**컴네 실습**

**1) 일방적인 종료를 했을 때 생길 수 있는 문제점 (7장)**

일방적인 종료란 tcp 연결을 하게되면 입력, 출력stream해서 2개가 만들어지는데 이 두개를 종료시키는 것이다. 일방적인 종료의 가장 큰 문제점은 상대방의 상태에 상관없이 close함수를 호출한다는 것인데, 만약 호스트 B가 보낼 데이터가 남아있었음에도 불구하고 호스트 A가 close 함수를 호출하면 그 이후에는 데이터를 받을 수 없다.

=> 이것을 보완하기 위해 half close가 나옴

\*half close:

입력, 출력 stream 중 하나만 종료

shutdown 함수 사용한다.

ex) 호스트 A가 왼쪽이라고 하면, 종료할 때 입력 버퍼로 받는 stream은 남겨두고 입출력 stream 2번째것(write쪽)만 닫아서 host B에게 EOF를 보낸다. EOF의 뜻은 호스트 A가 출력을 끝냈다는 걸 의미해서 호스트 B가 그것을 인지하게 하고 자신에게 남아있는 보낼 data를 보낸 뒤 호스트 B도 종료되게 한다.

shutdown으로 출력 스트림만 종료해도 됨.

**2) 다중 접속 서버**

멀티프로세스 기반 서버:

>> 다수의 프로세스를 생성하는 방식으로 서비스 제공.

클라이언트 여러 개가 연결 요청을 하게 되면 각 client마다 process를 새로 생성해서 데이터 송수신을 하게 하는 방법

멀티플렉싱 기반 서버:

>> 입출력대상을 묶어서 관리하는 방식으로 서비스 제공

소켓 여러 개를 하나의 프로세스에서 select 함수를 써서 쉽게 반복적으로 관리를 잘 해서 여러 개의 client한테 서비스를 제공한다.

멀티쓰레딩 기반 서버:

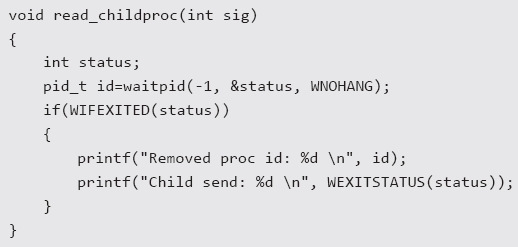
>>클라이언트의 수만큼 쓰레드를 생성하는 방식

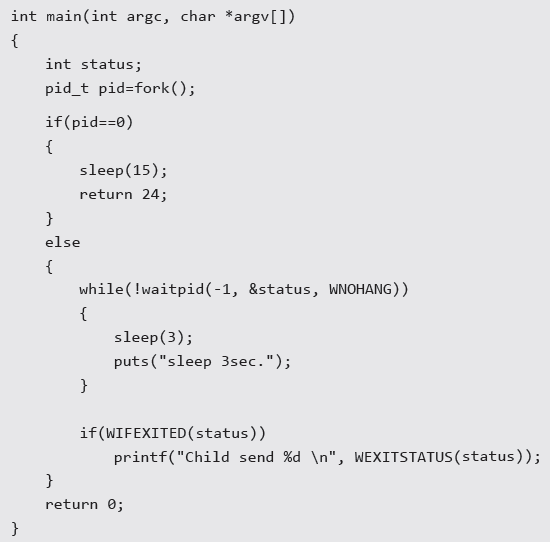
client의 수만큼 process가 아닌 thread를 생성해 데이터 송수신을 함

3) 좀비프로세스의 생성원인

자식 프로세스가 종료되면서 반환하는 상태 값이 부모 프로세스에게 전달되지 않으면 해당 프로세스는 소멸되지 않고 좀비가 된다. 그래서 resource를 계속 잡아먹고 있다. => wait 함수 사용

**4) 좀비프로세스를 없애는 필수 코드**

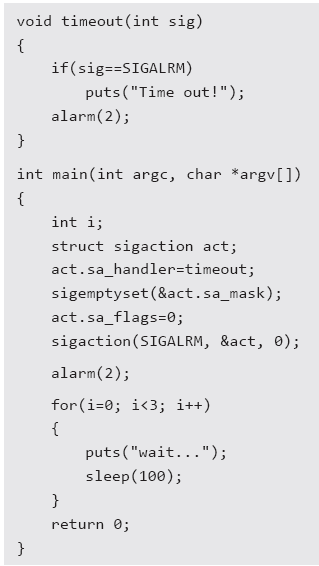




**5) 시그널**

signal(SIGCHLD, mychild): 자식 프로세스가 종료되면 mychild 함수를 호출해달라

signal보다 sigaction 더 많이 씀



**6) 멀티 프로세싱 서버의 과정**

부모프로세스에는 서버 소켓을 가지고 있다. 서버소켓에는 각각의 client들이 요청을 하게되면 그 연결 요청이 도착하는 게 서버 소켓이고 연결을 수락해주면 부모프로세스는 새로 프로세스를 만들어 그 자식 프로세스들이 각각의 client와 연결되어 송수신을 할 수 있도록 함

**7) 입출력 루틴 분할**

process 만들어서 어떤 프로세스는 read만 하고, 다른 프로세스는 write만 하게 한다.

