중간고사 정리

시스템 콜이란?

시스템 콜은 운영체제에서 제공하는 서비스에 대한 인터페이스를 제공하며,

사용자 프로세스는 시스템 콜을 통해 운영체제에서 제공하는 기능을 사용할 수 있다.

시스템 콜은 API를 통해 호출되며, 사용자 프로세스는 직접 호출할 수 없다.

대표적인 API로는 Windows API, POSIX API, Java API가 있다.

시스템 콜을 호출하는 프로그램에서 추가 정보가 필요할 경우, 매개 변수를 전달하며

이때 레지스터나 스택을 사용하여 전달한다.

링커와 로더?

링커와 로더는 운영체제에서 프로그램을 실행하는데 중요한 역할을 한다.

링커는 여러 Object 파일을 하나의 실행 파일로 결합하여 프로그램의 실행에 필요한 함수나 변수 등의 참조 관계를 해결하고 실행 파일을 만든다.

로더는 실행 파일을 메모리에 로드하여 실행하고, 보안 기능도 수행한다.

운영체제의 구조?

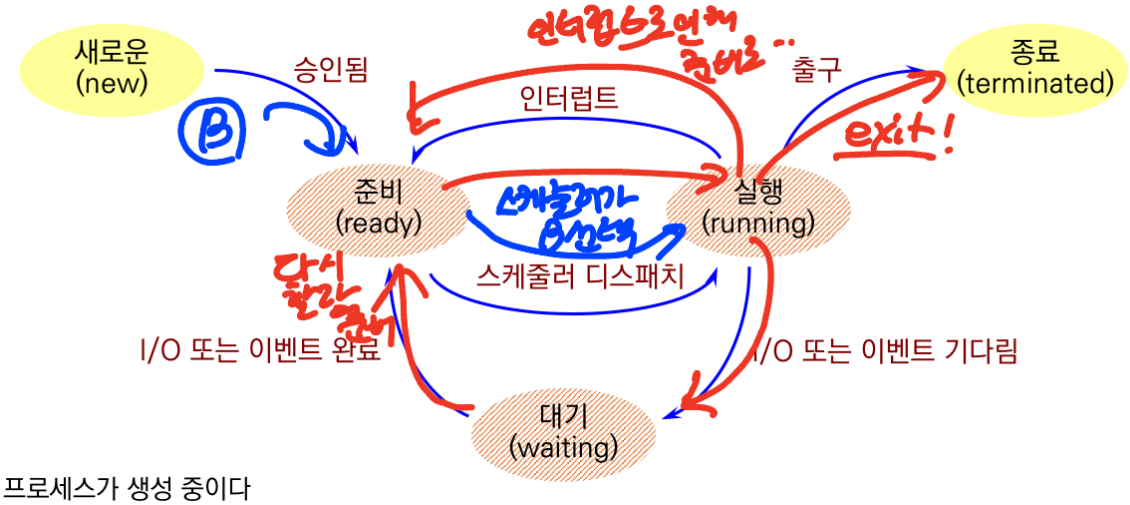
운영체제의 구조에는 모놀리식 구조와 마이크로 커널 구조가 있다.

모놀리식 구조는 커널의 모든 기능을 커널 내부에 포함시켜 운영체제를 구성하는 것을 말하며 오버헤드는 적지만 유지보수와 구현이 어렵다는 단점이 있다.

마이크로 커널 구조는 필요한 최소한의 기능만을 가지고 있으며, 나머지 기능들은 커널 외부에 모듈 형태로 구현한다.

이 구조는 높은 보안과 신뢰성을 제공하며 운영체제로의 확장이 용이하지만, 모듈 간의 통신에서 오버헤드가 발생할 수 있다.

프로세스 상태?



프로세스가 생성 중이다

프로세스는 CPU와 같은 자원이 필요하며 OS에서 상태를 관리하고 CPU 자원을 할당한다.

프로세스는 실행, 준비, 대기, 종료 상태를 가지며, 실행 상태는 CPU를 할당 받아 작업을 실행하고, 준비 상태는 실행 조건을 모두 갖추고 CPU를 할당 받기 위해 Ready Queue에서 대기하고, 대기 상태는 CPU 이외의 자원이 필요하여 CPU를 양도하고 대기하며, 이때 대기 중인 자원이 사용 가능해질 때까지 Waiting Queue에서 대기한다.

프로세스는 실행 상태에서 준비 상태, 대기 상태로 이동할 수 있고, 준비 상태에서만 다시 실행 상태로 돌아갈 수 있다. 인터럽트가 발생한 경우 실행에서 준비 상태로 이동한다.

대기 상태로 이동하는 예는 연산을 위해 입력을 기다려야 하는 경우이다.

