3.1 프로세스 개념

프로세스란 실행 중인 프로그램을 말한다

하나의 시스템은 일부는 사용자 코드를 실행하고 일부는 운영체제 코드를 실행하는 프로세스의 집합체이다

초창기 컴퓨터는 작업(job)을 실행하는 일괄처리 시스템이었고, 사용자 프로그램 또는 태스크(task)를 실행하는 시분할 시스템이 뒤를 이었다

3.1.1 프로세스

프로세스란 실행 중인 프로그램이다

프로세스의 현재 활동의 상태는 프로그램 카운터 값과 프로세서 레지스터의 내용으로 나타낸다

프로세스의 메모리 배치는 일반적으로 여러 섹션으로 구분된다

텍스트 섹션 – 실행 코드

데이터 섹션 – 전역 변수

힙 섹션 – 프로그램 실행 중에 동적으로 할당되는 메모리

스택 섹션 – 함수를 호출할 때 임시 데이터 저장장소(예: 함수 매개변수, 복귀 주소 및 지역 변수)

프로그램 그 자체는 프로세스가 아니다

프로그램은 명령어 리스트를 내용으로 가진 디스크에 저장된 파일(실행 파일이라고 불림)과 같은 수동적인 존재(passive entity)이다

프로세스는 다음에 실행할 명령어를 지정하는 프로그램 카운터와 관련 자원의 집합을 가진 능동적인 존재(active entity)이다

실행 파일이 메모리에 적재될 때 프로그램은 프로세스가 된다

두 프로세스들이 동일한 프로그램에 연관될 수 있지만 이들은 두 개의 별도의 실행 순서로 간주한다

3.1.2 프로세스 상태

프로세스는 실행되면서 그 상태가 변한다

New: 프로세스가 생성 중이다

Running: 명령어들이 실행되고 있다

Waiting: 프로세스가 어떤 이벤트(입출력 완료 또는 신호의 수신 같은)가 일어나기를 기다린다

Ready: 프로세스가 처리기에 할당되기를 기다린다

Terminated: 프로세스의 실행이 종료되었다

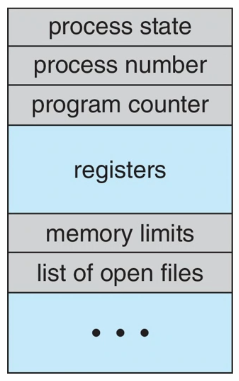
허느 한 순간에 한 CPU에서는 오직 하나의 프로세스만이 실행된다는 것을 인식하는 것이 중요하다

그렇지만, 많은 프로세스가 ready 및 waiting 상태에 있을 수 있다

3.1.3 프로세스 제어 블록

각 프로세스는 운영체제에서 프로세스 제어 블록(PCB, also called task control block)에 의해 표현된다

프로세스 제어 블록은 특정 프로세스와 관련된 여러 정보를 수록하며, 다음과 같은 것들을 포함한다



프로세스 상태: new, ready, running, waiting, 또는 halted(정지) 상태 등이다

프로그램 카운터: 다음에 실행할 명령어의 주소

CPU 레지스터들: 모든 프로세스 중심 레지스터들의 내용, 프로그램 카운터와 함께 이 상태 정보는 인터럽트 발생 시 저장되어야 한다

CPU 스케줄링 정보: 프로세스 우선순위, 스케줄 큐에 대한 포인터와 다른 스케줄 매개변수를 포함한다

메모리 관리 정보: 프로세스에 할당된 메모리

Accounting 정보: CPU 사용 시간과 경과된 시간, 시간 제한, 계정 번호, 잡 또는 프로세스 번호 등을 포함한다

I/O status 정보: 프로세스에 할당된 입출력 장치, 열린 파일의 목록

요약하자면 프로세스 제어 블록은 약간의 accounting 데이터와 함께 프로세스를 시작하거나 다시 시작시키는 데 필요한 모든 데이터를 위한 저장소의 역할을 한다

