

신입 개발자 기술면접 질문 : network



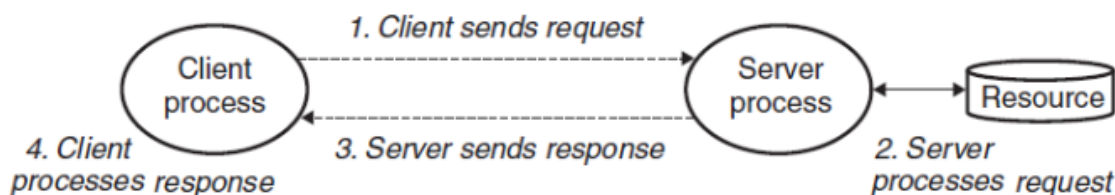
두 대 이상의 컴퓨터 간의 데이터를 주고 받을 수 있도록 프로그래밍을 하는 것을 네트워크 프로그래밍이라 한다.

인터넷 망을 통한 디바이스, 컴퓨터 간의 데이터를 주고 받는 것도 네트워크 프로그래밍이라 할 수 있다.

두 대 이상의 컴퓨터 간에 데이터를 주고 받으려면 물리적 연결이 필요한데 이러한 연결을 도울 수 있는 라이브러리를 소켓(socket)이라 한다.

두 대의 컴퓨터가 연결되면 아래 그림과 같은 상태가 된다.

클라이언트-서버 트랜잭션



클라이언트와 서버는 프로세스(실행 중인 프로그램)이며 모든 서버는 호스트이지만, 모든 호스트가 서버는 아니다.
* 호스트(host) : 네트워크 주소가 할당된 네트워크 노드. 네트워크에 연결이 확립된 모든 장치는 호스트의 자격이 있는 반면, 다른 장치(client)로부터 연결을 수락하는 호스트만 서버가 될 수 있다.

이렇게 연결이 되면 서로 데이터를 주고 받을 수 있다. 여기서 Server는 연결을 대기하는 컴퓨터이며 Client는 연결을 시도하는 컴퓨터를 의미한다. Server는 여러 Client를 연결 받아서 관리 할 수도 있다. 모든 데이터 간의 송수신 작업을 끝내고 소켓을 닫으면 두 대의 컴퓨터 연결은 해제된다.

Q&A

HTTP 프로토콜에 대해 설명해주세요.

HTTP(Hyper Text Transfer Protocol)이란 데이터를 주고 받기 위한 프로토콜이며, 서버/클라이언트 모델을 따릅니다. HTTP는 상태 정보를 저장하지 않는 Stateless의 특징과 클라이언트의 요청에 맞는 응답을 보낸 후 연결을 끊는 Connectionless의 특징을 가지고 있습니다.

HTTP (HyperText Transfer Protocol)는 웹에서 데이터를 교환하는 데 사용되는 주요 프로토콜입니다. 웹 브라우저와 웹 서버 간의 통신을 위해 설계되었으며, 대부분의 웹 트래픽을 처리합니다.

기본 작동 원리

- 클라이언트-서버 모델: HTTP는 클라이언트(보통 웹 브라우저)와 서버(웹사이트를 호스팅하는 컴퓨터) 간의 통신을 용이하게 합니다. 클라이언트는 요청(Request)을 보내고 서버는 응답(Response)을 제공합니다.
- 상태 비저장(Stateless): HTTP는 상태 비저장 프로토콜입니다. 각 요청은 독립적이며, 서버는 이전 요청의 상태를 저장하지 않습니다. 이는 웹 규모에서의 확장성을 높이지만, 상태 정보를 유지하기 위해 쿠키 같은 기술이 필요합니다.

주요 구성 요소

- 메소드(Method): 클라이언트가 수행하고자 하는 동작을 서버에 알립니다. 예를 들어, GET (자원 요청),

POST (자원 생성), PUT (자원 업데이트), DELETE (자원 삭제) 등이 있습니다.

- URL (Uniform Resource Locator): 웹상에서 자원을 식별하는 주소입니다.
- 헤더(Header): 요청이나 응답과 함께 전송되는 메타데이터로, 데이터 유형, 인증 정보 등을 포함합니다.
- 본문(Body): 실제 전송되는 데이터. 요청에서는 폼 데이터나 파일을 포함할 수 있으며, 응답에서는 클라이언트가 요청한 웹 페이지나 파일 등이 포함될 수 있습니다.

버전

HTTP/1.1: 가장 널리 사용되는 버전으로, 지속적 연결(Persistent Connections)과 파이프라이닝(Pipelining)을 지원합니다.

HTTP/2: 2015년에 표준화된 이 버전은 성능 향상을 위해 다중 요청/응답, 헤더 압축 등을 제공합니다.

HTTP/3: 최신 버전으로, 기존 TCP 대신 QUIC 프로토콜을 사용하여 지연 시간을 줄이고 성능을 향상시킵니다.

HTTP는 웹의 기본적인 동작 방식을 정의하며, 웹 기술의 핵심 구성 요소 중 하나입니다.

Q&A

HTTP와 HTTPS의 차이점은 무엇인가요?

HTTP와 HTTPS의 주요 차이점은 보안에서 나타납니다. HTTPS는 HTTP의 보안 버전이며, 다음과 같은 중요한 차이점을 가지고 있습니다:

암호화

- HTTP: 암호화되지 않고 평문으로 데이터를 전송합니다. 이로 인해 제3자가 데이터를 도청하거나 수정할 수 있어, 통신이 취약해집니다.
- HTTPS: 데이터를 전송하기 전에 암호화하여 보안을 강화합니다. 이는 SSL(Secure Sockets Layer) 또는 TLS(Transport Layer Security) 프로토콜을 사용하여 구현됩니다. 이 암호화는 데이터를 안전하게 보호하여, 중간자 공격(Man-In-The-Middle attacks)을 방지합니다.

포트

- HTTP: 기본적으로 80번 포트를 사용합니다.
- HTTPS: 기본적으로 443번 포트를 사용합니다.

인증

- HTTPS: 서버의 신원을 검증하기 위해 SSL/TLS 인증서를 사용합니다. 이 인증서는 신뢰할 수 있는 제3자(Certificate Authority, CA)에 의해 발급되며, 서버가 실제로 주장하는 기관 또는 개인임을 보증합니다.

성능

- HTTP: 암호화 과정이 없기 때문에 HTTPS보다 빠를 수 있습니다.
- HTTPS: 암호화/복호화 과정이 있어 약간의 성능 오버헤드가 발생할 수 있지만, 최신 기술과 알고리즘 덕분에 이 차이는 점점 줄어들고 있습니다.

사용 목적

- HTTP: 보안이 중요하지 않은 경우나 내부 네트워크와 같이 제어된 환경에서 사용됩니다.
- HTTPS: 온라인 거래, 개인 정보 전송, 로그인 정보 등 보안이 필요한 모든 데이터 교환에 사용됩니다.

요약하면, HTTPS는 HTTP에 비해 보안성이 훨씬 높으며, 웹에서 안전한 통신을 위해 권장됩니다.

Q&A

쿠키(Cookie)와 세션(Session)의 차이점에 대해 설명해주세요.

쿠키(Cookie)와 세션(Session)은 웹사이트가 사용자 정보를 저장하고 상태 정보를 유지하는 데 사용되지만, 그 방식과 사용 목적에 차이가 있습니다.

* 쿠키(Cookie)

정의: 사용자의 컴퓨터에 저장되는 작은 데이터 조각으로, 웹사이트가 사용자의 브라우저를 식별하고 상태 정보를 유지할 수 있게 합니다.

저장 위치: 클라이언트 측(사용자의 컴퓨터 또는 브라우저)에 저장됩니다.

수명: 설정 가능한 만료 날짜가 있으며, 만료 날짜가 지나면 자동으로 삭제됩니다. 만료 날짜를 지정하지 않으면 세션 쿠키가 되어 브라우저가 닫힐 때 삭제됩니다.

보안: 클라이언트 측에 저장되므로 중요한 정보를 저장하기에는 보안상 취약할 수 있습니다.

사용 예: 사용자의 선호도 저장, 로그인 상태 유지, 쇼핑 카트 기능 등에 사용됩니다.

* 세션(Session)

정의: 서버에서 생성되는 사용자별 고유한 식별자를 사용해 사용자의 상태 정보를 유지하는 방식입니다.

저장 위치: 서버 측에 저장됩니다. 사용자는 서버에 저장된 세션 ID를 쿠키 형태로 받아 브라우저에 저장하게 되며, 이 ID를 통해 서버는 각각의 사용자를 식별합니다.

수명: 일정 시간 동안 사용자와의 상호작용이 없으면 세션이 만료되어 삭제됩니다.

보안: 서버 측에 저장되므로 쿠키에 비해 보안이 더 강화됩니다. 중요한 정보는 서버에 저장되기 때문에 클라이언트 측의 조작이나 도난으로부터 보호됩니다.

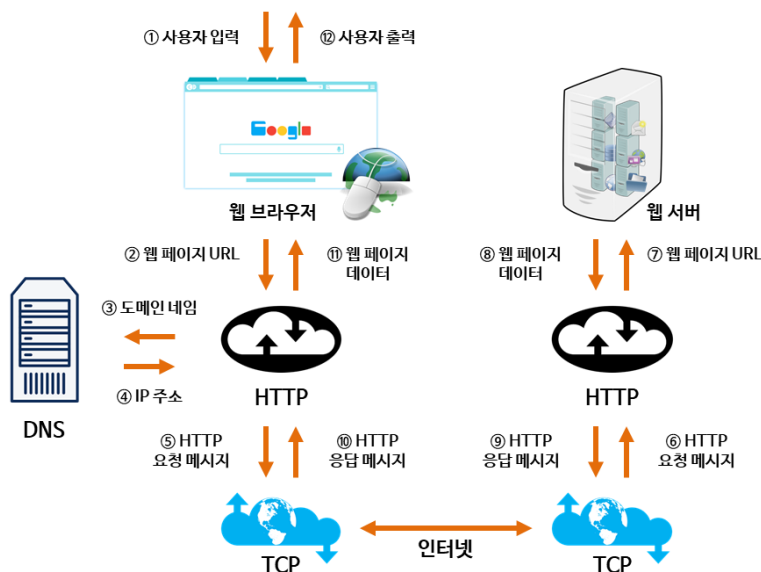
사용 예: 로그인 세션 관리, 사용자별 데이터 관리 등에 사용됩니다.

