

---

# **Network Programming 2019 Seminar**

## **5. TCP Client/Server Example**

---

# Network Programming

## 2019 Seminar

---

### 목차

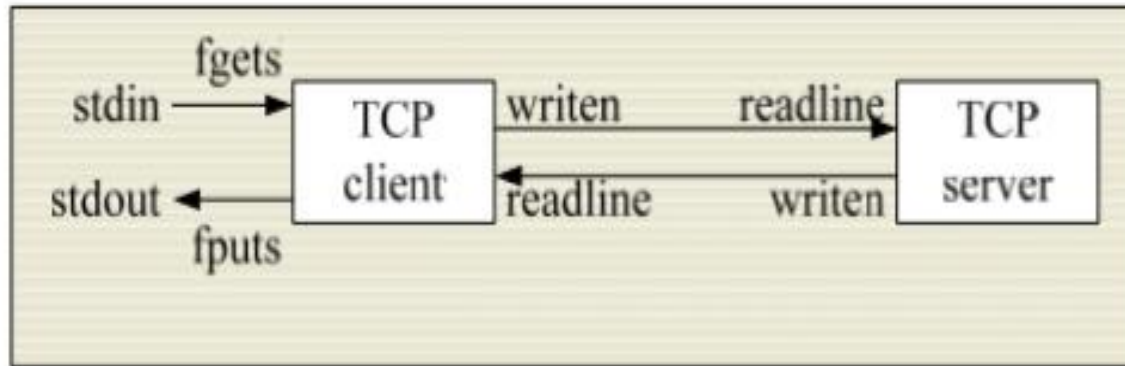
- 5.1 Introduction
- 5.2 TCP Echo Server: main Function
- 5.3 TCP Echo Server: str\_echo Function
- 5.4 TCP Echo Client: main Function
- 5.5 TCP Echo Client: str\_cli Function
- 5.6 Normal Startup
- 5.7 Normal Termination
- 5.8 POSIX Signal Handling
- 5.9 Handling SIGCHLD Signals
- 5.10 wait and waitpid Functions
- 5.11 Connection Abort before accept Returns
- 5.12 Termination of Server Process
- 5.13 SIGPIPE Signal
- 5.14 Crashing of Server Host
- 5.15 Crashing and Rebooting of Server Host
- 5.16 Shutdown of Server Host

# Network Programming

## 2019 Seminar

### 5.1 Introduction

Simple echo client and server



1. 클라이언트는 표준 입력장치로부터 문자열을 읽어 서버로 보낸다.
2. 서버는 입력된 행을 읽고 다시 클라이언트로 읽은 행을 재전송한다.
3. 클라이언트는 서버가 돌려준 행을 읽고 모니터로 출력한다.

Echo Server/client 를 배우는 이유?

# Network Programming

## 2019 Seminar

### 5.2 TCP Echo Server: main Function

<tcpserv01.c>

```
1 #include      "lnp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int      listenfd, connfd;
6     pid_t    childpid;
7     socklen_t clilen;
8     struct sockaddr_in cliaddr, servaddr;
9     listenfd = Socket (AF_INET, SOCK_STREAM, 0);
10    bzero(&servaddr, sizeof(servaddr));
11    servaddr.sin_family = AF_INET;
12    servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
13    servaddr.sin_port = htons (SERV_PORT);
14    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
15    Listen(listenfd, LISTENQ);
16    for ( ; ; ) {
17        clilen = sizeof(cliaddr);
18        connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);
19        if ( (childpid = Fork()) == 0 ) { /* child process */
20            Close(listenfd); /* close listening socket */
21            str_echo(connfd); /* process the request */
22            exit (0);
23        }
24        Close(connfd); /* parent closes connected socket */
25    }
26 }
```

# Network Programming

## 2019 Seminar

### 5.2 Concurrent Server(다중 접속 서버)의 구현방법들

- Multi Process 기반 서버                      다수의 프로세서를 생성하는 방식
- Multiplexing 기반 서버                      입출력 대상을 묶어서 관리하는 방식
- Multithreading 기반 서버                      클라이언트의 수만큼 스레드 생성하는 방식

#### Process란?

- 메모리 공간을 차지한 상태에서 실행중인 프로그램
- 운영체제의 관점에서 프로그램 흐름의 기본단위

#### Process ID란?

- 운영체제가 프로세스에게 부여한 2이상의 정수

```
root@server:~# ps au
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	937	0.0	0.1	24288	1816	tty1	Ss+	20:38	0:00	/sbin/agetty -
root	939	0.8	5.9	327084	59760	tty7	Ss+	20:38	0:08	/usr/lib/xorg/
root	1650	0.0	0.4	30004	4184	pts/6	Ss	20:39	0:00	bash
root	1908	0.0	0.3	45712	3324	pts/6	R+	20:57	0:00	ps au

# Network Programming

## 2019 Seminar

### 5.2 Concurrent Server(다중 접속 서버)의 구현방법들

**Pid\_t fork(void);**

성공 시 프로세스 ID, 실패 시 -1 반환

다른 프로그램을 바탕으로 프로세스 생성X 이미 실행중인 프로세스를 복사하는 것!!!

- 부모 프로세스                      fork 함수의 반환 값은 자식 프로세스의 ID
- 자식 프로세스                      fork 함수의 반환 값은 0

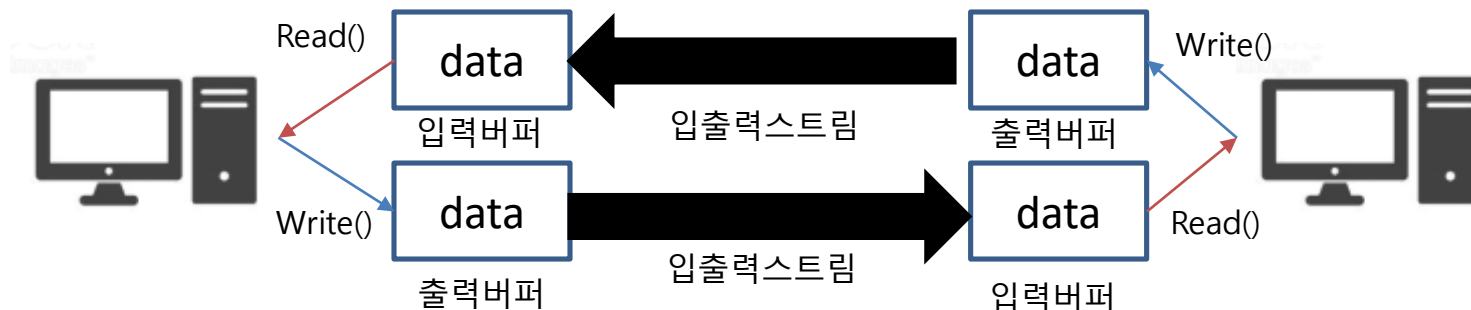
# Network Programming

## 2019 Seminar

### 5.3 TCP Echo Server: str\_echo Function

<str\_echo.c>

