**- OS(운영체제)가 무엇인지 설명해주실 수 있나요?**

- 운영체제는 하드웨어 자원을 관리하고, 응용 프로그램와 하드웨어 사이를 중재하는 인터페이스를 의미

+ 인터페이스 : 서로 다른 두 개의 시스템 사이에서 정보나 신호를 주고 받는 접점이나 경계를 의미

**- 커널이 무엇인지 설명하실 수 있나요?**

- 기본적으로 프로그램이 실행되기 위해서는 주기억장치에 적재된 상태여야 한다.

- 운영체제 역시 프로그램이기 때문에 메인 메모리에 적재가 되어야 하나, 운영체제의 크기가 너무 커서 전체를 다 메인 메모리에 올리게 된다면 비효율적. (메모리 공간의 낭비)

- 따라서 커널(kernel)이라고 하는 항상 필요한 운영체제의 핵심 부분만을 메인 메모리에 적재하여

운영체제를 사용.

- 즉, 커널은 메모리에 상주하는 운영체제의 핵심 부분이라고 할 수 있다.

**- 메모리의 구조를 영역에 따라서 설명해줄 수 있나요?**

1) 코드 영역(텍스트 영역) :

실행할 프로그램의 코드가 저장되는 영역으로, '텍스트' 영역. CPU는 해당 코드 영역에 저장된 명령어를 하나씩 가져가서 처리.

2) 데이터 영역(static 영역) :

전역 변수와 정적 변수가 저장되는 영역으로, 프로그램이 시작하는 동시에 할당, 프로그램이 종료되면 소멸.

3) 힙 영역 (Heap) :

사용자가 직접 관리하는 영역이며 메모리 공간이 동적으로 할당 및 헤제. 동적으로 할당되므로 new로 생성한 object가 많아질 수록 힙 영역의 크기가 늘어나게 된다.

4) 스택 영역 (Stack) :

함수의 호출에 따른 지역변수와 매개변수가 저장되는 영역, 컴파일 시 크기가 결정.

함수의 호출과 함께 할당, 함수의 호출이 종료되면 소멸.

**- 메모리의 힙 영역과 스택 영역의 차이에 대해 설명.**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**- 메모리의 힙 영역을 너무 크게 잡으면 어떤 일이 생길까요?**

- 힙 영역은 스택 영역과 다르게 직접 생성된 변수를 제거를 통해 메모리 관리가 필요

- 메모리의 힙 영역이 너무 크다면, 넓은 힙 영역에 존재하는 동적 변수에 대해 참조 여부를 확인해야 하기 때문에 Garbage Collect의 수행시간이 너무 길어지게 됩니다.

- Java의 JVM에서도 처음부터 힙 영역을 크게 잡지 않고, Heap 영역이 꽉 찼을 때 GC 진행 후 Heap 영역을 확장

**- 프로세스와 쓰레드에 대해 설명하고, 둘의 차이에 대해 설명.**

1) 프로세스

- 메인 메모리(주기억장치)에 적재되어 실행되는 프로그램을 의미.

- PCB, code, data, heap, stack 영역을 갖습니다.

- 프로세스별 각각의 영역을 갖기 때문에 별다른 동기화 작업이 필요하지 않습니다.

- 프로세스별 각각의 영역을 갖기 때문에 context switching 비용이 큽니다.

- 프로세스 쓰레드보다 무겁기 때문에 생성 시간이 더 깁니다.

- 한 프로세스에서 오류가 생기더라도 다른 프로세스에 영향을 주지 않습니다.

2) 쓰레드

- 한 프로세스 내의 실행 단위, 실행의 흐름을 의미.

- stack 영역만 별도로 갖고, 나머지 영역(code, data, heap)에 대해서는 해당 쓰레드를 포함한 프로세스의 자원을 공유.

- stack 영역 외부의 공유 자원에 대해 접근 시 동기화 작업이 필요.

- stack 영역 외부를 공유하므로 컨텍스트 스위칭 비용이 적다.

- 쓰레드는 프로세스보다 가볍기 때문에 생성 시간이 더 짧다.