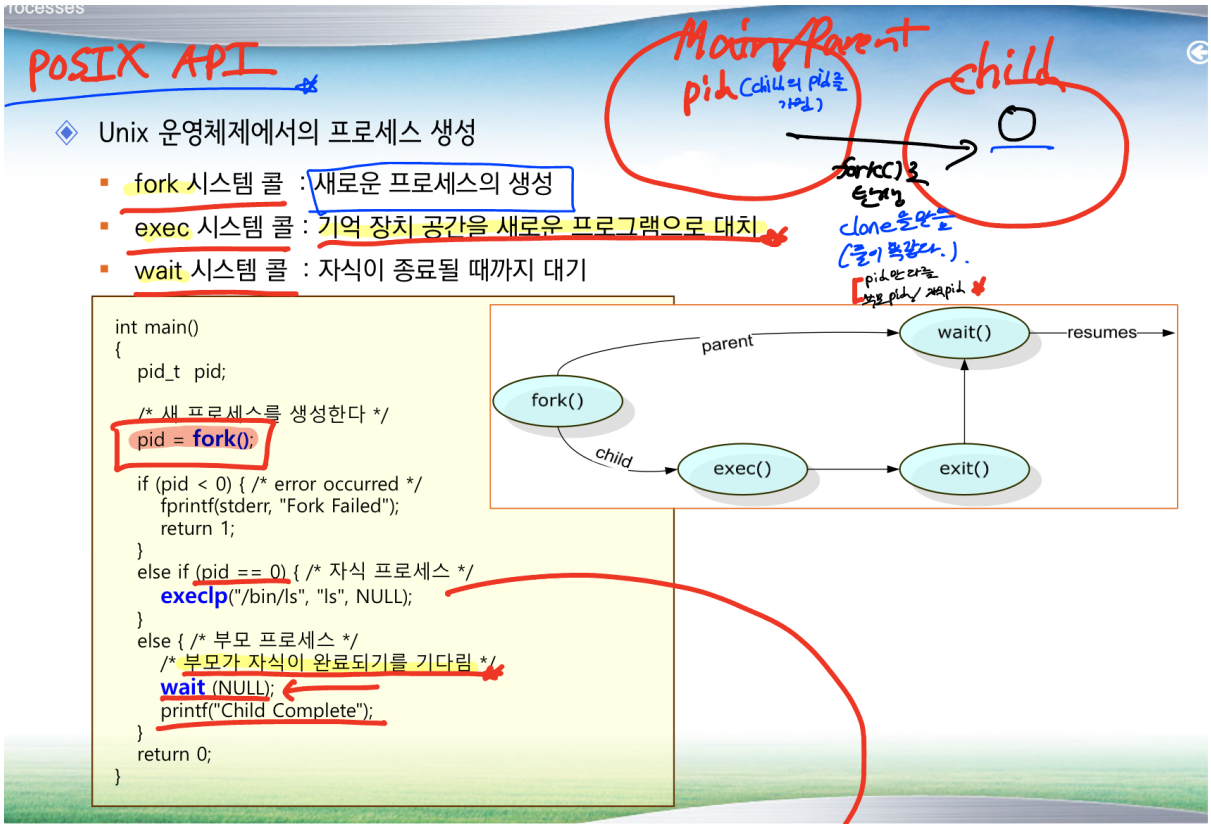
문맥 교환?

문맥은 CPU와 레지스터의 값, 프로세스 상태 등을 의미하며 해당 프로세스가 다시 실행될 때 중요한 역할을 한다.

문맥 교환이란, 실행 중인 프로세스의 문맥을 저장하고 대기 중인 프로세스의 문맥 정보를 복원하여 CPU를 할당하는 과정을 의미한다. 이때, 이전 프로세스의 문맥 정보를 PCB에 저장하게 된다.

문맥 교환 시간은 하드웨어 성능에 영향을 많이 받으며, 순수한 오버헤드다.

프로세스 생성과 종료 과정 + Unix POSIX API?



POSIX API에서는 아래와 같이 프로세스 생성과 종료 과정에서 필요한 함수를 제공한다.

프로세스 생성은 `fork()` 시스템 콜을 사용하여 이루어진다.

`fork()` 는 현재 프로세스(부모)와 동일한 복사본인 자식 프로세스를 생성한다.

위 자식 프로세스는 부모 프로세스와 동일한 프로그램 코드를 실행하며 각각의 프로세스 ID를 가지게 된다.

`exec()` 시스템 콜을 사용하여 현재 프로세스를 새로운 프로세스로 교체할 수 있다.

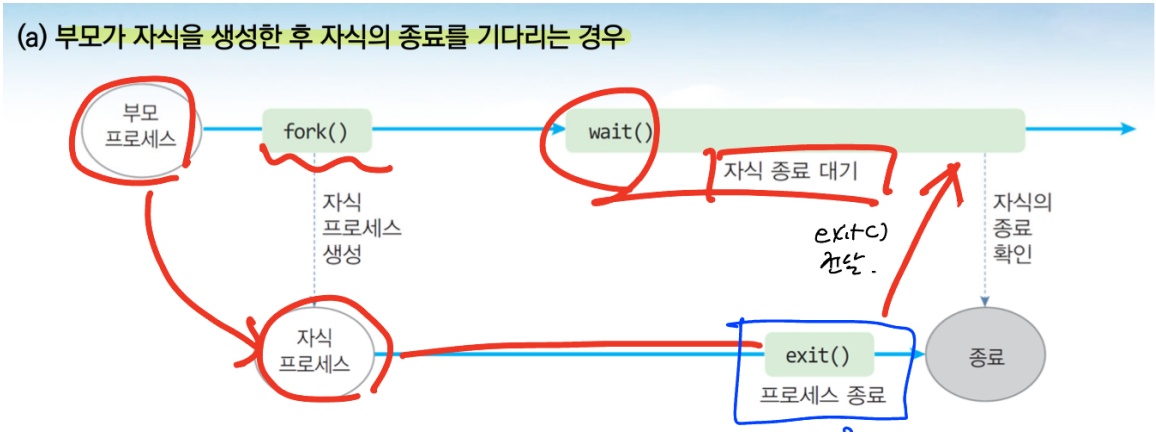
`exec()` 는 현재 프로세스를 새로운 프로세스로 덮어쓰기 때문에, 새로운 프로세스는 이전 프로세스와 완전히 다른 프로그램 코드를 실행하게 된다.