결론 : 쿠키와 세션은 웹사이트의 상태 유지 및 사용자 식별을 위해 함께 사용될 수 있습니다. 쿠키는 클라이언트 측에 정보를 저장하여 사용자 경험을 개인화하는 데 유용하지만, 중요한 정보의 보안을 위해서는 서버 측에 정보를 저장하는 세션을 사용하는 것이 바람직합니다.

	쿠키	세션
저장 위치	클라이언트(=접속자 PC)	웹 서버
저장 형식	text	Object
만료 시점	쿠키 저장시 설정, 브라우저가 종료되도, 만료 시점이 지나지 않으면 삭제되지 않음.	브라우저 종료시 삭제, 기간 지정 가능
사용하는 자원	클라이언트 리소스	웹 서버 리소스
용량 제한	총 300개 하나의 도메인 당 20개 하나의 쿠키 당 4KB(=4096byte)	서버가 허용하는 한 용량제한 없음
속도	세션보다 빠름	쿠키보다 느림
보안	세션보다 안 좋음	쿠키보다 좋음

Q&A

웹 동작 방식 이해도에 질문하고자 합니다. 예를 들어 www.korea.com에 접속할 때 생기는 과정에 대해 설명해주세요.



<https://mangkyu.tistory.com/91> 참조

이 그림은 사용자가 웹 브라우저를 통해 웹 페이지에 접속할 때 일어나는 일련의 과정을 설명하고 있습니다. 각 숫자는 웹 페이지 로딩 과정의 단계를 나타냅니다:

- 1) 사용자 요청: 사용자가 웹 브라우저에 URL을 입력하면, 웹 페이지 요청이 시작됩니다.
- 2) DNS 조회: 입력된 URL에 해당하는 서버의 IP 주소를 찾기 위해 Domain Name System(DNS) 서버에 조회를 합니다.
- 3) TCP 연결: DNS 조회를 통해 얻은 IP 주소로 TCP 연결을 시도합니다. 이 연결은 HTTP 데이터 교환의 기반이 됩니다.
- 4) HTTP 요청 메시지: 사용자의 브라우저가 웹 서버에 HTTP 요청을 보냅니다. 이 요청은 일반적으로 웹 페이지를 요청하는 GET 요청입니다.
- 5) 웹 서버: 웹 서버는 사용자의 요청을 받아 처리합니다.
- 6) HTTP 응답 메시지: 웹 서버는 요청된 웹 페이지 데이터를 HTTP 응답으로 사용자에게 보냅니다.
- 7) 웹 페이지 데이터: 웹 서버에서 전송된 데이터를 받습니다.
- 8) 웹 페이지 URL: 사용자가 요청한 웹 페이지의 URL입니다.
- 9) 사용자 브라우저: 웹 서버로부터 받은 데이터를 해석하여 사용자에게 웹 페이지로 표시합니다.

Q&A

TCP와 UDP 프로토콜의 차이를 설명해주세요.

TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)는 인터넷 프로토콜 스위트의 핵심 요소로, 데이터를 전송하는 방식에 있어 다른 접근 방식을 취합니다. 각각의 특징과 차이점은 다음과 같습니다:

1) TCP (Transmission Control Protocol)

- 연결 지향적: TCP는 데이터 전송을 시작하기 전에 통신을 위한 연결을 설정합니다. 이 과정에서 3-way handshake라고 하는 절차를 통해 연결을 확립합니다.
- 신뢰성 있는 데이터 전송: TCP는 패킷 손실, 중복, 오류를 감지하고 재전송을 요청함으로써 데이터의 정확

한 전송을 보장합니다.

- 순서 보장: 발송된 패킷의 순서대로 데이터를 재조합하여, 수신측에서 올바른 순서로 데이터를 받을 수 있도록 합니다.
- 흐름 제어 및 혼잡 제어: 네트워크의 현재 상태를 고려하여 데이터 전송 속도를 조절합니다. 이는 네트워크 혼잡을 방지하고 효율적인 데이터 흐름을 유지하는 데 도움이 됩니다.
- 사용 예: 웹 HTTP/HTTPS 통신, 이메일 전송(SMTP, POP3, IMAP), 파일 전송(FTP) 등 신뢰성이 중요한 애플리케이션에서 사용됩니다.

2) UDP (User Datagram Protocol)

- 비연결 지향적: UDP는 연결 설정 없이 데이터를 바로 전송합니다. 이는 빠른 전송을 가능하게 하지만, TCP와 달리 연결 상태를 유지하지 않습니다.
- 신뢰성 없는 데이터 전송: 패킷 손실이 발생해도 재전송을 하지 않습니다. 받는 쪽에서 패킷 손실을 처리해야 합니다.
- 순서 보장하지 않음: 패킷이 도착한 순서대로 처리되며, 발송된 순서와 다를 수 있습니다.
- 흐름 제어 및 혼잡 제어 없음: TCP와 달리 데이터 전송 속도를 조절하지 않아 네트워크 상태에 무관하게 데이터를 전송합니다.
- 사용 예: 실시간 스트리밍, 온라인 게임, VoIP(음성 통신) 등 지연 시간이 중요하거나 실시간성이 요구되는 애플리케이션에서 사용됩니다.

요약하자면, TCP는 신뢰성과 순서를 보장하는 반면에, UDP는 낮은 지연성과 오버헤드를 제공하지만 신뢰성은 낮습니다. 사용하는 애플리케이션의 요구 사항에 따라 적절한 프로토콜을 선택하게 됩니다.

프로토콜 종류	TCP	UDP
연결 방식	연결형 서비스	비연결형 서비스
패킷 교환 방식	가상 회선 방식	데이터그램 방식
전송 순서	전송 순서 보장	전송 순서가 바뀔 수 있음
수신 여부 확인	수신 여부를 확인함	수신 여부를 확인하지 않음
통신 방식	1:1 통신	1:1 OR 1:N OR N:N 통신
신뢰성	높다.	낮다.
속도	느리다.	빠르다.



Q&A

3 way-handshake와 4 way-handshake를 설명해주세요.

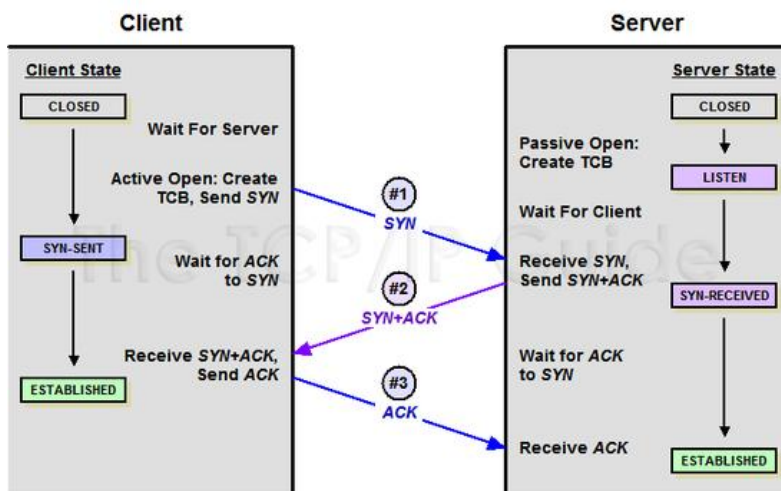
네트워크 통신에서 "3-way handshake"와 "4-way handshake"는 연결 설정 및 종료 절차에 사용되는 두 가지 다른 프로토콜입니다.

1) 3-way Handshake (TCP 연결 설정)

TCP/IP 네트워크에서 연결 지향적 통신을 수립할 때 사용되는 방법입니다. 3-way handshake는 다음과 같은 세 단계로 이루어집니다:

- SYN(Synchronize): 클라이언트가 서버에 연결을 요청하기 위해 SYN 패킷을 보내고, 자신의 초기 시퀀스 번호를 전송합니다.
- SYN-ACK(Synchronize-Acknowledgment): 서버는 SYN 패킷을 받고 클라이언트의 SYN을 확인하는 ACK 패킷과 함께 자신의 SYN 패킷을 클라이언트에게 보냅니다.
- ACK(Acknowledgment): 클라이언트는 서버의 SYN-ACK를 받고, 이를 확인하는 ACK 패킷을 서버에게 전송하여 연결이 설정됩니다.

이 세 단계를 통해 클라이언트와 서버는 서로의 존재를 확인하고, 통신을 위한 신뢰성 있는 연결을 구성합니다.



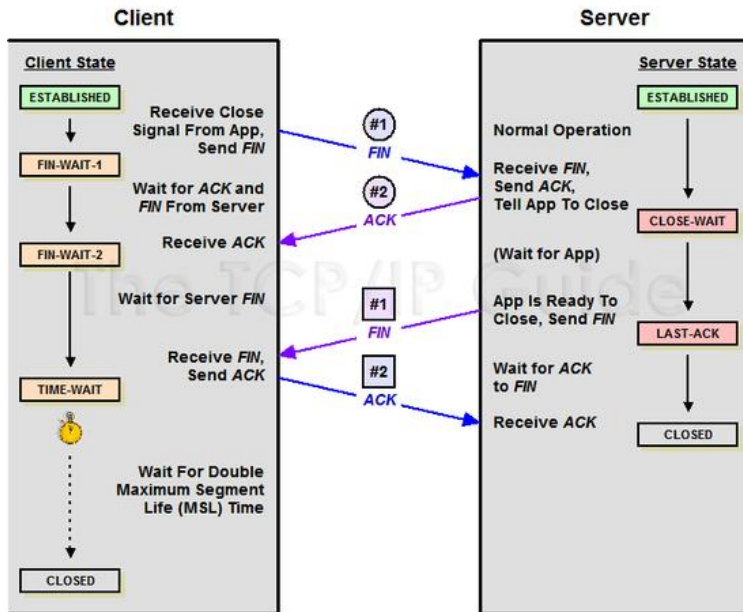
2) 4-way Handshake (무선 네트워크 보안 연결 설정)

특히 무선 네트워크에서 사용되는 WPA(Wi-Fi Protected Access)나 WPA2의 보안 프로토콜에서 보안 연결을 설정하기 위해 사용됩니다. 4-way handshake는 다음과 같은 네 단계로 이루어집니다:

- AP(Access Point) → 클라이언트: AP가 클라이언트에게 ANonce(연결할 때마다 다른 난수)를 보냅니다.
- 클라이언트 → AP: 클라이언트는 ANonce를 사용하여 PTK(Pairwise Transient Key)를 생성하고, SNonce(클라이언트가 생성한 난수)와 함께 PTK를 검증할 정보를 AP에게 보냅니다.
- AP → 클라이언트: AP는 클라이언트로부터 받은 정보를 사용하여 PTK를 생성하고, GTK(Group Temporal Key)를 전송하며, 클라이언트에게 PTK와 GTK가 설치되었다고 알립니다.
- 클라이언트 → AP: 클라이언트는 GTK를 설치하고, AP에게 PTK와 GTK의 설치 완료를 알립니다.