- 한 쓰레드에서 오류가 생기면, 나머지 프로세스 내의 쓰레드에 영향을 줄 수 있다.

**- 프로세스 제어 블록 (PCB)에 대해 설명.(데드락 관련)**

프로세스 제어 블록(Process Control Block, 줄여서 PCB)은 프로세스를 관리하기 위한 정보를 포함하는 OS 커널의 자료 구조.

- OS는 프로세스의 생성과 동시에 고유한 PCB 를 생성

- PCB가 프로세스의 중요한 정보를 포함하고 있기 때문에, 일반 사용자가 접근하지 못하도록 보호된 메모리 영역 안에 위치.

- 프로세스는 수행할 차례가 되면, CPU를 할당받아 작업을 처리. 작업을 처리하던 도중 프로세스의 시간이 모두 경과되거나 인터럽트가 발생하는 등의 프로세스 전환이 발생하면, 진행하던 작업을 저장하고 CPU 를 반환.

이 때, 수행하던 프로세스 관련 데이터들을 PCB 에 저장. 그리고 다시 프로세스의 수행 차례가 와서 CPU 를 할당받게 되면, PCB 에 저장되어있던 내용을 불러와 이전에 종료됐던 시점부터 다시 작업을 수행. 이렇게 사용하던 프로세스의 데이터를 PCB에 저장하고, 실행할 프로세스의 데이터를 PCB에서 불러오는 과정을 Context Switching 이라고 합니다.

- PCB 에 저장되는 정보는 아래와 같다.

1. 프로세스 식별자(Process ID, PID) : 프로세스 식별번호

2. 프로세스 상태 : new, ready, running, waiting, terminated 등의 상태를 저장

3. 프로그램 카운터 : 프로세스가 다음에 실행할 명령어의 주소

4. CPU 레지스터

5. CPU 스케쥴링 정보 : 프로세스의 우선순위, 스케줄 큐에 대한 포인터 등

6. 메모리 관리 정보 : 페이지 테이블 또는 세그먼트 테이블 등과 같은 정보를 포함

7. 입출력 상태 정보 : 프로세스에 할당된 입출력 장치들과 열린 파일 목록

8. 어카운팅 정보 : 사용된 CPU 시간, 시간제한, 계정번호 등

**- 크롬 브라우저에서 한 탭에 오류가 생기더라도, 다른 탭에 영향을 끼치지 않습니다. 그렇다면 각각의 크롬 탭은 프로세스일까요? 쓰레드일까요?**

크롬 탭이 쓰레드라면, 한 크롬 탭에서 오류가 생기면 다른 크롬 탭에도 영향이 갈 수 있다. 따라서 한 탭에 오류가 생기더라도 다른 탭에 영향을 끼치지 않는다는 특성에 의해, 크롬 탭은 별개의 프로세스인 것을 알 수 있다.

**- 멀티 쓰레드와 멀티 프로세스를 비교.**

1) 멀티 쓰레드

하나의 프로세스를 여러 개의 쓰레드로 구성해 자원을 공유하면서 작업을 수행하는 것을 의미.

멀티 프로세스보다 적은 메모리 공간을 차지하고, Context Switching이 빠른 장점이 있지만, 오류로 인해 하나의 쓰레드가 종료되면 전체 쓰레드가 종료될 수 있다는 점과 동기화 문제를 가지고 있습니다.

2) 멀티 프로세스

하나의 프로그램을 여러개의 프로세스로 구성해 프로세스가 병렬적으로 작업을 수행하는 것.

하나의 프로세스가 죽더라도 다른 프로세스에 영향을 끼치지 않지만, 멀티 쓰레드보다 많은 메모리 공간을 차지합니다.

3) 왜 멀티 쓰레드를 더 많이 사용하는가?

하나의 프로그램 안에서 여러 작업을 해결하는 것이 더욱 효율적이고, Context Switching 시, Stack 영역만 초기화하면 되기 때문에 더 빠릅니다. 프로세스 생성시 자원 할당하는 콜이 줄어들어 자원을 효율적으로 관리할 수 있는 장점-.

