**운영체제(Operating System) - 과제 #2**

**202195039**

**임서윤**

(1) 교재 그림 3.11의 프로세스 상태 모델에서 각 상태와 상태 간의 천이를 유발하는 이벤 트에 대해 자세히 설명하시오.

(2) 어떤 프로세스를 선점(preemption)한다는 것의 의미는 무엇인가?

프로세스를 \*\*선점(preemption)\*\*한다는 것은 운영체제에서 하나의 프로세스가 CPU를 사용하고 있는 중에, 다른 프로세스가 CPU를 차지할 수 있도록 현재 실행 중인 프로세스의 실행을 중단시키고, 새로운 프로세스가 CPU를 사용하도록 만드는 과정을 의미

(3) 프로세스 스와핑(process swapping)이란 무엇이며, 그 목적은 무엇인가?

프로세스 스와핑(process swapping)\*\*은 운영체제에서 **메모리 관리 기법** 중 하나로, 실행 중인 프로세스를 메인 메모리(주 메모리)에서 \*\*보조 기억 장치(주로 디스크)\*\*로 일시적으로 이동시키는 과정입니다. 스와핑의 목적은 물리 메모리의 용량이 부족할 때 여러 프로세스가 실행되도록 하는 것입니다.

 **메모리 자원 관리**: 물리 메모리가 부족할 때, 비활성 상태거나 대기 중인 프로세스를 보조 기억 장치로 이동시켜 활성 프로세스에게 더 많은 메모리를 할당할 수 있습니다.

 **다중 프로세스 실행 가능**: 동시에 많은 프로세스가 실행될 수 있도록 보조 기억 장치와 메모리를 효율적으로 활용하여 다중 작업 환경을 제공합니다.

 **성능 최적화**: 자주 사용되지 않거나 오랜 시간 동안 기다리는 프로세스를 메모리에서 내보내면, 중요한 작업을 위한 메모리 자원을 확보할 수 있습니다.

(4) 프로세스 이미지(process image)의 구성요소들에 대해 자세히 설명하시오.

**텍스트 영역 (Text Segment)**

* **프로그램 코드**가 저장되는 영역입니다.
* 프로세스가 실행하는 기계어 명령어가 들어 있으며, 이는 보통 읽기 전용으로 설정되어 있습니다. 프로그램의 실행 흐름에 따라 이 영역의 코드를 CPU가 읽고 실행합니다.
* 여러 프로세스가 동일한 프로그램을 실행할 때, 코드가 변경되지 않기 때문에 이 영역은 **공유**될 수 있습니다.

**2. 데이터 영역 (Data Segment)**

* **전역 변수와 정적 변수**가 저장되는 영역입니다.
* 이 데이터는 프로그램이 실행되기 전부터 할당되며, 실행 동안 계속 유지됩니다.
* 보통 두 부분으로 나뉩니다:
  + **초기화된 데이터 영역 (Initialized Data)**: 초기값이 할당된 전역 변수들이 저장됩니다.
  + **초기화되지 않은 데이터 영역 (Uninitialized Data, BSS)**: 초기값이 없는 전역 변수와 정적 변수들이 포함됩니다.

**3. 힙 영역 (Heap Segment)**

* 프로세스 실행 중 **동적으로 할당된 메모리**가 저장되는 영역입니다.
* 사용자가 메모리 할당 함수(예: malloc, new)를 호출하여 런타임에 메모리를 요청하면 이 영역에서 할당됩니다.
* 힙은 위쪽으로 확장되며, 프로세스가 종료될 때까지 유지됩니다.
* 메모리 해제가 이루어지지 않으면 메모리 누수가 발생할 수 있습니다.

**4. 스택 영역 (Stack Segment)**

* **함수 호출과 관련된 데이터**가 저장되는 영역으로, 지역 변수, 함수의 인자값, 리턴 주소 등을 포함합니다.
* 스택은 함수가 호출될 때마다 새로운 프레임이 생성되며, 함수가 종료되면 해당 프레임이 제거됩니다.
* 스택은 아래쪽으로 확장되며, LIFO(Last In First Out) 구조로 작동합니다.
* 스택 오버플로우는 재귀 함수나 비정상적으로 많은 함수 호출이 이루어질 때 발생할 수 있습니다.