이 4-way handshake를 통해 클라이언트와 AP는 암호화된 통신 채널을 구축하며, 네트워크에 안전하게 연결

됩니다.



Q&A OSI 7 layer와 각 계층에 대해 아는 대로 설명해주세요.

OSI 7 Layer

TCP/IP 4 Layer

L7	응용 계층 (Application Layer)
L6	표현 계층 (Presentation Layer)
L5	세션 계층 (Session Layer)
L4	전송 계층 (Transport Layer)
L3	네트워크 계층 (Network Layer)
L2	데이터 링크 계층 (Data Link Layer)
L1	물리 계층 (Physical Layer)

L4	응용 계층 (Application Layer)
L3	전송 계층 (Transport Layer)
L2	인터넷 계층 (Internet Layer)
L1	네트워크 액세스 (Network Access Layer)

- 7 계층(응용 계층) : 사용자에게 통신을 위한 서비스 제공. 인터페이스 역할
- 6 계층(표현 계층) : 데이터의 형식(Format)을 정의하는 계층 (코드 간의 번역을 담당)
- 5 계층(세션 계층) : 컴퓨터끼리 통신을 하기 위해 세션을 만드는 계층
- 4 계층(전송 계층) : 최종 수신 프로세스로 데이터의 전송을 담당하는 계층 (단위 :Segment) (ex. TCP, UDP)
- 3 계층(네트워크 계층) : 패킷을 목적지까지 가장 빠른 길로 전송하기 위한 계층 (단위 :Packet) (ex. Router)
- 2 계층(데이터링크 계층) : 데이터의 물리적인 전송과 에러 검출, 흐름 제어를 담당하는 계층 (단위 :frame) (ex. 이더넷)
- 1 계층(물리 계층) : 데이터를 전기 신호로 바꾸어주는 계층 (단위 :bit) (장비: 케이블,리피터,허브)

Q&A HTTP Method 종류와 각 메소드의 기능에 대해 설명해주세요

HTTP 메소드는 웹 서버와의 통신에서 클라이언트가 수행하려는 동작을 정의합니다. 다음은 주요 HTTP 메소드와 그 기능에 대한 설명입니다:

GET : 리소스를 조회할 때 사용합니다. 서버에게 리소스의 정보를 요청하며, 데이터를 서버로부터 클라이언트로만 전송합니다. 데이터를 URL의 쿼리 스트링을 통해 서버에 전달할 수 있습니다.

GET 요청은 안전하며, 멍등(idempotent)입니다, 즉 같은 요청을 여러 번 해도 동일한 결과를 가져옵니다.

2) POST : 새로운 리소스를 생성할 때 사용합니다. 서버로 데이터를 전송하여 리소스를 생성하거나 업데이트합니다. POST 요청은 멍등이 아닙니다. 같은 POST 요청을 여러 번 할 경우, 여러 리소스가 생성될 수 있습니다.

3) PUT : 리소스를 업데이트할 때 사용합니다. 기존 리소스를 새 데이터로 대체합니다. PUT 요청은 멍등입니다. 같은 PUT 요청을 여러 번 하더라도 결과는 동일합니다.

4) DELETE : 리소스를 삭제할 때 사용합니다. 지정된 URI의 리소스를 제거합니다. DELETE 요청 역시 멍등으로, 요청을 여러 번 수행해도 한 번 삭제된 리소스는 변하지 않습니다.

5) HEAD : 리소스의 헤더 정보만을 조회할 때 사용합니다. GET과 비슷하지만, 리소스의 본문은 반환하지 않습니다. HEAD 요청은 안전하고 멍등입니다.

6) OPTIONS : 대상 리소스에 대한 통신 옵션을 조회할 때 사용합니다. 서버가 지원하는 메소드를 알기 위해 사용되며, CORS(Cross-Origin Resource Sharing)에서 사전 요청(preflight)으로 사용됩니다.

7) PATCH : 리소스의 부분적인 업데이트를 수행할 때 사용합니다. PUT이 전체 리소스를 대체하는 것과 달리, PATCH는 일부만 변경합니다. PATCH 요청의 멍등성은 구현에 따라 달라질 수 있습니다.

이외에도 다양한 HTTP 메소드가 있지만, 실제 웹 개발에서 가장 흔히 사용되는 것들은 위에 나열한 7가지입니다.

Q&A

세션 기반 인증과 토큰 기반 인증의 차이에 대해 얘기해주세요.

세션 기반 인증과 토큰 기반 인증은 웹 애플리케이션에서 사용자 인증을 관리하는 두 가지 접근 방식입니다.

1) 세션 기반 인증

작동 방식: 사용자가 로그인하면, 서버는 사용자의 로그인 정보를 기반으로 세션을 생성하고, 세션 ID를 발급합니다. 이 ID는 쿠키에 저장되어 클라이언트에게 반환됩니다. 사용자가 서버에 요청을 보낼 때마다 이 쿠키가 서버로 전송되고, 서버는 쿠키에 있는 세션 ID를 통해 사용자를 인증하고 필요한 정보를 세션에서 가져옵니다.

저장 위치: 사용자의 세션 정보는 서버의 메모리 또는 데이터베이스에 저장됩니다.

장점: 상태를 유지할 수 있으며, 서버에서 사용자의 상태를 정확히 제어할 수 있습니다.

단점: 서버의 자원을 많이 사용하며, 확장성 측면에서 제한이 있을 수 있습니다(세션 정보를 여러 서버 간에 공유해야 하는 경우).

2) 토큰 기반 인증

작동 방식: 사용자가 로그인하면, 서버는 사용자의 인증 정보를 검증하고, 인증 토큰(예: JWT, JSON Web Token)을 발급합니다. 이 토큰은 사용자의 인증 정보, 권한 등이 암호화되어 포함되어 있습니다. 사용자는 이 토큰을 클라이언트에 저장하고(주로 HTTP 헤더의 Authorization 필드를 사용하여), 서버로 요청을 보낼 때마다 토큰을 함께 전송합니다. 서버는 토큰을 검증하여 사용자를 인증합니다.

저장 위치: 클라이언트 측(예: 웹 스토리지, 로컬 스토리지)에 토큰을 저장합니다.

장점: 세션을 저장할 필요가 없어 서버의 자원을 덜 사용하며, 여러 시스템에서 토큰을 공유할 수 있어 확장성이 높습니다.

단점: 토큰이 탈취될 경우, 보안 문제가 발생할 수 있으며, 토큰이 만료되기 전까지는 서버 측에서 사용자의 상태를 제어하기 어렵습니다.

각 방식은 그 특성에 따라 적합한 환경과 요구 사항이 다릅니다. 세션 기반 인증은 전통적인 웹 애플리케이션에서 많이 사용되는 반면, 토큰 기반 인증은 RESTful API, SPA(Single Page Application), 모바일 애플리케이션 등 다양한 플랫폼과의 통합이 필요한 현대적인 애플리케이션 개발에 주로 사용됩니다.

Q&A

세션 기반 인증과 토큰 기반 인증은 각각 어느 경우에 적합한가요?

세션 기반 인증과 토큰 기반 인증은 서로 다른 환경과 요구 사항에 맞춰 적합한 경우가 있습니다.

* 세션 기반 인증의 적합한 경우:

- 서버 중심 애플리케이션: 서버가 사용자 상태를 관리하는 전통적인 웹 애플리케이션에서 잘 작동합니다.
- 서버 리소스 제어: 서버 측에서 사용자의 로그인 상태를 더 잘 제어하고 싶을 때 유리합니다.
- 단일 서버 환경: 애플리케이션이 단일 서버로 운영되거나, 세션 정보를 쉽게 공유할 수 있는 작은 규모의 서비스에서 적합합니다.
- 스케일 아웃보다 스케일 업: 서버의 수를 늘리기보다는 서버의 성능을 강화하는 방식으로 확장할 계획이 있을 때 좋습니다.

* 토큰 기반 인증의 적합한 경우:

- 분산 시스템: 마이크로서비스 아키텍처와 같이 여러 서비스가 분산되어 운영되는 환경에서 잘 맞습니다.
- 크로스 플랫폼: 다양한 플랫폼(웹, 모바일, IoT)에서 동일한 방식으로 인증을 처리해야 할 때 유리합니다.
- 스케일 아웃: 서버의 수를 늘려 시스템을 수평적으로 확장할 필요가 있을 때 적합합니다.
- 스테이트리스(stateless) 서버: 각 요청이 독립적으로 처리되어야 하는 RESTful API에 적합합니다.
- SSO(Single Sign-On): 여러 시스템에 걸쳐 사용자 인증을 공유해야 할 때, 토큰 기반 인증은 SSO 구현에 이상적입니다.

보안, 확장성, 관리 용이성 및 개발 편의성 등의 요소를 고려하여, 각 애플리케이션의 목적과 환경에 맞는 인증 방식을 선택해야 합니다.

Q&A

Stateful한 세션 기반의 인증 방식을 사용하게 된다면 어떠한 단점이 있을까요?

Stateful한 세션 기반 인증 방식은 서버가 사용자의 상태(세션)를 유지하고 있어야 한다는 점에서 다음과 같은 단점이 있을 수 있습니다:

- 서버 부하: 각 사용자별로 세션을 생성하고 유지해야 하기 때문에 서버의 메모리 부하가 증가할 수 있습니다. 사용자가 많아질수록 이 부담은 비례하여 증가합니다.
- 확장성 문제: 세션 정보가 특정 서버에 저장되기 때문에, 수평적 확장(스케일 아웃)이 어려워질 수 있습니다. 여러 서버 간에 세션을 공유하려면 추가적인 인프라가 필요하며, 이는 복잡성과 비용을 증가시킵니다.
- 로드 밸런싱의 복잡성: 로드 밸런서를 사용하여 트래픽을 분산할 때 세션 지속성(session persistence) 또는 스틱키 세션(sticky sessions)을 구성해야 하는 등의 추가적인 설정이 필요할 수 있습니다.
- 세션 타임아웃 관리: 세션 타임아웃은 사용자가 일정 시간 동안 활동하지 않을 때 세션을 만료시키는 방

법입니다. 이를 관리하는 것은 또 다른 도전 과제가 될 수 있습니다.