**- 쓰레드마다 독립적으로 할당하는 2가지에 대해 설명.**

쓰레드는, 독립적으로 스택 영역과 PC Register를 갖고 있습니다. 그 이유는 쓰레드의 정의와 컨텍스트 스위칭에 있습니다.

1) 독립적으로 스택 영역을 갖는 이유 : 스택 영역은 함수의 실행과 관련이 있습니다. 독립적으로 스택을 갖는다는 것은 곧 독립적으로 함수 호출이 가능하다는 것을 의미합니다. 즉, 프로세스 내의 독립적인 실행 흐름을 만들기 위해 쓰레드마다 스택 영역을 갖습니다.

2) 독립적으로 PC Register를 갖는 이유 :

PC Register 값은, 쓰레드가 명령어를 어디까지 수행했는 지를 나타냅니다. 쓰레드는 CPU를 할당받았다가, 스케줄러에 의해서 선점당해 컨텍스트 스위칭이 일어납니다. 따라서 다시 해당 쓰레드가 실행될 때 어디까지 실행이 되었는지 알아야 하므로 독립적으로 PC Register 값을 갖게 됩니다.

**- 멀티 쓰레드 환경에서의 주의해야 할 점에 대해 설명할 수 있나요?**

멀티 쓰레드는 서로 한 프로세스 내의 자원을 공유하므로 동기화를 항상 고려해야 합니다.

**- 동기와 비동기의 차이에 대해 설명.**

- 동기는 요청과 그 결과가 동시에 일어난다는 약속입니다. 요청을 하면, 시간이 얼마가 걸리던지 요청한 자리에서 결과가 주어져야 합니다. 설계가 매우 간단하고 직관적이지만, 결과가 주어질 때까지 아무것도 못하고 대기해야 합니다.

- 정확한 설명으로는, A함수가 B함수를 호출할 때, B함수의 결과를 A가 처리하는 것.

- 비동기는 반대로 요청과 결과가 동시에 일어나지 않을 것이라는 약속입니다. 동기보다 복잡한 방식이지만, 그 시간동안 다른 작업이 가능하므로 자원을 효율적으로 사용할 수 있다는 장점이 있다.

- 정확한 설명으로는, A함수가 B함수를 호출할 때, B함수의 결과를 B가 처리하고 A에게 callback으로 알려주는 것.

**- Blocking과 Non-Blocking에 대해 설명.**

- Blocking은, A함수가 B함수를 호출할 때, B 함수가 자신의 작업이 종료되기 전까지 A 함수에게 제어권을 돌려주지 않는 것을 의미. 즉, B 함수가 종료될 때까지 A 함수는 다른 일을 수행할 수 없다.

- Non-Blocking은, A함수가 B함수를 호출할 때, B함수가 A 함수에게 곧바로 제어권을 돌려주게 됩니다. A 함수는 곧바로 제어권을 가지고 있으므로 다른 일을 수행할 수 있다.

- 따라서 Blocking != Sync, Non-blocking != Async 입니다.

- 동기/비동기 = 호출당한 함수의 종료를 누가 처리하느냐

- blocking/non-blocking = 제어권이 누구에게 돌아가느냐

**- 프로세스의 종류는 무엇이 있는지 설명.**

1. 자식 프로세스 : fork로 자식 프로세스를 만든 상태, 부모의 데이터, 힙, 스택, PCB 복사

2. 데몬 프로세스 : 백그라운드에서 동작하면서 특정한 서비스를 제공하는 프로세스

3. 고아 프로세스 : 부모 프로세스가 먼저 종료되어 고립된 자식 프로세스

4. 좀비 프로세스 : 자식 프로세스가 종료되었음에도 불구하고 부모 프로세스로부터 작업 종료에

대한 승인을 받지 못한 프로세스

**- Race Condition과 Critical Section이 무엇이고, 경쟁상태를 막기 위해 어떤 방법을 사용하는지 설명.**

- 두 개 이상의 쓰레드가 공유자원에 대해 접근하려고 서로 경쟁하는 것을 'Race Condition(경쟁상태)'

- 이 공유자원이 존재하는 구간으로 Race Condition이 일어날 수 있는 영역을 'Critical Section(임계영역)'

- 이러한 임계영역에 대한 경쟁상태를 제거하기 위해서 한 공유자원에 대해서 한 쓰레드에만 접근을 허락하도록 하는 '상호배제(Mutual Exclusion)'를 사용.