```
1 #include "unp.h"
2 void
3 str_echo(int sockfd)
4 {
5     ssize_t n;
6     char buf[MAXLINE];
7
8     again:
9     while ( (n = read(sockfd, buf, MAXLINE) ) > 0)
10         Writen(sockfd, buf, n);
11
12     if (n < 0 && errno == EINTR)
13         goto again;
14     else if (n < 0)
15         err_sys("str_echo: read error");
16 }
```



# Network Programming

## 2019 Seminar

### 5.4 TCP Echo Client: main Function

<tcpcli01.c>

```
-----
1 #include    "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int     sockfd;
6     struct sockaddr_in servaddr;
7     if (argc != 2)
8         err_quit("usage: tcpcli <IPaddress>");
9     sockfd = Socket(AF_INET, SOCK_STREAM, 0);
10    bzero(&servaddr, sizeof(servaddr));
11    servaddr.sin_family = AF_INET;
12    servaddr.sin_port = htons(SERV_PORT);
13    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
14    Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
15    str_cli(stdin, sockfd);    /* do it all */
16    exit(0);
17 }
-----
```

사람이 알기 쉬운 IP주소를  
binary 형태로 알기 쉽게 바꾼다



# Network Programming

## 2019 Seminar

### 5.5 TCP Echo Client: str\_cli Function

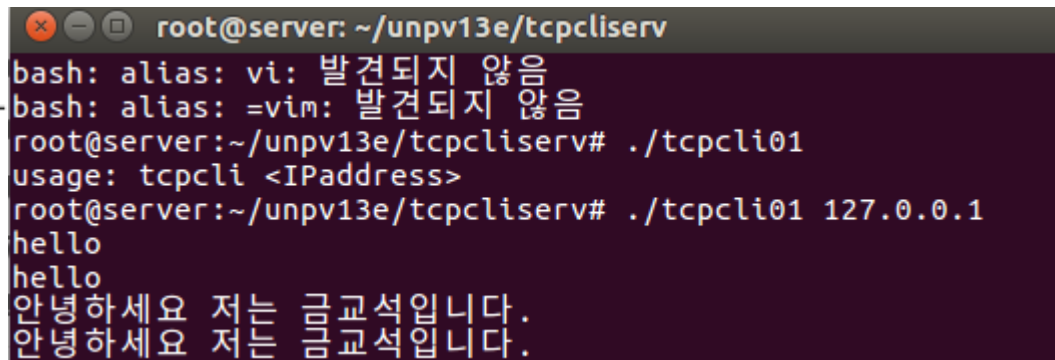
<str\_cli.c>

```
-----
1 #include    "unp.h"

2 void
3 str_cli(FILE *fp, int sockfd)
4 {
5     char    sendline[MAXLINE], recvline[MAXLINE];

6     while (Fgets(sendline, MAXLINE, fp) != NULL) {
7         Writen(sockfd, sendline, strlen (sendline));
8         if (Readline(sockfd, recvline, MAXLINE) == 0)
9             err_quit("str_cli: server terminated prematurely");

