

TCP Client/Server Example

2016707013 손재혁

차례

1. Introduction
2. TCP Echo Server : main Function
3. TCP Echo Server : str_echo Function
4. TCP Echo Client : main Function
5. TCP Echo Client : str_cli Function
6. Normal Startup
7. Normal Termination
8. POSIX Signal Handling
9. Handling SIGCHLD Signals
10. wait and waitpid function
11. Connection Abort before accept Returns
12. Termination of Server Process
13. SIGPIPE Signal
14. Crashing and Server Host
15. Crashing and Rebooting of Server Host
16. Shutdown of Server Host

Before Start

0. Download

먼저 아래의 링크에서 소스코드의 압축 파일을 받아준뒤 압축을 풀어 줍니다.

<http://unpbook.com/src.html> 에서 download 하시거나

아래의 명령어를 통해 다운로드합니다.

```
$ wget http://unpbook.com/src/unpv13e.tar.gz
```

압축해제

```
$ tar -xvf unpv13e.tar.gz
```

config.guess 파일을

http://mclab.hufs.ac.kr/wiki/HOWTO/UNP_Library에서

제공되는 파일로 바꿔줍니다

```
$ cd unpv13e/
```

```
$ rm config.guess
```

```
$ wget ftp://mclab.hufs.ac.kr/config.guess
```

1. Install

```
$ ./configure
```

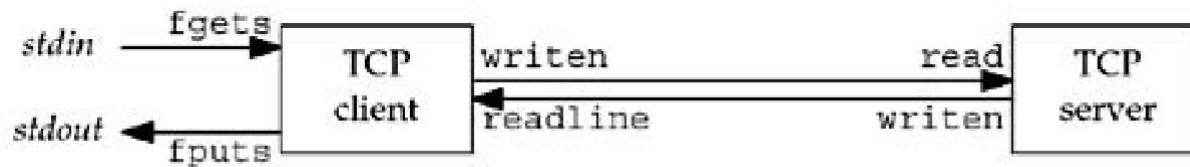
```
$ cd ./lib
```

```
$ vi unp.h // SERV_PORT 수정 (기본 9877)
```

```
$ make
```

<https://wowan.tistory.com/107>

Figure 5.1. Simple echo client and server.



1. 표준 입력장치를 통해 문자열을 입력받는다.
2. 클라이언트는 문자열을 서버에게 넘겨준다.
3. 서버는 문자열을 읽고 클라이언트에게 다시 넘겨준다.
4. 클라이언트는 표준 출력장치를 통해 받은 문자열을 출력한다.

TCP Echo Server : main Function

```
#include "unp.h"

int
main(int argc, char **argv)
{
    int listenfd, connfd;
    pid_t childpid;
    socklen_t cliilen;
    struct sockaddr_in cliaddr, servaddr;

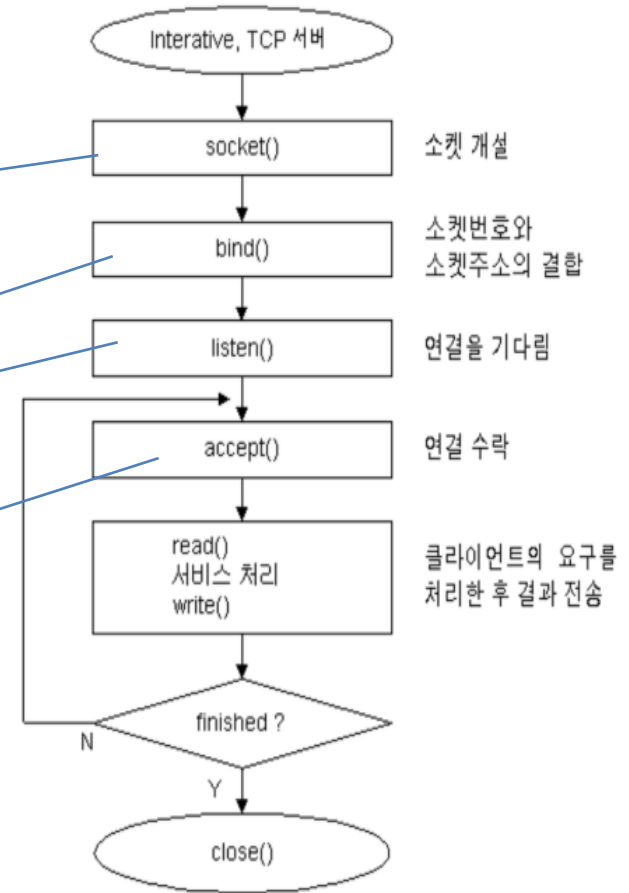
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);

    Bind(listenfd, (SA *)&servaddr, sizeof(servaddr));
    Listen(listenfd, LISTENQ);

    for ( ; ; ) {
        cliilen = sizeof(cliaddr);
        connfd = Accept(listenfd, (SA *)&cliaddr, &cliilen);

        if ( (childpid = Fork()) == 0 ) { /* child process */
            Close(listenfd); /* close listening socket */
            str_echo(connfd); /* process the request */
            exit(0);
        }
        Close(connfd); /* parent closes connected socket */
    }
}
```



