운영체제

프로세스와 스레드 차이

프로세스는 메모리 상에서 실행중인 프로그램을 말하며, 스레드는 이 프로세스 안에서 실행되는 흐름 단위를 말한다.

프로세스마다 최소 하나의 스레드를 보유하고 있으며, 각각 별도의 주소공간을 독립적으로 할당받는다. (code, data, heap, stack)

스레드는 이중에 stack만 따로 할당받고 나머지 영역은 스레드끼리 서로 공유한다.

[요약]

프로세스 : 자신만의 고유 공간과 자원을 할당받아 사용

스레드 : 다른 스레드와 공간과 자원을 공유하면서 사용

멀티 프로세스로 처리 가능한 걸 굳이 멀티 스레드로 하는 이유는?

프로세스를 생성하여 자원을 할당하는 시스템 콜이 감소함으로써 자원의 효율적 관리가 가능함

프로세스 간의 통신(IPC)보다 스레드 간의 통신 비용이 적어 작업들 간 부담이 감소함

대신, 멀티 스레드를 사용할 때는 공유 자원으로 인한 문제 해결을 위해 '동기화'에 신경써야 한다.

교착상태(DeadLock)가 무엇이며, 4가지 조건은?

프로세스가 자원을 얻지 못해 다음 처리를 하지 못하는 상태를 말한다.

시스템적으로 한정된 자원을 여러 곳에서 사용하려고 할 때 발생하는 문제임

교착상태의 4가지 조건은 아래와 같다.

- 상호배제 : 프로세스들이 필요로 하는 자원에 대해 배타적 통제권을 요구함 Mutual exclusion
- 점유대기 : 프로세스가 할당된 자원을 가진 상태에서 다른 자원 기다림 Hold and wait
- 비선점 : 프로세스가 어떤 자원의 사용을 끝날 때까지 그 자원을 뺏을 수 없음 No preemption
- 순환대기 : 각 프로세스는 순환적으로 다음 프로세스가 요구하는 자원을 갖고 있음 Circular wait

이 4가지 조건 중 하나라도 만족하지 않으면 교착상태는 발생하지 않음

(순환대기는 점유대기와 비선점을 모두 만족해야만 성립함. 따라서 4가지가 서로 독립적이진 않음)

교착상태 해결 방법 4가지

- 예방
- 회피
- 무시
- 발견

메모리 계층 (상-하층 순)

레지스터
캐시
메모리
하드디스크

메모리 할당 알고리즘 First fit, Next fit, Best fit 결과

- First fit : 메모리의 처음부터 검사해서 크기가 충분한 첫번째 메모리에 할당
- Next fit : 마지막으로 참조한 메모리 공간에서부터 탐색을 시작해 공간을 찾음
- Best fit : 모든 메모리 공간을 검사해서 내부 단편화를 최소화하는 공간에 할당

페이지 교체 알고리즘에 따른 페이지 폴트 방식

OPT : 최적 교체. 앞으로 가장 오랫동안 사용하지 않을 페이지 교체 (실현 가능성 희박)

FIFO : 메모리가 할당된 순서대로 페이지를 교체

LRU : 최근에 가장 오랫동안 사용하지 않은 페이지를 교체

LFU : 사용 빈도가 가장 적은 페이지를 교체

NUR : 최근에 사용하지 않은 페이지를 교체

외부 단편화와 내부 단편화란?

외부 단편화 : 작업보다 많은 공간이 있더라도 실제로 그 작업을 받아들일 수 없는 경우 (메모리 배치에 따라 발생하는 문제)

내부 단편화 : 작업에 필요한 공간보다 많은 공간을 할당받음으로써 발생하는 내부의 사용 불가능한 공간

가상 메모리란?

메모리에 로드된, 실행중인 프로세스가 메모리가 아닌 가상의 공간을 참조해 마치 커다란 물리 메모리를 갖는 것처럼 사용할 수 있게 해주는 기법

페이징과 세그먼테이션이란?

페이징

페이지 단위의 논리-물리 주소 관리 기법. 논리 주소 공간이 하나의 연속적인 물리 메모리 공간에 들어가야하는 제약을 해결하기 위한 기법 논리 주소 공간과 물리 주소 공간을 분리해야함(주소의 동적 재배치 허용), 변환을 위한 MMU 필요

특징 : 외부 단편화를 없앨 수 있음. 페이지가 클수록 내부 단편화도 커짐

세그먼테이션

사용자/프로그래머 관점의 메모리 관리 기법. 페이징 기법은 같은 크기의 페이지를 갖는 것 과는 다르게 논리적 단위(세그먼트)로 나누므로 미리 분할하는 것이 아니고 메모리 사용할 시점에 할당됨

뮤텍스, 세마포어가 뭔지, 차이점은?

