


Operating Systems 문답

# Index	7
📅 CreatedAt	
👤 Person	 Ally Hyeseong Kim
☀️ Status	In progress
🏷️ Tags	Computer Science Operating Systems
📅 UpdatedAt	

References

tech_interview.zip/OperatingSystem.md at main · 4z7l/tech_interview.zip

✔️ 취준하면서 모았던 면접 질문 모음집 ✔️. Contribute to 4z7l/tech_interview.zip development by creating an account on GitHub.

https://github.com/4z7l/tech_interview.zip/blob/main/직무/OperatingSystem.md

4z7l/
tech_interview.zip

✔️ 취준하면서 모았던 면접 질문 모음집 ✔️

Contributor 1 Issue 1 Stars 1k Forks 132

Interview_Question_for_Beginner/OS at master · JaeYeopHan/Interview_Question_for_Beginner

:boy: :girl: Technical-Interview guidelines written for those who started studying programming. I wish you all the best. :space_invader: - Interview_Question_for_Beginner/OS at master · JaeYeopHan/...

https://github.com/JaeYeopHan/Interview_Question_for_Beginner/tree/master/OS#프로세스와-스레드의-차이

Tech In

면접 시리즈5 - 운영체제

Java, JPA, Spring을 주로 다루고 공유합니다.

B <https://backtony.github.io/interview/2021-12-13-interview-20/#해결책>

References

- 프로세스와 스레드의 차이점에 대해 설명해주세요.
 - stack을 스레드마다 독립적으로 할당하는 이유는 무엇인가요?
 - PC 레지스터를 스레드마다 독립적으로 할당하는 이유는 무엇인가요?
- 멀티프로세스와 멀티스레드의 차이점에 대해 설명해주세요.
 - Java에서 사용하는 Thread는 어떤건가요?
 - Spring에서는 어떻게 되어있나요?
 - context switching에 대해 설명해주세요.
 - Java에서 스레드가 늘어나 context switching이 자주 일어나는 문제를 어떻게 해결할 수 있나요?
 - 유저 모드와 커널 모드에 대해 설명해주세요.
- 스케줄러에 대해 설명해주세요.
- CPU 스케줄러에 대해 설명해주세요.
- 동기와 비동기에 대해 설명해주세요.
- 동기화에 대해 설명해주세요.
 - 교착 상태에 대해 설명해주세요.
- 메모리 관리 전략에 대해 설명해주세요.
 - 메모리가 고갈되면 어떤 현상이 발생할까요?
 - CPU 사용률을 계속 체크하는 이유는 무엇인가요?
- 가상 메모리에 대해 설명해주세요.
 - 페이지 교체가 이루어지는 과정을 설명해주세요.
- 캐시의 지역성에 대해 설명해주세요.
 - 캐싱 라인에 대해 설명해주세요.

1. 프로세스와 스레드의 차이점에 대해 설명해주세요.

프로세스는 실행 중인 프로그램으로 고유한 공간과 자원을 할당 받아 사용합니다. 스레드는 프로세스 안에서 실행되는 여러 흐름으로 프로세스 내의 자원을 공유하고 고유한 stack을 할당받습니다.

좀 더 자세히 설명하자면,

프로세스는 함수의 매개변수, 복귀주소, 로컬변수 등의 자료를 가지는 stack, 전역변수를 가지는 data, 프로세스 실행 중에 동적으로 할당되는 heap을 가집니다. 그리고 운영체제는 프로세스를 관리하기 위해 프로세스 생성과 동시에 고유한 프로세스 제어 블록(PCB)을 생성합니다. PCB는 특정 프로세스에 대한 중요한 정보를 가지는 운영체제의 자료구조입니다. 프로세스가 CPU를 할당받아 작업을 처리하다 프로세스 전환이 발생하면 PCB에 작업 내용을 저장한 후 CPU를 반환하고 다시 CPU를 할당받으면 PCB에 저장되어있던 내용을 불러와 이전에 종료된 시점부터 재시작하게 됩니다. PCB에 저장되는 정보는, 프로세스 ID, 프로세스 상태, 프로세스가 다음에 실행할 명령어 주소인 프로그램 카운터, CPU 스케줄링 정보, 메모리 관리 정보, 입출력 상태 정보, 어카운팅 정보가 있습니다.