- 복구성: 서버가 다운되거나 재시작될 경우, 메모리에 저장된 세션 정보가 손실될 수 있으며, 이는 사용자에게 세션을 다시 시작하라는 요구로 이어질 수 있습니다.
- 보안 취약성: 세션 쿠키가 탈취되면, 공격자가 사용자로 위장할 수 있습니다. 세션 하이재킹(Session Hijacking)과 같은 공격에 취약할 수 있습니다.

이러한 단점들 때문에 많은 현대 애플리케이션들은 Stateless한 토큰 기반 인증 방식을 선호하고 있습니다. 하지만, 세션 기반 인증 방식도 적절히 관리하고 보안 조치를 취한다면 안전하고 효과적인 인증 수단이 될 수 있습니다.

Q&A 웹 프로그래밍에서 JWT 토큰에 대해 설명해주세요.

JWT (JSON Web Token)는 웹 프로그래밍에서 자주 사용되는 토큰 기반 인증 방식의 한 형태입니다. 이는 소규모의 정보를 JSON 객체 형태로 안전하게 전송하기 위한 컴팩트하고 독립적인 방법을 제공합니다. JWT는 주로 사용자 인증 및 정보 교환에 사용됩니다.

JWT { JSON WEB TOKEN }

With Love By
@Sec_v8

1 { "what": "JSON" }

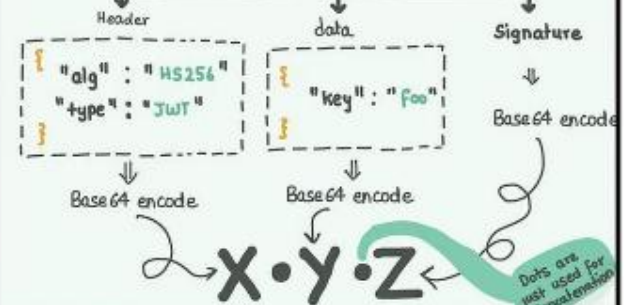
* A file format to store data in Key: value format

```
{
  "Key1": "value1",
  "Key2": ["val1", "val2"],
  "Key3": {
    "Nested Json",
    [ "Json", "Json" ]
  }
}
```

Key has to be string
can be string
or list of string or Json
another Json
or list of Json
Just a data Structure :)

2 JWT Structure

3 parts



JWT = [Header].[Data].[Signature]

X I am just Encoded Header

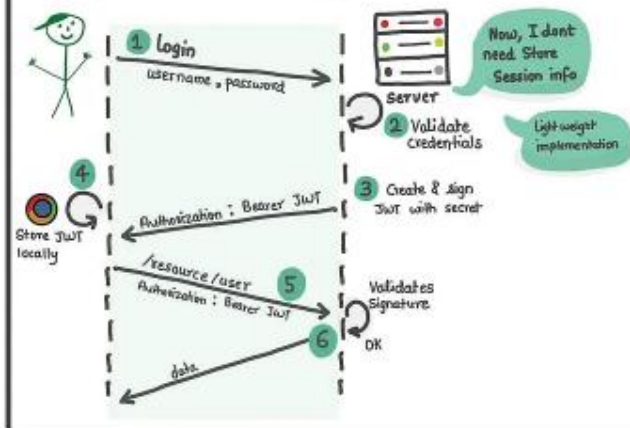
Y I am just Encoded Data

My Keys are also called "CLAIMS"

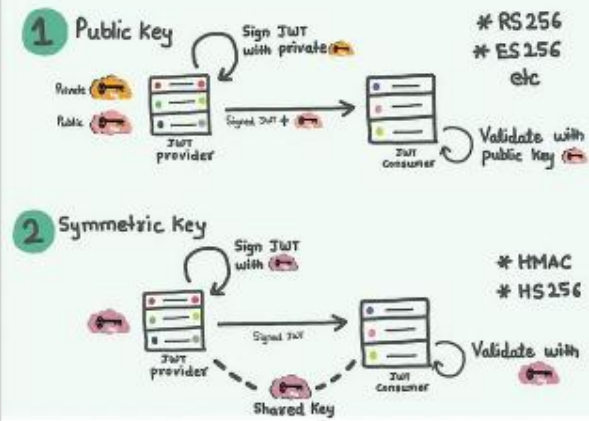
You know there are a few predefined Claims

Z I am Encoded Signature Read above below

3 Working?



4 Signing Alg



SecurityZines.com In Collaboration with ByteByteGo

<https://blog.bytebytego.com/p/ep69-explaining-json-web-token-jwt>

구조 : JWT는 세 부분으로 구성됩니다:

- 1) Header (헤더): 토큰의 유형(주로 JWT)과 해싱 알고리즘(예: HMAC SHA256, RSA) 정보를 담고 있는 JSON 객체입니다.
- 2) Payload (페이로드): 실제로 전달하고자 하는 데이터(클레임)가 담겨 있습니다. 클레임은 사용자 식별 정보, 발행자, 만료 시간 등 여러 정보를 포함할 수 있습니다.
- 3) Signature (서명): 헤더와 페이로드를 합친 후, 서버의 비밀키를 사용하여 해싱한 결과입니다. 이 서명을 통해 토큰의 무결성과 인증이 보장됩니다.

작동 방식

- 1) 토큰 생성: 사용자가 인증(로그인 등)을 요청하면, 서버는 사용자의 정보를 기반으로 JWT를 생성하여 반환합니다.

- 2) 토큰 전송: 클라이언트는 서버로부터 받은 JWT를 저장해두고, 이후 서버에 요청을 할 때 HTTP 헤더에 토큰을 포함시켜 전송합니다.
- 3) 토큰 검증: 서버는 요청에 포함된 JWT의 서명을 검증하고, 페이로드를 해석하여 사용자의 요청을 처리합니다.

장점

- 확장성: 세션 정보를 서버에 저장할 필요가 없어, 서비스의 확장성이 높아집니다.
- 자가 포함(Self-contained): JWT는 필요한 모든 정보를 자체적으로 가지고 있어, 별도의 정보 조회 없이 토큰 자체로 인증 정보를 검증할 수 있습니다.
- 언어 독립적: JSON 포맷을 사용하기 때문에, 다양한 프로그래밍 언어에서 쉽게 사용할 수 있습니다.

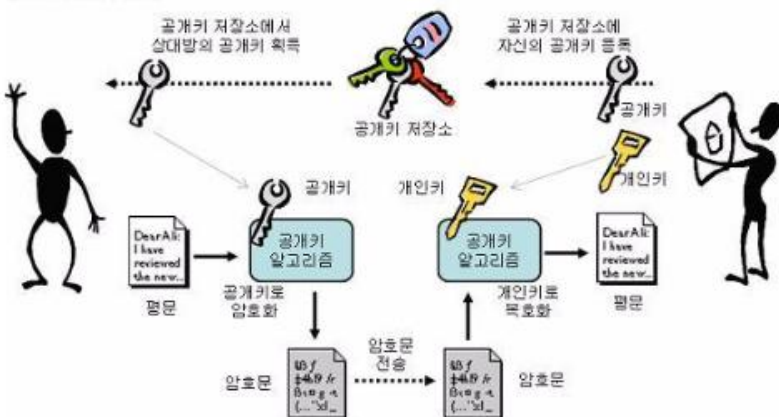
단점

- 길이: 토큰이 클레임 정보를 모두 포함하고 있어서 HTTP 헤더의 크기가 커질 수 있습니다.
- 보안: 토큰이 탈취될 경우, 탈취된 토큰의 유효 기간 동안은 보안 문제가 발생할 수 있습니다.

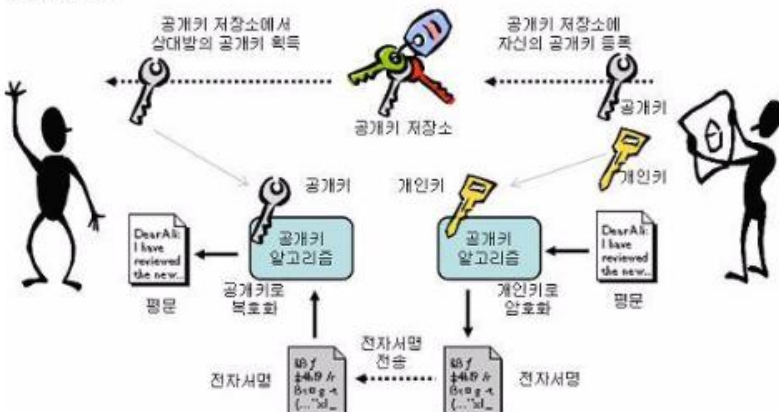
JWT는 Stateless한 인증 방식으로, 서버의 부하를 줄이고, 다양한 플랫폼에서 유연하게 인증을 처리할 수 있게 해주는 현대적인 인증 방식입니다.

Q&A 대칭키, 비대칭키 암호화 방식에 대해 설명해주세요.

공개키 암호화 과정



전자서명 과정



<http://cryptocat.tistory.com/2>

대칭키와 비대칭키는 양방향 암호화 방식이며, 대칭키는 암호화와 복호화에 같은 암호 키를 쓰는 알고리즘입니다. 이는 중간에 누군가 암호 키를 가로채면 암호화된 정보가 유출될 수 있다는 단점이 있는데, 이런 문제

를 보완한 새로운 방식이 비대칭키(공개키)입니다.

비대칭키는 암호화와 복호화할 때 키를 서로 다른 키로 사용하는 암호화 알고리즘입니다. 타인에게 절대 노출되어서는 안되는 개인키(private key)와 공개적으로 개방되어 있는 공개키(public key)를 쌍으로 이룬 형태입니다.

대칭키 암호화 방식과 비대칭키 암호화 방식은 정보를 안전하게 전송하고 저장하기 위한 두 가지 주요 암호화 방법입니다.

1) 대칭키 암호화 방식 (Symmetric Key Encryption)

정의: 대칭키 암호화에서는 데이터를 암호화하고 복호화하는 데 동일한 키를 사용합니다.

작동 원리: 암호화하려는 데이터에 대칭키를 사용하여 암호화하며, 복호화할 때도 같은 키를 사용하여 원본 데이터를 얻습니다.

장점: 처리 속도가 빠르고 효율적이며, 대량의 데이터를 암호화하는 데 적합합니다.

단점: 키를 안전하게 교환하고 관리하는 것이 어렵습니다. 암호화와 복호화에 동일한 키를 사용하기 때문에, 키가 노출되면 보안이 위협받을 수 있습니다.