따라서, `exec()` 를 사용하면 새로운 프로세스를 생성하지 않고도 프로세스를 교체할 수 있다. 만약 `fork()` 시스템 콜 이후 `exec()` 시스템 콜을 사용하는 경우가 확실하다면,

copy-on-write(COW) 방식을 사용하여 최소한의 복사만 이루어지게할 수 있다.

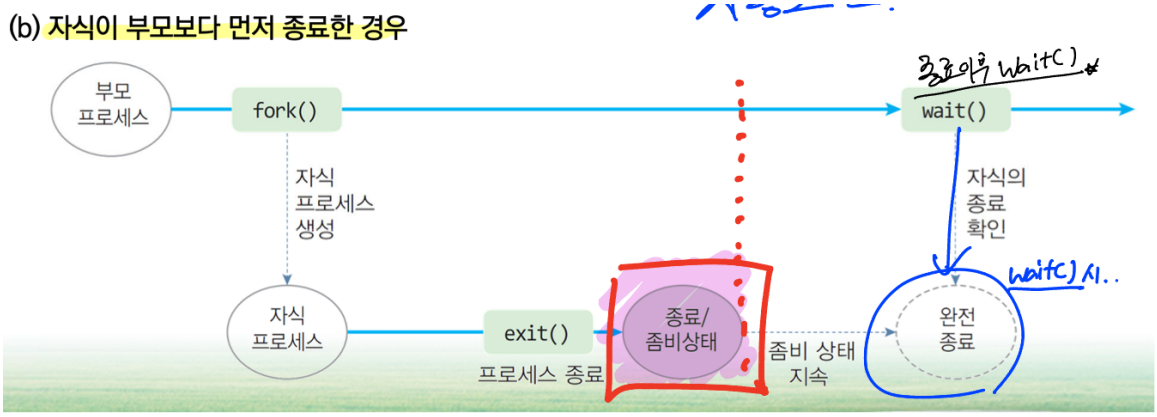
`exit()` 시스템 콜을 사용하여 프로세스 종료가 된다. `exit()` 은 OS에게 종료 상태값을 전달한다. 이 종료 상태값은 부모 프로세스가 `wait()` 시스템 콜을 호출하여 확인할 수 있다.

`wait()` 시스템 콜은 자식이 종료될 때까지 대기하며, 부모 프로세스는 자식 프로세스의 종료 상태 값을 확인하고 필요에 따라 후속 작업을 진행한다.



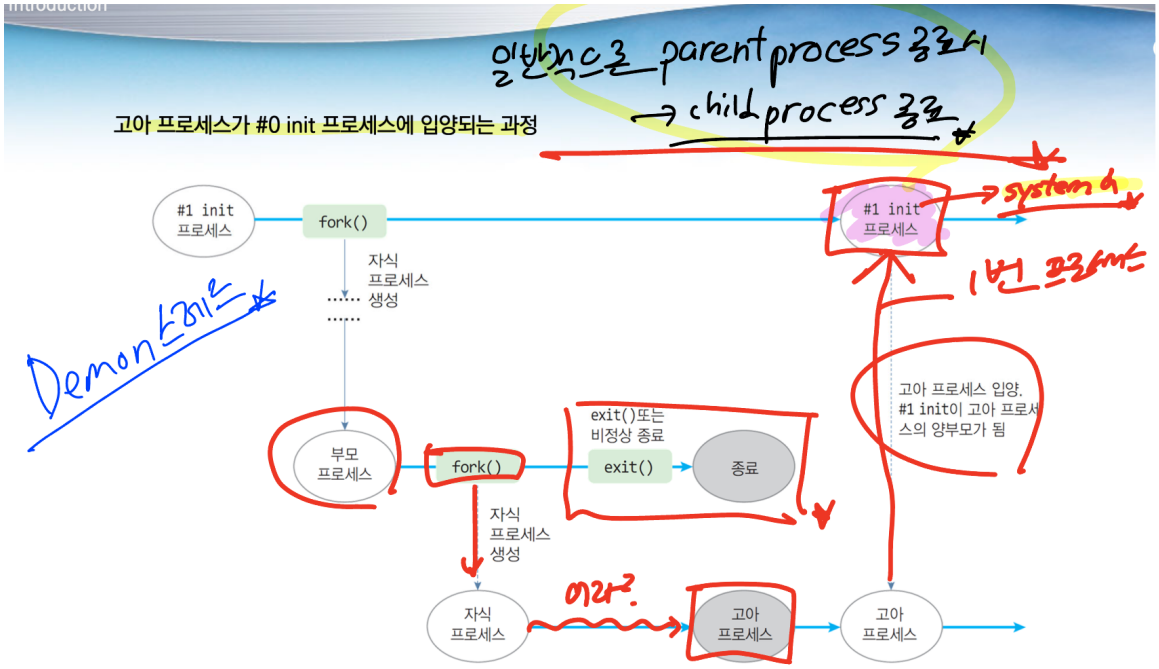
부모가 자식을 생성한 후 자식의 종료를 기다리는 일반 적인 경우, `fork()` 시스템 콜을 사용하여 자식 프로세스를 생성하고, `wait()` 상태로 자식 프로세스의 종료 상태를 기다리게 된다.

자식 프로세스가 `exit()` 되면 종료 상태값을 전달 받고 부모 프로세스는 하던 일을 계속 진행 한다.



자식 프로세스가 생성된 후, 부모 프로세스가 `wait()` 시스템 콜을 통해 자식 프로세스의 상태를 확인하기 전에 자식 프로세스가 종료된 경우, 자식 프로세스는 좀비 프로세스가 된다.