스레드는 스레드 ID, 프로그램 카운터, 레지스터 집합 정보를 담은 TCB와 stack을 가집니다. 스레드는 프로세스의 자원을 공유하여 중복되는 자원의 생성을 최소화해 멀티스레딩을 할 수 있습니다. 이때, 스레드는 독립적인 작업을 수행해야하므로 각자의 stack, PC 레지스터 값을 가집니다.

1.1. stack을 스레드마다 독립적으로 할당하는 이유는 무엇인가요?

stack은 함수 호출 시 인자 값, 반환 주소, 변수 등을 저장하는 메모리 공간입니다. 프로세스 안에서 스레드는 독립적인 작업 흐름을 갖기 위해서는 독립적으로 함수를 호출할 수 있어야 합니다. 따라서 스레드마다 독립적으로 stack을 할당받습니다.

1.2. PC 레지스터를 스레드마다 독립적으로 할당하는 이유는 무엇인가요?

PC 레지스터는 스레드가 어디까지 명령을 수행했는지에 대한 정보를 가지고 있습니다. 스레드는 CPU를 할당받았다가 스케줄러에 의해 다시 선점됩니다. 따라서 연속적으로 명령을 처리하지 못해 각 스레드가 어디까지 명령을 수행했는지에 대한 정보가 필요하므로 독립적으로 PC 레지스터를 할당받습니다.

3. 멀티프로세스와 멀티스레드의 차이점에 대해 설명해주세요.

멀티프로세스는 여러개의 프로세스가 동시에 실행되는 방법이고, 멀티스레드는 여러개의 스레드가 동시에 실행되는 방법입니다.

멀티프로세스와 멀티스레드는 동시에 여러 작업을 수행한다는 점에서 같지만, 멀티프로세스는 자원을 공유하지 않고 멀티스레드는 자원을 공유한다는 점에서 다릅니다.

좀 더 자세히 설명하자면,

멀티스레드는 자원을 공유하므로 멀티프로세스에 비해 메모리 공간과 시스템 자원 소모가 덜합니다. 스레드 간에 통신이 필요할 때도 전역변수나 힙 영역 등을 통해 데이터를 주고받을 수 있습니다. 따라서 멀티스레드가 멀티프로세스에 비해 자원 소모가 줄어들어 응답시간이 단축됩니다. 이러한 이유로 프로세스의 작업을 여러 스레드로 나누어 실행합니다.

반면, 스레드가 자원을 공유하기 때문에 발생하는 문제점이 있습니다. 다른 스레드가 사용하고 있는 공유 변수를 또 다른 스레드가 사용할 때 다른 스레드에 의해 값이 수정될 수 있습니다. 따라서 멀티스레드 환경에서는 동기화 작업이 필요합니다. 동기화를 통해 작업 처리 순서를 컨트롤 할 수 있습니다. 이때 과도한 락으로 인해 병목현상이 발생하지 않도록 주의해야 합니다.

멀티스레드는 커널 레벨의 스레드가 유저 레벨의 스레드를 어떻게 관리하는지에 따라 여러 모델이 있습니다. 커널 레벨의 스레드가 여러 개의 유저 레벨의 스레드를 관리하는 1:N 모델의 경우, 유저 레벨의 스레드 하나가 I/O 작업이 필요해 커널에 액세스하면, 프로세스의 다른 스레드가 같이 block 됩니다. 반면, 커널 레벨의 스레드가 하나의 유저 레벨 스레드를 관리하는 1:1 모델의 경우, 유저 레벨의 스레드가 커널에 액세스하더라도, 다른 스레드는 block되지 않습니다.

3.1. Java에서 사용하는 Thread는 어떤건가요?

Java에서 new Thread로 생성하는 객체는 사용자 레벨의 스레드이지만 커널 스레드와 1:1 연결되는 모델입니다. 따라서 자바의 스레드는 하나가 block되어도 다른 스레드가 block되지 않습니다.

