코드를 읽어들이고, 아래와 같이 출력하시오.

2-3.

라운드 로빈 스케줄링 구현

출력 형식

CPU clock 진행에 따라,

로드된 프로세스 정보: "%04d CPU: Loaded PID: %03d\tArrival: %03d\tCodesize: %03d\tPC: %03d\n"

프로세스 종료: "%04d CPU: Process is terminated\tPID:%03d\tPC:%03d\n"

CPU 작업 전환: "%04d CPU: Switched\tfrom: %03d\tto: %03d\n"

CPU 작업 전환이 안되는 경우: "%04d CPU: Not Switched\tPID: %03d\n"

라운드가 바뀌는 경우: "%04d CPU: ROUND ENDS. Recharge the Timeslices\n"

모든 작업 종료 후, 각 프로세스별 정보 출력: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"

모든 작업 종료 후, 최종 리포트: "PID: %03d\tARRIVAL: %03d\tCODESIZE: %03d\tWAITING: %03d\tRESPONSE: %03d\n"



JOTA: 2-3 with test2.bin

```
ubuntu@41983:~/hw2$ cat test2.bin | ./os2-3
0000 CPU: Loaded PID: 000      Arrival: 000      Codesize: 002      PC: 000
0000 CPU: Loaded PID: 100      Arrival: 000      Codesize: 002      PC: 000
0002 CPU: Process is terminated PID:000 PC:001
0007 CPU: Switched      from: 000      to: 100
0007 CPU: Loaded PID: 001      Arrival: 004      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 002      Arrival: 005      Codesize: 002      PC: 000
0007 CPU: Loaded PID: 080      Arrival: 006      Codesize: 002      PC: 000
0012 CPU: Switched      from: 100      to: 080
0022 CPU: Process is terminated PID:080 PC:001
0027 CPU: Switched      from: 080      to: 001
0040 CPU: Loaded PID: 081      Arrival: 040      Codesize: 002      PC: 000
0045 CPU: Switched      from: 001      to: 081
0055 CPU: Process is terminated PID:081 PC:001
0060 CPU: Switched      from: 081      to: 001
0102 CPU: Switched      from: 001      to: 002
0152 CPU: ROUND ENDS. Recharge the Timeslices
0157 CPU: Switched      from: 002      to: 001
0212 CPU: Switched      from: 001      to: 002
0242 CPU: Process is terminated PID:002 PC:001
0242 CPU: ROUND ENDS. Recharge the Timeslices
0247 CPU: Switched      from: 002      to: 001
0297 CPU: ROUND ENDS. Recharge the Timeslices
0297 CPU: Not Switched  PID: 001
0317 CPU: Process is terminated PID:001 PC:001
PID: 100      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
PID: 081      ARRIVAL: 040      CODESIZE: 002      WAITING: 005      RESPONSE: 005
PID: 080      ARRIVAL: 006      CODESIZE: 002      WAITING: 006      RESPONSE: 006
PID: 002      ARRIVAL: 005      CODESIZE: 002      WAITING: 157      RESPONSE: 097
PID: 001      ARRIVAL: 004      CODESIZE: 002      WAITING: 143      RESPONSE: 023
PID: 000      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
*** TOTAL CLOCKS: 0317 IDLE: 0045 UTIL: 85.80% WAIT: 62.20 RESPONSE: 26.20
```



2-3 Detailed Result (1/2)

```
ubuntu@41983:~/hw2$ cat test2.bin | ./os2-3-dbg
```

```
0 0 2
```

```
0 2
```

```
1 4 2
```

```
0 170
```

```
2 5 2
```

```
0 80
```

```
80 6 2
```

```
0 10
```

```
81 40 2
```

```
0 10
```

```
100 0 2
```

```
255 0
```

```
Start Processing. loaded_procs = 5
```

```
0000 CPU: Loaded PID: 000      Arrival: 000      Codesize: 002      PC: 000
```

```
0000 CPU: Loaded PID: 100      Arrival: 000      Codesize: 002      PC: 000
```

```
0000 CPU: OP_CPU START len: 002 ends at: 0002
```

```
0002 CPU: Increase PC      PID:000 PC:000
```

```
0002 CPU: Process is terminated PID:000 PC:001
```

```
0002 CPU: Reschedule      PID: 000      Timeslice: 00      Status: 04
```

```
0007 CPU: Switched      from: 000      to: 100
```

```
0007 CPU: Loaded PID: 001      Arrival: 004      Codesize: 002      PC: 000
```

```
0007 CPU: Loaded PID: 002      Arrival: 005      Codesize: 002      PC: 000
```

```
0007 CPU: Loaded PID: 080      Arrival: 006      Codesize: 002      PC: 000
```

```
0007 CPU: Reschedule      PID: 100      Timeslice: 00      Status: 01
```

```
0012 CPU: Switched      from: 100      to: 080
```

```
0012 CPU: OP_CPU START len: 010 ends at: 0022
```

```
0022 CPU: Increase PC      PID:080 PC:000
```

```
0022 CPU: Process is terminated PID:080 PC:001
```

```
0022 CPU: Reschedule      PID: 080      Timeslice: 00      Status: 04
```

```
0027 CPU: Switched      from: 080      to: 001
```

```
0027 CPU: OP_CPU START len: 170 ends at: 0197
```

```
0040 CPU: Loaded PID: 081      Arrival: 040      Codesize: 002      PC: 000
```

```
0040 CPU: Reschedule      PID: 001      Timeslice: 37      Status: 02
```

```
0045 CPU: Switched      from: 001      to: 081
```

```
0055 CPU: Increase PC      PID:081 PC:000
```

```
0055 CPU: Process is terminated PID:081 PC:001
```

```
0055 CPU: Reschedule      PID: 081      Timeslice: 00      Status: 04
```