부모 프로세스가 `wait()` 시스템 콜로 확인 전까지 PCB는 남아서 좀비 상태가 지속되므로, 부모 프로세스가 `wait()` 시스템 콜을 호출하여 상태를 확인하고 메모리에서 제거해야 한다.



부모 프로세스가 종료되었지만, 자식 프로세스가 아직 종료되지 않은 상태이다.

이 경우, 자식 프로세스는 부모 프로세스를 기다리지 않는 대신 새로운 부모 프로세스가 필요하다. 이 때, 자식 프로세스는 #1 init 프로세스에게 입양된다.

멀티 스레드 모델?

스레드 지원은 사용자 수준 및 커널 수준에서 제공된다.

다대일 모델은 여러 개의 사용자 수준 스레드를 하나의 커널 스레드에 매핑하여 문맥 교환에 대한 비용을 줄일 수 있으나, 여러 스레드를 동시에 실행할 수 없다.

일대일 모델은 각 사용자 수준 스레드에 하나의 커널 스레드를 할당하며, 여러 스레드를 동시에 실행할 수 있어 성능이 좋지만, 많은 수의 스레드를 생성하는 경우 커널 스레드 수가 증가하여 성능 저하가 발생할 수 있다.

다대다 모델은 여러 개의 사용자 수준 스레드를 여러 개의 커널 스레드에 매핑하며, 다대일과 일대일 모델의 단점을 보완할 수 있지만, 커널 스레드의 수가 증가하여 성능 저하가 발생할 수 있다.

스레드 라이브러리 (Unix / Pthreads)?

Pthreads는 Unix OS에서 멀티 스레드 프로그래밍 구현을 위해 사용되는 스레드 라이브러리다.

`pthread_create()` 함수를 이용하여 스레드를 생성할 수 있으며,

`pthread_join()` 함수를 사용하여 스레드가 종료될 때까지 대기할 수 있다.

스레드 풀?

스레드 풀은 프로세스가 시작될 때 미리 일정한 수의 스레드를 생성하고,

서버가 요청을 받으면 하나의 스레드 할당 작업을 진행한다.

스레드 풀의 장점으로는 스레드 생성 및 삭제 오버헤드가 감소하며,

작업 처리 지연이 감소한다.

스케줄링 알고리즘?

스케줄링 알고리즘 비교 기준에는 CPU 이용률, 처리량, 총처리 시간, 대기 시간 등이 있다.

CPU 이용률과 처리량은 늘려야 하며, 총처리 시간과 대기시간은 줄여야 한다.

스케줄링 알고리즘의 종류는 다음과 같다.

1. 선입 선처리 스케줄링 / FCFS (First-Come, First-Served)
2. 최단 작업 우선 스케줄링 / SJF (Shortest-job-first)
3. 우선 순위 스케줄링 / Priority
4. 라운드-로빈 스케줄링 / 시분할(time sorrting)
5. 다단계 큐, 다단계 피드백 큐 스케줄링

선입 선처리 스케줄링은 먼저 도착한 프로세스가 먼저 실행되는 스케줄링 알고리즘이다.

큐에 들어온 순서대로 처리하므로 비선점형 스케줄링 알고리즘이며, 평균 대기 시간이 긴 경우가 있다. 실행 시간이 긴 프로세스가 먼저 도착하게되면, 그 다음 도착하는 짧은 실행 시간

의 프로세스들도 긴 실행 시간의 프로세스가 끝날때 까지 대기 해야되는 현상이 발생하는데, 이를 **호위 효과**라 한다.

최단 작업 우선 스케줄링은 실행 시간이 가장 짧은 프로세스를 우선으로 처리하는 스케줄링 알고리즘이다.

평균 대기 시간이 가장 적으며, 선점형과 비선점형 두 가지 방식이 있다.

비선점형은 실행중인 프로세스가 CPU 버스트를 완료할 때까지 선점하지 않는다.

선점형은 새로운 프로세스가 현재 실행되고 있는 프로세스의 남은 시간보다 짧은 버스트를 가지면 현재 실행중인 프로세스를 선점한다. 따라서, **새로운 프로세스가 도착하는 시점**에 재평가가 이루어진다.

SJF의 가장 치명적인 단점은 **다음 CPU 버스트의 길이를 알 방법이 없다는 것**이다.

따라서 각 프로세스의 실행 시간을 추정하고 이를 기반으로 스케줄링을 수행하게 된다.

또한 SJF의 특성으로 인해, 긴 수행 시간을 가진 작업은 대기열에서 오래 기다려야 될 수 도 있다. 이를 **기아 상태**라고 한다.

라운드-로빈 스케줄링은 시간 할당량을 정해놓고,

해당 시간이 지나면 선점하여 다른 프로세스에게 CPU를 할당하는 스케줄링 알고리즘이다.

시간 할당량은 10~100ms로 정해지는데, 해당 시간이 자나면 일시 중단되고 다른 작업이 실행 되고 각 작업 간 문맥 교환이 일어난다.

시간 할당량이 크면 FCFS와 유사한 성능을 보이며, 시간 할당량이 작으면 매우 많은 문맥 교환이 일어나 오버헤드가 커지게 된다.

우선 순위 스케줄링은 CPU에서 가장 높은 우선 순위를 가진 프로세스를 먼저 처리하는 알고리즘이며 선점형이거나 비선점형일 수 있다.

선점형 우선순위 스케줄링 알고리즘은 새로 도착한 프로세스의 우선 순위가 **현재 실행되는 프로세스의 우선 순위보다 높으면 CPU를 선점**한다.

비선점형 우선 순위 스케줄링 알고리즘은 단순히 **준비 큐의 머리 부분에 새로운 프로세스**를 넣는다.

