

과제 #4 : Scheduling

○ 과제 목표

- 스케줄링 이해
- SSU OS에서 Multi-level Feedback Queue 스케줄링 구현
 - ✓ 각 프로세스가 속한 큐의 우선 순위(level)에 기반하여 프로세스를 스케줄링

○ 기본 배경 지식

- 스케줄링
 - ✓ 스케줄링은 다중 프로그래밍을 가능하게 하는 운영체제의 동작 기법
 - ✓ 운영체제는 프로세스들에게 CPU 등의 자원 배정을 적절히 함으로써 시스템의 성능을 개선할 수 있음
 - ✓ Non-preemptive Scheduling
 - 어떤 프로세스가 CPU를 할당 받으면 그 프로세스가 종료되거나 자발적으로 중지될 때까지 계속 실행되도록 보장
 - 일괄 처리 시스템에 적합하며, CPU 사용 시간이 긴 프로세스가 CPU 사용 시간이 짧은 여러 프로세스를 오랫동안 대기시킬 수 있으므로, 처리율이 떨어질 수 있다는 단점이 있음
 - ✓ Preemptive Scheduling
 - 어떤 프로세스가 CPU를 할당받아 실행 중에 특정한 이벤트(Time quantum, I/O 등)로 프로세스의 실행을 중지시키고 CPU를 다른 프로세스에게 할당
 - 빠른 응답시간을 요하는 대화식 시분할 시스템에 적합하며 긴급한 프로세서에게 CPU를 할당할 수 있음
- Multi Level Feedback Queue 스케줄러
 - ✓ 여러 개의 준비 큐를 둘 수 있으며, 준비 큐마다 다른 CPU 할당 시간을 부여하며 시간 안에 완료되지 못한 프로세스들은 다음 단계의 큐에 들어감
 - ✓ 상위 단계의 큐일수록 우선순위가 높고 시간할당량이 적음
 - ✓ 작업 시간이 짧은 프로세스(입출력 중심의 프로세스(I/O bound job) 등)가 우선순위가 높으며 오래 기다린 프로세스에게도 긴 CPU 할당 시간을 부여 할 수 있음

○ 과제 내용

- 본 과제에서는 최하위 단계의 큐를 제외한 모든 큐에서는 FCFS(First Come, First Served) 스케줄링 기법을 사용하고, 최하위 단계의 큐에서는 라운드 로빈(RR) 스케줄링 기법을 사용
- 주어진 코드는 FIFO스케줄러로 구현되어 있음
 - ✓ 내부 자료구조를 이해하고 응용하여 SSU OS Multi Level Feedback Queue 스케줄러를 구현 할 것
- 0번 프로세스(idle process)는 schedule() 함수만 호출하도록 구현
- 프로세스는 0~10번의 pid 순서대로 생성되며 생성된 순서대로 레디 큐에 진입함
- schedule()가 호출되어 0번 프로세스가 선택되면 sched_find_que()를 통해 다음번에 실행할 프로세스를 선택하고 schedule() -> switch_process()를 호출
- schedule()가 호출되어 0번이 아닌 프로세스가 선택되면 switch_process()를 호출

○ 과제 수행 내용

- (0) 스케줄링에 사용되는 자료구조 및 변수
 - ✓ struct process 구조체 : 멤버 변수에 스케줄링을 위한 적절한 값 할당
 - time_slice : 프로세스가 스케줄링 된 이후 다시 스케줄링 되기 전까지 CPU를 사용한 시간(tick)으로 time quantum을 의미하는 unsigned int형 변수
 - time_used : 프로세스가 생성된 이후 CPU를 사용한 총합 시간(tick)으로 unsigned long long형 변수
 - time_sleep : I/O 처리(proc_sleep() 호출) 시, 프로세스가 깨어날 시간(tick)으로 unsigned long long형 변수
 - que_level : int형 변수로 0,1,2 중 하나의 값을 가질 수 있음
- (1) schedule() 수정
 - ✓ schedule()에서 0번 프로세스만이 다른 프로세스들을 스케줄링 해야 하며 , 0번이 아닌 프로세스는 0번 프로세스로 스케줄링 되어야 함
 - ✓ Context Switching 도중에는 다른 인터럽트가 발생하더라도 인터럽트 핸들러가 수행되지 않도록 해야 함
 - ✓ 출력 형식은 아래 과제수행 결과와 동일한 필요는 없음
- (2) sched_find_que() 구현
 - ✓ 항상 레벨1 레디 큐 리스트(주어진 코드에서 전역 변수로 level_que[1]로 선언)가 비어있는지 먼저 탐색하고 레벨2 레디 큐 리스트(주어진 코드에서 전역 변수로 level_que[2]로 선언)를 탐색해야 함. 참고로 레벨0 레디 큐 리스트는 주어진 코드에서 전역 변수로 level_que[0]로 선언되어 있음
 - ✓ 레벨1 레디 큐 리스트가 비어 있지 않으면 그 리스트를 get_next_proc()의 인자로 호출
 - ✓ 레벨1 레디 큐 리스트가 비어 있으면 레벨2 레디 큐 리스트가 비어있는지 탐색하고 레벨2 레디 큐 리스트가 비어 있지 않으면 그 리스트를 get_next_proc()의 인자로 호출하고 레벨2 레디 큐 리스트가 비어있으면 다시 레벨1 레디 큐 리스트 탐색
- (3) proc_create() 일부 구현
 - 0번 프로세스의 경우 init_proc()을 통해서 레벨0 레디 큐 리스트에 들어감
 - 0번이 아닌 프로세스가 생성되면 레벨1 레디 큐 리스트에 들어감
 - 모든 process는 생성될 때 time_used, time_slice는 항상 0으로 세팅됨
 - 각 레디 큐를 리스트 구조체로 구현
 - list.h, list.c를 참고
- (4) 실행되는 프로세스 상황 출력 구현
 - ✓ kernel1_proc(), kernel2_proc(), kernel3_proc(), kernel4_proc(), kernel5_proc(), kernel6_proc(), kernel7_proc(), kernel8_proc(), kernel9_proc(), kernel10_proc()은 이미 구현되어 있음
 - ✓ 0번 프로세스가 다음 수행으로 선택한 프로세스에 대한 pid를 출력 (과제 수행 결과의 실행 예시에서 'Selected process = #num' 부분)
 - ✓ 프로세스가 하위 레벨 큐로 들어가는 경우 관련 정보 출력
 - ✓ 각 스케줄링마다 종료 되지 않은 모든 프로세스들에 대한 정보는 "#= (pid), t= (time_slice) , u= (used_time), q = (que_level)" 형식으로 출력
 - ✓ I/O시 해당 프로세스의 Pid 와 I/O 시작 시각(tick) 출력
 - ✓ 출력 양식은 아래 과제 수행 결과와 동일한 구조여야 함