TCP Echo Server : main Function

```
listenfd = Socket(AF_INET, SOCK_STREAM, 0);  
  
bzero(&servaddr, sizeof(servaddr));  
servaddr.sin_family      = AF_INET;  
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);  
servaddr.sin_port        = htons(SERV_PORT);
```

```
#include <sys/socket.h>  
int socket(int domain, int type, int protocol);
```

domain : 통신 영역 지정 ex) IPv4, IPv6..

type : 소켓 타입 ex) SOCK_STREAM(TCP), SOCK_DGRAM(UDP)..

protocol : 프로토콜 타입 ex) IPPROTO_TCP, IPPROTO_UDP, 0(type에서 정해진 경우)

리턴 값 : 생성 실패 -> -1 반환

 생성 성공 -> 0이상의 값(소켓 디스크립터)

AF_INET = IPv4 인터넷 프로토콜

AF_INET6 = IPV6 인터넷 프로토콜

AF_LOCAL = local 통신을 위한 UNIX 프로토콜

TCP Echo Server : main Function

```
listenfd = Socket(AF_INET, SOCK_STREAM, 0);  
  
bzero(&servaddr, sizeof(servaddr));  
servaddr.sin_family      = AF_INET;  
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);  
servaddr.sin_port        = htons(SERV_PORT);
```

bzero(&servaddr , sizeof(servaddr)) = memset(&servaddr, 0 , sizeof(servaddr))

servaddr.sin_family = AF_INET -> socket에서 처럼 통신 영역 지정

servaddr.sin_addr.s_addr = htonl(INADDR_ANY) -> 자신의 IP 할당

INADDR_ANY는 랜카드가 두개 이상이여서 IP주소가 2개 이상이 아닌경우 보통의 경우에 사용

servaddr.sin_port = htons(SERV_PORT) -> 사용할 포트 저장.

What is 'htonl'/'htons'?

What is 'htonl'/'htons'?

빅/스몰 엔디안 :

- 빅 엔디안 : 높은 메모리 주소에 뒤의 값 저장
- 스몰 엔디안 : 낮은 메모리 주소에 뒤의 값 저장.

Ex) char* char = "abcde";

- 빅 엔디안 : 메모리가 낮은 곳부터 a,b,c,d 저장
- 스몰 엔디안 : 메모리가 높은 곳 부터 a,b,c,d 저장

전송 시 : 빅엔디안 / 수신 시 : 스몰 엔디안

- > network 통신 시 방식이 다르면 문제 발생, 데이터가 달라지기 때문.
- > read, write함수 내부에서 구현 완료.
- > 포트 번호(2 byte), IP(4 byte)를 처리.

htonl / htons :

'host to network long/short'

- > 리틀엔디안 에서 빅엔디안으로 long(4 byte) / short(2 byte) 만큼 변경.