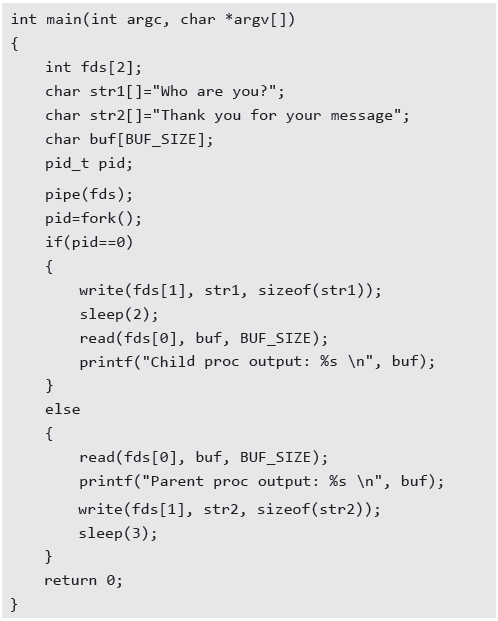
장점: 구현의 용이성

<11장>

**8) pipe2**

(파이프 함수를 통해 메모리 공간을 만들고 파일 디스크립터를 통해 그 메모리에 접근한다.)

하나의 파이프를 가지고 부모-자식이 서로 data를 읽고 쓰기 모두 하려고 했을 때 발생하는 문제점



하나의 파이프를 이용해서 양방향 통신을 하는 경우, 데이터를 쓰고 읽는 타이밍이 매우 중요해진다. 그런데 이를 컨트롤 하는 것은 사실상 불가능하기 때문에 이는 적절한 방법이 될 수 없다. 왼쪽의 예제에서 sleep 함수의 호출문을 주석처리 해 버리면 문제가 있음을 쉽게 확인할 수 있다.

9) waitpid의 options = WNOHANG 의미 <10장>

wait 함수는 블로킹 상태에 빠질 수 있는 반면, waitpid 함수는 블로킹 상태에 놓이지 않게끔 할 수 있다는 장점이 있다.

waitpid(pid\_t pid, int \*statloc, int options)에서 options 자리에 WNOHANG을 인자로 주면 종료된 자식 프로세스가 존재하지 않아도 blocking 상태에 있지 않고 0을 반환하면서 함수를 빠져나온다. 그러면 다른 일을 할 수 있게 됨

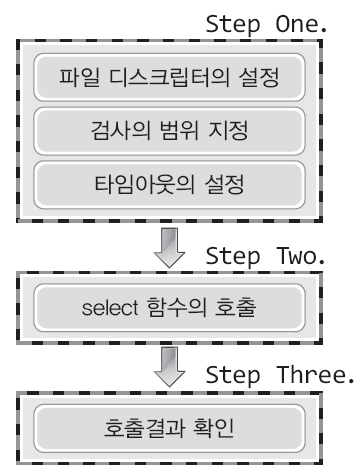
<12장>

**10) 멀티 프로세스 서버의 단점**

프로세스의 빈번한 생성은 context switching이 많이 일어나기 때문에 성능의 저하로 이어진다.

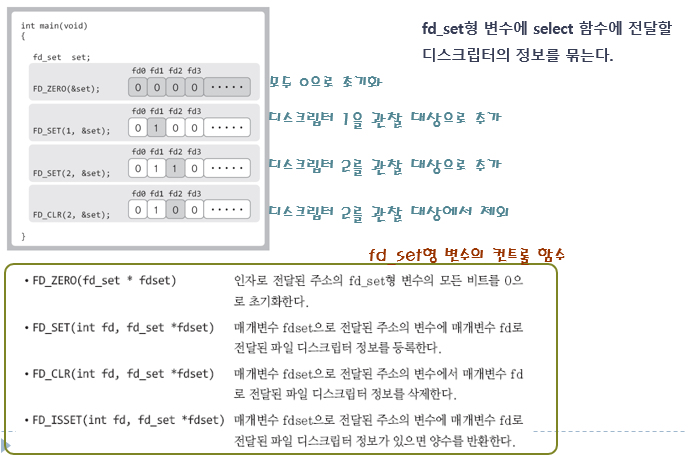
어떤 프로세스가 먼저 실행이 될지 흐름을 알기 힘들기 때문에 구현의 어려움