자바 1.2 이전에는 JVM이 직접 스레드 스케줄링을 하는 green thread 방식을 사용해 1:N으로 연결되는 모델을 사용했습니다. 그러면 green thread 하나가 block되면 프로세스의 모든 스레드가 block되는 문제가 있었습니다. 그래서 지금은 1:1 모델을 사용해 OS의 스케줄링에 맡겨 사용합니다.

3.2. Spring에서는 어떻게 되어있나요?

Spring 5 버전 이전에는 하나의 스레드가 하나의 요청을 동기 방식으로 처리하고 그동안 block되는 멀티스레드 방식을 사용했습니다. 하지만, 스레드가 block되는 시간이 길어지면 비효율적이어서 5버전부터는 webFlux가 도입되었습니다. 노드의 싱글 스레드를 비동기로 non-block하는 방식의 장점을 가져와, webFlux는 기존의 멀티 스레드 방식을 유지하면서 각 요청에 하나의 스레드가 대응되는 것이 아니라, 싱글 스레드처럼 다수의 요청을 처리하게 되는 구조입니다.

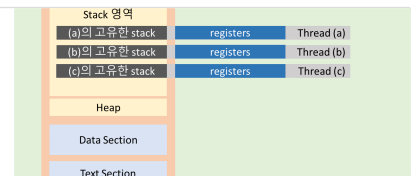
따라서, 복잡한 요구사항이 많은 실시간 게임이나 영상처리는 스프링을 사용하는 것이 유리하고, 간단한 요구사항이 많은 CRUD 기반 서비스는 node.js를 사용하는 것이 유리합니다.

3.3. context switching에 대해 설명해주세요.

Context Switching이란?

Processor가 수행하고 있는 Task(Process, Thread)의 context를 저장하고 다음 진행할 Task의 context를 읽어 수행하는 과정을 Context Switching이라고 부른다. 자세히 알아보자.

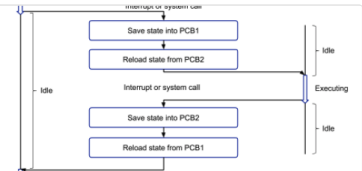
<https://velog.io/@curiosity806/Context-Switching으로-알아보는-process와-thread>



컨텍스트 스위칭이 일어나는 일

P1이 실행중이고 P2가 준비상태임. 일부 중단이 발생하면 컨텍스트를 저장한 후 프로세스 P1을 실행 중 에서 준비 상태로 전환하고 P2를 준비에서 실행중 상태로 전환해야함 언제 발생? 1. 인터럽트 : CPU는 디스크에서 데이터를 읽을 것을 요청하고, 인터럽트가 있는 경우 컨텍스트 스위칭은 인터럽트를 처리하는데

<https://applefarm.tistory.com/139>



CPU에 존재하는 레지스터들은 현재 실행 중인 프로세스나 스레드에 대한 정보로 채워집니다. 따라서 프로세스나 스레드가 스케줄러에 의해 CPU를 반환하고 새로운 프로세스나 스레드가 CPU를 할당받는 과정에서 CPU의 정보를 바꿔주는 과정을 context switching이라고 합니다.

프로세스의 context switching은 현재 상태를 전부 PCB에 저장하고 다음에 실행될 프로세스의 PCB를 복원해 해당 PCB의 PC 레지스터 값에 따라 작업을 수행합니다.

하지만 스레드의 context switching은 동일한 프로세스 안에서 발생하는 경우 공유하는 자원을 제외하고 레지스터 상태와 스택 주소를 가리키는 스택 포인터 값을 TCB에 저장하고 CPU를 반환합니다. 그리고 그 다음 실행될 스레드가 CPU를 점유해 PC 레지스터 값에 따라 해당 스레드의 작업을 수행합니다.

