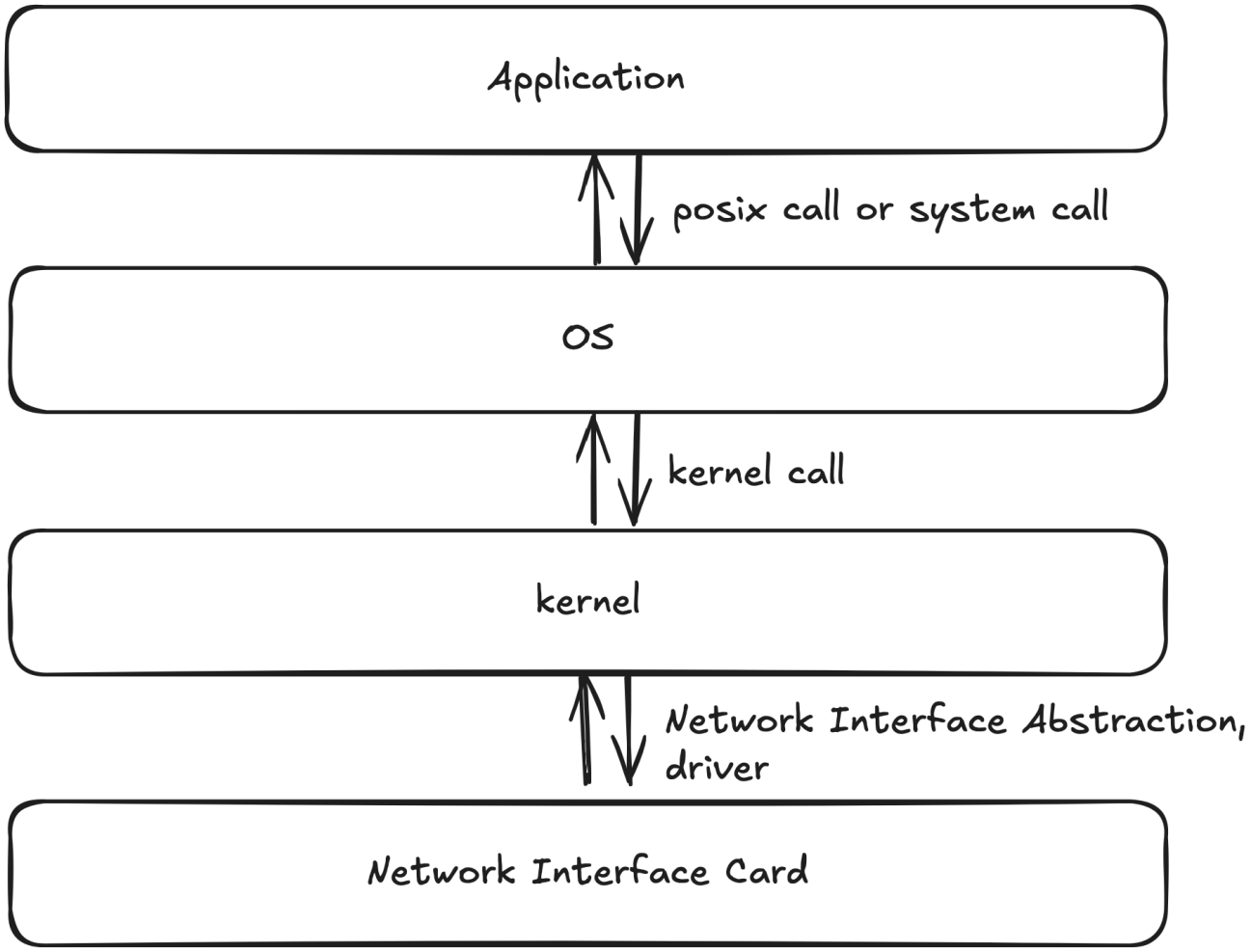
UNIX Network System Call

역사

- **초기(1960년대)**: 컴퓨터 간 통신 시스템이 없었고, 네트워크 시스템 콜의 개념은 존재하지 않음.
  - **특이점**: 프로세스 간 통신은 공유 메모리나 직접적인 하드웨어 연결을 통해서만 가능.
- **ARPANET와 TCP/IP 개발(1970~1980년대)**: TCP/IP 기반의 네트워크 프로토콜 개발로, 네트워크 시스템 호출 개념 도입. 프로세스 간 통신을 추상화하여 다양한 기기 간 연결을 지원.
  - **특이점**: TCP/IP 프로토콜과 소켓 인터페이스가 통합된 네트워크 시스템 콜이 개발되면서, 원격 기기 간 통신 가능.
- **BSD Unix와 소켓 API 등장(1983년)**: BSD Unix에서 **소켓(socket)** 개념을 도입한 네트워크 시스템 콜 등장. UNIX 시스템에서 네트워크 통신을 위한 표준 시스템 콜로 자리잡음.
  - **특이점**: 네트워크 프로그래밍을 위한 **소켓 인터페이스** 도입. 이를 통해 TCP/UDP를 사용하는 네트워크 통신이 간편해짐.
- **POSIX 표준화(1990년대)**: 네트워크 통신을 포함한 시스템 콜의 표준화. 소켓을 비롯한 네트워크 API가 POSIX 표준에 포함됨.
  - **특이점**: 다양한 운영체제 간 네트워크 시스템 콜의 일관성 확보.
- **IPv6 지원(2000년대)**: IPv4 주소 고갈 문제 해결을 위한 **IPv6 지원 시스템 콜** 추가. 기존 네트워크 시스템 콜 확장 및 새로운 콜 도입.
  - **특이점**: 기존 IPv4 시스템 콜과 호환성을 유지하면서, 대규모 주소 공간을 제공하는 IPv6 지원.