10        Fputs(recvline, stdout);
11    }
12 }
```

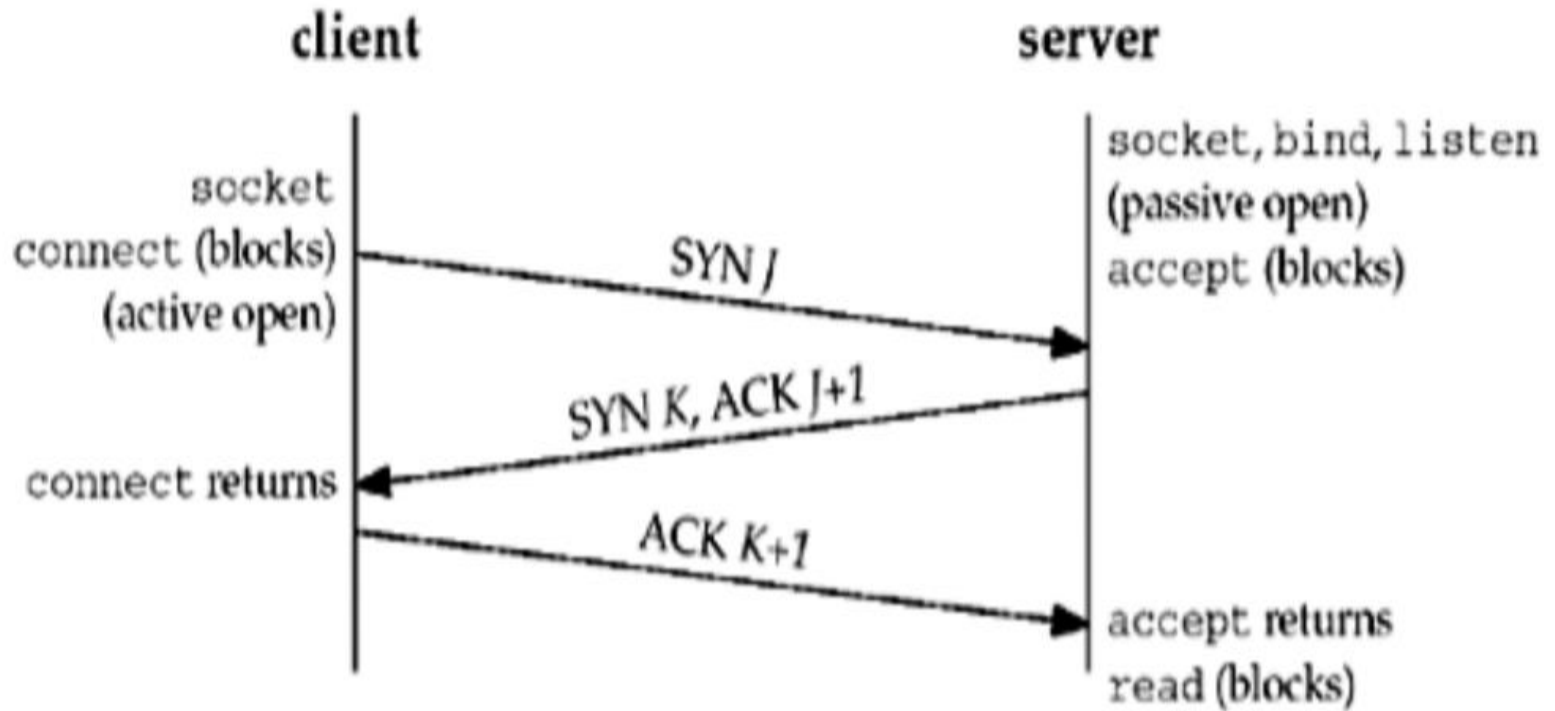


```
root@server: ~/unpv13e/tcpcliserv
bash: alias: vi: 발견되지 않음
bash: alias: =vim: 발견되지 않음
root@server:~/unpv13e/tcpcliserv# ./tcpcli01
usage: tcpcli <IPaddress>
root@server:~/unpv13e/tcpcliserv# ./tcpcli01 127.0.0.1
hello
hello
안녕하세요 저는 금교석입니다.
안녕하세요 저는 금교석입니다.
```

# Network Programming

## 2019 Seminar

### 5.6 Normal Startup(connection)



# Network Programming

## 2019 Seminar

### 5.6 Normal Startup(connection)

#### 1. Server 의 Listing socket의 상태

TCP통신을 하는 서비스의  
연결상태를 보여줌

```
root@server: ~/unpv13e/tcpcliserv
bash: alias: vi: 발견되지 않음
bash: alias: =vim: 발견되지 않음
root@server:~/unpv13e/tcpcliserv# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:9877                  *:.*                     LISTEN
```

#### 2. Client의 socket과 connect 호출

```
root@server: ~/unpv13e/tcpcliserv
root@server:~/unpv13e/tcpcliserv# ./tcpcli01 127.0.0.1
```

#### 3. Three-way handshake 시작!!!

#### 4 완료 후 connect return & accept return

# Network Programming

## 2019 Seminar

### 5.6 Normal Startup(block)

1. Client Str\_cli() 호출 fget() block!!!
2. 서버 accept() 반환 시 서버 fork() 호출/ 자식 프로세스는 str\_echo() 호출 read() block!!!
3. 서버는 accept() 다음 클라이언트 호출을 block!!

```
root@server: ~/unpv13e/tcpcliserv
bash: alias: vi: 발견되지 않음
bash: alias: =vim: 발견되지 않음
root@server:~/unpv13e/tcpcliserv# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:9877                  *:                        LISTEN
tcp        0      0 server:domain           *:                        LISTEN
tcp        0      0 localhost:9877           localhost:47800         ESTABLISHED
tcp        0      0 localhost:47800          localhost:9877          ESTABLISHED
```

Server  
child

client

# Network Programming

## 2019 Seminar

### 5.6 Normal Startup

