서술형 과제 **(2**주차**)**

최규형

1. 다음과 같이 스케쥴링 정책이 적용되어 있는 **General Purpose** 운영체제가 있습니다**.** 다음 정책이 실제 어떻게 동작할지를 가능한 상세히 기술해주세요**.**

- Round Robin 스케쥴링 정책
  - 100ms 마다 다음 프로세스로 교체
  - Ready Queue, Running Queue, Block Queue 존재
  - 각 Queue 는 FIFO 정책으로 동작함
- 인터럽트로 선점 가능 (선점형 스케쥴링 기능 지원)
- 타이머 인터럽트를 지원하며, 타이머 칩에서는 1ms 마다 인터럽트를 발생시킴
- CPU만 실행하는 A, B, C 프로세스가 Ready Queue 에 기술한 순서대로 들어간 상태임
  - 각 프로세스가 총 실행해야 하는 시간은 다음과 같음
  - A 는 200ms, B 는 500ms, C 는 300ms

➡ 위에서 기술한 사항 외에 추가 가정이 꼭 필요한 경우에는 각자 가정한 상황을 기술하고, 동작 방식을 기술하세요.

Round Robin, 작업단위 = 100ms
Ready-Running-Block Queue (All FIFO)
*프로세스 **B**의 **3**번째 작업단위 실행 도중 **(50ms)** 시스템 콜 인터럽트 명령을 **CPU**가 실행한다고 가정

1. Ready Queue의 첫번째 원소 Process A -> Running State Queue 전환
   a. Process A 1작업단위(100ms)만큼 실행
   b. 1작업단위(100ms)에 해당하는 프로세스A가 Ready Queue에 세번째 원소로 생성
   c. A 잔여 작업량 = 100ms
2. Ready Queue의 첫번째 원소인 Process B -> Running State Queue 전환
   a. Process B 1작업단위(100ms)만큼 실행
   b. 1작업단위(100ms)에 해당하는 프로세스B가 Ready Queue에 세번째 원소로 생성
   c. B 잔여 작업량 = 400ms
3. Ready Queue의 첫번째 원소인 Process C -> Running State Queue 전환
   a. Process C 1작업단위(100ms)만큼 실행
   b. 1작업단위(100ms)에 해당하는 프로세스C가 Ready Queue에 세번째 원소로 생성
   c. C 잔여 작업량 = 200ms
4. Ready Queue의 첫번째 원소인 Process A -> Running State Queue 전환
   a. Process A 1작업단위(100ms)만큼 실행
   b. Process A는 모두 실행되었으므로 **Ready Queue**에 더 이상 생성되지 않음
   c. A 잔여 작업량 **= 0ms**
5. Ready Queue의 첫번째 원소인 Process B -> Running State Queue 전환
   a. Process B 1작업단위(100ms)만큼 실행

    b. 1작업단위(100ms)에 해당하는 프로세스B가 Ready Queue에 2번째 원소로 생성
    c. B 잔여 작업량 = 300ms
6. Ready Queue 첫번째 원소인 Process C -> Running State Queue 전환
    a. Process C, 1작업단위(100ms) 만큼 실행
    b. 1작업단위(100ms)에 해당하는 프로세스C가 Ready Queue에 2번째 원소로 생성
    c. C 잔여 작업량 = 100ms
7. Ready Queue의 첫번째 원소인 Process B > Running State Queue 전환
    a. Process B 0.5작업단위(50ms)만큼 실행
    b. 시스템 콜 인터럽트
        i. Process B 실행 중단
            1. (Process B) Running State Queue -> Ready State Queue
        ii. eax 레지스터에 시스템 콜 번호 input
        iii. ebx 레지스터에 시스템 콜에 해당하는 인자값 input
        iv. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
        v. CPU는 사용자 모드에서 커널 모드로 전환
        vi. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
            1. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
            2. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
        vii. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
        viii. 해당 시스템 콜 함수를 실행
        ix. CPU는 커널 모드에서 사용자 모드로 복귀
        x. Process B 재실행
            1. (Process B) Ready State Queue -> Running State Queue
    c. Process B 0.5작업단위(50ms) 만큼 실행
    d. 1작업단위(100ms)에 해당하는 프로세스B가 Ready Queue에 2번째 원소로 생성
    e. B 잔여 작업량 = 200ms
8. Ready Queue 첫번째 원소인 Process C > Running State Queue 전환
    a. Process C, 1작업단위(100ms) 만큼 실행
    b. Process C는 모두 실행되었으므로 **Ready Queue**에 더 이상 생성되지 않음
    c. C 잔여 작업량 **= 0ms**
9. Ready Queue의 첫번째 원소인 Process B -> Running State Queue 전환
    a. Process B 1작업단위(100ms)만큼 실행
    b. 1작업단위(100ms)에 해당하는 프로세스B가 Ready Queue에 1번째 원소로 생성
    c. B 잔여 작업량 = 100ms
10. Ready Queue의 첫번째 원소인 Process B -> Running State Queue 전환
    a. Process B 1작업단위(100ms)만큼 실행
    b. Process B는 모두 실행되었으므로 **Ready Queue**에 더 이상 생성되지 않음
    c. B 잔여 작업량 **= 0ms**