3.4. Java에서 스레드가 늘어나 context switching이 자주 일어나는 문제를 어떻게 해결할 수 있나요?

[Spring Boot] Tomcat 알아보기 (2) Java Thread Pool

스프링은 어떻게 동시에 수많은 요청을 처리할까요? 지난 포스팅에서는 동시 요청 처리에 대해 알아보기 전에 Tomcat의 Servlet Container 역할을 살펴보았습니다. 더보기 [Spring Boot] Tomcat 알아보기 - (1) Servlet과 Servlet Container 이번 포스팅에서는 Thread Pool의 개념과 Tomcat

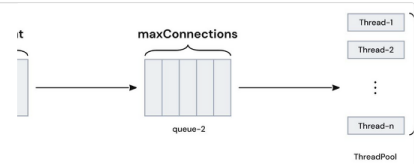
<https://e-una.tistory.com/74>



[Spring Boot] Tomcat 알아보기 (3) Tomcat Thread Pool

스프링은 어떻게 동시에 수많은 요청을 처리할까요? 지난 포스팅에서는 Servlet Container와 Java Thread Pool에 대해 알아보았습니다. 더보기 [Spring Boot] Tomcat 알아보기 (1) Servlet과 Servlet Container [Spring Boot] Tomcat 알아보기 (2) Java Thread Pool 이번 포스팅에서

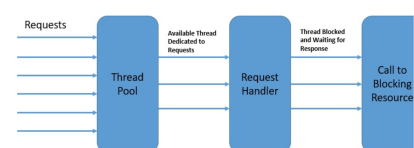
<https://e-una.tistory.com/75>



Thread per request VS EventLoop Model in Spring

Spring Web MVC는 Servlet API와 Servlet Container를 위해 만들어진 Spring 프레임워크 속 웹 프레임워크입니다. 하지만 Servlet Container(주로 Tomcat)에 의해 요청을 처리하기 때문에 요청 당 Thread 하나가 매핑

<https://velog.io/@jihoson94/EventLoopModelInSpring>



Java에서 1:1 모델을 사용하므로 context switching을 할 경우, 커널 레벨 스레드의 스위칭이 필요하므로 context switching 비용이 더 크게 발생합니다. 커널 레벨 스레드는 독립적인 스택을 가져 커널에서 직접 관리하므로 별도의 PCB가 존재합니다. 따라서 context switching이 발생할 경우, 프로세스 context switching과 유사하게 커널이 관리하는 PCB를 교체해야하므로 비용이 더 커 스레드가 많아지면 비용이 부담됩니다. 따라서 이 문제를 해결하기 위해 스레드를 미리 만들어두고 작업 큐를 사용해 작업을 처리하는 스레드 풀 패턴이 도입되었습니다.

스레드 풀은 미리 설정한 사이즈만큼의 스레드를 생성해 요청이 오면 작업 큐에 넣어두고 유휴 상태의 스레드가 있다면 큐에서 작업을 꺼내 스레드에 할당하고, 없다면 대기합니다. 만약 작업 큐가 꽉차게 되면 스레드를 새로 생성합니다. 이때, 스레드 풀의 사이즈와 작업큐의 사이즈가 최대값에 도달하면 요청을 거절합니다. Tomcat도 3.2 버전 이후부터는 스레드 풀을 사용합니다.

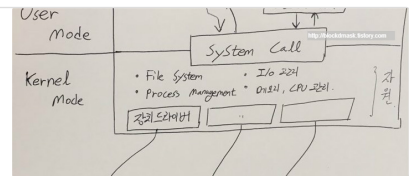
Tomcat 8.5 버전 이후부터 nonblocking I/O를 사용했는데, 그럼에도 스레드 풀 모델이 완전한 nonblocking을 만들기 어려워 요청 당 하나의 스레드의 한계에 따라 Spring 5버전부터는 Spring WebFlux가 생겼다고합니다.

3.2. 유저 모드와 커널 모드에 대해 설명해주세요.

[운영체제] 유저모드와 커널모드에 대해서.

안녕하세요, BlockDMask입니다. 오늘은 운영체제의 유저모드와 커널모드에 대해서 알아보도록 하겠습니다. 글로 먼저 설명을 하고, 그림을 통해서 설명하겠습니다. 제가 학교에서 수강했던 Unix system programming 수업과 OS 수업에서 배운 내용을 정리했습니다. 혹시 내용에 이상한 점이 있으면 댓글로 지

<https://blockdmask.tistory.com/69>



우선, 운영체제의 커널은 CPU, 메모리 등의 자원을 관리하기 위해 스케줄링, IO, 메모리 관리 등을 수행합니다. 따라서 커널은 중요한 자원을 관리하기 때문에, 이 자원에 접근하지 못하도록 유저모드와 커널모드 2가지 모드로 나뉘어져 있습니다. 유저모드는 코드를 작성하고 프로세스를 실행하는 등의 작업을 할 수 있는 영역으로 프로그램의 자원에 침범하지 못하고, 커널모드는 모든 자원에 접근하고 명령할 수 있습니다.