예: AES(Advanced Encryption Standard), DES(Data Encryption Standard)

2) 비대칭키 암호화 방식 (Asymmetric Key Encryption)

정의: 비대칭키 암호화에서는 데이터를 암호화하는 데 공개 키를 사용하고, 복호화하는 데 개인 키(비밀 키)를 사용합니다. 공개 키와 개인 키는 수학적으로 관련되어 있지만, 한 키로부터 다른 키를 유추하는 것은 계산상 불가능합니다.

작동 원리: 보안이 필요한 통신에서 사용자는 자신의 개인 키를 비밀로 유지하고, 공개 키를 타인과 공유합니다. 데이터는 공개 키로 암호화되어 전송되며, 수신자만이 개인 키로 복호화할 수 있습니다.

장점: 키 교환 문제를 해결할 수 있습니다. 공개 키는 안전하게 공유될 수 있으며, 개인 키는 수신자만이 알고 있습니다.

단점: 대칭키 암호화 방식에 비해 처리 속도가 느립니다. 복잡한 수학 연산을 필요로 하기 때문에, 대량의 데이터를 암호화하는 데는 적합하지 않을 수 있습니다.

예: RSA, ECC(Elliptic Curve Cryptography)

결합 사용 : 실제 애플리케이션에서는 두 암호화 방식을 결합하여 사용하는 경우가 많습니다. 예를 들어, SSL/TLS 같은 보안 프로토콜에서는 키 교환을 위해 비대칭키 암호화를 사용하고, 세션 중에 데이터를 암호화하기 위해 대칭키 암호화를 사용합니다. 이러한 결합은 두 방식의 장점을 활용하면서 단점을 보완하는 효과적인 방법입니다.

Q&A

Connection Timeout과 Read Timeout의 차이에 대해 설명해주세요.

Connection Timeout과 Read Timeout은 네트워크 통신에서 시간 초과를 관리하는 두 가지 중요한 개념입니다. 이 두 종류의 타임아웃은 통신 과정에서 발생할 수 있는 다양한 문제를 처리하는 데 사용됩니다.

1) Connection Timeout

정의: Connection Timeout은 클라이언트가 서버에 연결을 시도할 때 설정된 시간 동안 연결이 성공적으로 수립되지 않으면 발생하는 타임아웃입니다.

목적: 네트워크 지연, 서버 과부하, 서버 가용성 문제 등으로 인해 연결 요청이 적시에 처리되지 않을 경우, 클라이언트가 무한정 대기하지 않고 일정 시간 후에 연결 시도를 중단하게 합니다.

사용 예: 사용자가 웹 서버에 접속을 시도할 때, 설정된 시간(예: 30초) 내에 TCP 연결이 완료되지 않으면 연결 시도를 중단합니다.

2) Read Timeout

정의: Read Timeout은 연결이 성공적으로 수립된 후, 클라이언트가 서버로부터 데이터를 읽기 시작한 시점부터 설정된 시간 동안 데이터를 수신하지 못하면 발생하는 타임아웃입니다.

목적: 서버가 요청을 받고 응답을 시작한 후에도 데이터 전송에 지연이 발생할 수 있는데, 이 경우 클라이언트가 무한정 데이터를 기다리지 않도록 합니다.

사용 예: 웹 페이지를 로드할 때, 서버가 응답을 시작한 후 설정된 시간(예: 60초) 내에 모든 데이터를 수신하지 못하면 타임아웃 오류가 발생합니다.

결론 : Connection Timeout은 연결 과정에서, Read Timeout은 연결 후 데이터 전송 과정에서 발생합니다. 이러한 타임아웃 설정은 네트워크 상황이나 서비스의 특성에 따라 적절히 조정되어야 하며, 사용자에게 더 나은 경험을 제공하고, 서버 자원을 효율적으로 관리하는 데 도움이 됩니다..

Q&A

공인(public) IP와 사설(private) IP의 차이에 대해 설명해주세요.

공인 IP는 ISP(인터넷 서비스 공급자)가 제공하는 IP 주소이며, 외부에 공개되어 있는 IP주소 입니다.

사설 IP는 일반 가정이나 회사 내 등에 할당된 네트워크 IP 주소이며, IPv4의 주소부족으로 인해 서브네팅된 IP이기 때문에 라우터(공유기)에 의해 로컬 네트워크상의 PC나 장치에 할당됩니다.

공인 IP 주소(Public IP Address)와 사설 IP 주소(Private IP Address)는 네트워크 상에서 디바이스를 식별하기 위해 사용되는 IP 주소의 두 가지 유형입니다. 이들의 주요 차이점은 다음과 같습니다:

1) 공인 IP 주소 (Public IP Address)

- 정의: 인터넷상에서 유일하게 식별되는 IP 주소입니다. 전 세계적으로 고유하며, 인터넷에 연결된 장치가 다른 네트워크와 통신할 때 사용됩니다.

- 할당: ISP(Internet Service Provider, 인터넷 서비스 제공업체) 또는 해당 기관으로부터 할당받습니다.

- 목적: 인터넷상의 다른 장치와의 통신을 가능하게 하며, 외부에서 접근할 수 있는 서버나 장치에 주로 사용됩니다.

- 예: 웹사이트 서버, 이메일 서버, 공유기의 외부 인터페이스 등

2) 사설 IP 주소 (Private IP Address)

- 정의: 특정 조직 또는 가정 내 네트워크에서만 사용되도록 예약된 IP 주소 범위입니다. 사설 IP 주소를 사용하는 장치는 인터넷상에서 직접 식별할 수 없으며, 로컬 네트워크 내에서만 고유합니다.

- 할당: 네트워크 관리자나 자동으로 DHCP(Dynamic Host Configuration Protocol)에 의해 내부 네트워크 내에서 할당됩니다.

- 목적: 내부 네트워크에서 장치들이 서로 통신하는 데 사용되며, 공인 IP 주소의 부족 문제를 완화하는 데 도움이 됩니다.

- 예: 가정이나 사무실의 컴퓨터, 프린터, 로컬 서버 등

* IP 주소 범위

공인 IP 주소: 전 세계적으로 고유한 범위

사설 IP 주소:

10.0.0.0 ~ 10.255.255.255

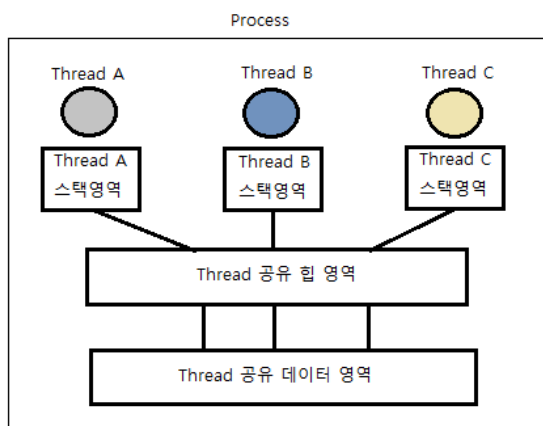
172.16.0.0 ~ 172.31.255.255

192.168.0.0 ~ 192.168.255.255

공인 IP 주소는 인터넷을 통한 전 세계적인 통신에 사용되며, 사설 IP 주소는 개별 네트워크 내에서 장치 간의 통신에 사용됩니다. 대개, 네트워크 내부에서는 사설 IP 주소를 사용하고, 인터넷으로 나가기 위해서는 NAT(Network Address Translation)를 통해 공인 IP 주소로 변환되어 통신합니다.

Q&A

프로세스와 스레드의 차이에 대해 설명해주세요.



프로세스는 실행 중인 프로그램을 말하며, 완벽히 독립적이기 때문에 메모리 영역(Code, Data, Heap, Stack)을 다른 프로세스와 공유하지 않습니다. 프로세스는 최소 1개의 스레드(메인 스레드)를 가지고 있습니다.

스레드는 프로세스 내에서 Stack만 따로 할당 받고, 그 이외의 메모리 영역(Code, Data, Heap)영역을 공유하기 때문에 다른 스레드의 실행 결과를 즉시 확인할 수 있습니다. 스레드는 프로세스 내에 존재하며 프로세스가 할당받은 자원을 이용하여 실행됩니다.

프로세스와 스레드는 운영 체제에서 실행되는 프로그램의 두 가지 기본적인 실행 단위입니다. 이들은 자원 관리와 실행 방식에서 다음과 같이 차이가 있습니다:

1) 프로세스 (Process)

정의: 실행 중인 프로그램의 인스턴스입니다. 프로세스는 독립된 메모리 영역(코드, 데이터, 스택, 힙 등)을 가지며, 운영 체제로부터 자원을 할당받아 사용합니다.

독립성: 각 프로세스는 서로 독립적인 메모리 공간을 가집니다. 한 프로세스에서 발생하는 오류나 문제가 다른 프로세스에 영향을 미치지 않습니다.

통신 방법: 프로세스 간 통신(IPC, Inter-Process Communication)을 통해 서로 정보를 교환할 수 있으며, 이는 파이프, 소켓, 공유 메모리 등을 통해 이루어집니다.

자원 사용: 프로세스는 상대적으로 많은 자원을 사용하며, 생성과 관리가 스레드에 비해 무겁고 비용이 많이 듭니다.

2) 스레드 (Thread)

정의: 프로세스 내에서 실행되는 실행 흐름의 단위로, 프로세스의 자원을 공유하여 사용합니다. 프로세스 내의 스레드들은 해당 프로세스의 메모리, 자원, 컨텍스트를 공유합니다.

공유성: 같은 프로세스 내의 스레드들은 코드, 데이터, 힙 영역을 공유하며, 각자의 스택을 갖습니다.

통신 방법: 같은 프로세스 내의 스레드들은 자연스럽게 메모리 공간을 공유하기 때문에, 별도의 IPC 메커니즘 없이 서로 통신할 수 있습니다.

자원 사용: 스레드는 프로세스에 비해 생성과 관리가 가볍고 자원을 효율적으로 사용합니다. 스레드 간의 컨텍스트 전환도 프로세스간 전환보다 빠릅니다.

결론 : 프로세스는 독립적인 실행 단위로, 각기 독립된 메모리 공간에서 실행됩니다. 스레드는 프로세스 내에서 실행되는 더 경량화된 실행 단위로, 같은 프로세스 내의 스레드들은 메모리를 공유합니다. 스레드는 자원의 효율성과 실행 속도의 향상을 위해 사용되며, 멀티쓰레딩을 통해 여러 작업을 병렬로 처리할 수 있습니다.

Q&A

멀티 프로세스와 멀티 스레드의 특징에 대해 설명해주세요.

멀티 프로세스와 멀티 스레드는 시스템의 자원을 활용하여 여러 작업을 동시에 처리하는 두 가지 접근 방식입니다. 이들의 특징은 다음과 같습니다:

1) 멀티 프로세스 (Multi-Process)

독립성: 각 프로세스는 독립된 메모리 공간을 가지며, 하나의 프로세스에서 발생한 오류가 다른 프로세스에 영향을 미치지 않습니다.