```
linux % ps -t pts/6 -o pid,ppid,tt,stat,args,wchan
```

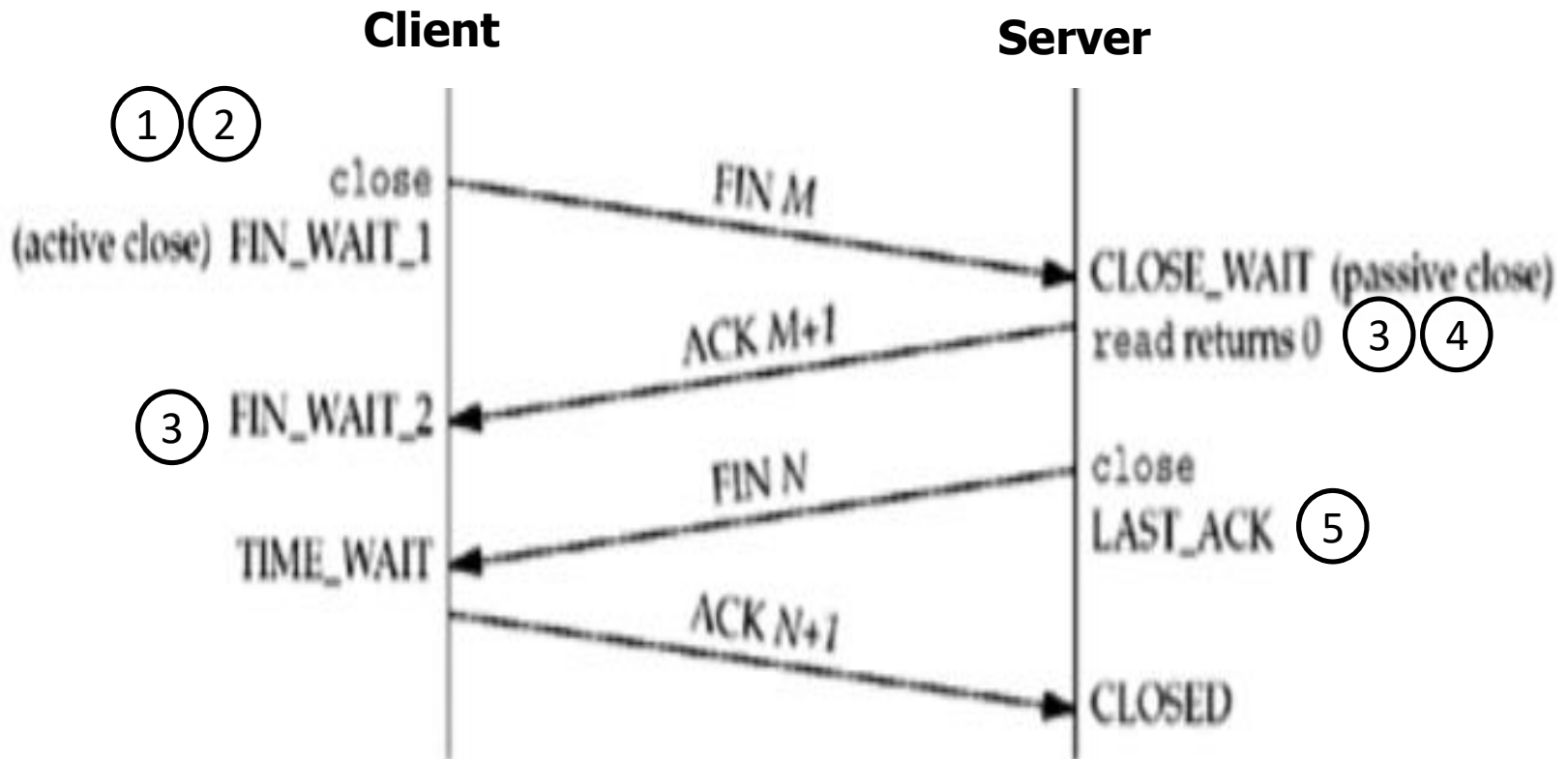
PID	PPID	TT	STAT	COMMAND	WCHAN
22038	22036	pts/6	S	-bash	wait4
17870	22038	pts/6	S	./tcpserv01	wait_for_connect
19315	17870	pts/6	S	./tcpserv01	tcp_data_wait
19314	22038	pts/6	S	./tcpcli01 127.0	read_chan

- S = sleeping
- PID = processID PPID = parent processID

# Network Programming

## 2019 Seminar

### 5.7 Normal Termination



# Network Programming

## 2019 Seminar

### 5.7 Normal Termination

1. client에서 EOF를 치면 fget() return and str\_cli() return
2. Client exit() 호출
3. Client 이 닫히면서 Server에게 Fin 전달(termination 시작)  
Client socket= `FIN_WAIT_2state`      Server = `CLOSE_WAIT`
- 4 Server 가 FIN을 받게 되면 server child 는 block 된다.(readline())  
readline = 0 return      str\_echo = return
5. exit() 함수로 Server child 닫힌다.
6. Child 가 닫히면서 connected socket이 닫힌다 그때 서버는 ACK와 Fin을 보낸다.

```
root@server: ~
bash: alias: vi: 발견되지 않음
bash: alias: =vim: 발견되지 않음
root@server:~# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:9877                  :::*                     LISTEN
tcp        0      0 server:domain           :::*                     LISTEN
tcp        0      0 localhost:47804          localhost:9877           TIME_WAIT
```

Client TIME\_WAIT 상태 반면 listing socket은 연결 대기

# Network Programming

## 2019 Seminar

### 5.7 Normal Termination

7. Zombi process를 없애기 위해서 signal을 보내야한다.

```
linux % ps -t pts/6 -o pid,ppid,ttty,stat,args,wchan

  PID  PPID TT      STAT COMMAND          WCHAN
 22038 22036 pts/6   S    -bash            read_chan
 17870 22038 pts/6   S    ./tcpserv01       wait_for_connect
 19315 17870 pts/6   Z    [tcpserv01 <defu do_exit
```

zombie를 처리할 수 있는 코드가 존재하지 않음!!!



# Network Programming


## 2019 Seminar


### 5.8 POSIX Signal Handling

#### Zombie Process란?

- 프로세스 생성 이 후 일을 다하고도 사라지지 않는 프로세스
- 시스템의 중요한 리소스를 차지하여 큰 문제를 야기

#### Zombie 의 생성 이유?(fork())

- 인자를 전달하면서 exit를 호출하는 경우
  - Main 함수에서 return문을 실행하면서 값을 반환하는 경우
- 
- 운영체제에게 전달



운영체제는 반환되는 값이 자식 프로세스를 생성한 부모 프로세스에게 전달될 때 까지 소멸 시키지 않는다.

#### Zombie 해결 방안!!!

- 부모 프로세스의 적극적인 요청!!!!

# Network Programming

## 2019 Seminar

### 5.10 wait and waitpid Functions

**Pid\_t wait(int \* statloc);**

성공 시 종료된 자식 프로세스의 ID, 자식의 종료 상태

실패 시 -1 반환

이미 종료된 자식 프로세스가 있다면 자식 프로세스가 종료되면서 반환하는 값이 매개변수로 전달!!!

메크로 함수를 통해 값의 분리 과정

- WIFEXITED                      자식 프로세스가 정상 종료한 경우 참을 반환
- WEXITSTATUS                  자식 프로세스의 전달 값을 반환

특징

Wait 함수는 호출된 시점에서 종료된 자식 프로세스가 없으면 첫번째 존재하는 children 종료할 때 까지 wait를 block시킨다.

# Network Programming

## 2019 Seminar

### 5.10 wait and waitpid Functions