우선 순위 스케줄링에서 발생하는 문제점은 무한 봉쇄/기아 상태이다.

무한 봉쇄는 우선 순위가 낮은 프로세스가 CPU 자원을 할당 받지 못해 영원히 대기하는 상태를 말한다.

무한 봉쇄를 방지하는 방법으로 우선 순위를 동적으로 변경해주는 방법이 존재하며, **에이징(Aging)**이라고 한다

에이징을 통해 우선 순위를 점진적으로 증가시키게 된다.

다단계 큐 스케줄링 알고리즘은 **작업들을 다른 그룹에 할당 하는 것**으로 **큐를 여러 개 사용**하는 알고리즘이다.

각 큐는 자신만의 스케줄링 알고리즘을 가지고 있으며, 우선 순위나 작업 유형에 따라 다르게 설정할 수 있다.

다단계 큐 피드백 스케줄링 알고리즘은 다단계 큐 스케줄링 알고리즘에서 확장된 버전이다.

프로세스의 우선 순위와 CPU 사용 시간에 따라 다른 큐로 이동할 수 있다.

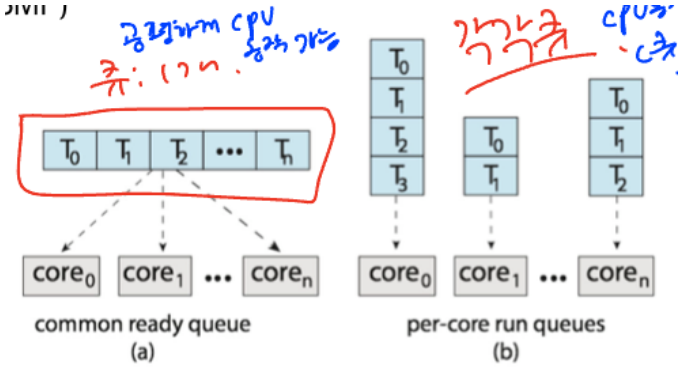
다중 처리 스케줄링?

다중 처리 스케줄링은 크게 두가지로 나뉜다.

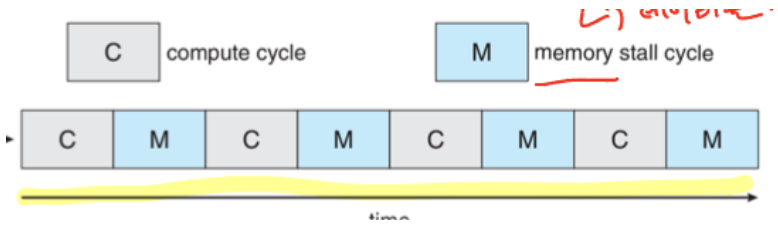
1. 비대칭 다중 처리 (AMP)
2. 대칭 다중 처리 (SMP)

비대칭 다중 처리는 하나의 프로세서가 작업을 처리하는 것을 말하며, 하나의 프로세서만 큐에 접근하므로 문제점을 발생시킬 가능성이 적어진다.

대칭 다중처리는 각 프로세서가 독자적으로 스케줄링하는 것을 말하며, 하나의 공동 큐를 사용하거나 각각 큐를 할당하는 방법이 있다.



다중 코어 프로세서?



하나의 프로세서 안에 여러 개의 프로세서 코어를 장착한 구조이다.

각 코어가 메모리를 공유하므로 동시에 코어가 메모리에 접근하려고 하는 경우 충돌이 발생해 메모리가 멈추는데 이를 메모리 멈춤 현상이라고 한다.

부하 균등화?

SMP에서 부하 균등화는 각 CPU가 일을 골고루 처리할 수 있도록 하는 것을 말한다.

부하 균등화를 위한 방법으로 pull, push 방식이 있는데

pull 방식은 작업 처리 능력이 높은 프로세서가 남은 작업을 가져와 처리하는 방식이고,

push 방식은 과부하인 프로세서가 작업을 덜 바쁜 프로세서로 이동하는 방식을 말한다.

위 방식이 가능한 이유는 OS는 CPU가 얼마나 바쁜지 체크가 가능하기 때문이다.

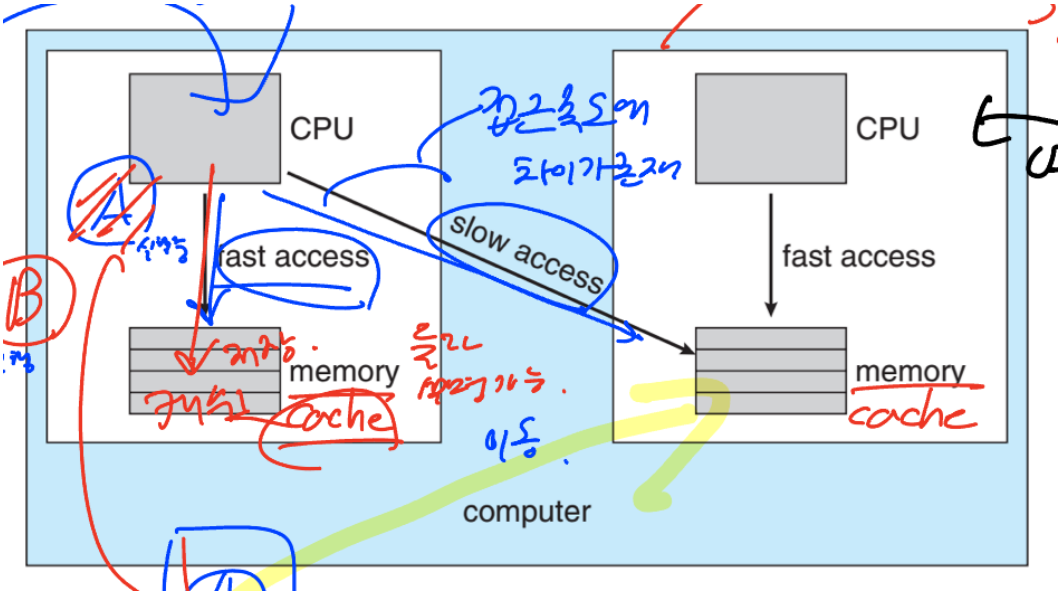
처리기 친화성?