안정성: 프로세스간 격리로 인해 시스템의 안정성이 높아집니다. 하나의 프로세스에 문제가 생겨도 다른 프로세스의 실행에는 영향을 주지 않으며, 해당 프로세스만 재시작하면 됩니다.

자원 사용과 오버헤드: 각 프로세스는 별도의 메모리와 시스템 자원을 사용하기 때문에, 자원 사용량이 많고 컨텍스트 전환 시 오버헤드가 큼니다.

통신 비용: 프로세스 간 통신(IPC)에는 상대적으로 높은 비용이 들며, 통신 방법에는 파이프, 소켓, 공유 메모리 등이 있습니다.

2) 멀티 스레드 (Multi-Thread)

자원 공유: 멀티 스레드는 하나의 프로세스 내에서 메모리와 자원을 공유하므로, 스레드 간 데이터 공유가 쉽고 자원을 효율적으로 사용할 수 있습니다.

응답성: 하나의 스레드가 대기 상태에 있더라도, 다른 스레드는 계속 작업을 수행할 수 있어 시스템의 응답성이 향상됩니다.

오버헤드 감소: 스레드의 생성과 컨텍스트 전환은 프로세스에 비해 자원 소모가 적고 빠르기 때문에 오버헤드가 줄어듭니다.

위험 요소: 메모리와 자원을 공유하기 때문에, 하나의 스레드에서 발생한 문제가 전체 프로세스에 영향을 줄 수 있습니다. 따라서 동기화와 같은 문제가 중요한 고려 사항이 됩니다.

결론 : 멀티 프로세스는 프로세스간의 격리를 통해 시스템의 안정성을 높이는 반면, 자원 사용량과 오버헤드가 큰 단점이 있습니다. 반면, 멀티 스레드는 자원 공유와 오버헤드 감소의 장점을 가지고 있지만, 스레드 간의 충돌 및 동기화 문제를 적절히 관리해야 합니다. 선택은 애플리케이션의 요구 사항, 시스템의 자원, 처리해야 할 작업의 성격에 따라 달라집니다.

Q&A

멀티 스레드의 동시성과 병렬성을 설명해주세요.

멀티 스레드 프로그래밍에서 동시성(Concurrency)과 병렬성(Parallelism)은 관련이 깊지만 서로 다른 개념입니다.

다. 이 두 용어는 여러 작업을 어떻게 처리하는지에 대한 접근 방식을 설명합니다.

1) 동시성 (Concurrency)

- 정의: 동시성은 여러 작업이 시간상으로 겹쳐서 실행되는 것을 의미합니다. 단일 프로세서에서도 동시성을 구현할 수 있으며, 이는 여러 작업이 번갈아 가며 실행되어 마치 동시에 처리되는 것처럼 보이게 합니다.
- 목적: 주로 응답성을 높이고, 자원 사용을 최적화하기 위해 사용됩니다. 사용자 인터페이스, I/O 작업, 네트워크 통신 등에서 효과적입니다.
- 작동 원리: 하나의 코어에서 여러 스레드가 시분할(time-slicing)을 통해 실행됩니다. 각 스레드는 짧은 시간 동안 실행되고, 일정 시간 후에 다른 스레드로 전환되어 프로세스의 자원을 공유하면서 작업합니다.

2) 병렬성 (Parallelism)

- 정의: 병렬성은 여러 작업이 정말로 동시에 실행되는 것을 의미합니다. 이를 위해서는 여러 프로세서 또는 코어가 필요하며, 각각의 코어에서 다른 작업이 동시에 수행됩니다.
- 목적: 계산 속도를 향상시키고, 대량의 데이터 처리 작업을 빠르게 수행하기 위해 사용됩니다. 고성능 컴퓨팅, 대규모 데이터 처리, 과학적 계산에서 주로 활용됩니다.
- 작동 원리: 다중 코어 또는 CPU에서 각 스레드 또는 프로세스가 동시에 별도의 자원을 사용하여 작업을 수행합니다.

* 상호 관계 : 동시성과 병렬성은 종종 함께 사용되어 시스템의 성능을 향상시킵니다. 예를 들어, 멀티코어 시스템에서 동시에 실행되는 여러 스레드는 병렬적으로 처리할 수 있는 작업량을 증가시키며, 이는 전체적인 처리 속도를 개선합니다.

하지만 모든 동시적인 작업이 병렬적일 수 있는 것은 아니며, 작업의 종류와 의존성에 따라 실제 실행 방식이 결정됩니다.

결국, 동시성은 작업 처리의 효율성과 시스템의 응답성을 높이는 데 중점을 두고, 병렬성은 실제로 여러 작업을 동시에 수행하여 처리 성능을 향상시키는 데 초점을 맞춥니다.

Q&A

멀티 스레드 환경에서의 주의사항을 설명해주세요.

멀티 스레드 환경에서는 다양한 장점이 있지만, 주의해야 할 사항들도 있습니다. 여기에는 동기화, 데이터 공유, 성능 등에 관련된 다음과 같은 주의사항이 포함됩니다:

1. 동기화 문제

- 경쟁 상태(Race Condition): 여러 스레드가 동시에 같은 데이터에 접근하여 수정할 때 발생하는 문제입니다. 경쟁 상태를 방지하기 위해 적절한 동기화 메커니즘(예: 뮤텍스, 세마포어)을 사용해야 합니다.
- 데드락(Deadlock): 여러 스레드가 서로 다른 스레드가 보유한 자원을 기다리며 블록되는 상황입니다. 데드락을 피하기 위해서는 자원 할당 순서를 고정하거나 타임아웃을 설정하는 등의 전략이 필요합니다.

2. 자원 공유 및 관리

- 공유 자원 관리: 스레드들은 프로세스 내의 메모리와 자원을 공유합니다. 공유 자원에 대한 접근을 잘 관리해야 하며, 필요한 경우 스레드 간의 자원 접근을 동기화해야 합니다.
- 스레드 안전(Thread Safety): 공유되는 데이터 또는 객체는 스레드 안전하게 설계되어야 합니다. 스레드 안전한 객체는 여러 스레드에서 동시에 접근해도 데이터 일관성과 무결성을 유지합니다.

3. 성능 최적화

- 스레드 오버헤드: 스레드의 수가 너무 많으면 컨텍스트 전환 등으로 인한 오버헤드가 증가할 수 있습니다. 따라서 스레드의 수를 적절히 관리하고, 필요한 경우 스레드 풀을 사용하여 오버헤드를 줄일 수 있습니다.

- 부하 균형(Load Balancing): 모든 쓰레드에 균등하게 작업을 분배하여 효율적으로 시스템 자원을 활용해야 합니다.

4. 예외 처리

- 예외 처리: 쓰레드에서 발생하는 예외는 해당 쓰레드 내에서 적절히 처리해야 합니다. 예외가 다른 쓰레드로 전파되지 않도록 주의해야 합니다.

5. 메모리 관리

- 메모리 누수: 쓰레드에서 사용한 자원이나 메모리는 쓰레드 종료 시 반드시 정리해야 합니다. 메모리 누수는 시간이 지남에 따라 시스템 성능을 저하시킬 수 있습니다.

멀티 쓰레드 환경을 효과적으로 관리하기 위해서는 이러한 주의사항을 잘 이해하고, 적절한 설계와 프로그래밍 기법을 적용해야 합니다.

Q&A

콘보이 현상(convoy effect)이란 무엇이고, 콘보이 현상이 발생할 수 있는 CPU 스케줄러 알고리즘은 무엇인지 설명해주세요.

콘보이 현상이란 작업 시간이 긴 프로세스가 먼저 큐에 도착해서 다른 프로세스의 실행 시간이 전부 늦춰져 효율성을 떨어뜨리는 현상을 말합니다.

FCFS(First-Come First Served) 스케줄링은 비선점형으로, 순차적으로 먼저 큐에 들어온 작업부터 실행하므로 콘보이 현상이 발생할 수 있습니다.

FCFS(First Come First Served)

- 먼저 온 순서대로 처리한다.
- 비선점형(Non Preemptive) 스케줄링: 일단 CPU를 잡으면 CPU burst가 완료될 때까지 CPU를 반환하지 않는다. 할당되었던 CPU를 반환될 때만 스케줄링이 이루어진다.

문제점: Convoy Effect(호위 효과)

소요시간이 긴 프로세스가 먼저 도달하여 효율성을 낮추는 현상이 발생한다.

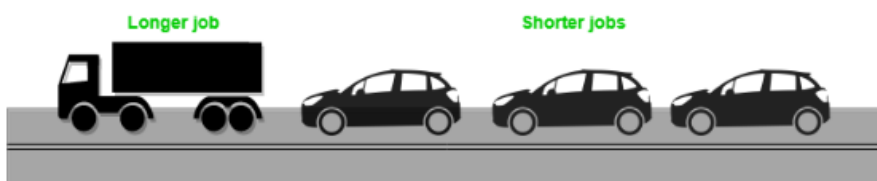


Figure - The Convoy Effect, Visualized

<https://velog.io/@leeesangheee>

콘보이 현상(Convoy Effect)은 컴퓨터 과학, 특히 멀티 프로그래밍과 CPU 스케줄링에서 발생하는 현상으로, 여러 짧은 작업이 하나의 긴 작업을 기다려야 하는 상황을 말합니다. 이 현상은 특히 선점되지 않는 스케줄링 알고리즘에서 발생할 수 있으며, 시스템의 전반적인 성능 저하로 이어질 수 있습니다.

* 콘보이 현상의 특징

- 긴 작업 우선 실행: 시스템이 긴 작업을 먼저 처리하게 되면, 그 작업이 완료될 때까지 다른 짧은 작업들이 실행을 기다려야 합니다.
- 자원 활용도 저하: 한 작업이 CPU를 독점하고 있으면, 다른 작업들은 CPU뿐만 아니라 입출력 장치와 같은 다른 자원에도 접근할 수 없게 되어 자원 활용도가 떨어집니다.

- 응답 시간 증가: 짧은 작업이긴 하나 대기하는 시간이 길어지므로 사용자나 시스템의 응답 시간이 증가합니다.

* 콘보이 현상을 유발할 수 있는 CPU 스케줄러 알고리즘

: FCFS (First-Come, First-Served): 가장 간단한 스케줄링 알고리즘으로, 먼저 도착한 프로세스를 먼저 서비스합니다. 이 알고리즘은 선점이 되지 않기 때문에, 하나의 긴 작업이 CPU를 오랫동안 점유하고 있으면, 다른 짧은 작업들이 그 작업이 끝나기를 기다려야 하므로 콘보이 현상이 발생하기 쉽습니다.