3.2 프로세스 스케줄링

다중 프로그래밍의 목적은 CPU 이용을 최대화하기 위하여 항상 어떤 프로세스가 실행되도록 하는 데 있다

시분할의 목적은 각 프로그램이 실행되는 동안 사용자가 상호 작용할 수 있도록 프로세스들 사이에서 CPU 코어를 빈번하게 교체하는 것이다

이 목적을 달성하기 위해 프로세스 스케줄러는 코어에서 실행 가능한 여러 프로세스 중에서 하나의 프로세스를 선택한다

현재 메모리에 있는 프로세스 수를 다중 프로그래밍 정도라고 한다

I/O 바운드 프로세스는 계산에 소비하는 것보다 I/O에 더 많은 시간을 소비하는 프로세스이다

반대로 CPU 바운드 프로세스는 계산에 더 많은 시간을 사용하여 I/O 요청을 자주 생성하지 않는다

3.2.1 스케줄링 큐

프로세스가 시스템에 들어가면 ready 큐에 들어가서 준비 상태가 되어 CPU 코어에서 실행되기를 기다린다

I/O 완료와 같은 특정 이벤트가 발생하기를 기다리는 프로세스는 wait 큐에 삽입된다

새 프로세스는 처음에 ready 큐에 놓인다. 프로세스는 실행을 위해 선택되거나 또는 디스패치 될 때까지 기다린다.

프로세스에 CPU 코어가 할당되고 running 상태가 되면, 여러 이벤트 중 하나가 발생할 수 있다

- 프로세스가 I/O 요청을 공표한 다음 I/O wait 큐에 놓일 수 있다

- 프로세스는 새 자식 프로세스를 만든 다음 자식의 종료를 기다리는 동안 wait 큐에 놓일 수 있다

- 인터럽트 또는 타임 슬라이스가 만료되어 프로세스가 코어에서 강제로 제거되어 ready 큐로 돌아갈 수 있다

3.2.3 문맥 교환; Context Switch

문맥 교환은 CPU가 하나의 프로세스를 다른 프로세스로 전환할 때 발생한다

CPU가 다른 프로세스로 전환할 때, 시스템은 반드시 이전 프로세스의 상태를 저장하고(state save) 문맥 교환(context switch)를 통해 저장되어 있던 새 프로세스의 상태를 불러와야 한다

CPU 코어를 다른 프로세스로 교환하려면 이전의 프로세스의 상태를 보관하고 새로운 프로세스의 상태를 복구하는 작업이 필요하다

이 작업을 문맥 교환(context switch)이라고 한다

문맥 교환이 일어나면, 커널은 과거 프로세스의 문맥을 PCB에 저장하고, 실행이 스케줄된 새로운 프로세스의 저장된 문맥을 복구한다

문맥 교환이 진행될 동안 시스템이 아무런 유용한 일을 못 하기 때문에 문맥 교환 시간은 순수한 오버헤드이다

문맥 교환 시간은 하드웨어의 지원에 크게 좌우된다

일부 하드웨어는 CPU당 여러 개의 레지스터 집합을 제공한다 -> 여러 문맥을 한 번에 불러옴

운영체제가 복잡할수록, 문맥 교환 시 해야 할 작업의 양이 더 많아진다

3.3 프로세스에 대한 연산

대부분 시스템 내의 프로세스들은 병행 실행될 수 있으며, 반드시 동적으로 실행되고, 제거되어야 한다

그러므로 운영체제는 프로세스 생성 및 종료를 위한 기법을 제공해야 한다

3.3.1 프로세스 생성

생성하는 프로세스를 부모 프로세스라 부르고, 새로운 프로세스는 자식 프로세스라고 부른다

이 새로운 프로세스들은 각각 다시 다른 프로세스들을 생성할 수 있으며, 그 결과 프로세스의 트리를 형성한다

대부분의 현대 운영체제들은 프로세스 식별자(pid)를 통해 프로세스를 식별 및 관리한다

언제나 pid가 1인 systemd 프로세스가 모든 사용자 프로세스의 루트 부모 프로세스 역할을 수행하고 시스템이 부트될 때 생성되는 첫 번째 사용자 프로세스이다