만약 빅엔디안에서 리틀 엔디안으로 변경 -> 'ntohl' / 'ntohs'

TCP Echo Server : main Function

```
listenfd = Socket(AF_INET, SOCK_STREAM, 0);  
  
bzero(&servaddr, sizeof(servaddr));  
servaddr.sin_family = AF_INET;  
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);  
servaddr.sin_port = htons(SERV_PORT);
```

포트 넘버는 2 byte 이기 때문에 htons

IP 주소는 4 byte이기 때문에 htonl

TCP Echo Server : str_echo Function

```
1 #include      "unp.h"

2 void
3 str_echo(int sockfd)
4 {
5     ssize_t n;
6     char    buf[MAXLINE];

7     again:
8     while ( (n = read(sockfd, buf, MAXLINE)) > 0)
9         Writen(sockfd, buf, n);

10    if (n < 0 && errno == EINTR)
11        goto again;
12    else if (n < 0)
13        err_sys("str_echo: read error");
14 }
```

TCP Echo Client : main Function

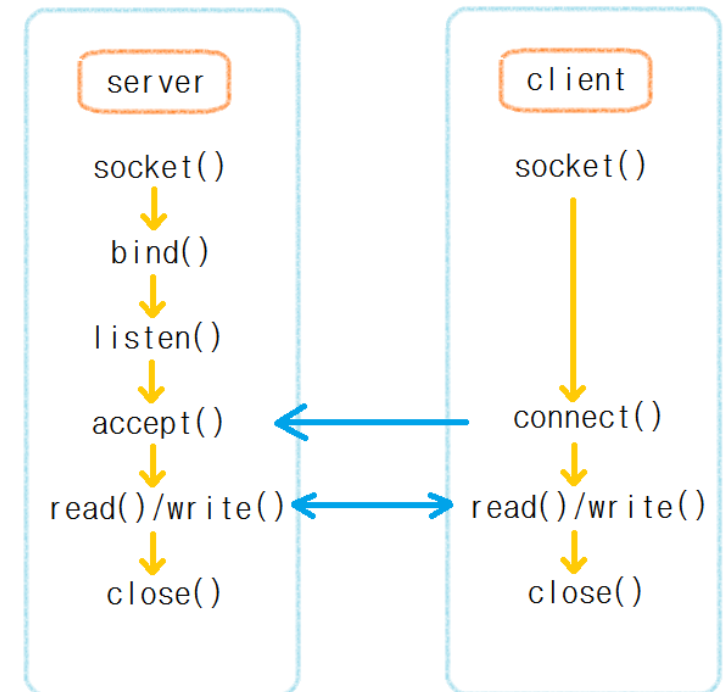
```
#include <unistd.h>

int
main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in servaddr;

    if (argc != 2)
        err_quit("usage: tcpcli <IPaddress>");

    sockfd = Socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    Connect(sockfd, (SA *)&servaddr, sizeof(servaddr));
    str_cli(stdin, sockfd);      /* do it all */
    exit(0);
}
```



TCP Echo Client : main Function

```
Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
```

```
int inet_pton(int af, const char *src, void *dst);
```