**11) select함수의 step**

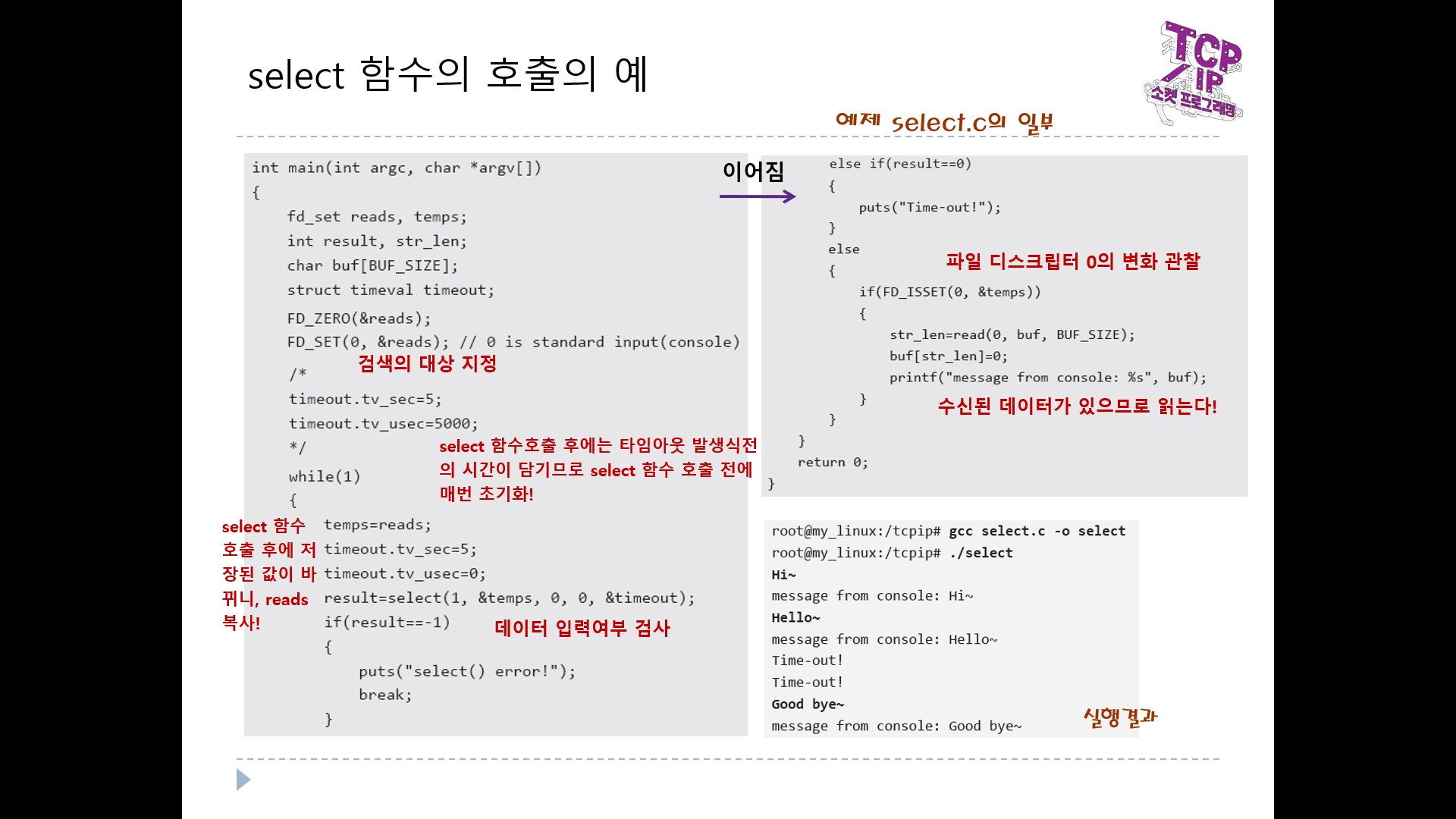


select함수를 쓰고나서 fd set이 어떻게 변했는지

각각의 메크로 함수는 어떤 의미를 가지고 있는지



select 함수를 쓰면 값이 변하기 때문에 실제로 select함수의 인자로 넘겨주는 것은 copy된 fdset을 넘겨줌.



timeout을 어디에 넣어주면 될지!

**12) process vs thread**

<processing>

단점 1. 프로세스의 부담

프로세스의 생성에는 많은 리소스가 소모된다.

일단 프로세스가 생성되면, 프로세스간의 컨텍스트 스위칭으로 인해서 성능이 저하된다.

컨텍스트 스위칭은 프로세스의 정보를 하드디스크에 저장 및 복원하는 일이다.