**- DeadLock(교착상태)에 대해 설명하고, 해결 방법에 대해 설명.**

- DeadLock(교착상태)는, 두 개 이상의 프로세스나 쓰레드가 서로 자원을 기다리면서 무한히 대기하는 상태를 의미.

- DeadLock이 발생하기 위한 조건은 4가지가 있다.

1. 상호 배제 (Mutual Exclusion) : 한 자원에 여러 프로세스가 동시에 접근할 수 없다.

2. 점유 대기 (Hold and Wait) : 하나의 자원을 소유한 상태에서 다른 자원을 기다린다.

3. 비선점 (Non-preemption) : 프로세스가 어떤 자원의 사용을 끝낼 때까지 프로세스의 자원을 뺏을 수 없다.

4. 순환 대기 (Circular Wait) : 각 프로세스가 순환적으로 다음 프로세스가 요구하는 자원을 갖고 있다.

교착상태를 해결하는 방법 : 위의 4가지 조건 중 하나라도 제거하면 막을 수 있다.

Race Condition 제거를 위해서 '상호 배제' 조건을 제거하기는 어렵다.

1. 예방(Prevention) :

- 교착 상태 조건 중 하나를 제거함으로 데드락이 발생하지 않도록 예방하는 방식

- 자원 낭비가 심하다.

2. 회피(Avoidance) :

- 데드락 발생 가능성을 인정하면서도 적절하게 회피하는 방식

- 은행원 알고리즘 : 프로세스가 자원을 요구할 때 시스템은 자원을 할당한 후에도 안정 상태로

남아있게 되는지를 사전에 검사하는 알고리즘

발생하지 않으면 자원을 할당하고, 발생하면 다른 프로세스가 자원을 해제할 때까지 대기한다.

- 안전상태: 시스템이 교착상태를 일으키지 않으면서 각 프로세스가 요구한 최대 요구량만큼

필요한 자원을 할당해 줄 수 있는 상태

- 항상 안전 상태를 유지할 수 있다는 장점이 있으나 최대 자원 요구량을 미리 알아야 하고

항상 불안전 상태를 방지해야 하므로 자원 이용도가 낮다.

1. 회복(Recovery) :

- 은행원 알고리즘과 유사하게 시스템의 자원 할당 상태를 갖고 데드락이 발생했는지

여부를 탐색한다.

- 탐지했다면 데드락을 회복한다.

- 데드락 상태의 프로세스 모두 중단시키기

- 프로세스를 하나씩 중단 시킬 때마다 탐지 알고리즘으로 데드락을 탐지하면서 회복시키기

- 자원 섬점을 통해 해결하기(교착 상태에 있는 프로세스가 점유하고 있는 자원을 선점하여

다른 프로세스에 할당)

**- Mutex(뮤텍스)와 Semaphore(세마포어)에 대해 설명(동기화 종류)**

- 뮤텍스

오직 1개만의 스레드(또는 프로세스)만 접근 할 수 있다.

1개만 접근 가능하므로 반드시 락을 획득한 프로세스가 락을 해제해야 한다.

- 세마포어

세마포어 변수만큼의 스레드(또는 프로세스)가 접근 할 수 있다.

현재 수행중인 프로세스가 아닌 다른 프로세스가 세마포어를 해제할 수 있다.

이진 세마포어는 뮤텍스와 동일하다고 볼 수 있다. 이진 세마포어가 아닌 나머지 세마포어를 카운팅

세마포어라고 부른다.

- 뮤텍스와 세마포어를 사용해 임계영역에서의 경쟁상태를 제거할 수 있습니다.

**- 선점(preemption)과 비선점(non-preemption)이 무엇인지 설명.**

선점 방식은, 특정 프로세스가 실행 중이더라도 CPU 할당을 뺏어 다른 프로세스에 부여할 수 있는 방식입니다.