프로세스가 새로운 프로세스를 생성할 때, 두 프로세스를 실행시키는 데 두 가지 가능한 방법이 존재한다

1. 부모는 자식과 병행하게 실행을 계속한다
2. 부모는 일부 또는 모든 자식이 실행을 종료할 때까지 기다린다

새로운 프로세스들의 주소 공간 측면에서 볼 때 다음과 같은 두 가지 가능성이 있다

1. 자식 프로세스는 부모 프로세스의 복사본이다(자식 프로세스는 부모와 똑 같은 프로그램과 데이터를 가진다)
2. 자식 프로세스가 자신에게 적재될 새로운 프로그램을 가지고 있다

UNIX 예시

1. 새로운 프로세스는 fork() 시스템 콜로 생성된다  
   새로운 프로세스는 원래 프로세스의 주소 공간의 복사본으로 구성된다  
   이 기법은 부모 프로세스가 쉽게 원래 프로세스와 통신할 수 있게 한다
2. Fork() 시스템 콜 다음에 두 프로세스 중 한 프로세스가 exec() 시스템 콜을 사용하여 자신의 메모리 공간을 새로운 프로그램으로 교체한다  
   exec() 시스템 콜은 이진 파일을 메모리로 적재(load)하고 그 프로그램을 실행한다  
   이와 같은 방법으로 두 프로세스는 통신을 할 수 있으며, 그들 각자의 길을 간다
3. 부모는 wait() 시스템 콜로 자식 프로세스가 끝나길 기다린다

3.3.2 프로세스 종료

프로세스가 마지막 문장의 실행을 끝내고, exit() 시스템 콜을 사용하여 운영체제에 자신의 삭제를 요청하면 종료한다

1. 프로세스는 부모 프로세스에 상태 값(통상 정수 값)을 반환할 수 있다(wait() 시스템 콜을 통해)
2. 프로세스의 리소스는 할당 해제되고 운영체제로 반납된다

종료되었지만 부모 프로세스가 아직 wait() 호출을 하지 않은 프로세스를 좀비(zombie) 프로세스라고 한다

부모 프로세스가 wait()를 호출하는 대신 종료한다면 자식 프로세스는 고아(orphan) 프로세스가 된다

UNIX는 고아 프로세스의 새로운 부모 프로세스로 init 프로세스를 지정함으로써 이 문제를 해결한다

3.4 프로세스 간 통신

운영체제 내에서 실행되는 프로세스들은 독립적이거나 또는 협력적인 프로세스들일 수 있다

프로세스가 시스템에서 실행 중인 다른 프로세스들과 데이터를 공유하지 않는 프로세스는 독립적이다

프로세스가 시스템에서 실행 중인 다른 프로세스들에 영향을 주거나 받는다면 이는 협력적인 프로세스들이다. 데이터 공유를 포함해서

분명히 다른 프로세스들과 자료를 공유하는 프로세스는 상호 협력적인 프로세스이다

협력적 프로세스들은 데이터를 교환할 수 있는, 즉 서로 데이터를 보내거나 받을 수 있는 프로세스 간 통신(IPC) 기법이 필요하다

프로세스 간 통신에는 기본적으로 공유 메모리(shared memory)와 메시지 전달(message passing)의 두 가지 모델이 있다

공유 메모리에 모델에서는 협력 프로세스들에 의해 공유되는 메모리의 영역이 구축된다. 프로세스들은 그 영역에 데이터를 읽고 쓰고 함으로써 정보를 교환할 수 있다

메시지 전달 모델에서는 통신이 협력 프로세스들 사이에 교환되는 메시지를 통해서 이루어진다

