Week 2 과제 답안

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

(답안)

① A, B, C 프로세스는 모두 CPU에서만 실행하므로 저장 매체에 접근하기 위해 Blocked가 되는 경우는 없다. (Block Queue에 들어가는 경우는 없다.)
② 프로세스 간 우선순위는 동일, FIFO 정책만을 따른다.
③ 타이머는 1ms마다 인터럽트를 발생하고, 100ms 마다 스케쥴러에 의해 실행할 프로세스가 교체된다.
④ 실행은 최대 1개 프로세스로 한다.

시간 (ms)	Ready Queue	Running	Block Queue
0	B, C	A	-
100	C, A	B	-
200	A, B	C	-
300	B, C	A	-
400	C	B	-
500	B	C	-
600	C	B	-
700	B	C	-
800	-	B	-
900	-	B	-
1000	-	-	-

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

(답안)

우선 CPU 가 가상 메모리 주소를 MMU 에 요청하면 MMU 는 TLB 에 최근에 프로세스에서 그 가상 메모리 주소와 물리 메모리 주소를 매핑한 적이 있는지 찾아보고 있다면 그 물리 메모리 주소로 변환하고, 없다면 CR3 레지스터에 있는 값을 이용해서 그 프로세스의 page table 에 있는 값과 매핑을 시켜서 그 물리 메모리 주소로 변환한다. 이 때 만약 요구 페이징 시스템에 의해서 아직 물리 메모리 주소와 매핑이 되지 않았다면 (valid-invalid 값 확인해서 i 라면), 페이지 폴트 인터럽트가 일어나게 되고 그때가 돼서야 필요한 페이지가 물리 메모리에 올려지고 page table 이 업데이트가 된다. 그리고 더 이상 필요하지 않은 페이지는 다시 저장매체에 저장한다.

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

(답안)

컴퓨터의 전원버튼을 누르면 메인보드에 전력이 들어오며, 메인보드에 부착된 장치들에게 전력이 공급된다. 이후 CPU가 ROM(Read-Only Memory)에 저장된 펌웨어(Firmware)인 BIOS(Basic Input/Output System)코드를 실행시킨다. 실행된 BIOS는 POST(Power On Self Test)라는 주변 하드웨어 체크한다. 부팅매체(저장 매체 중 커널 이미지가 저장된 것)를 선택하고 부팅매체의 MBR에 저장된 부팅정보를 읽어오는 부트스트랩(Bootstrap)을 실행한다. 부트스트랩(Bootstrap)과정으로 RAM에 부트로더(Bootloader)가 올라가고, 부트로더는 디스크에 있는 OS(커널) 코드를 복사해 메모리에 붙여서 OS를 실행한다. 이후 최초 프로세스(init)이 실행되고 fork를 통해 쉘 프로그램이 실행된다.

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

(답안)

쉘 입력 동작 관련

① 쉘 프로그램 커맨드 창에 사용자가 키보드 입력

② 'l' , 's' 각각 입력마다 키보드 인터럽트 발생하며 Shell 커맨드 창에 커서 깜박임 및 입력 대기 프로세스는 동작을 중지하고 커서 위치에 해당 문자 입력해주고 다시 입력 대기 상태(커서 깜박임)로 바뀐다.

③ 'enter키' 입력 시 키보드 인터럽트 발생하며 커맨드 창에 입력대기 상태는 동작을 중지하고 Shell에서

커맨드창에 현재까지 입력된 'ls' 명령어를 수집

④ Shell 프로세스에서 ls동작을 위한 프로세스를 생성 후 ls 명령어 API 기능을 동작

ls 명령어 동작 관련

① ls를 위한 프로세스가 getcwd() 시스템 콜을 호출

② getcwd() 시스템콜은 현재 디렉토리 정보를 dentry로 부터 로드

③ ls를 위한 프로세스가 opendir(cwd) 시스템 콜을 호출

④ opendir(cwd) 시스템콜은 dentry로 부터 불러온 현재 디렉토리 정보를 read 하기 위해 파일 정보를 스캔 후 이 정보를 dir 변수에 저장

⑤ ls를 위한 프로세스가 readdir(dir) 시스템 콜을 호출

⑥ ls를 위한 프로세스가 현재 연결된 디렉토리에 있는 모든 파일 정보를 로드

⑦ Shell 창에 파일명을 print 하고 프로세스 종료