프로세스가 실행되는 동안 프로세스는 유저모드와 커널모드를 왔다갔다하는데, 유저모드에서 실행되다가 커널에서의 작업이 필요할 때 system call을 이용해 커널모드로 요청하고 그 요청에 대한 결과를 sysmtem call로 반환해 유저모드로 반환합니다.

4. 스케줄러에 대해 설명해주세요.

스케줄러는 프로세스에 메모리와 CPU를 할당하는 스케줄링 작업을 하는 방법으로 세가지 종류가 있습니다.

우선, 프로세스를 스케줄링하기 위해 queue를 사용하는데 모든 프로세스가 있는 작업 큐, 메모리 내에 있으며 CPU 할당을 기다리는 ready 큐, I/O 작업을 기다리는 device 큐 세가지 종류가 있습니다. 스케줄러는 이 queue에 프로세스를 넣고 빼는 작업을 하며 장기, 단기, 중기 스케줄러가 있습니다.

우선, job(장기) 스케줄러는 일반적으로 디스크에 임시로 저장되어있는 프로세스 중에 메모리를 할당하여 ready queue로 보낼 프로세스를 결정하는 역할을 합니다. 요즘의 운영체제는 프로세스가 장기 스케줄러 없이 바로 메모리를 할당해 ready queue에 넣어준다고 알고있습니다.

그리고 CPU(단기) 스케줄러는 ready queue에서 CPU를 할당해 run할 프로세스를 결정하는 역할을 합니다.

마지막으로 swapper(중기) 스케줄러는 메모리 공간 확보를 위해 프로세스를 메모리에서 디스크로 이동시키는 swapping 작업을 합니다. 이때, blocked 상태는 다른 I/O 작업을 기다리는 상태이기 때문에 스스로 ready 상태가 될 수 있지만, swapping된 프로세스는 외부적인 요인으로 중지되었기 때문에 스스로 돌아갈 수 없습니다.

5. CPU 스케줄러에 대해 설명해주세요.

CPU 스케줄러는 메모리를 할당받아 ready 큐에서 기다리는 프로세스 중에서 CPU를 할당할 프로세스를 결정하는 역할을 합니다. CPU 스케줄링 알고리즘으로는 CPU를 할당받은 프로세스가 CPU를 반납하기 전까지 뺏지 않는 방법인 비선점 방식과 CPU를 뺏을 수 있는 선점 방식이 있습니다.

우선 선점 방식으로는 남은 CPU 점유 시간이 가장 짧은 프로세스에 CPU를 먼저 할당하는 SRTF 방식과 할당 시간만큼 CPU를 할당하고 완료하지 못하면 다시 ready queue에 넣는 round robin 방식이 있습니다. SRTF 방식은 점유 시간이 긴 프로세스는 계속 작업을 완료하지 못할 수 있다는 문제가 있고, 라운드로빈 방식은 점유 시간이 너무 작아질 경우 잦은 context switching 일어날 수 있다는 문제가 있습니다. 요즘은 round robin 방식을 사용한다고 알고있습니다.

비선점 방식으로는 ready queue에 넣은 순서대로 실행하는 FCFS 방식과 CPU 점유 시간이 가장 짧은 순서대로 실행하는 SJF 방식이 있습니다. FCFS 방식은 CPU 점유 시간이 긴 프로세스가 먼저 도달해 비효율적으로 할당될 수 있다는 문제가 있고, SJF 방식은 선점 방식의 SRTF 방식과 마찬가지로 CPU 점유 시간이 긴 프로세스는 작업을 완료하지 못할 수 있다는 문제가 있습니다.

선점과 비선점 방식 모두에 사용되는 방식으로는 우선순위로 실행되는 Priority 방식이 있습니다. 이 방식은 우선순위가 낮은 경우 CPU에 할당되지 못할 수 있다는 문제가 있어 우선순위가 낮은 프로세스도 실행되도록 대기 기한을 만들어서 해결할 수 있습니다.