-input

Int af : address family지정. Ex) IPv4, IPv6

const char * src : 문자열 형태의 IP주소

-output

Void *dst : *src를 binary 형태로 변환 후 복사할 메모리의 포인터

TCP Echo Client : str_cli Function

```
#include "unp.h"

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

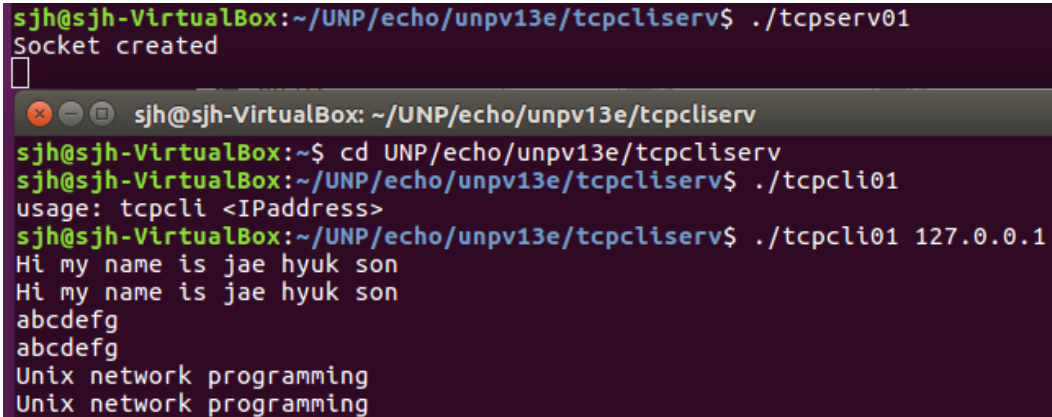
    while (Fgets(sendline, MAXLINE, fp) != NULL) {

        Writen(sockfd, sendline, strlen(sendline));

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

        Fputs(recvline, stdout);

    }
}
```



```
sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv$ ./tcpcliserv01
Socket created
[sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv]
sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv$ cd UNP/echo/unpv13e/tcpcliserv
sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv$ ./tcpcli01
usage: tcpcli <IPaddress>
sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv$ ./tcpcli01 127.0.0.1
Hi my name is jae hyuk son
Hi my name is jae hyuk son
abcdefg
abcdefg
Unix network programming
Unix network programming
```

Normal Stratup

1. Server 의 listening list 출력

```

sjh@sjh-VirtualBox:~/UNP/echo/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 sjh-VirtualBox:domain  *:                       LISTEN
tcp        0      0 localhost:ipp           *:                       LISTEN
tcp        0      0 localhost:48472         localhost:9877          TIME_WAIT
tcp        0      0 10.0.2.15:40810         ec2-35-163-249-50:https ESTABLISHED
tcp6       0      0 ip6-localhost:ipp      [::]:*                  LISTEN
/* Define some port number that can be used for client servers */##131 ##src/lib/unp.h##
#define SERV_PORT          9877 /* TCP and UDP client-servers */##132 ##src/lib/unp.h##
#define SERV_PORT_STR      "9877" /* TCP and UDP client-servers */##133 ##src/lib/unp.h##
#define UNIXSTR_PATH        "/tmp/unix.str" /* Unix domain stream cli-serv */##134 ##src/lib/unp.h##
#define UNIXDG_PATH         "/tmp/unix.dg" /* Unix domain datagram cli-serv */##135 ##src/lib/unp.h##
/* $$$.ix [LISTENQ]~constant,~definition~of$$ */
/* $$$.ix [MAXLINE]~constant,~definition~of$$ */
/* $$$.ix [MAXSOCKADDR]~constant,~definition~of$$ */
/* $$$.ix [BUFSIZE]~constant,~definition~of$$ */
/* $$$.ix [SERV_PORT]~constant,~definition~of$$ */
/* $$$.ix [UNIXSTR_PATH]~constant,~definition~of$$ */
/* $$$.ix [UNIXDG_PATH]~constant,~definition~of$$ */

/* Following shortens all the type casts of pointer arguments */##136 ##src/lib/unp.h##
#define SA struct sockaddr##137 ##src/lib/unp.h##