- **현대(2010년대 이후)**: 클라우드 컴퓨팅 및 컨테이너화된 환경에서 **고성능 네트워크 통신**을 지원하기 위한 **비동기 네트워크 시스템 콜**과 네트워크 성능 최적화를 위한 다양한 기술 발전.
  - **특이점**: 네트워크 성능을 극대화하기 위해 비동기 시스템 콜(`epoll`, `io_uring`)과 같은 고성능 네트워크 API 도입.

동작 과정



소켓 (Socket) 인터페이스

소켓 인터페이스는 BSD Unix에서 처음 도입된 네트워크 통신 시스템 콜의 표준 인터페이스로, 다양한 프로토콜(TCP, UDP 등)을 통해 네트워크 간 통신을 가능하게 합니다.

소켓의 동작 구조

1. **소켓 생성** (`socket()`):
   - 네트워크 통신을 위한 소켓을 생성합니다.
   - `int socket(int domain, int type, int protocol)` 로 호출되며, 소켓을 사용할 프로토콜, 통신 방식(TCP/UDP) 등을 설정.
2. **주소 할당** (`bind()`):
   - 소켓에 IP 주소와 포트를 할당
   - 서버 소켓의 경우 필수적으로 IP와 포트
3. **연결 대기** (`listen()`):

- 서버에서 클라이언트의 연결 요청을 기다리는 상태로 설정
- 연결 요청이 있을 때까지 블로킹 모드로 동작

4. **연결 수락** (`accept()`):
   - 클라이언트의 연결 요청을 수락하고, 해당 클라이언트와의 통신을 위한 소켓을 반환
5. **데이터 송수신** (`send()`, `recv()`):
   - 생성된 소켓을 통해 데이터를 송수신
   - TCP 소켓의 경우 연결 상태에서 데이터를 주고받을 수 있으며, UDP 소켓은 비연결형 방식으로 작동.
6. **소켓 종료** (`close()`):
   - 소켓을 종료하고, 해당 자원을 해제