```
int main() {
    pid_t childPid;
    int status,i;

    childPid = fork();

    if(childPid > 0) { // 부모 프로세스
        pid_t waitPid;
        printf("부모 PID : %ld, pid : %d %d \n", (long)getpid(), childPid, errno);

        for(i=0; i<5; i++) {
            sleep(1);
        }

        waitPid = wait(&status);

        if(waitPid == -1) {
            printf("에러 넘버 : %d \n", errno);
            perror("wait 함수 오류 반환");
        }
        else {
            if(WIFEXITED(status)) {
                printf("wait : 자식 프로세스 정상 종료 %d\n", WEXITSTATUS(status));
            }
            else if(WIFSIGNALED(status)) {
                printf("wait : 자식 프로세스 비정상 종료 %d\n", WTERMSIG(status));
            }
        }

        printf("부모 종료 %d %d\n", waitPid, WTERMSIG(status));
    }
    else if(childPid == 0) { // 자식 프로세스
        printf("자식 PID : %ld \n", (long)getpid());

        printf("자식 종료 \n");
        exit(0);
    }
    else { // fork 실패
        perror("fork Fail! \n");
        return -1;
    }

    return 0;
}
```

# Network Programming

## 2019 Seminar

### 5.10 wait and waitpid Functions

**Pid\_t waitpid(pid\_t pid, int \* statloc int options);**

성공 시 종료된 자식 프로세스의 ID, 자식의 종료 상태

실패 시 -1 반환

- Pid                      종료를 확인하고자 하는 자식 프로세스의 ID
- Statloc                wait 함수의 매개변수 statloc
- Options                WNOHANG이 전달되면 종료된 자식프로세스가 없어도 no Blocking

특징

어느 프로세스를 기다릴 것 인지와 블록 할 것인지를 조절 가능하다.

# Network Programming

## 2019 Seminar

### 5.8 POSIX Signal Handling

문제

자식 프로세스가 언제 종료될 줄 알고 waitpid()를 계속 호출 하고 있을 것인가???



자식 프로세스 종료의 인식 주체는 운영체제!!!

부모 프로세스!!! 네가 생성한 자식 프로세스가 종료 되었어!!!

#### signal란?

특정 상황이 발생했음을 알리기 위해 운영체제가 프로세스에게 전달하는 메시지

#### Handling란?

그 메시지에 반응하여 메시지와 연관된 작업이 진행되는 것

# Network Programming

## 2019 Seminar

---

### 5.8 POSIX Signal Handling

#### Signal의 종류

- 하나의 프로세스가 다른 프로세스에게
- 커널이 프로세스에게

#### Signal disposition(action)

- Signal handler 함수가 Signal이 SIGKILL과 SIGSTOP일 경우는 알지 못한다.
- Signal이 SIG\_IGN이면 Signal을 무시가능하다.
- SIG\_DEF는 기본적인 disposition인데 신호를 받으면 프로세스를 종료시킨다.
- SIGCHLD 와 SIGURG일 때 무시한다.

## Network Programming 2019 seminar

# Network Programming

## 2019 Seminar

### 5.8 POSIX Signal Handling

**Void(\*[signal](#)(int signo,void(\*func)(int)))(int) ;**

시그널 발생시 호출 되도록 이전에 등록된 함수의 포인터 반환

- |           |                               |
|-----------|-------------------------------|
| • 첫번째 인자  | 특정 상황에 대한 정보                  |
| • 두번째 인자  | 특정 상황에서 호출될 함수의 주소값           |
|           |                               |
| • SIGALRM | alarm 함수 호출을 통해서 등록된 시간이 된 상황 |
| • SIGINT  | CTRL+C가 입력된 상황                |
| • SIGCHLD | 자식 프로세스가 종료된 상황               |

#### Signal 함수 예시

자식 프로세스가 종료되면 hello 함수를 호출해 달라!!!

-> [Signal](#)(SIGCHLD,hello)



# Network Programming

## 2019 Seminar

### 5.8 POSIX Signal Handling

Int **sigaction**(int singno, const struct sigaction \*act, struct sigaction \*oldact);

성공 시 0 실패 시 -1 반환

- Signo signal 함수와 동일한 시그널의 정보를 인자로 전달
- Act 첫번째 인자로 전달된 상수에 해당하는 시그널 발생 시 호출될 함수의 정보 전달
- Oldact 이전에 등록된 시그널 핸들러의 함수 포인터를 얻는데 사용되는 인자

Signal은 유닉스 운영체제별로 약간의 차이가 존재 하지만 sigaction은 차이 X

Struct sigaction

```
{  
    void(*sa_handler)(int);  
    sigset_t sa_mask;  
    int sa_flags;  
}
```

시그널 핸들러의 함수  
포인터를 0으로 초기화 작업  
필요로 초기화 작업 필요

# Network Programming

## 2019 Seminar

### 5.8 POSIX Signal Handling

Sigaction 사용예제