2. 다음과 같이 메모리 정책이 적용되어 있는 **General Purpose** 운영체제가 있습니다**.** 다음 정책이 실제 내부적으로 어떻게 동작할지를 가능한 상세히 기술해주세요**.**

- 페이징 시스템을 기반으로 한 가상 메모리 시스템 지원
- 요구 페이징을 지원하며, 하드웨어에서는 MMU와 TLB 칩을 지원함

➡ 위에서 기술한 상황에서 특정 프로세스가 실행 후, CPU 상에서 가상 주소로 메모리에서 데이터를 가져와야 할때, 이 동작이 어떻게 진행될지를 가능한 상세히 기술해주세요

1. 프로세스 생성 시 페이지 테이블 정보가 생성됨
    a. 물리 메모리에 테이블 정보가 올라감 (적재)
2. PCB는 해당 페이지 테이블의 base address를 갖고 있으며, 이를 통해 테이블 접근 가능
3. 프로세스 구동 시, 해당 페이지 테이블의 base address가 CR3 레지스터에 저장
4. CPU가 가상 주소에 접근을 시도하면
    a. MMU 하드웨어 장치가 페이지 테이블 base address에 접근
        i. 페이지 테이블에는 가상 주소와 물리 주소 간 매핑 정보가 담겨 있음
        ii. 가상 주소 v = (p, d)
            1. p = 가상 메모리 페이지 번호
            2. d = p 안에서 참조하는 위치 = 변위(offset)
    b. 가상 주소를 물리 주소로 변환
        i. (1) CPU가 접근 시도하는 가상 주소가 포함된 page number가 프로세스의 페이지 테이블에 존재하는지 확인
        ii. (2) page number가 존재하면, 해당 페이지가 매핑된 첫번째 물리 주소 p'를 찾아냄
        iii. (3) p' + d 연산을 통하여 실제 물리 주소값을 획득
    c. 획득한 물리 주소를 CPU에 전달

3. 다음과 같은 운영체제가 설치된 컴퓨터가 있을 때**,** 사용자가 컴퓨터를 켰을 때부터**,** 운영체제가 프로세스를 실행하고 쉘을 실행할 때까지 어떻게 동작할지를 가능한 상세히 기술해주세요**.**

- 컴퓨터는 BIOS를 지원하고, 부팅을 지원하는 bootstrap loader 가 별도로 설치되어 있음
- 운영체제 커널은 실행후, 최초 프로세스(init)를 운영체제 코드상에서 바로 실행시키며,
- 이후에는 fork() 기능을 지원하며, 쉘 프로그램을 실행시킴
- 쉘 프로그램을 통해 사용자는 각 프로그램을 실행시킬 수 있음

➡ 위에서 기술한 사항 외에 추가 가정이 꼭 필요한 경우에는 각자 가정한 상황을 기술하고, 동작 방식을 기술하세요.

1. 사용자가 컴퓨터 전원을 킴
2. BIOS가 컴퓨터 하드웨어를 초기화
3. BIOS가 저장매체 맨 앞단의 특정 size에 해당하는 부분(=MBR)으로부터 부트 로더 프로그램을 메모리에 불러온다
    a. 이 과정에서 MBR의 파티션 테이블로부터 어떤 파티션이 운영체제가 있는 파티션(=메인 파티션)인지를 식별, 확인
4. 부트 로더 프로그램 실행
    a. 메인 파티션의 부트섹터에 접근하여 부트 코드를 불러와 메모리에 불러온다
5. 부트 코드 실행
    a. 메인 파티션의 운영체제 커널 이미지(운영체제 실행파일)의 주소를 알아낸다
    b. 주소값으로부터 운영체제 커널 이미지(운영체제 실행파일)를 메모리에 불러온다
    c. 운영체제 커널 이미지의 가장 앞 부분으로 PC(Program Counter)를 옮긴다
6. 운영체제 커널 이미지 실행
    a. 최초 프로세스(init)을 운영체제 코드상에서 즉시 실행
    b. 쉘 프로그램 실행
        i. 새로운 사용자 요청이 올 때마다 fork() 시스템콜 호출
        ii. fork() 함수를 통해 기존(부모) 프로세스로부터 새로운(자식) 프로세스 생성 -> 각 사용자 요청에 즉시 대응
        iii. 여러 프로세스 간 통신을 위한 IPC(Inter Process Communication) 기법 적용