단점 2. 데이터 교환이 어렵다

프로세스간 메모리가 독립적으로 운영되기 때문에 프로세스간 데이터 공유 불가능!

따라서 운영체제가 별도로 제공하는 메모리 공간을 대상으로 별도의 IPC 기법 적용 -> pipe 사용

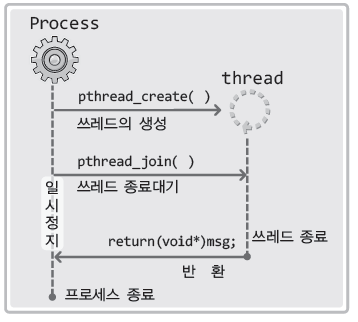
<thread>

프로세스보다 가벼운, 경량화된 프로세스이다. 때문에 컨텍스트 스위칭이 빠르다.

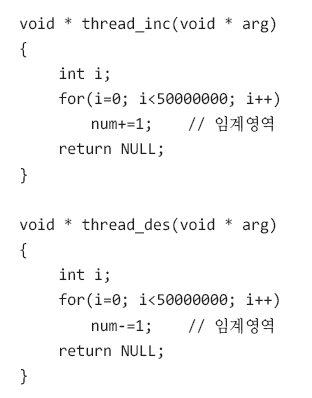
쓰레드 별로 메모리 공유가 가능하기 때문에 별도의 IPC 기법 불필요

**13) thread의 lifecycle**

pthread\_join의 사용해서 pthread를 잘 받아줘야한다.



**14) 임계영역**

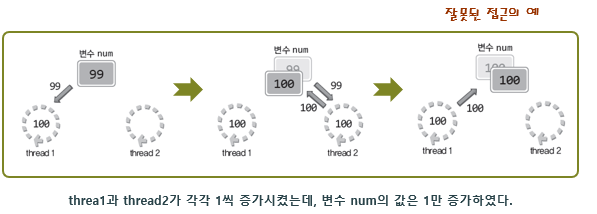


임계영역은 둘 이상의 쓰레드가 동시에 실행하면 문제를 일으키는 영역이다. 왼쪽에서 보이는 바와 같이, 서로 다른 문장임에도 불구하고 동시에 실행이 되는 상황에서도 문제는 발생할 수 있기 때문에 임계영역은 다양하게 구성이 된다.

>> int sum이라는 전역변수에 2개의 thread가 동시에 접근해 메모리를 공유할 수 있었다.

>> context switching이 언제 일어날지 알 수 없기 때문에 2개 이상의 thread가 전역변수에 동시 접근할 때 문제가 발생한다.

**15) 둘 이상의 thread 동시접근 문제점**



정상적인 접근이라면 num에 0이 들어가야 하는데 동시접근으로 인해 문제가 생긴다.

변수 값을 나에게 복사해서 가져와서 값을 변환시키고 다시 전해준다.

99인 값에 1을 더해서 100을 가지고 있는데 저장하기 전에 thread2가 우선권을 가져가서 다른 작업을 하면 우리가 원했던 결과와 다른 값이 들어갈 수 있다.

따라서 공유변수에 접근할 때는, 동시 접근을 할 때는, synchronize를 잘 맞춰 줘야한다

**16) 동기화가 필요한 대표적인 상황**

1. 동일한 메모리 영역으로의 동시접근이 발생하는 상황

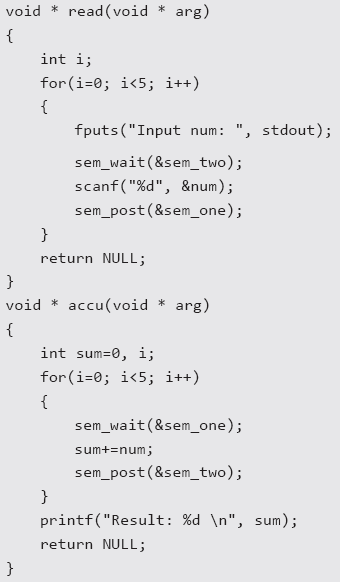
2. 동일한 메모리 영역에 접근하는 쓰레드의 실행순서를 지정해야 하는 상황

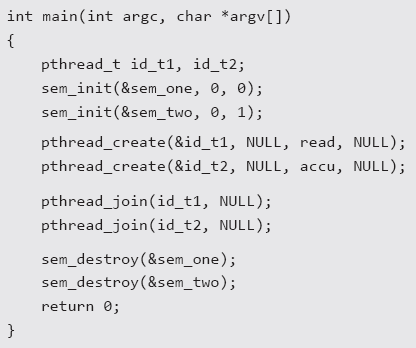
=> mutex, semaphore 사용함

**17) semaphore vs mutex**

post: 값에 +1

pend: 값에 -1





sem\_wait, sem\_post, 인자들이 어느 위치에 있는지 알기