```

2. Client에서 connect 호출

```

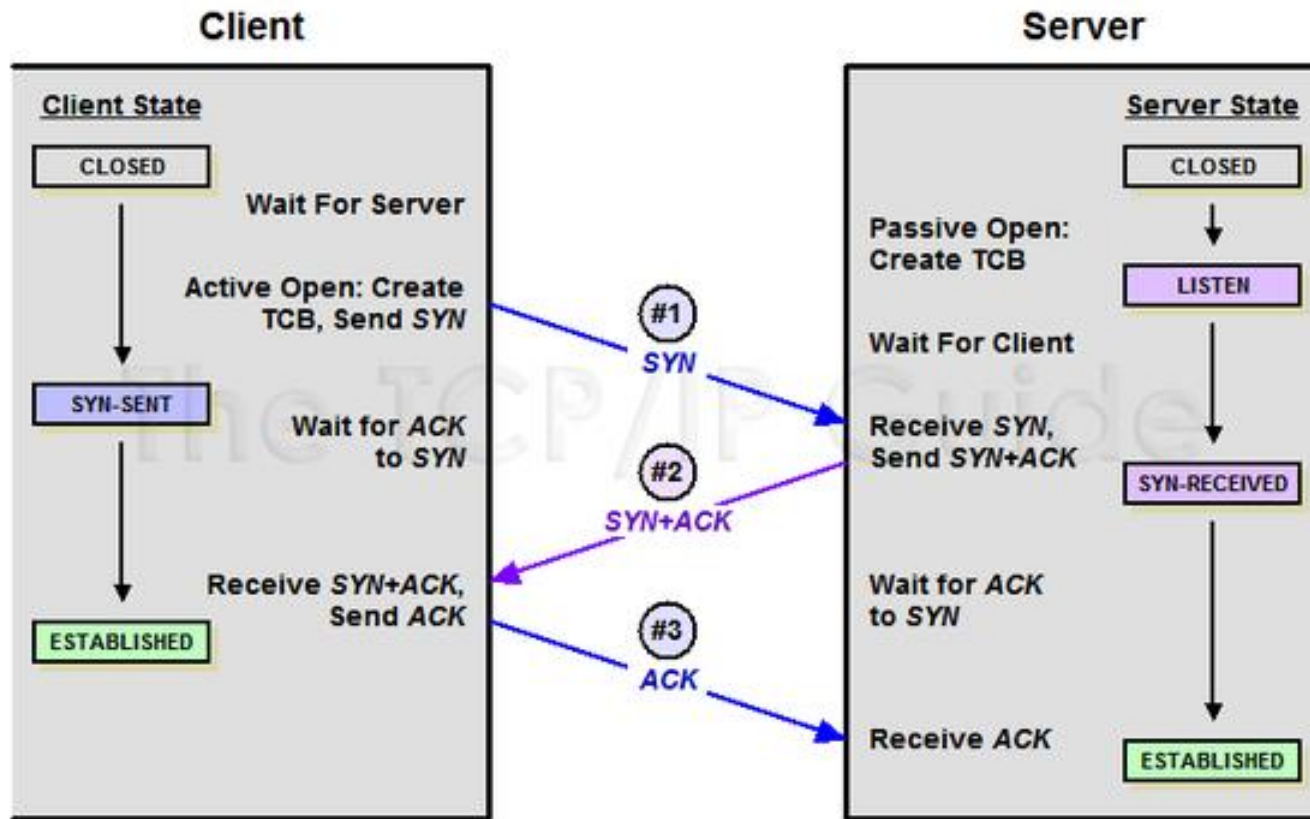
sjh@sjh-VirtualBox:~/UNP/echo/unpv13e/tcpcliserv$ ./tcpcli01 127.0.0.1