4. 다음과 같은 **General Purpose** 운영체제에서**,** 사용자가 커멘드창을 오픈한 후**, ls (dir)** 커멘드를 키보드로 작성할 때**,** 실제 내부적으로 어떻게 동작하는지 가능한 상세히 기술해주세요

- 사용자가 커멘드창을 오픈하면, 자동으로 쉘프로그램이 실행됨
- ls 명령은 윈도우의 dir 과 마찬가지로 해당 디렉토리의 정보를 보여주는 프로그램임
- 해당 쉘에서는 키보드를 입력하면, 해당 키보드 문자가 화면에 표시하며, 엔터를 누르면 그동안 입력받은 문자열을 명령으로 인식하고, 해당 명령을 실행함
- 스케쥴링 방식은 선점형을 지원하며, 기본적으로 멀티 태스킹을 지원함
- 인터럽트를 지원하며, 키보드 인터럽트와 타이머 인터럽트를 지원함
- 타이머 인터럽트는 1ms 마다 발생함
- 사용자는 키보드로 l 과 s 그리고 엔터키를 누를 때 각기 인터럽트가 발생함
- ls 프로그램이 실행되면, 내부적으로 필요시 시스템콜을 호출할 수 있음

➡ 위에서 기술한 사항 외에 추가 가정이 꼭 필요한 경우에는 각자 가정한 상황을 기술하고, 동작 방식을 기술하세요.

1. 사용자 커맨드창 오픈
2. 사용자 키보드 l 입력
    a. 키보드(하드웨어) 인터럽트 발생
    b. eax 레지스터에 시스템 콜 번호 input

    c. ebx 레지스터에 시스템 콜에 해당하는 인자값 input
    d. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
    e. CPU는 사용자 모드에서 커널 모드로 전환
    f. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
        i. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
        ii. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
    g. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
    h. 해당 시스템 콜 함수를 실행
    i. CPU는 커널 모드에서 사용자 모드로 복귀
    j. 해당 키보드 문자를 화면에 표시
3. 사용자 키보드 s 입력
    a. 키보드(하드웨어) 인터럽트 발생
    b. eax 레지스터에 시스템 콜 번호 input
    c. ebx 레지스터에 시스템 콜에 해당하는 인자값 input
    d. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
    e. CPU는 사용자 모드에서 커널 모드로 전환
    f. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
        i. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
        ii. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
    g. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
    h. 해당 시스템 콜 함수를 실행
    i. CPU는 커널 모드에서 사용자 모드로 복귀
    j. 해당 키보드 문자를 화면에 표시
4. 사용자 키보드 enter 입력
    a. 키보드(하드웨어) 인터럽트 발생
    b. eax 레지스터에 시스템 콜 번호 input
    c. ebx 레지스터에 시스템 콜에 해당하는 인자값 input
    d. 소프트웨어 인터럽트 명령 호출하면서 CPU opcode 0x80(인터럽트 번호) 넘겨줌
    e. CPU는 사용자 모드에서 커널 모드로 전환
    f. IDT(Interrupt Descriptor Table)에서 0x80에 해당하는 주소(함수)를 찾아 실행
        i. 컴퓨터 부팅시 운영체제가 인터럽트 각각의 번호와 실행 코드를 가리키는 주소를 IDT에 기록
        ii. 운영체제 내부 코드이므로 현재 사용자 모드라면 커널 모드로 전환 필요함
    g. system_call() 함수의 eax로부터 시스템 콜 번호를 찾아, 해당 번호에 맞는 시스템 콜 함수로 이동
    h. 해당 시스템 콜 함수를 실행
    i. CPU는 커널 모드에서 사용자 모드로 복귀
    j. 현재까지 입력된 ls 문자열을 명령으로 인식

k. 현재 디렉토리의 정보를 보여주는 함수를 실행