```
struct sigaction act_int, act_quit ;
```

```
act_int.sa_handler = sigint_handler ;
```

```
sigemptyset(&act_int.sa_mask) ;
```

```
sigaddset(&act_int.sa_mask, SIGQUIT) ;
```

```
act_int.sa_flags = 0;
```

```
if ( sigaction(SIGINT, &act_int, NULL) < 0 )
```

```
{
```

```
    printf("sigaction() error\n") ;
```

```
    exit(-1) ;
```

```
}
```

→ 시그널 핸들러의 함수

→ 모든 비트를 0으로 초기화 작업

필요

→ 0으로 초기화 작업 필요

→ 함수 사용

# Network Programming

## 2019 Seminar

### 5.9 Handling SIGCHLD Signals

```
Signal (SIGCHLD, sig_chld)

<sigchldwait.c>
-----
1 #include      "unp.h"

2 void
3 sig_chld(int signo)
4 {
5     pid_t    pid;
6     int      stat;

7     pid = wait(&stat);
8     printf("child %d terminated\\", pid);
9     return;
10 }
-----
```

1. Client에서 EOF를 치면 Client는 FIN을 서버에게 보내고 서버는 ACK를 날린다.
2. FIN을 수신하면 readline이 종료되면서 child도 종료된다.
3. SIGCHLD신호가 오면 accept를 부르면서 Block이 된다. Wait은 자식의 PID를 얻고 종료된다.
4. errno==EINTR이지만 parent은 erro를 처리하지 않고 종료된다.

# Network Programming

## 2019 Seminar

---

### 5.9 Handling SIGCHLD Signals (Handling interrupted System call)

#### System call 이란??

프로세스가 운영체제 에게 **필요한 행동을 요청**할 수 있도록 운영체제에서 정의한 함수  
하드웨어에 직접 접근해서 **필요한 기능을 사용할** 수 있게 요청(accept)

#### System call이 호출 시 시그널이 발생한다면???

Signal handler가 작동하면서 시스템 콜은 에러를 반환한다.

System call이 호출 시 에러 발생!!

에러가 발생한 원인에 대한 값 = 변수 errno

(Errno == EINTR)는 시스템 콜 수행 중 인터럽트가 걸려 수행이 중단!!!

# Network Programming

## 2019 Seminar

---

### 5.9 Handling SIGCHLD Signals

```
for ( ; ; ) {  
    clilen = sizeof (cliaddr);  
    if ( (connfd = accept (listenfd, (SA *) &cliaddr, &clilen)) < 0) {  
        if (errno == EINTR)  
            continue;          /* back to for () */  
        else  
            err_sys ("accept error");  
    }  
}
```

# Network Programming

## 2019 Seminar

### 5.10 wait and waitpid Functions