System Call, Time Quantum, Interrupt 등에 의해 선점이 일어납니다.

비선점 방식은, 특정 프로세스가 실행 중이면 끝나기 전까지는 절대로 다른 프로세스에게 CPU를 뺏기지 않는 방식입니다.

**- 동시성과 병렬성이 어떻게 다른지 설명.**

-동시성은 멀티 프로그래밍 (Multi-Programming)에서 나온 개념으로, 주기억장치에 여러 프로세스를 적재해서 Context Switching을 통해 동시에 실행되는 것 처럼 보이게 하는 것을 의미합니다. 실제로는 동시에 실행되는 것이 아니라, 번갈아 실행하는 것입니다. 싱글 코어에서 멀티스레드를 동작시키기 위한 방식입니다.

- 병렬성은 멀티 프로세싱 (Multi-Processing)에서 나온 개념으로, 실제로 동시에 여러 프로세스를 병렬적으로 실행하는 방식입니다. 병렬적으로 실행하기 위해서는 CPU가 멀티코어여야 합니다.

**- Interrupt(인터럽트)에 대해 설명.**

- Interrupt(인터럽트)는 프로그램을 실행하고 있는 도중에 입출력 요청 혹은 예외 상황을 처리해야 하면 실행하던 프로그램을 멈추고 CPU가 해당 작업을 처리하도록 하는 것을 의미.

**- System Call은 무엇인가요?**

System Call(시스템 콜)은 사용자나 응용프로그램이 커널에서 제공하는 기능을 사용하기 위한 인터페이스

**- 메모리의 종류에 대해 설명하고, 종류가 여러가지인 이유에 대해 설명.**

- CPU에 가까운 순서대로 레지스터, 캐시, 주기억장치, 보조기억장치가 있다.

- 물리적 메모리의 종류가 많은 이유는, 접근 속도에 따른 차이를 두기 위해서.

(레지스터 > 캐시 > 주기억장치 > 보조기억장치)

**- 메모리의 fit의 종류 4가지에 대해 설명.**

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

Worst fit : 남은 공간 중에 가장 큰 공간에 할당

**- 요구 페이징(Demand Paging)이 무엇인지 설명.**

- 요구 페이징(Demand Paging)은 프로그램 실행 시작 시에 프로그램 전체를 물리 메모리에 적재하는 대신, 초기에 필요한 것들만 적재하는 전략을 의미합니다. 가상 메모리와 페이지 개념을 활용해 관리

**- Thrashing(쓰레싱)에 대해 설명해주세요.**

- 페이지 부재율이 높은 상태를 의미

- 페이지 부재가 발생하면, 페이지 교체 혹은 페이지 로드가 일어납니다. 다양한 프로세스가 메모리에 올라오면서 메모리의 유효 사용 공간은 줄어들고, CPU의 가동 시간이 올라가면서 자원을 최대한 사용하게 됩니다.

- 그러나 메모리에 프로세스가 너무 많아지면, 프로세스당 물리 메모리를 사용할 수 있는 프레임의 개수가 줄어들고, 페이지가 물리 메모리에 적게 올라온 프로세스는 명령이 진행될 때 마다 Page Fault가 일어나 페이지 교체가 일어나게 됩니다. 이 경우엔 CPU 사용률이 줄어들게 됩니다.

- 페이지를 교체하는 과정에서는 CPU를 사용하지 않고, 페이지 교체만을 계속해서 실행하면, 운영체제는 CPU가 놀고 있으므로 더 많은 프로세스를 메모리에 올리려고 하며 악순환이 일어나게 됩니다. 이러한 현상을 Thrashing(쓰레싱)이라고 합니다.

- 이를 해소 하기 위해서 Working Set 알고리즘과 Page Fault Frequency 알고리즘을 사용합니다.

- Working Set : 대부분의 프로세스가 특정 페이지만 집중적으로 참조하는 특성을 이용, 일정 시간 동안 참조되는 페이지 개수를 파악하고, 그 페이지 수만큼 여분 프레임이 확보되면 그 때 페이지를 메모리에 올리는 알고리즘 입니다.