소켓 시스템 콜과 커널 호출

1. **`socket()`**:
   - `int socket(int domain, int type, int protocol)` 을 호출하면 새로운 소켓이 생성되고, 커널에서 네트워크 프로토콜 스택에 맞는 소켓을 관리합니다.
2. **`bind()`**:
   - **`bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`** 는 소켓에 특정 IP 주소와 포트를 할당합니다.
   - 커널에서는 소켓이 사용하는 네트워크 인터페이스와 해당 주소를 연동하여 관리
3. **`listen()`**:
   - 서버 소켓이 클라이언트의 연결 요청을 받을 준비가 되었음을 알립니다.
   - **`listen(int sockfd, int backlog)`** 로 호출하며, `backlog` 는 대기할 최대 연결 수를 나타냅니다.
4. **`accept()`**:
   - 클라이언트로부터의 연결 요청을 받아들입니다.
   - **`accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`** 로 호출하며, 새로운 연결된 소켓을 반환합니다.
5. **`send()`**, **`recv()`**:
   - **`send(int sockfd, const void *buf, size_t len, int flags)`** 는 데이터를 전송하고, **`recv(int sockfd, void *buf, size_t len, int flags)`** 는 데이터를 수신합니다.
6. **`close()`**:
   - **`close(int sockfd)`** 는 사용한 소켓을 닫고, 커널에서 해당 자원을 해제합니다.

주요 네트워크 시스템 콜

1. 소켓 생성 (`socket`)

시스템 호출: `socket()`

- **정의**: `int socket(int domain, int type, int protocol)`
- 소켓을 생성하는 시스템 호출입니다.
- `domain` : 통신 영역(IPv4: `AF_INET` , IPv6: `AF_INET6` 등)
- `type` : 소켓 타입(TCP: `SOCK_STREAM` , UDP: `SOCK_DGRAM` 등)
- `protocol` : 사용 프로토콜(TCP, UDP 등)

커널 호출: `sock_create()`

- 커널에서 소켓을 생성하는 함수입니다. 사용자 모드에서 호출된 `socket()` 이 이 커널 함수를 호출합니다.

---

2. 소켓에 주소 할당 (`bind`)

시스템 호출: `bind()`

- **정의**: `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`
- 소켓에 특정 IP 주소와 포트를 할당하는 시스템 호출입니다. 서버 소켓에서 필수적으로 사용됩니다.

커널 호출: `sock_bind()`

- 커널에서 소켓에 주소를 연결하는 함수입니다. 소켓이 특정 네트워크 인터페이스와 연결되도록 설정합니다.

---

3. 연결 대기 (`listen`)

시스템 호출: `listen()`

- **정의**: `int listen(int sockfd, int backlog)`
- 서버 소켓이 클라이언트의 연결 요청을 대기할 수 있게 설정하는 시스템 호출입니다.

커널 호출: `sock_listen()`

- 커널에서 연결 요청을 처리하기 위해 소켓을 대기 상태로 전환하는 함수입니다.

---

4. 연결 수락 (`accept`)

시스템 호출: `accept()`

- **정의**: `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`
- 클라이언트 연결 요청을 수락하고, 새로운 연결된 소켓을 반환하는 시스템 호출입니다.

커널 호출: `sock_accept()`

- 커널에서 클라이언트 연결을 처리하고, 해당 연결에 대한 새로운 소켓을 생성하는 함수입니다.

5.데이터 송수신 (`send`, `recv`)

시스템 호출: `send()`, `recv()`

- **정의**:
  - `send()`: `ssize_t send(int sockfd, const void *buf, size_t len, int flags)`
  - `recv()`: `ssize_t recv(int sockfd, void *buf, size_t len, int flags)`
- 소켓을 통해 데이터를 전송(`send`)하거나 수신(`recv`)하는 시스템 호출입니다.

커널 호출: `sock_sendmsg()`, `sock_recvmsg()`

- 커널에서 소켓을 통해 데이터를 송수신하는 함수입니다.