```
<tcpserver04.c>
-----
1 #include    "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int    listenfd, connfd;
6     pid_t  childpid;
7     socklen_t cliilen;
8     struct sockaddr_in cliaddr, servaddr;
9     void    sig_chld(int);
10
11     listenfd = Socket (AF_INET, SOCK_STREAM, 0);
12
13     bzero (&servaddr, sizeof(servaddr));
14     servaddr.sin_family = AF_INET;
15     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
16     servaddr.sin_port = htons(SERV_PORT);
17
18     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
19
20     Listen(listenfd, LISTENQ);
21
22     Signal (SIGCHLD, sig_chld); /* must call waitpid() */
23
24     for ( ; ; ) {
25         cliilen = sizeof(cliaddr);
26         if ( (connfd = accept (listenfd, (SA *) &cliaddr, &cliilen)) < 0)
27         {
28             if (errno == EINTR)
29                 continue; /* back to for() */
30             else
31                 err_sys("accept error");
32         }
33
34         if ( (childpid = Fork()) == 0) { /* child process */
35             Close(listenfd); /* close listening socket */
36             str_echo(connfd); /* process the request */
37             exit(0);
38         }
39         Close (connfd); /* parent closes connected socket */
40     }
41 }
```

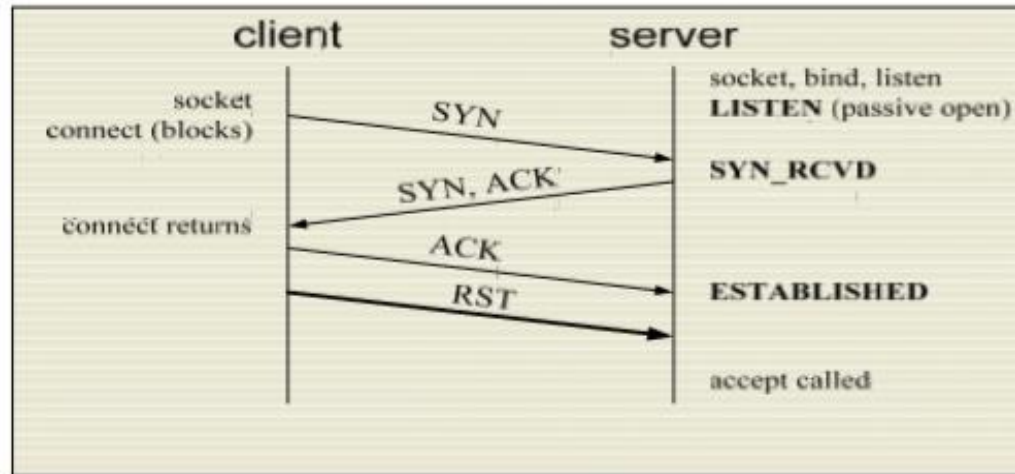
Zombie process를 없애기 위해

Accept 호출 시 시그널에  
대한 예외처리

# Network Programming

## 2019 Seminar

### 5.11 Connection Abort before accept Returns



1. 3way handshaking 마지막 ACK를 서버가 수신 후
2. Incomplete 큐에 있던 클라이언트 연결 요청을 complete 큐에 옮긴다.
3. Complete 큐 맨 앞에서 부터 accept()시스템 콜을 리턴 함
4. Connect 소켓 생성

문제

갑자기 클라이언트 프로세스 종료!!! 연결이 되지도 않았는데

클라이언트에게 **RST(RESET)**메세지 서버로 도달

그 연결 요청 정보를 accept() 리턴할 때 -1반환!! Error 메시지 설정!!

# Network Programming

## 2019 Seminar

---

### 5.11 Connection Abort before accept Returns

#### RST란?

재설정을 하는 과정이며 양방향에서 동시에 일어나는 중단 작업.  
비 정상적인 세션 연결 끊기에 해당한다

#### RST의 송신 이유

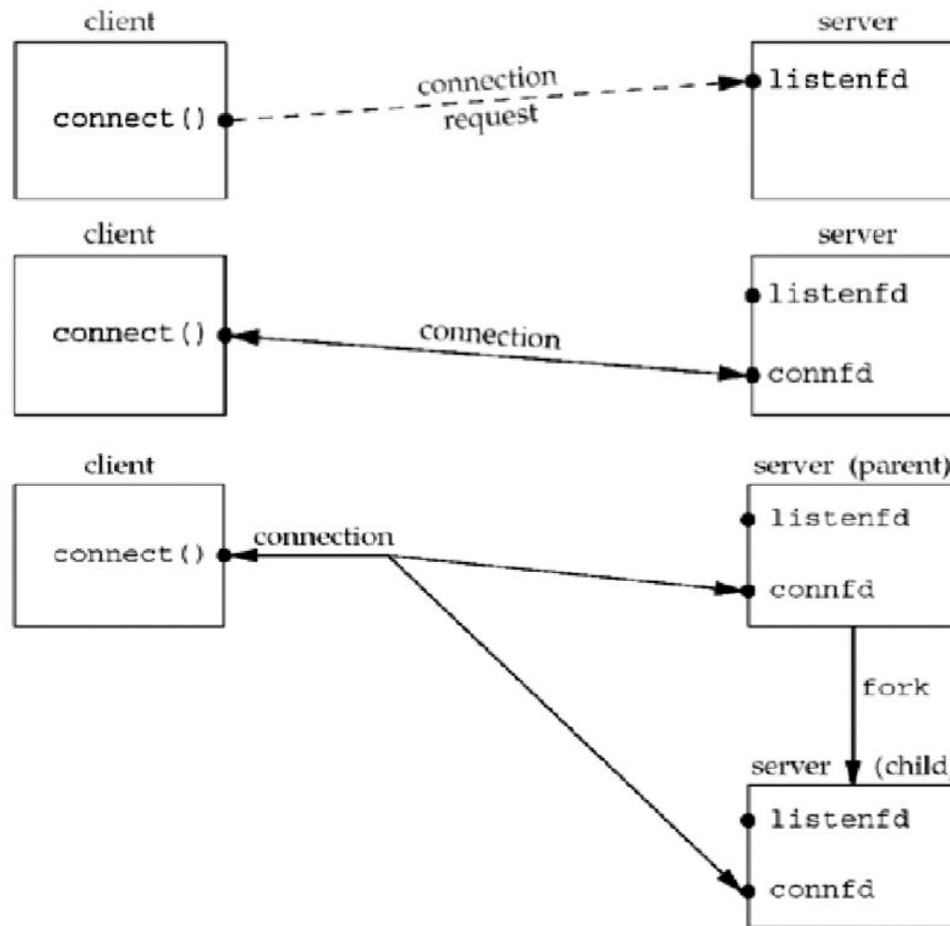
- 연결을 맺고 있지 않은 호스트로부터 TCP 패킷 온 경우.
- 부정확한 순서 번호나 승인 번호 필드를 가지는 메시지가 수신한 경우.
- 연결을 기다리는 프로세스가 없는 포트로 SYN 메시지를 받은 경우.



# Network Programming

## 2019 Seminar

### 5.11 Connection Abort before accept Returns

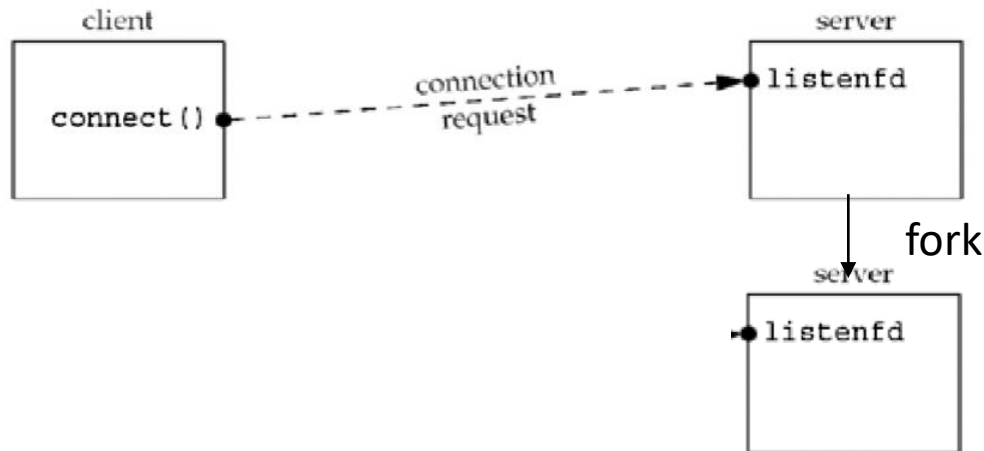


# Network Programming

## 2019 Seminar

### 5.11 Connection Abort before accept Returns

문제 발생!!!



서버 부모 프로세서는 connect 소켓을 닫고  
서버 자식 프로세서는 listen 소켓을 닫는 것으로 구성  
아무 일도 하지 않는 프로세스가 여러 개 생성

의미 없는 프로세스가 계속 생성 문제 발생

# Network Programming

## 2019 Seminar

### 5.12 Termination of Server Process

1. 서버와 클라이언트를 시작하고 한줄을 작성한다.
2. 서버 child를 없애면 클라이언트는 FIN을 받게 되고 ACK를 전달한다.
3. SIGCHLD이 서버 parent에게 가고 처리한다.
4. 클라이언트는 아무 일도 없고 클라이언트는 FIN을 받고 ACK