6. 동기와 비동기에 대해 설명해주세요.

Blocking & Non-Blocking, Sync&Async

Blocking과 Non-Blocking, Sync와 Async는 비슷한 부분이 많지만 어떤 것에 관심사를 두고 있는지에 따라 구분되는 개념이기 때문에 헷갈릴 수 있고 구분하기 어려운 개념 중 하나이다. 그래서 각각에 대한 개념과 차이점을 알아보면 정리를 해보겠다.

<https://velog.io/@sdb016/Blocking-Non-Blocking-SyncAsync>

velog

동기와 비동기는 요청 후 요청 결과를 요청하거나 반환하는 쪽에 따라 구분됩니다. 동기는 요청한 쪽에서 작업 완료 여부를 확인하고, 비동기는 요청 받은 쪽에서 작업 완료 여부를 반환합니다.

이때, blocking과 nonblocking은 제어권을 갖는 쪽에 따라 구분됩니다. blocking은 요청한 쪽에서 제어권을 갖고, nonblocking은 요청 받은 쪽에서 제어권을 갖습니다.

따라서, 동기-blocking 방식은 요청하는 쪽에서 제어권을 갖고 작업 완료 여부를 확인하므로, 요청 후 반환되는 동안 계속 대기합니다. 동기-nonblocking 방식은 요청하는 쪽에서 제어권을 갖고 요청 받은 쪽에서 작업 완료 여부를 반환하므로, 요청 후 반환되는 동안 계속 대기합니다. 이 방식은 동기-blocking 방식과 유사한데, 주로 비동기-nonblocking 방식을 추구하다가 의도치 않게 동기-blocking이 되는 경우입니다. 대표적인 예로 node.js와 MySQL인데요, Node.js는 비동기로 non-blocking하는 방식으로 구현되어 있는데, DB 호출시에는 MySQL 드라이버를 호출해 blocking 방식이 된다고 알고 있습니다.

동기-nonblocking 방식은 요청받는 곳에서 제어권을 갖고 요청하는 쪽에서 작업 완료 여부를 확인하므로, 요청 후 요청받은 쪽에서 바로 반환해 제어권을 요청한 쪽으로 넘겨 요청한 쪽에서 다른 작업을 수행하면서 계속 작업 완료 여부를 요청합니다. 비동기-nonblocking 방식은 요청받은 쪽에서 제어권을 갖고 작업 완료 여부를 반환하므로 요청 후 요청받은 쪽에서 바로 반환해 제어권을 요청한 쪽으로 넘겨 요청한 쪽에서 다른 작업을 수행하고, 요청받은 쪽에서 작업을 완료하면 반환합니다.

7. 동기화에 대해 설명해주세요.

멀티 스레딩과 같이 자원을 공유할 때 자원을 동시에 접근하게 될 경우, 동기화 문제가 발생합니다. 같은 자원을 동시에 접근할 때, 한 곳에서 자원을 변경하면 다른 곳에서도 변경될 수 있습니다.

이때, 공유되는 자원의 영역을 임계 영역이라고 하는데, 임계 영역을 동시에 사용하기 위한 방법으로는 뮤텍스와 세마포어가 있습니다. 뮤텍스는 하나의 자원에 1개의 스레드 혹은 프로세스가 접근할 수 있는 방법이고, 세마포어는 자원이 가진 count 만큼의 스레드 혹은 프로세스가 접근할 수 있는 방법입니다.

처음에는 spin lock 방식을 사용해 자원을 점유할 수 있는지를 반복해서 확인해 CPU 시간을 낭비했습니다. 하지만 현재는 자원을 점유하지 못한 프로세스나 스레드는 block되고 점유할 수 있을 때 다시 깨우는 방식을 사용하고 있습니다.

7.1. 교착 상태에 대해 설명해주세요.

교착상태는 두 개 이상의 프로세스나 스레드가 서로 자원을 기다리면서 무한히 대기하는 상태를 말합니다. 교착상태의 조건으로는 한 자원에 동시에 접근할 수 없는 상호 배제, 하나의 자원을 점유한 상태에서 다른 자원을 기다리는 점유 대기, 자원을 뺏을 수 없는 비선점, 순환적으로 자원을 대기하는 순환 대기가 있습니다.