```

3. 3-Way Handshaking

Normal Stratup

3. 3-Way Handshaking



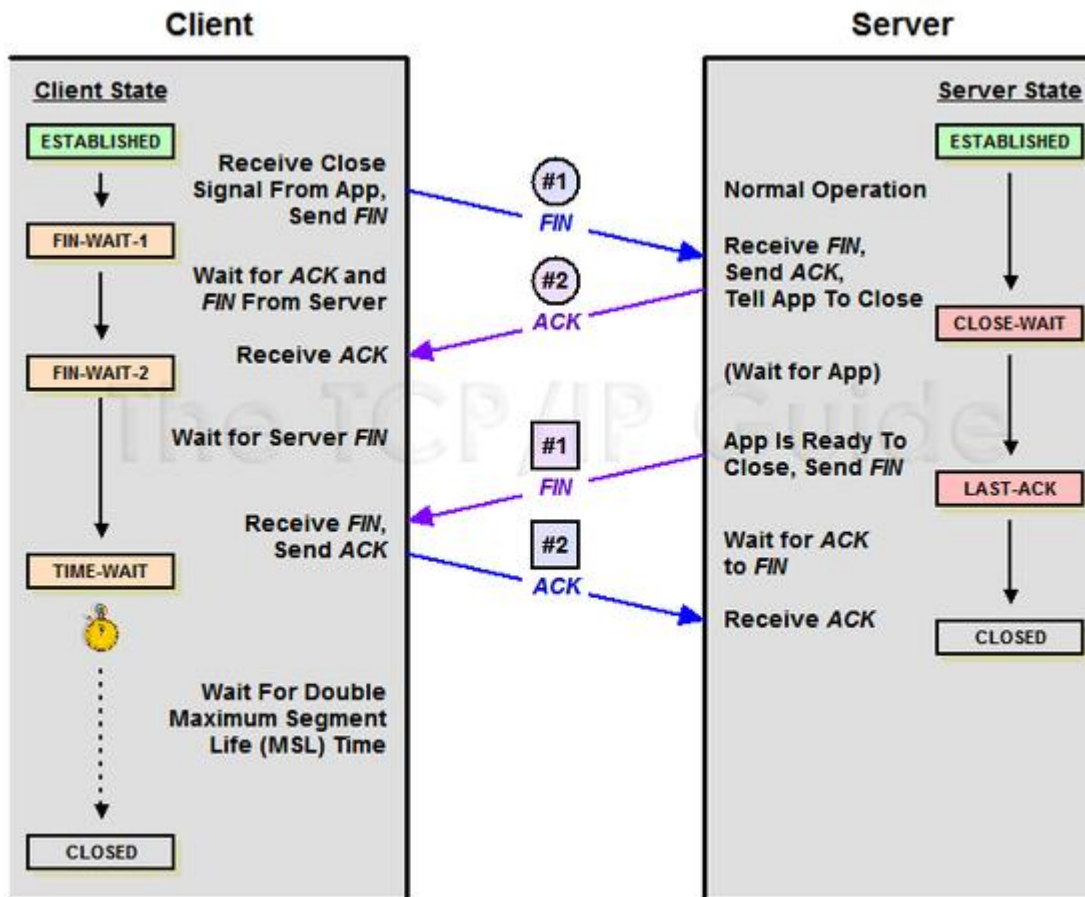
Normal Stratup

4. client str_cli() 호출
fgets() block -> 아직 입력이 없기 때문.
5. Server에서 accept반환
6. fork() 호출. 자식 프로세스 생성
7. 자식 프로세스가 str_echo()호출
read() block -> 아직 읽을 것이 없기 때문에
8. server parent는 accept() 다시 호출.
9. 다음 클라이언트 호출 대기.

```
sjh@sjh-VirtualBox:~/UNP/echo/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 sjh-VirtualBox:domain  *:                        LISTEN
tcp        0      0 localhost:ipp           *:                        LISTEN
tcp        0      0 localhost:48492         localhost:9877          ESTABLISHED
tcp        0      0 localhost:9877          localhost:48492         ESTABLISHED
```


Normal Termination

1. Client에서 EOF(End of File) 입력 시 `fgets()`, `str_cli()` return
2. Client `exit()` 호출
3. 4-way handshaking 시작



Normal Termination

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

자식 프로세스 종료 signal이 없음



자식 프로세스가 Zombie 프로세스가 됨



Signal Handling 필요

POSIX Signal Handling

Zombie Proces란?

- 프로세스 생성 후, 작업 종료 후에도 사라지지 않은 프로세스
- 시스템의 리소스를 차지, 오류 발생 가능.

Zombie 생성 이유

- 자식프로세스가 exit호출
- 자식프로세스의 Main 함수에서 return 실행 후 값 반환.
- exit return값과 main함수의 return값을 운영체제에 전달.
- 운영체제가 받은 return 값들을 부모 프로세스에게 전달 전까진 프로세스를 소멸 시키지 않음.
- 운영체제는 부모 프로세스의 요청이 있기전엔 전달 X.

Zombie 프로세스를 없애기 위해선?

부모 프로세스가 운영체제에게 정보 전달 요청.

POSIX Signal Handling

POSIX Signal 이란?

- 운영체제에 쓰이는 제한된 형태의 프로세스간 통신.

Sigaction() 함수란?

- `signal()`보다 향상된 기능을 제공하는 시그널 처리 함수.