```
linux % netstat -a | grep 9877
```

```
tcp        0      0 *:9877                ::*                LISTEN
tcp        0      0 localhost:9877        localhost:43604    FIN_WAIT2
tcp        1      0 localhost:43604      localhost:9877    CLOSE_WAIT
```

5. Client는 RST를 받지 못한다. Error message를 받게 된다.

```
linux % tcpcli01 127.0.0.1      start client
another line                    we then type a second line to the client
str_cli : server terminated
prematurely
```

6. Client가 끝이 나면 모든 descriptor가 닫히게 된다.

6. 해결

Select와 poll 함수

# Network Programming

## 2019 Seminar

### 5.13 SIGPIPE Signal

문제

클라이언트가 read에서 오는 오류를 무시하고 서버에 write를 하면 어떻게 될까??

SIGPIPE Signal이 발생

<str\_cli1.c>

```
-----
1 #include      "unp.h"

2 void
3 str_cli(FILE *fp, int sockfd)
4 {
5     char    sendline [MAXLINE], recvline [MAXLINE];

6     while (Fgets(sendline, MAXLINE, fp) != NULL) {

7         Writen(sockfd, sendline, 1);
8         sleep(1);
9         Writen(sockfd, sendline + 1, strlen(sendline) - 1);

10        if (Readline(sockfd, recvline, MAXLINE) == 0)
11            err_quit("str_cli: server terminated prematurely");

12        Fputs(recvline, stdout);
13    }
14 }
-----
```

서버로부터 RTS를  
받기 위해

# Network Programming

## 2019 Seminar

### 5.13 SIGPIPE Signal

```
linux % tcpclll 127.0.0.1

hi there          we type this line

hi there          this is echoed by the server

                  here we kill the server child

bye              then we type this line

Broken pipe       this is printed by the shell
```

1. Hi there를 입력시 hi there가 정상적으로 서버로 에코됨
2. 서버 자식 프로세스를 kill
3. 클라이언트는 서버로 FIN 메시지를 받음
4. Fgets()로 키보드 입력 받고 첫번째 write실행해 서버에 데이터를 1바이트 보냄
5. 서버 커널은 클라이언트에게 RST메세지를 전송
6. RST를 받은 소켓에 다시 write하면 SIGPIPE 시그널 발생

디폴트 액션으로 프로세스 종료 따라서 프로세스 종료를 막기 위해 [SIGPIPE 설정](#)


# Network Programming

## 2019 Seminar

### 5.14 Crashing of Server Host

문제

클라이언트는 서버가 Crashing 됐는지 안됐는지 알 수 없다.



시간이 오래 걸린다.

방법

Fgets()->write()->read() block?? (FIN도 RESET도 오지 않는다면)

12번 정도(9분) 재전송 client 커널은 타임아웃!!!(상대방이 없다)

Read() 시스템 콜을 -1 리턴 후 **errno ETIMEDOUT** 설정

방법

라우터를 타고 목적지 까지 가는 와중에 그 ip로는 갈수 없다는 라우팅 정보를 가지고 있는 라우터를 통하게 될 경우 그 라우터 에서 클라이언트에게 **ICMP메세지**를 돌려주게 된다.

# Network Programming

## 2019 Seminar

### 5.15 Crashing and Rebooting of Server Host

문제

갑자기 서버가 다운되고 켜졌다고 했을 때



서버는 그전에 설정 되었던 소켓에 대한 정보가 사라진다.

방법

Client : Fgets()->write()->서버에게 데이터를 보냄->read()

Server: 그런 소켓 없는데??? RTS보냄

Read() 시스템 콜을 -1 리턴 후 **errno ETIMEDOUT** 설정


# Network Programming

## 2019 Seminar

### 5.16 Shutdown of Server Host

문제

서버 프로세스가 동작 중일 때 서버 호스트를 끈다면??



SIGTERM 신호를 모든 프로세스에게 보냄

방법

실행 중인 프로세스 정리 및 시간 부여

SIGTERM을 만나고도 종료하지 못하면 SIGKILL 신호를 종료 시킴

모든 open descriptor를 닫는다.

해결

서버 프로세스의 종료를 알기 위해 select와 poll 함수 사용



# Network Programming 2019 Seminar

---

## 참고자료

<https://simsimjae.tistory.com/126>

[http://blog.daum.net/\\_blog/BlogTypeView.do?blogid=0YW8F&articleno=65&categoryId=4&regdt=20130109190217](http://blog.daum.net/_blog/BlogTypeView.do?blogid=0YW8F&articleno=65&categoryId=4&regdt=20130109190217)

<https://kldp.org/node/25770>

<https://luckyowu.tistory.com/133>

유닉스 시스템&네트워크 프로그래밍

열혈TCP/IP 소켓 프로그래밍

<https://codetravel.tistory.com/30>