4

27-40 : 23

60-97 : 20

157-207 : 60

247-317: 40



2-3 Detailed Result (2/2)

```
0060 CPU: Switched      from: 081      to: 001
0060 CPU: OP_CPU START len: 157 ends at: 0217
0097 CPU: RR times up   PID: 001 Timeslice: 000
0097 CPU: Reschedule   PID: 001      Timeslice: 00   Status: 02
0102 CPU: Switched      from: 001      to: 002
0152 CPU: RR times up   PID: 002 Timeslice: 000
0152 CPU: Reschedule   PID: 002      Timeslice: 00   Status: 02
0152 CPU: ROUND ENDS. Recharge the Timeslices
0157 CPU: Switched      from: 002      to: 001
0207 CPU: RR times up   PID: 001 Timeslice: 000
0207 CPU: Reschedule   PID: 001      Timeslice: 00   Status: 02
0212 CPU: Switched      from: 001      to: 002
0242 CPU: Increase PC   PID:002 PC:000
0242 CPU: Process is terminated PID:002 PC:001
0242 CPU: Reschedule   PID: 002      Timeslice: 00   Status: 04
0242 CPU: ROUND ENDS. Recharge the Timeslices
0247 CPU: Switched      from: 002      to: 001
0247 CPU: OP_CPU START len: 070 ends at: 0317
0297 CPU: RR times up   PID: 001 Timeslice: 000
0297 CPU: Reschedule   PID: 001      Timeslice: 00   Status: 02
0297 CPU: ROUND ENDS. Recharge the Timeslices
0297 CPU: Not Switched PID: 001
0317 CPU: Increase PC   PID:001 PC:000
0317 CPU: Process is terminated PID:001 PC:001
PID: 100      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
PID: 081      ARRIVAL: 040      CODESIZE: 002      WAITING: 005      RESPONSE: 005
PID: 080      ARRIVAL: 006      CODESIZE: 002      WAITING: 006      RESPONSE: 006
PID: 002      ARRIVAL: 005      CODESIZE: 002      WAITING: 157      RESPONSE: 097
PID: 001      ARRIVAL: 004      CODESIZE: 002      WAITING: 143      RESPONSE: 023
PID: 000      ARRIVAL: 000      CODESIZE: 002      WAITING: 000      RESPONSE: 000
*** TOTAL CLOCKS: 0317 IDLE: 0045 UTIL: 85.80% WAIT: 62.20 RESPONSE: 26.20
```

4
27-40 : 23
60-97 : 20
157-207 : 60
247-317: 40

2-4. Evaluation



스케줄링 성능 분석 및 평가: FIFO, SRJF, Round-Robin

- Response time, waiting time, CPU utilization 등 다양한 측면에서 여러 스케줄링 기법을 평가하고 비교, 분석할 것
 - 다양한 상황에 대해 실험하고, 결과를 비교
 - os-gen-cpu.c 를 활용하여 원하는 프로세스들을 생성하거나,
 - 시뮬레이터 코드 내에서 직접 프로세스 정보를 설정해서 사용하여도 되고,
 - hex editor 를 사용하여 직접 바이너리 파일을 만들어도 됨
 - Real-time class 프로세스는 없다고 가정 -> 분석을 단순하게 하기 위함
- 양식 및 방식 등 모든 다른 사항은 자유
- JOTA 제출 없음
- 결과를 도출하는데 사용한 소스 코드 (os2-4.c) 및 기타 데이터 제출
 - os2-4.c 는 FIFO, SRJF, RR 을 모두 구현되어있는 os2-3.c 를 기반으로 하고,
 - 원하는 스케줄러를 선택하여 수행하고
 - 성능 평가에 필요한 수치들을 출력할 수 있도록 작성