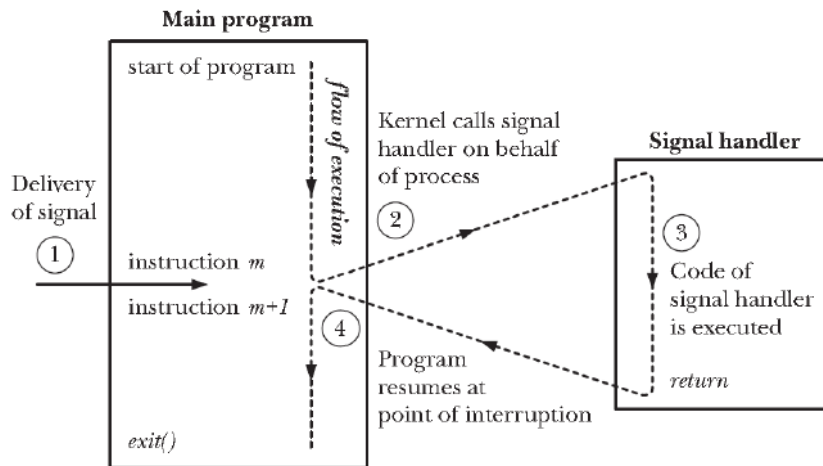


Figure 20-1: Signal delivery and handler execution

POSIX Signal Handling

Sigaction 구조체

```

struct sigaction {
    void (*sa_handler)(int);    // 시그널을 처리하기 위한 핸들러. SIG_DFL, SIG_IGN 또는 핸들러 함수
    void (*sa_sigaction)(int, siginfo_t *, void *); // 밑의 sa_flags가 SA_SIGINFO일때
                                                // sa_handler 대신에 동작하는 핸들러
    sigset_t sa_mask;           // 시그널을 처리하는 동안 블록화할 시그널 집합의 마스크
    int sa_flags;                // 아래 설명을 참고하세요.
    void (*sa_restorer)(void); // 사용해서는 안됩니다.
}

```

int sa_flags : 시그널이 어떻게 처리돼야 하는지 제어

Sigaction() 함수

헤더	signal.h
형태	int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
인수	int signum 시그널 번호 struct sigaction *act 설정할 행동. 즉, 새롭게 지정할 처리 행동 struct sigaction *oldact 이전 행동, 이 함수를 호출하기 전에 지정된 행동 정보가 입력됩니다.
반환	0 성공 -1 실패

POSIX Signal Handling

```

/* include signal */
#include "unp.h"

Sigfunc *
signal(int signo, Sigfunc *func)
{
    struct sigaction act, oact;

    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (signo == SIGALRM) {
#ifdef SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT; /* SunOS 4.x */
#else
        act.sa_flags |= SA_RESTART; /* SVR4, 44BSD */
#endif
    }
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
/* end signal */

Sigfunc *
Signal(int signo, Sigfunc *func) /* for our signal() function */
{
    Sigfunc *sigfunc;

    if ((sigfunc = signal(signo, func)) == SIG_ERR)
        err_sys("signal error");
    return(sigfunc);
}

```

Handling SIGCHLD Signals

SIGCHLD

- 자식프로세스는 종료하면 부모 프로세스에게 SIGCHLD 시그널을 보냄.

```
Signal (SIGCHLD, sig_chld);
```

- 위의 함수를 통해 SIGCHLD신호를 받으면 sig_chld 함수 호출

Handling SIGCHLD Signals

sig_chld 함수.

```
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 }
```

wait 함수를 통해 종료된 자식 프로세스의 정보를 받는다.



부모 프로세스가 정보를 전달받으면 운영체제가 자식 프로세스 소멸

wait and waitpid Functions

```
pid_t wait(int* status);
```

int* status : 자식 프로세스의 상태. 필요 없으면 NULL

return

-1 : wait 실패

다른 값 : 종료된 자식 프로세스의 ID, 종료 상태.

종료된 자식프로세스 존재 :

종료된 자식프로세스 값 반환.