한 처리기에서 다른 처리기로의 이동을 피하고, 같은 처리기에서 프로세서를 실행시키려 하는것을 말한다.

연성 친화성은 가능하다면 동일한 처리기에서 처리하는 것이 목적으로 이주가 가능하다.

강성 친화성은 프로세스가 다른 처리기로 이주하지 않게 지정하는 것으로 이주로 인한 추가 비용 발생을 방지하는 것이다.

시스템의 메인 메모리 구조는 처리기 친화성 이슈에 영향을 미칠 수 있는데, 처리기에 바로 붙어 있는 메모리에 접근하는 시간과, 옆에 있는 메모리에 접근하는 시간의 차이로 인해 발생한다.



실시간 컴퓨팅의 종류?

연성 실시간 시스템은 모든 dealine를 만족시키는 것을 보장하지 않으며 일반적인 시스템에서 요구되는 빠른 응답시간과 같은 목표를 충족시키는 것이 주요 목표이고, 웹 서버 등이 이에 해당한다.

경성 실시간 시스템은 모든 deadline을 엄격하게 만족해야한다.

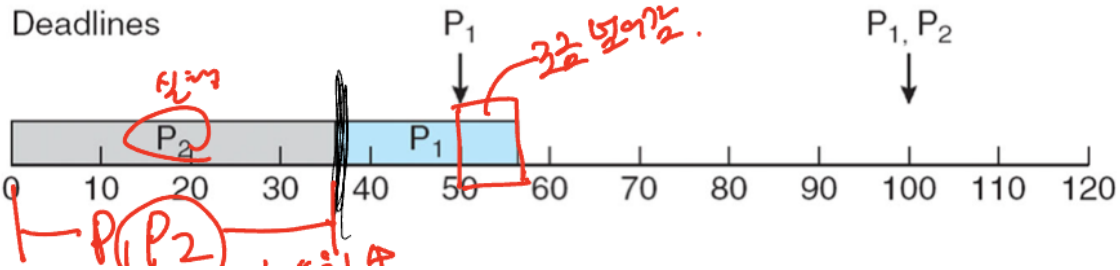
작업 처리 시간이 예측 가능해야한다. 대표적으로 원자력, 로켓 발사, 항공 제어 등이 있다.

실시간 CPU 스케줄링?

실시간 CPU 스케줄링에는 두 가지 종류가 있다.

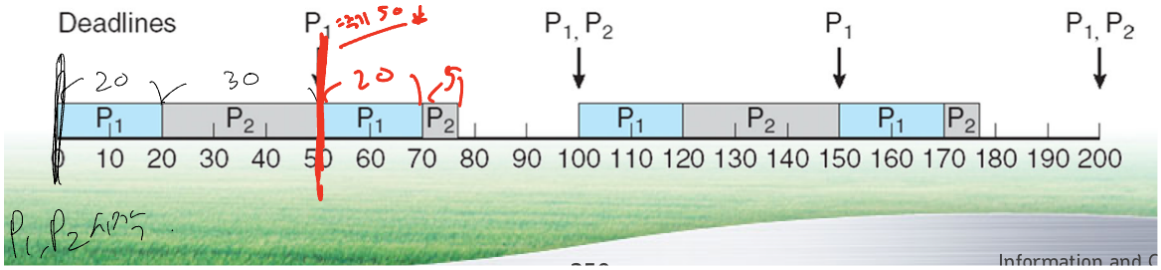
1. Rate Monotonic 스케줄링 (RMS)
2. Earliest Deadline First Scheduling (EDF)

기존에는 우선순위 정책을 이용하여 스케줄링을 진행



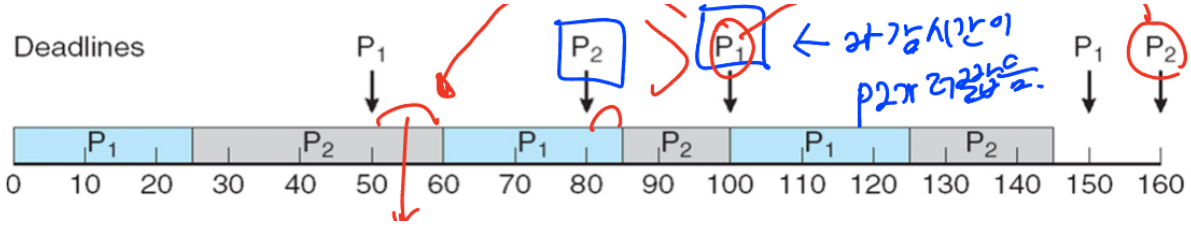
주기를 넘겨서 실행하는 경우가 존재한다.

Rate Monotonic Scheduling은 주기가 짧은 프로세스에 높은 우선순위를 부여하는 것이다.



주기를 넘겨서 실행하는 경우가 존재한다.

Earliset Deadline First Scheduling은 마감시간에 따라 우선순위를 동적으로 부여하는 방법이다.



마감시간(주기)에 따라서 우선 순위를 부여하는 방법이다.

경쟁 상황?