* 콘보이 현상의 해결 방안

- 시간 할당량(time quantum) 사용: 라운드 로빈(Round Robin)과 같은 선점형 스케줄링을 사용하여, 각 프로세스에게 동일한 시간 할당량을 부여하고 시간이 초과하면 다음 프로세스로 넘어가게 하여, 모든 프로세스가 공정하게 CPU 시간을 할당받을 수 있게 합니다.

- 우선순위 스케줄링: 각 프로세스에 우선순위를 부여하여 더 높은 우선순위의 작업을 먼저 처리하되, 우선순위가 낮은 작업이 무한히 대기하지 않도록 우선순위를 조정하는 방식을 적용할 수 있습니다.

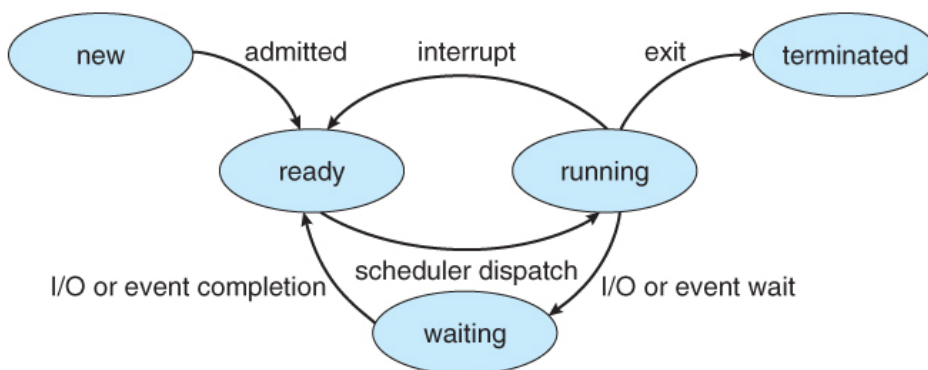
콘보이 현상은 시스템의 효율성과 공정성을 저해할 수 있으므로, 스케줄링 알고리즘을 적절히 선택하고 구현하는 것이 중요합니다.

Q&A

선점형 스케줄링과 비선점형 스케줄링의 차이를 설명해주세요.

선점형 스케줄링(Preemptive Scheduling)과 비선점형 스케줄링(Non-Preemptive Scheduling)은 운영 체제에서 프로세스 또는 스레드를 관리하는 두 가지 주요 방법입니다. 이들은 CPU 사용 시간의 배분 방식에서 차이를 보입니다.

1. 수행 → 대기 **interrupt**
2. 수행 → 준비 **time-out**
3. 대기 → 준비 **wake-up**
4. 수행 → 종료 * 대기 → 수행 **dispatch**



1) 선점형 스케줄링

- 정의: 선점형 스케줄링에서는 한 프로세스가 CPU를 점유하고 있을 때, 운영 체제가 더 높은 우선순위의 프로세스가 도착하면 현재 실행 중인 프로세스를 중단하고 새 프로세스에 CPU를 할당할 수 있습니다.

- 특징:

동적으로 CPU를 재분배할 수 있어 시스템의 반응 시간이 빠릅니다.

시스템의 자원을 보다 효율적으로 활용할 수 있으며, 긴급한 작업을 즉시 처리할 수 있습니다.

컨텍스트 스위칭(Context Switching)이 빈번하게 발생할 수 있습니다.

- 적용 예: 라운드 로빈(Round-Robin), 우선순위 스케줄링(Priority Scheduling), 멀티레벨 큐 스케줄링(Multilevel Queue Scheduling)

2) 비선점형 스케줄링

- 정의: 비선점형 스케줄링에서는 한 번 CPU를 할당받은 프로세스가 작업을 완료하거나 특정 상태(예: 입출력 대기)로 전환될 때까지 CPU를 계속 사용합니다.
- 특징:
 - 한 번 시작된 프로세스는 중간에 다른 프로세스에 의해 중단되지 않습니다.
 - 컨텍스트 스위칭 비용이 선점형에 비해 적으나, 긴급한 작업 처리에는 불리할 수 있습니다.
 - 우선순위가 높은 프로세스가 도착해도, 현재 실행 중인 프로세스가 완료될 때까지 기다려야 합니다.
- 적용 예: FCFS(First-Come, First-Served), SJF(Shortest Job First)

결론 : 선점형 스케줄링은 시스템이 더 반응적이고 유연하게 운영되도록 하지만, 컨텍스트 스위칭으로 인한 오버헤드가 증가할 수 있습니다.

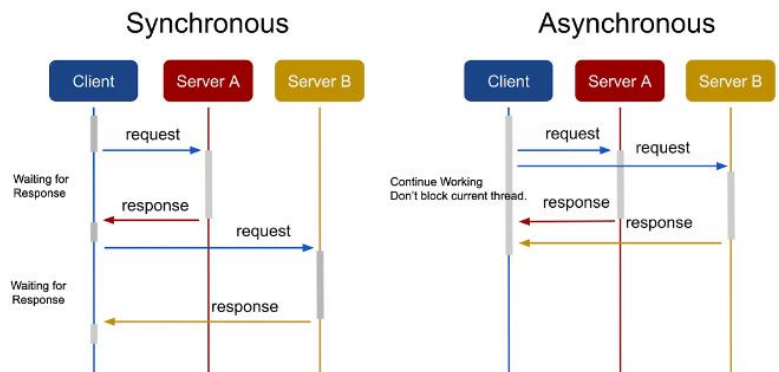
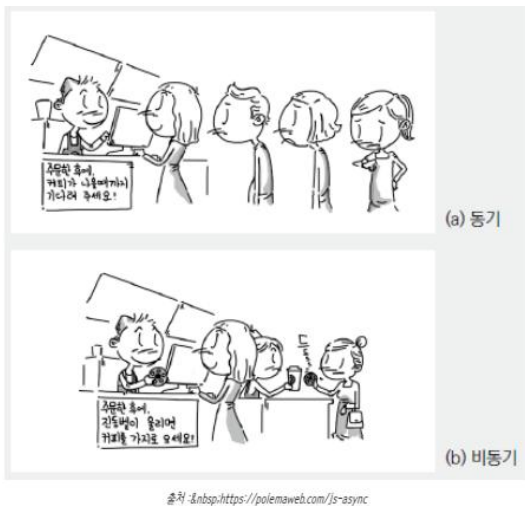
비선점형 스케줄링은 구현이 간단하고 예측 가능하지만, 특정 프로세스가 CPU를 독점할 위험이 있으며, 시스템의 전반적인 성능에 영향을 줄 수 있습니다.

적절한 스케줄링 방식의 선택은 시스템의 요구사항, 프로세스의 특성, 성능 목표 등을 고려하여 결정해야 합니다.

선점형은 하나의 프로세스가 다른 프로세스 대신에 CPU를 차지할 수 있음을 말하고, 비선점형은 하나의 프로세스가 끝나지 않으면 다른 프로세스는 CPU를 사용할 수 없음을 말합니다.

Q&A

동기와 비동기의 차이에 대해 설명해주세요.



동기는 순차적, 직렬적으로 테스크를 수행하고, 비동기는 병렬적으로 테스크를 수행합니다.

동기(Synchronous)와 비동기(Asynchronous)는 특히 컴퓨팅 및 통신에서 작업 처리 방식의 차이를 나타내는 개념입니다. 이 두 방식은 작업의 실행 시점과 완료에 대한 대응 방법에서 주요한 차이를 보입니다.

1) 동기 (Synchronous)

- 정의: 동기 방식에서는 한 작업이 완료될 때까지 기다리고, 해당 작업이 끝나야 다음 작업을 시작할 수

있습니다.

- 특징:

작업의 수행이 순차적으로 이루어집니다.

호출된 작업이 완료될 때까지 호출자는 블로킹(대기) 상태가 됩니다.

작업의 실행 과정과 결과가 예측 가능하며, 관리가 비교적 간단합니다.

- 적용 예: 전통적인 함수 호출, 데이터베이스 쿼리 실행, 파일 시스템 작업 등

2) 비동기 (Asynchronous)

- 정의: 비동기 방식에서는 한 작업을 시작한 후 완료 여부와 관계없이 즉시 다음 작업을 시작할 수 있습니다.

- 특징:

작업의 시작과 완료가 독립적으로 이루어지며, 병렬 처리가 가능합니다.

호출된 작업이 백그라운드에서 수행되며, 완료 시 콜백 함수나 이벤트, 프로미스를 통해 결과를 처리합니다.

시스템의 자원을 효율적으로 사용할 수 있으며, 응답성이 향상됩니다.

- 적용 예: 자바스크립트의 비동기 프로그래밍, 네트워크 요청, I/O 작업 등

결론 : 동기 방식은 작업 처리의 단순함과 예측 가능성에서 장점을 가지지만, 자원 활용과 응답성 측면에서 비효율적일 수 있습니다.

비동기 방식은 동시에 여러 작업을 처리할 수 있는 능력에서 장점을 가지며, 시스템의 자원을 보다 효율적으로 사용하고, 대규모 처리에 적합합니다. 하지만 작업의 관리와 처리가 복잡해질 수 있습니다.

적절한 방식을 선택하는 것은 작업의 성격, 필요한 응답 시간, 시스템의 설계와 목표에 따라 달라집니다.

Q&A

Critical Section(임계영역)에 대해 설명해주세요.

Critical Section(임계영역)은 멀티쓰레딩 환경에서 여러 스레드가 동시에 접근하면 안 되는 코드 영역을 말합니다. 임계영역은 공유 자원(데이터 구조, 파일, 하드웨어 자원 등)을 사용하는 코드 부분을 포함하며, 이 영역에서의 동시 실행은 데이터의 일관성과 무결성을 해칠 수 있습니다.

* 임계영역의 특징

- 공유 자원 접근: 임계영역은 하나 이상의 스레드가 공유 자원에 접근하는 코드를 포함합니다.

- 동시성 문제: 여러 스레드가 동시에 임계영역에 접근하려 할 때, 경쟁 상태(Race Condition)와 같은 동시성 문제가 발생할 수 있습니다.

- 동기화 필요성: 임계영역에 대한 접근을 제어하기 위해 동기화 메커니즘(예: 뮤텝스, 세마포어, 모니터 등)이 필요합니다.