**5. 커널 데이터 (Kernel Data)**

* 프로세스 실행에 필요한 운영체제의 정보가 포함된 영역입니다. 이 영역은 주로 커널 모드에서 접근이 가능합니다.
* 프로세스 제어 블록(PCB, Process Control Block)도 여기에 포함되며, 이 블록은 프로세스의 현재 상태, 프로세스 ID, 우선순위, 스케줄링 정보, 레지스터 값, 열려 있는 파일 정보 등을 저장합니다.
* 운영체제가 프로세스를 관리하고 스케줄링할 때 필요한 정보를 담고 있습니다.

**6. 프로세스 상태 (Processor State)**

* **CPU 레지스터 값**, 프로그램 카운터(PC), 스택 포인터(SP), 플래그 레지스터 등이 포함됩니다.
* 프로세스가 실행 중단되고 나중에 다시 재개될 때, 중단 시점의 정확한 상태를 기억하여 그 지점에서 실행을 재개할 수 있도록 유지됩니다.
* 컨텍스트 스위칭 시 중요한 역할을 하며, 프로세스의 문맥을 유지하는 데 필요합니다.

**7. 열려 있는 파일 정보**

* 해당 프로세스가 열어둔 파일이나 네트워크 연결 등의 자원에 대한 정보가 포함됩니다.
* 각 프로세스는 열려 있는 파일의 목록을 유지하며, 파일 디스크립터 테이블이 이에 해당합니다. 이 테이블에는 파일의 핸들, 위치, 접근 권한 등의 정보가 저장됩니다.

**8. 환경 변수 (Environment Variables)**

* 프로세스의 실행 환경을 설정하는 변수들이 저장됩니다.

(5) 시간 5(time 5)에서 처리기와 메모리를 제외한 다른 시스템 자원들은 사용되고 있지 않다고 가정한다. 그리고 다음 이벤트들을 고려한다.

시간 5: P1이 3번 디스크 장치로부터 읽는 명령을 수행한다.

시간 15: P5의 타임 슬라이스(time slice)가 만료된다.

시간 18: P7이 3번 디스크 장치에 쓰는 명령을 수행한다.

시간 20: P3이 2번 디스크 장치로부터 읽는 명령을 수행한다.

시간 24: P5이 3번 디스크 장치에 쓰는 명령을 수행한다.

시간 28: P5이 스왑-아웃된다.

시간 33: 2번 디스크 장치로부터 인터럽트가 발생하고, P3의 읽기가 완료된다.

시간 36: 3번 디스크 장치로부터 인터럽트가 발생하고, P1의 읽기가 완료된다.

시간 38: P8이 종료된다.

시간 40: 3번 디스크 장치로부터 인터럽트가 발생하고, P5의 쓰기가 완료된다.

시간 44: P5가 스왑-인(swap-in)된다.

시간 48: 3번 디스크 장치로부터 인터럽트가 발생하고, P7의 쓰기가 완료된다.

시간 22, 시간 37, 시간 47에서 각 프로세스가 어떤 상태에 있는지 설명하여라. 만약 한 프로세스가 블록(blocked)되어 있다면 그 프로세스를 블록시킨 이벤트에 대해서도 설명하시오(Hint: Gantt-chart 형태로 각 프로세스의 상태를 추적한다).

**시간 22의 프로세스 상태:**

* **P1**: 3번 디스크 장치에서 읽기 작업을 대기 중이므로 **블록됨**.
  + 이벤트: 3번 디스크 장치에서의 읽기 완료를 대기 중.
* **P3**: 2번 디스크 장치에서 읽기 작업을 대기 중이므로 **블록됨**.
  + 이벤트: 2번 디스크 장치에서의 읽기 완료를 대기 중.
* **P5**: 3번 디스크 장치에 쓰는 명령을 수행 중이므로 **블록됨**.
  + 이벤트: 3번 디스크 장치에서의 쓰기 완료를 대기 중.
* **P7**: 3번 디스크 장치에 쓰는 명령을 수행 중이므로 **블록됨**.
  + 이벤트: 3번 디스크 장치에서의 쓰기 완료를 대기 중.
* **P8**: CPU 사용 중일 가능성 있음 (**실행 중**).