동시에 여러 개의 프로세스가 동일한 자료를 접근하여 조작하고 그 실행 결과가 접근이 발생한 특별한 순서에 의존하는 상황을 말한다. 그러므로 프로세스들이 동기화되도록 할 필요가 있다.

임계구역?

각 프로세스들은 임계구역 이라고 부르는 코드 부분을 포함하고 있으며, 해당 부분에서는 공유 자원을 사용하는 상황에 있어서 다른 스레드가 해당 자원을 사용하지 못하도록 보호하는 데 목적이 있다.

임계 구역 문제 해결안으로는 상호 배제, 진행, 한계 대기가 있다.

상호배제는 여러개의 프로세스나 스레드가 공유 자원을 동시에 접근하지 못하도록 하는 기법이다.

진행은 실행 중이지 않은 프로세스들 중 진입 하려는 임계구역에 관심이 있는 프로세스 간 서로 경쟁하여 임계 구역에 들어오는 것이다.

한계 대기는 특정 프로세스가 무한히 대기 하는 것을 방지하고 실행 기회를 갖게 하는 것이다. 또한, OS에서는 임계구역 문제를 해결하기 위해 뮤텍스, 세마포, 모니터 등의 동기화 도구를 제공한다.

동기화를 위한 하드웨어 지원?

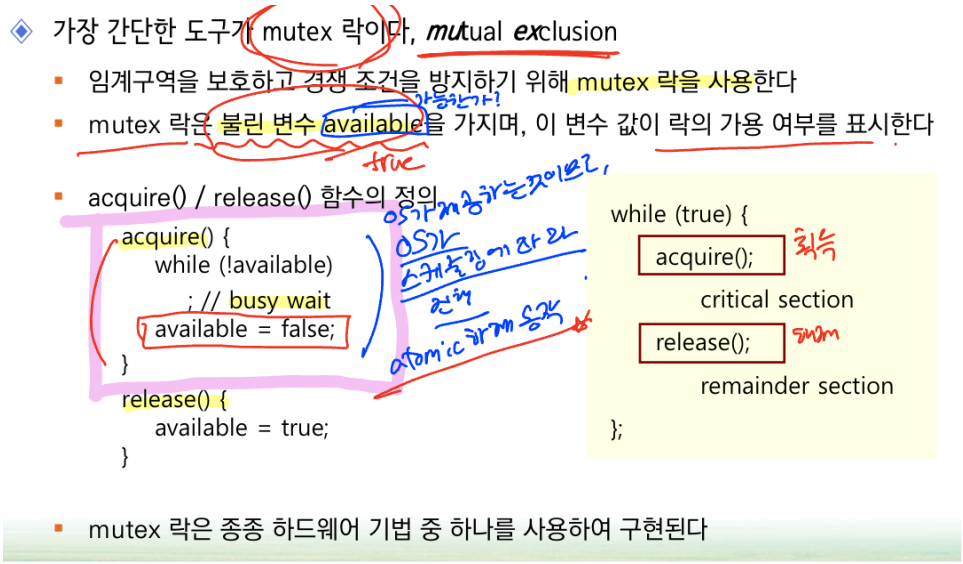
원자적 명령어와 CAS(compare-and-swap)가 존재한다.

원자적 명령어는 실행 중에 중단될 수 없는 원자적 연산을 지원하는 명령어 이며,

CAS 명령어는 메모리에 저장된 값을 비교하고 일치하면 새로운 값을 저장하는 기능을 지원한다.

위 기능을 이용해 소프트웨어 기반 동기화 기술(뮤텍스 등)을 개선하며 효율적인 동기화 기술을 구현할 수 있다.

뮤텍스 락?



임계 구역을 보호 하고 경쟁 상황을 방지하기 위해 등장한 동기화 도구이다.

뮤텍스 락은 boolean 변수 available을 가지며, 잠금 해제와 잠금 상태를 의미한다.

뮤텍스를 사용하면 공유 자원에 대한 접근 권한이 있는 (락을 확득한) 스레드만 공유 자원에 접근할 수 있게 된다.

다른 스레드들은 뮤텍스가 잠겨 있는 동안 대기 상태에 놓이게 된다.

busy-wait은 뮤텍스를 점유하고 있는 스레드가 뮤텍스를 해제할 때까지 다른 스레드가 기다리는 방식이다.

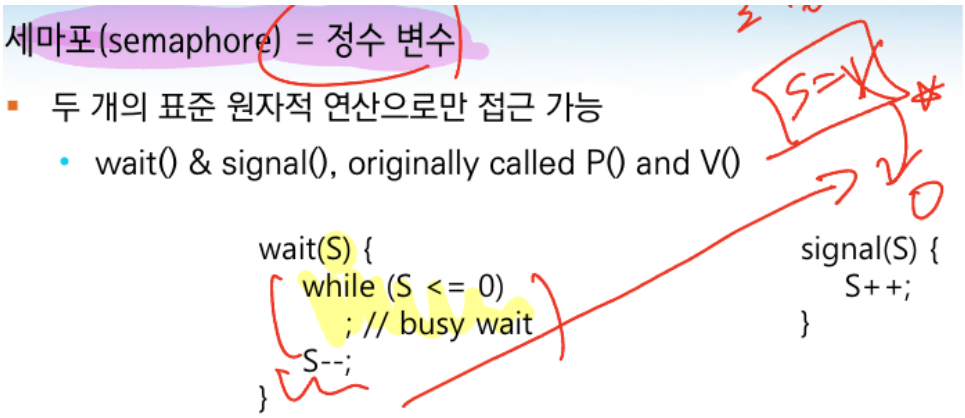
이 방식은 스레드가 자원을 점유하고 있는 동안 다른 스레드가 작업을 수행하지 못해, 성능과 자원에 영향을 준다.

spinlock은 뮤텍스를 점유하고 있는 스레드가 뮤텍스를 해제할 때까지 다른 스레드가 계속해서 뮤텍스를 확인하는 방법이다. 이 방법은 busy-wait에 비해 자원을 덜 낭비하는 방식이다.