세마포어

운영체제에서 공유 자원에 대한 접속을 제어하기 위해 사용되는 신호 공유자원에 접근할 수 있는 최대 허용치만큼만 동시에 사용자 접근 가능 스레드들은 리소스 접근 요청을 할 수 있고, 세마포어는 카운트가 하나씩 줄어들게 되며 리소스가 모두 사용중인 경우(카운트=0) 다음 작업은 대기를 하게 된다

뮤텍스

상호배제, 제어되는 섹션에 하나의 스레드만 허용하기 때문에, 해당 섹션에 접근하려는 다른 스레드들을 강제적으로 막음으로써 첫번재 스레드가 해당 섹션을 빠져나올 때까지 기다리는 것 (대기열(큐) 구조라고 생각하면 됨)

차이점

- 세마포어는 뮤텍스가 될 수 있지만, 뮤텍스는 세마포어가 될 수 없음
- 세마포어는 소유 불가능하지만, 뮤택스는 소유가 가능함
- 동기화의 개수가 다름

Context Switching이란?

하나의 프로세스가 CPU를 사용 중인 상태에서 다른 프로세스가 CPU를 사용하도록 하기 위해, 이전의 프로세스 상태를 보관하고 새로운 프로세스의 상태를 적재하는 작업

한 프로세스의 문맥은 그 프로세스의 PCB에 기록됨

사용자 수준 스레드 vs 커널 수준 스레드 차이는?

사용자 수준 스레드

장점 : context switching이 없어서 커널 스레드보다 오버헤드가 적음 (스레드 전환 시 커널 스케줄러 호출할 필요가 없기 때문)

단점 : 프로세스 내의 한 스레드가 커널로 진입하는 순간, 나머지 스레드들도 전부 정지됨 (커널이 스레드의 존재를 알지 못하기 때문에)

커널 수준 스레드

장점 : 사용자 수준 스레드보다 효율적임. 커널 스레드를 쓰면 멀티프로세서를 활용할 수 있기 때문이다. 사용자 스레드는 CPU가 아무리 많아도 커널 모드의 스케줄이 되지 않으므로, 각 CPU에 효율적으로 스레드 배당할 수가 없음

단점 : context switching이 발생함. 이 과정에서 프로세서 모드가 사용자 모드와 커널 모드 사이를 움직이기 때문에 많이 돌아다닐 수록 성능이 떨어지게 된다.

가상메모리란?

프로세스에서 사용하는 메모리 주소와 실제 물리적 메모리 주소는 다를 수 있음

따라서 메모리 = 실제 + 가상 메모리라고 생각하면 안됨

메모리가 부족해서 가상메모리를 사용하는 건 맞지만, 가상메모리를 쓴다고 실제 메모리처럼 사용하는 것은 아님

실제 메모리 안에 공간이 부족하면, **현재 사용하고 있지 않은 데이터를 빼내어 가상 메모리에 저장해두고, 실제 메모리에선 처리만 하게 하는 것이 가상 메모리의 역할**이다.

즉, 실제 메모리에 놀고 있는 공간이 없게 계속 일을 시키는 것. 이를 도와주는 것이 '가상 메모리'

fork()와 vfork()의 차이점은?

fork()는 부모 프로세스의 메모리를 복사해서 사용

vfork()는 부모 프로세스와의 메모리를 공유함. 복사하지 않기 때문에 fork()보다 생성 속도 빠름. 하지만 자원을 공유하기 때문에 자원에 대한 race condition이 발생하지 않도록 하기 위해 부모 프로세스는 자식 프로세스가 exit하거나 execute가 호출되기 전까지 block된다

Race Condition이란?

두 개 이상의 프로세스가 공통 자원을 병행적으로 읽거나 쓸 때, 공용 데이터에 대한 접근이 순서에 따라 실행 결과가 달라지는 상황

Race Condition이 발생하게 되면, 모든 프로세스에 원하는 결과가 발생하는 것을 보장할 수 없음. 따라서 이러한 상황은 피해야 하며 상호배제나 임계구역으로 해결이 가능하다.

리눅스에서 시스템 콜과 서브루틴의 차이는?

우선 커널을 확인하자

커널은 하드웨어를 둘러싸고 있음

즉, 커널은 하드웨어를 제어하기 위한 일종의 API와 같음

서브루틴(Subr>outine)은 우리가 프로그래밍할 때 사용하는 대부분의 API를 얘기하는 것

```
stdio.h에 있는 printf나 scanf
string.h에 있는 strcmp나 strcpy
```

서브루틴과 시스템 콜의 차이는?

서브루틴이 시스템 콜을 호출하고, 시스템 콜이 수행한 결과를 서브루틴에 보냄

시스템 콜 호출 시, 커널이 호출되고 커널이 수행한 임의의 결과 데이터를 다시 시스템 콜로 보냄

즉, 진행 방식은 아래와 같다.

```
서브루틴이 시스템 콜 호출 → 시스템 콜은 커널 호출
→ 커널은 자신의 역할을 수행하고 나온 결과 데이터를 시스템 콜에게 보냄
→ 시스템 콜이 다시 서브루틴에게 보냄
```

실무로 사용할 때 둘의 큰 차이는 없음(api를 호출해서 사용하는 것은 동일)