**2. 시간 37의 프로세스 상태:**

* **P1**: 3번 디스크 장치에서 읽기가 완료되었고, CPU 할당을 대기하므로 **대기 중**.
* **P3**: 2번 디스크 장치에서 읽기가 완료되었고, CPU 할당을 대기하므로 **대기 중**.
* **P5**: 여전히 스왑 아웃 상태로 **스왑 아웃** 상태.
* **P7**: 3번 디스크 장치에 쓰는 명령을 수행 중이므로 **블록됨**.
  + 이벤트: 3번 디스크 장치에서의 쓰기 완료를 대기 중.
* **P8**: 종료됨 (**종료**).

**3. 시간 47의 프로세스 상태:**

* **P1**: CPU 할당을 대기 중이므로 **대기 중**.
* **P3**: CPU 할당을 대기 중이므로 **대기 중**.
* **P5**: 스왑 인되어 CPU 할당을 대기 중이므로 **대기 중**.
* **P7**: 아직 3번 디스크 장치에서의 쓰기가 완료되지 않았으므로 **블록됨**.
  + 이벤트: 3번 디스크 장치에서의 쓰기 완료를 대기 중.

(6) UNIX 운영체제에서 fork() & exec() 시스템 호출에 대해 자세히 설명하시오

fork()\*\*는 현재 실행 중인 프로세스를 복제하여 \*\*새로운 프로세스(자식 프로세스)\*\*를 생성하는 시스템 호출입니다. 자식 프로세스는 부모 프로세스의 **거의 모든 상태를 복사**하며, 새로 생성된 자식 프로세스는 부모 프로세스와 동일한 메모리 공간, 파일 디스크립터, 환경 변수를 가지지만 독립적으로 실행됩니다.

**fork()의 주요 특징:**

* **프로세스 ID**: 자식 프로세스는 새로운 고유한 프로세스 ID(PID)를 갖습니다. 부모 프로세스는 fork() 호출의 반환값으로 자식 프로세스의 PID를 받으며, 자식 프로세스는 fork()가 **0**을 반환합니다.
* **주소 공간**: 자식 프로세스는 부모 프로세스의 **주소 공간을 복사**하지만, 이후에는 독립적인 메모리 공간을 가집니다. 자식 프로세스가 부모 프로세스의 변수를 수정해도 부모 프로세스에는 영향을 주지 않습니다.
  + 이 과정은 실제로는 **Copy-On-Write** 방식으로 구현되는 경우가 많습니다. 즉, 자식 프로세스가 메모리의 데이터를 변경하기 전까지는 부모와 메모리를 공유하며, 실제로 데이터가 수정되면 복사됩니다.
* **프로세스 상태**: 자식 프로세스는 부모 프로세스와 동일한 파일 디스크립터, 현재 디렉토리, 환경 변수를 복사합니다. 하지만 자식과 부모는 **서로 독립적**으로 실행됩니다.
* **실행 순서**: fork()를 호출한 후, 부모 프로세스와 자식 프로세스는 **동시에 병렬로** 실행될 수 있습니다.

\*exec()\*\*는 프로세스가 **다른 프로그램을 실행**하도록 하는 시스템 호출입니다. fork()에 의해 생성된 자식 프로세스가 새로운 프로그램을 실행하려면 exec()를 사용합니다. exec()는 호출된 프로세스의 **현재 코드, 데이터, 스택**을 완전히 대체하며, 새로운 프로그램으로 바뀝니다.

exec() 함수는 여러 가지 변형이 있으며, 그 중 가장 흔히 사용되는 것은 execl(), execv(), execle(), execve() 등입니다. 이들은 새로운 프로그램을 실행할 때 **프로그램의 경로**, **명령어 인자**, 그리고 **환경 변수**를 넘기는 방법에 따라 차이가 있습니다.

**exec()의 주요 특징:**

* **현재 프로세스 대체**: exec() 호출은 현재 실행 중인 프로세스를 완전히 대체합니다. 즉, 호출한 프로세스는 더 이상 기존의 코드를 실행하지 않고, 새로 지정된 프로그램의 코드로 대체됩니다.
* **PID 유지**: exec()는 새로운 프로그램을 실행하지만, 새로운 프로그램은 **기존 프로세스의 PID**를 유지합니다. 즉, 새로운 프로그램이 실행되어도 프로세스 ID는 바뀌지 않습니다.
* **프로세스 상태 유지**: 파일 디스크립터, 환경 변수 등 일부 프로세스 상태는 유지됩니다. 하지만 프로그램의 코드, 데이터, 스택은 새로운 프로그램으로 대체됩니다.