멀티 코어 시스템에서는 spinlock 방식이 선호되기도 한다.

코어가 1개인 경우에는 스핀락은 무조건 좋지 않는 방법이다. 코어가 여러개인 경우에도, 시스템의 전체적인 성능으로 보면 좋지 않으나, 그걸 수행하는 프로세스 입장에서는 특별한 경우 스핀락 형태가 선호될 수도 있다.

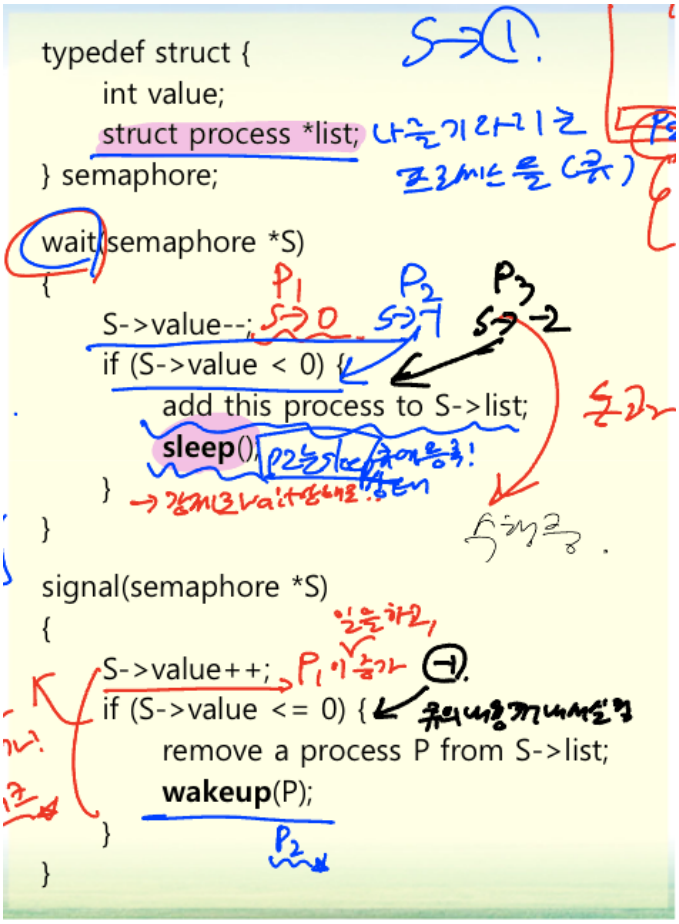
세마포?



공유 자원에 접근할 수 있는 정수 변수로 원자적 연산으로만 접근 가능한 것을 의미한다.

이진 세마포와 카운팅 세마포가 존재하며 이진 세마포는 0 또는 1의 값만 가질 수 있는 세마포이다.

카운팅 세마포의 값은 양의 정수값을 가지며 유한한 개수를 가진 자원에 대한 접근을 제어하는데 사용 가능하다.



세마포에서 spinlock없이 구현을 하는 방법에는 waiting queue를 사용하는 방법이 존재한다.

`sleep()` 연산은 waiting queue에 해당 프로세스를 추가하고 해당 프로세스가 실행 가능할 때까지 프로세스를

차단한다.

`wakeup()` 연산은 waiting queue에 대기 중인 프로세스 하나를 선택하여 실행 가능한 상태로 만든다.

spinlock과 달리 sleep과 wakeup 연산을 사용하면 프로세스가 차단되어 있을 때 CPU를 점유하지 않기 때문에 효율적인 시스템 동작을 가능하게 한다.

해당 방법의 단점은 `wakeup()`을 하게된 프로세스가 준비 큐에 들어갈 때 준비 큐에 어떤 프로세스가 있을 지 모르므로, 제일 먼저 실행 된다는 보장이 없다.

모니터?

모니터는 공유 자원을 감싸는 자료구조로서, 모니터 내부에서 공유 자원에 대한 접근이 이루어지도록 한다.

모니터를 사용하면 프로그래머가 동기화 함수를 직접 호출하지 않아도 된다.

모니터 내부에서 위 동작을 수행하기 때문이다.

교착상태?

두개 이상의 프로세스가 서로 상대방의 작업이 끝나기만을 기다리고 있어, 모두 무한정 대기하는 상황을 의미한다.

우선순위 역전?

하위 우선순위를 가진 프로세스가 상위 우선순위를 가진 프로세스와의 동기화를 위해 점유하고 있는 자원을 해제하지 않고 대기하면서 상위 우선 순위를 가진 프로세스가 해당 자원을 요청하는 경우 발생하는 문제이다.

이때, 하위 우선순위를 가진 프로세스가 자원을 해제하지 않으므로 상위 우선 순위를 가진 프로세스는 대기를 계속하게 되며, 무한 대기 상태에 빠지게 된다.

우선순위 역전을 해결하는 방법에는 **우선순위 상속 프로토콜**이 있다.

더 높은 우선순위 프로세스가 필요로 하는 자원을 접근하는 모든 프로세스들에게 문제가 된 자원의 사용이 끝날 때 까지 더 높은 우선순위를 상속 받는 것이다.

대표적인 우선순위 역전 문제 예시로 Mars Pathfinder이 있다.