교착 상태를 해결하는 방법은 예방, 회피, 검출 및 회복하는 방법이 있습니다.

우선 교착 상태가 일어나지 않도록 예방하는 방법은, 한 프로세스나 스레드가 여러 자원에 접근해야 할 때, 모든 자원에 동시에 요청하지 않고 필요한 자원들만 한번에 요청하는 방법이 있습니다.

교착 상태를 회피하는 방법은, 프로세스나 스레드가 자원을 접근해야 할 때, 해당 자원을 점유한 후 교착 상태가 발생하는지 확인하는 방법입니다.

교착 상태를 검출하고 회복하는 방법은, 교착 상태를 발견했을 때 교착 상태의 프로세스나 스레드를 모두 중단시키거나 자원을 선점하는 방식으로 회복할 수 있습니다.

8. 메모리 관리 전략에 대해 설명해주세요.

리눅스 커널의 이해에서 메모리 세그먼트 방식에 대한 질문입니다 | KLDP

 <https://kldp.org/node/124903>

각 프로세스는 독립된 메모리 공간을 갖고, 운영체제나 다른 프로세스의 메모리에 접근할 수 없습니다. 따라서 메모리 공간을 효율적으로 사용하기 위해 메모리 관리 전략이 필요합니다. 이때 사용되는 전략을 swapping이라고 하는데, CPU의 할당이 끝난 프로세스를 물리 메모리에서 디스크로 보내고 다른 프로세스의 메모리를 적재하는 방법입니다. swapping이 반복되면 프로세스가 할당되는 공간들이 달라 물리 메모리 상에서 자유 공간들이 생기는 단편화가 발생합니다. 이 공간을 한쪽으로 모으는 방식을 압축 방식이라고 하는데, 프로세스가 적재된 공간을 한쪽으로 모으는 과정이 비효율적입니다. 따라서 이 자유공간을 해결하기 위해 메모리 주소 공간을 나누는 방법으로 페이징과 세그멘테이션을 사용합니다.

페이징은 물리 메모리와 프로세스의 메모리 공간을 동일한 크기로 나누어 프로세스를 페이지 단위로 쪼개 물리 메모리의 쪼개진 프레임에 불연속적으로 적재하는 방법입니다. 이때, 불연속적으로 적재하므로 페이지 테이블을 사용해 물리 메모리에 적재된 프로세스의 페이지 번호와 물리 메모리에서의 시작 주소를 저장해 관리합니다. 페이징 방법은 프로세스에서 할당받은 프레임 안에 공간이 남는 내부 단편화 문제가 생길 수 있습니다. 예를 들어, 프로세스 페이지 크기가 0.5이고 프로세스가 필요

한 메모리 크기가 1.8이라고 할 때, $0.5 * 3 \leq 1.8 \leq 0.5 * 4$ 이므로 4개의 프레임에 적재되고, 마지막 프레임에 $0.2(0.5 * 4 - 1.8)$ 만큼의 여유 공간이 남게됩니다.

세그멘테이션은 프로세스의 메모리 공간을 다른 크기로 나누어 다양한 크기를 가지는 세그먼트를 물리 메모리에 적재합니다. 이때, 다양한 크기의 세그먼트를 적재하고 해제하므로 세그먼트 사이에 공간이 남는 외부 단편화 문제가 생길 수 있습니다.

요즘에는 페이지를 작게 만들면 내부단편화 문제가 해소될 수 있으므로 페이징 방식을 주로 사용합니다.

8.1. 메모리가 고갈되면 어떤 현상이 발생할까요?

프로세스들의 swapping이 잦아지면서 CPU 사용률이 하락하게 되고, 운영체제는 CPU가 놓고 있는 것을 보고 프로세스를 추가하는 스레싱 현상이 발생합니다. 이때, 스레싱이 해소되지 않으면 Out of Memory 상태가 되어 중요도가 낮은 프로세스를 종료합니다.