- Page Fault Frequency : Page Fault 퍼센트의 상한과 하한을 둡니다. 상한을 넘으면 페이지에게 지급하는 프레임 개수를 늘리고, 하한을 넘으면 지급 프레임 개수를 줄입니다.

- 또 CPU 사용률과 메모리 적재량을 함께 체크해야 쓰레싱 유무를 확인할 수 있게 됩니다.

**- 메모리가 고갈되면 일어나는 현상에 대해 설명.**

메모리가 고갈되었지만, 프로세스를 실행해야 하기 때문에 Swap이 활발해집니다. 따라서 CPU 이용률이 하락하고, OS는 CPU 이용률이 낮으므로 오히려 프로세스를 추가하게 되는 쓰레싱 현상이 발생합니다. 이를 해소하지 못하면 Out Of Memory 상태로 판단해 중요도가 낮은 프로세스를 강제로 종료하게 합니다.

**- 캐시(cache) 메모리를 왜 사용하는지, CPU의 적중률을 높이기 위해 어떤 원리를 사용하는지에 대해 설명.**

- CPU와 메모리 사이의 속도 차이를 완화하기 위해 사용합니다.

- 메모리의 데이터를 미리 가져와 저장해두는 임시 장소로, 캐시의 크기는 메인 메모리보다 작기 때문에 앞으로 사용될 것으로 예상하는 데이터를 미리 저장하는 것이 중요한 포인트

**- DMA(Direct Memory Access)에 대해 설명.**

데이터의 이동에는 CPU의 개입이 필요합니다. 그러므로 다량의 데이터가 이동하는 과정에서는 CPU에 많은 부하가 걸리게 됩니다. 따라서 CPU를 거치지 않고 직접 데이터를 전송하는 방법인 DMA를 사용합니다.

**- 메모리 할당 중, 연속(Contiguous) 방식과 불연속(Non-Contiguous) 방식에 대해 설명.**

- 연속 할당 방식은, 프로세스 이미지가 분리되지 않은 온전한 프로세스를 메인 메모리에 적재하는 방식입니다. 연속 할당 방식에는 크게 2가지가 있으며 '고정 길이 할당'과 '가변 길이 할당'이 있습니다.

- 고정 길이 할당 : 메모리를 고정된 길이로 파트를 분리하고, 프로세스를 해당 파트에 삽입합니다. 고정된 길이의 파트 내에 프로세스가 들어가므로, 여백이 생길 수 있는데 이것을 '내부 단편화'라고 합니다.

- 가변 길이 할당 : 메모리를 프로세스의 길이에 알맞게 파트를 분리하고, 해당 프로세스를 적재합니다. 가변된 길이이므로 내부 단편화는 발생하지 않지만, 가변적인 길이 할당에 의해 일부분이 남으므로 '외부 단편화'가 발생합니다.

- 불연속 할당 방식은, 프로세스가 연속된 이미지가 아닌 분리된 이미지로 구성되어 메인 메모리에 적재하는 방식입니다. '페이징'과 '세그멘테이션' 방식이 있습니다.

- 페이징 : 프로세스를 동일한 크기의 페이지로 분리, 메모리는 해당 페이지와 동일한 크기의 프레임으로 분리해서 메인 메모리에 불연속적으로 저장하는 방식입니다. 만약 프로세스가 13MB이고 페이지가 4MB라면, 한 페이지는 1MB가 됩니다. 프레임은 무조건 4MB로 분리했으므로 3MB의 '내부단편화'가 발생할 수 있습니다. (연속 할당 방식보다는 적게 발생합니다.)

- 세그멘테이션 : 프로세스를 가변적인 크기의 세그먼트로 분리하고 메모리에 적재합니다. 주로 논리적인 블록단위 세그먼트로 Code, Data, Stack & Heap으로 분리할 수 있습니다. 세그멘테이션 역시 '외부 단편화'가 발생합니다.

- 추가적으로 페이징과 세그멘테이션 방식은 연속적이지 않으므로, 각 조각이 실제 메모리의 위치에 있는지 알아야 하므로 페이지/세그먼트 테이블이 존재합니다.