- (5) 보고서에 기재되어야 할 내용

- ✓ 과제 수행 내용(4)에서 수행한 10개의 프로세스들의 총 수행 시간과 I/O 시점 및 시간을 파악하여 수행될 시뮬레이션 결과를 아래 표 형식으로 표현. 프로그램 구현 후 실행 결과를 쓰는 것이 아니라 수작업으로 계산한 결과를 쓰는 것이므로 실행 결과와 실행 tick의 차이가 있을 수 있음

- PID

- 프로세스 시작/종료 시각(tick)

- I/O 제외 총 수행시간(tick)

- I/O 시작 시각(tick)

- 아래 스케줄링 결과 시뮬레이션 예시는 임의의 예시에 대한 것으로 본 과제의 정답 중 일부가 아님

PID (순서)	시작시각 (tick)	종료시각 (tick)	I/O 제외 총 수행시간 (tick)	I/O 시작 시각 (tick)
1	0	1200	200	80
2	40	1000	120	70
3	70	3000	300	1000, 2700
...

<스케줄링 결과 시뮬레이션>

```

Bochs x86 emulator, http://bochs.sourceforge.net/
===== initialization complete =====
create done
# = 1 t= 0 u= 0 q= 1, # = 2 t= 0 u= 0 q= 1, # = 3 t= 0 u= 0 q= 1, # = 4 t=
0 u= 0 q= 1, # = 5 t= 0 u= 0 q= 1, # = 6 t= 0 u= 0 q= 1, # = 7 t= 0 u= 0
q= 1, # = 8 t= 0 u= 0 q= 1, # = 9 t= 0 u= 0 q= 1, # = 10 t= 0 u= 0 q= 1
Selected process : 1
proc1 change the queue(1->2)
# = 1 t= 40 u= 40 q= 2, # = 2 t= 0 u= 0 q= 1, # = 3 t= 0 u= 0 q= 1, # = 4 t=
0 u= 0 q= 1, # = 5 t= 0 u= 0 q= 1, # = 6 t= 0 u= 0 q= 1, # = 7 t= 0 u= 0
q= 1, # = 8 t= 0 u= 0 q= 1, # = 9 t= 0 u= 0 q= 1, # = 10 t= 0 u= 0 q= 1
Selected process : 2
Proc 2 I/O at 30
proc 2 's que is 1
proc2 change the queue (2->1)
# = 1 t= 40 u= 40 q= 2, # = 2 t= 0 u= 30 q= 1, # = 3 t= 0 u= 0 q= 1, # = 4 t=
0 u= 0 q= 1, # = 5 t= 0 u= 0 q= 1, # = 6 t= 0 u= 0 q= 1, # = 7 t= 0 u= 0
q= 1, # = 8 t= 0 u= 0 q= 1, # = 9 t= 0 u= 0 q= 1, # = 10 t= 0 u= 0 q= 1
Selected process : 3
  
```

<수행 결과>

(# = pid, t = 스케줄 된 이후 CPU 사용시간, u = 프로세스의 CPU 총 사용시간)

○ 구현 시 주의할 점

- printf()의 인자가 4개 이상일 경우 간혹 에러 발생 가능하기 때문에 여러 번의 printf()를 사용할 것

○ 과제제출

- 2018년 10월 14일 (일) 23시 59분까지 제출
- 배점 기준
 - ✓ 보고서 15%
 - 개요 2%
 - 상세 설계 명세(기능 명세 포함) 10%
 - 실행 결과 3%
 - ✓ 소스코드 85%
 - 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
 - 실행 여부 80%
- 최소 구현사항
 - ✓ 과제 수행 내용 (1), (2), (3), (4), (5)