* 임계영역 관리의 중요성

- 데이터 일관성 보장: 동기화를 통해 한 번에 하나의 스레드만 임계영역을 실행할 수 있도록 하여 데이터의 일관성과 무결성을 유지할 수 있습니다.

- 시스템 안정성 유지: 임계영역 관리를 통해 시스템의 안정성을 유지하고, 예기치 않은 오류를 방지할 수 있습니다.

* 임계영역 처리 방법

- 뮤텝스(Mutex): 뮤텝스는 잠금(Lock) 메커니즘을 사용하여 임계영역에 대한 접근을 단일 스레드로 제한함

니다.

- 세마포어(Semaphore): 세마포어는 뮤텍스를 일반화한 개념으로, 동시에 임계영역에 접근할 수 있는 스레드의 수를 제한할 수 있습니다.
- 모니터(Monitor): 모니터는 스레드의 동기화를 위해 사용되는 고수준의 동기화 메커니즘으로, 자동으로 임계영역의 동기화를 관리합니다.

임계영역의 적절한 관리는 멀티쓰레딩 프로그램의 올바른 동작을 위해 필수적이며, 잘못된 관리는 데이터 손실, 시스템 오류, 성능 저하 등의 문제를 일으킬 수 있습니다.

임계 영역이란 프로세스간에 공유자원을 접근하는데 있어 문제가 발생하지 않도록 한번에 하나의 프로세스만 이용하게끔 보장해줘야 하는 영역을 말합니다. 임계 영역 문제를 해결하기 위해서는 아래의 3가지 조건을 충족해야 합니다.

- 상호 배제(Mutual exclusion) - 하나의 프로세스가 임계 영역에 들어가 있다면 다른 프로세스는 들어갈 수 없어야 한다.
- 진행(Progress) - 임계 영역에 들어간 프로세스가 없는 상태에서 들어가려 하는 프로세스가 여러 개라면 어느 것이 들어갈지 결정 해주어야 한다.
- 한정 대기(Bounded waiting) - 다른 프로세스의 기아를 방지하기 위해, 한 번 임계 구역에 들어간 프로세스는 다음 번 임계 영역에 들어갈 때 제한을 두어야 한다.

Q&A

뮤텍스(Mutex)와 세마포어(Semaphore)의 차이에 대해 설명해주세요.

세마포어(Semaphore)	뮤텍스(Mutex)
<p>The state diagram of a counting semaphore.</p>	<p>The state diagram of a mutual exclusion (mutex) semaphore.</p>
- 사용 자원을 카운트하며 모든 자원이 사용되면 카운트 0	- 자원을 한 프로세스만 할당, 종료 시 다음 프로세스 할당

뮤텍스는 Lock을 사용해 하나의 프로세스나 스레드를 단독으로 실행하게 합니다.

반면에 세마포어는 공유자원에 세마포어 변수만큼의 프로세스(또는 스레드)가 접근할 수 있습니다.

세마포어의 변수 → 공유자원의 개수를 나타내는 변수

뮤텍스(Mutex)와 세마포어(Semaphore)는 멀티쓰레딩 환경에서 동기화를 위해 사용되는 두 가지 주요 메커니즘입니다. 이들은 임계 영역에 대한 접근을 제어하는 방식에서 차이를 보입니다.

1) 뮤텍스 (Mutex)

- 정의: 뮤텍스는 Mutual Exclusion(상호 배제)의 약자로, 한 번에 하나의 스레드만 임계 영역에 접근할 수

있게 하는 동기화 메커니즘입니다.

- 특징:

뮤텍스는 잠금(Lock)과 해제(Unlock)의 개념을 사용합니다.

임계 영역에 접근하는 스레드가 뮤텍스를 얻기 위해 잠그고(락), 작업이 끝나면 해제합니다.

뮤텍스는 소유 개념이 있어, 잠금을 한 스레드만이 뮤텍스를 해제할 수 있습니다.

2) 세마포어 (Semaphore)

- 정의: 세마포어는 동시에 임계 영역에 접근할 수 있는 스레드의 수를 제한하는 카운팅 메커니즘입니다.

- 특징:

세마포어는 신호(Signal)를 사용하여 리소스의 가용성을 나타냅니다.

카운팅 세마포어는 동시에 여러 스레드가 접근할 수 있도록 허용할 수 있으며, 이는 리소스의 수량을 나타낼 수 있습니다.

세마포어는 소유 없음 개념으로, 잠금과 해제가 다른 스레드에 의해 이루어질 수 있습니다.

* 뮤텍스와 세마포어의 주요 차이점

- 소유권: 뮤텍스는 잠금을 획득한 스레드가 소유하며, 그 스레드만이 뮤텍스를 해제할 수 있습니다. 반면, 세마포어는 소유 개념이 없어 어떤 스레드든 세마포어를 증가시킬 수 있습니다.

- 용도: 뮤텍스는 상호 배제를 목적으로 한 스레드 간의 동기화에 주로 사용됩니다. 세마포어는 리소스의 수량을 제어하는데 사용되며, 여러 스레드가 동시에 임계 영역에 접근할 수 있게 할 때 사용됩니다.

뮤텍스와 세마포어는 동기화를 위한 중요한 도구이지만, 사용 상황과 목적에 따라 적절히 선택하여 사용해야 합니다.

Q&A

페이지 교체 알고리즘에 대해 설명해주세요.

페이지 교체 알고리즘은 운영 체제의 메모리 관리에서 중요한 역할을 하는데, 주로 가상 메모리 시스템에서 사용됩니다. 이 알고리즘은 물리적 메모리(RAM)가 가득 찼을 때, 어떤 페이지(메모리 블록)를 교체하여 새로운 페이지를 불러올지 결정하는 방식을 정의합니다. 여기에는 여러 종류의 페이지 교체 알고리즘이 있으며, 각각의 특징과 목적이 다릅니다.

* FIFO (First-In, First-Out)

- 원리: 가장 먼저 메모리에 올라온 페이지를 가장 먼저 교체합니다.

- 특징: 구현이 간단하지만, 자주 사용되는 페이지도 교체될 수 있어 비효율적일 수 있습니다.

* LRU (Least Recently Used)

- 원리: 가장 오랫동안 사용되지 않은 페이지를 교체합니다.

- 특징: 자주 사용되는 페이지를 메모리에 유지함으로써 효율적인 페이지 관리를 할 수 있지만, 구현 복잡성이 높을 수 있습니다.

* LFU (Least Frequently Used)

- 원리: 사용 빈도가 가장 적은 페이지를 교체합니다.

- 특징: 빈도 기반으로 결정하기 때문에, 초기에 많이 사용되다가 사용되지 않는 페이지가 메모리에 남아있을 수 있습니다.

* MFU (Most Frequently Used)

- 원리: 사용 빈도가 가장 높은 페이지를 교체하는데, 이는 자주 사용되는 페이지일수록 메모리에 머무를 가능성이 높다는 가정에 기반합니다.
 - 특징: 이론적으로는 사용 빈도가 높은 페이지가 최근에 메모리에 적재되었을 가능성이 높다고 보는데, 이는 반드시 최적의 선택은 아닐 수 있습니다.
- * OPT (Optimal Page Replacement)
- 원리: 미래에 가장 오랫동안 사용되지 않을 페이지를 교체합니다.
 - 특징: 실제 시스템에서는 미래의 사용 패턴을 예측하기 어렵기 때문에 이상적인 경우에만 사용할 수 있습니다.
- * Clock Algorithm (시계 알고리즘)
- 원리: LRU의 근사로, 각 페이지에 접근 비트를 두고, 시계 바늘처럼 순회하면서 접근 비트가 0인 페이지를 교체합니다.
 - 특징: LRU보다 구현이 간단하며, 성능이 비교적 잘 나올 수 있습니다.

페이지 교체 알고리즘의 선택은 시스템의 특성, 메모리 사용 패턴, 성능 요구사항 등에 따라 달라질 수 있습니다. 알고리즘의 목표는 페이지 폴트(page fault)를 최소화하여 메모리 접근 시간을 줄이고 시스템의 전반적인 성능을 향상시키는 것입니다.

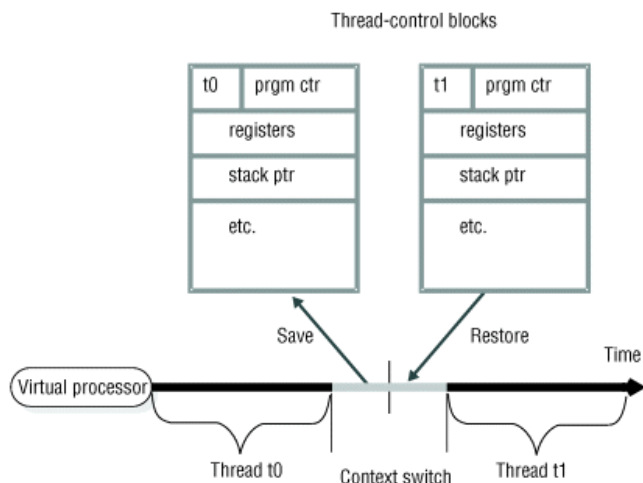
LFU와 MFU는 실제 사용에 잘 쓰이지 않습니다. 구현에 상당한 비용이 들고, 최적 페이지 교체 정책을 (LRU 만큼) 제대로 유사하게 구현해내지 못하기 때문입니다.

Q&A

컨텍스트 스위칭(Context Switching)에 대해 간략하게 핵심만 설명해주세요.

컨텍스트 스위칭(Context Switching)은 운영 체제가 CPU를 사용하는 한 프로세스(또는 스레드)에서 다른 프로세스(또는 스레드)로 전환하는 과정을 말합니다. 이 과정에서 현재 실행 중인 프로세스의 상태(컨텍스트)를 저장하고, 다음 실행할 프로세스의 상태를 불러와 CPU가 이어서 작업을 수행할 수 있도록 합니다. 컨텍스트 스위칭은 다중 프로세스 및 다중 스레드 환경에서 시스템의 동시성과 공유 자원의 효율적 사용을 가능하게 하는 핵심 메커니즘입니다.

멀티프로세스 환경에서 CPU가 어떤 하나의 프로세스를 실행하고 있는 상태에서 인터럽트 요청에 의해 다음 우선 순위의 프로세스가 실행되어야 할 때 기존의 프로세스의 상태 또는 레지스터 값(Context)을 저장하고 CPU가 다음 프로세스를 수행하도록 새로운 프로세스의 상태 또는 레지스터 값(Context)을 교체하는 작업을 말합니다.



* 컨텍스트 : CPU가 해당 프로세스를 실행하기 위한 해당 프로세스의 정보들.