(7) UNIX 운영체제에서 fork() 시스템 호출을 처리하는 중에 운영체제는 자식 프로세스의 ID를 부모 프로세스에 반환한다. 즉, 하나의 fork() 시스템 호출은 프로세스 트리를 생 성하는데 그 트리에서 부모 프로세스는 자식 프로세스를 가리킨다. 이때 세 번의 연속 적인 fork() 시스템 호출을 수행할 경우 생성되는 프로세스 트리를 그려라

fork(); // A fork(); // B fork(); // C 각 생성된 프로세스에 해당하는 fork() 문장의 레이블을 붙이시오.

(8) 아래의 그림에 대해 운영체제 커널의 실행 방식과 응용 프로세스와 커널간의 인터페이스 방식을 비교하여 설명하시오.

(9) 프로세스와 쓰레드의 차이점에 대해 설명하여라.

| **특징** | **프로세스(Process)** | **쓰레드(Thread)** |
| --- | --- | --- |
| **정의** | 독립적인 실행 단위 | 프로세스 내의 경량 실행 단위 |
| **메모리 공유** | 프로세스 간에 메모리 공간을 공유하지 않음 | 같은 프로세스 내에서 메모리 공간을 공유 |
| **주소 공간** | 각 프로세스는 독립된 메모리 주소 공간을 가짐 | 같은 프로세스의 쓰레드들은 주소 공간을 공유 |
| **자원 소유** | 각 프로세스는 파일, 메모리 등 자원을 독립적으로 소유 | 쓰레드는 프로세스의 자원을 공유 |
| **통신 방식** | IPC (공유 메모리, 메시지 큐, 파이프 등) | 같은 프로세스 내에서 메모리를 공유하여 빠르게 통신 |
| **오버헤드** | 프로세스 생성과 컨텍스트 스위칭 시 오버헤드 큼 | 쓰레드 생성과 컨텍스트 스위칭 시 오버헤드 작음 |
| **독립성** | 다른 프로세스와 독립적으로 실행 | 같은 프로세스 내 쓰레드들은 서로 영향을 줄 수 있음 |
| **충돌 영향** | 한 프로세스가 충돌해도 다른 프로세스에 영향 없음 | 한 쓰레드가 충돌하면 같은 프로세스 내 모든 쓰레드에 영향 |
| **성능** | 상대적으로 느림 | 상대적으로 빠름 |
| **사용 사례** | 독립적인 프로그램 실행 (브라우저, 텍스트 편집기 등) | 병렬 처리 및 멀티태스킹 (멀티스레드 서버 등) |

(10) 멀티프로세스를 대신해 멀티쓰레딩을 사용하면 무슨 장점이 있는가?

멀티프로세스를 대신해 멀티쓰레딩을 사용하면 **자원 효율성, 통신 속도, 컨텍스트 스위칭 속도, 응답성, 성능** 측면에서 많은 이점을 제공합니다.

(11) ULT, KLT 그리고 LWP(Light-Weigh Process)를 비교하여 설명하여라.

 **ULT**는 **가벼운 오버헤드**와 **빠른 실행**을 장점으로 하지만, **블로킹 문제**와 **병렬성 제한**이 단점입니다.

 **KLT**는 **병렬 처리**와 **블로킹 문제 해결**에 강점이 있지만, **높은 오버헤드**가 발생합니다.

 **LWP**는 **ULT와 KLT의 장점을 결합**한 방식으로, **유연성과 병렬성**을 동시에 제공하면서도 관리 오버헤드가 적당한 수준입니다.

(12) 두 개의 서브 루틴을 호출하는 메인 루틴을 가진 프로그램 있다고 가정한다. 서브 루 틴은 병행적으로 실행될 수 있다. 이러한 프로그램을 구현할 수 있는 두 개의 가능한 접근 방법(하나는 쓰레드를 사용하고 다른 하나는 쓰레드를 사용하지 않음)을 제시하시 오.

(13) 사용자 수준 쓰레드와 커널 수준 쓰레드가 일대일로 사상되어 있는 상황에서, 한 프로 세스 내에서 하나 이상의 쓰레드가 블록킹 시스템 호출을 수행하고 다른 쓰레드들은 계 속 수행되고 있는 환경을 생각해보자. 왜 이러한 모델이 단일 처리기 기계에서 환경에 서 멀티스레드 프로그램이 단일 쓰레드 프로그램보다 더 빨리 수행할 수 있는지 설명하 시오.