8.2. CPU 사용률을 계속 체크하는 이유는 무엇인가요?

특정 시점만 체크할 경우 CPU 사용률이 높아 CPU 사용률이 급격히 떨어지는 구간을 발견하지 못할 수 있습니다. 계속 체크할 경우 해당 구간을 발견할 수 있고 메모리 적재량을 같이 확인한다면 스레싱 현상을 확인해 자원을 추가적으로 배치하는 등의 해결방안을 마련할 수 있습니다.

9. 가상 메모리에 대해 설명해주세요.

가상 메모리는 프로세스 전체를 메모리에 적재하지 않아도 실행할 수 있는 방법입니다.

프로세스 전체를 물리 메모리 적재해야 할 때에는, 물리 메모리의 용량보다 큰 프로세스를 실행할 수 없었습니다. 그래서 가상 메모리를 사용하면 필요하지 않은 부분을 적재하지 않아도 프로세스를 실행시킬 수 있습니다.

가상 메모리 시스템에서는 프로세스를 디스크에서 물리 메모리에 적재하는 대신 초기에 실행하는데 필요한 부분만 적재하는 방법인 요구 페이징을 사용합니다. 페이지들은 페이지에 의해 관리되며 실행 중에 필요한 페이지를 그때마다 적재하는데, 따라서 한번도 접근되지 않은 페이지는 물리 메모리에 적재되지 않습니다.

9.1. 페이지 교체가 이루어지는 과정을 설명해주세요.

가상 메모리 시스템에서 요구 페이징 방법을 사용하므로, 필요한 페이지가 없을 경우 디스크에서 해당 페이지를 가져와야 합니다. 이때, 물리 메모리 공간이 모두 사용 중이면 페이지 교체가 필요합니다. 페이지 교체는 다음과 같은 과정을 거칩니다.

우선, 디스크에서 필요한 페이지의 위치를 찾고 빈 페이지 프레임을 찾습니다. 이때, 페이지 교체가 필요하면 페이지 교체 알고리즘을 통해 교체할 페이지를 찾아 디스크로 이동하고 페이지 테이블을 수정합니다. 그리고 새로운 페이지를 적재한 후 테이블을 수정합니다.

페이지 교체 알고리즘은 다음과 같이 정리할 수 있습니다.

우선, 가장 간단한 알고리즘으로 가장 먼저 적재된 순서대로 선택하는 FIFO 방법이 있습니다. 이때, 처음부터 활발히 사용되는 페이지가 적재되어있다면 페이지 교체가 더 많이 발생할 수 있습니다.

따라서 이 문제를 해결하기 위해 앞으로 오랫동안 사용되지 않을 페이지를 선택하는 최적 페이지 교체 방법이 있습니다. 하지만 이는 구현이 어려워 가장 근사한 알고리즘으로 마지막으로 사용된 시점에 가장 오래된 페이지를 선택하는 방법을 사용합니다.

10. 캐시의 지역성에 대해 설명해주세요.

캐시는 메모리와 CPU 사이에 위치해 속도가 느린 메모리와 속도가 빠른 CPU 사이의 속도 차이를 완화하기 위한 메모리로, 자주 사용되는 데이터를 저장합니다. 따라서 캐시의 성능은 CPU가 이후에 참조할 데이터를 많이 담고 있어 hit이 많아질수록 좋다고 측정합니다. 따라서 캐시의 적중률을 극대화하기 위해 데이터 지역성 원리를 사용합니다.

지역성이란 메모리 내의 데이터를 균일하게 접근하는 것이 아니라, 특정 부분을 어느 순간에 집중적으로 참조한다는 특성입니다. 지역성은 시간 지역성과 공간 지역성이 있는데, 시간 지역성은 최근에 참조된 주소가 곧 다시 참조된다는 특성이고, 공간 지역성은 참조된 주소의 인접한 주소가 다시 참조된다는 특성입니다. 이러한 지역성을 바탕으로 현재 접근하고 있는 주소의 근처 데이터들을 캐시에 저장한다면 캐시의 적중률을 높일 수 있습니다.

10.1. 캐싱 라인에 대해 설명해주세요.

캐시는 데이터를 바로 접근할 수 있어야 합니다. 따라서 캐시에 저장하는 데이터의 메모리 주소 등을 기록하는 태그를 저장하는데, 이 태그의 묶음을 캐싱 라인이라고 합니다.