종료된 자식프로세스 존재X :

첫번째 존재하는 자식 프로세스 종료까지 wait함수 block.

wait and waitpid Functions

```
pid_t waitpid(pid_t pid, int* status, int options);
```

pid_t pid : 자식 프로세스의 id

pid < -1 : -pid에 해당하는 프로세스 그룹 id의 모든 자식프로세스 종료 대기.

pid = -1 : 모든 자식프로세스 종료 대기.

pid = 0 : 현재 프로세스의 프로세스 그룹의 모든 자식 프로세스 종료 대기.

pid > 0 : pid에 해당하는 프로세스 종료 대기.

int* status : 자식 프로세스의 상태. 필요 없으면 NULL

int options : 다음 option 동작 수행 (OR 연산 사용가능)

WNOHANG : 자식 프로세스가 종료되길 기다리지 않는다.

0 : 자식프로세스의 종료 대기

return 값은 wait()함수와 동일.

wait and waitpid Functions

```
pid_t waitpid(pid_t pid, int* status, int options);
```

int* status

WIFEXITED : 자식 프로세스가 정상적으로 종료된 경우 true반환

WEXITSTATUS : WIFEXITED가 true를 반환 시 사용.

자식 프로세스의 종료 상태 반환.

WIFSINALED : 자식 프로세스가 signal에 의해 종료시 true반환.

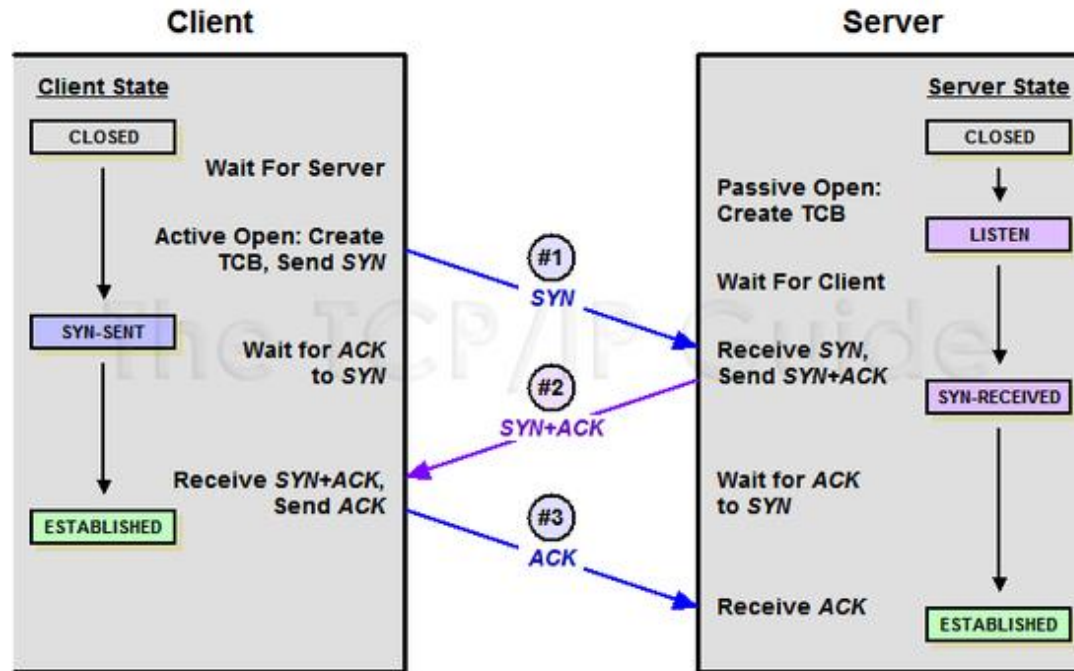
WTERMSIG : WIFSINALED가 true반환 시 사용.

자식 프로세스를 종료시킨 sinal 숫자 반환

WCOREDUMP : WIFSINALED가 true반환 시 사용.

자식 프로세스가 core dump생성시 true 반환.

Connection Abort before accept Returns



1. 3-way handshaking 마지막 ACK가 서버에 수신.
2. Connect 소켓 생성
3. 문제 발생.
4. 클라이언트에서 RST메세지가 서버로 도달.
5. RST메세지가 서버에서 accept()함수 리턴할 때 -1리턴
-> 오류로 간주.

Connection Abort before accept Returns

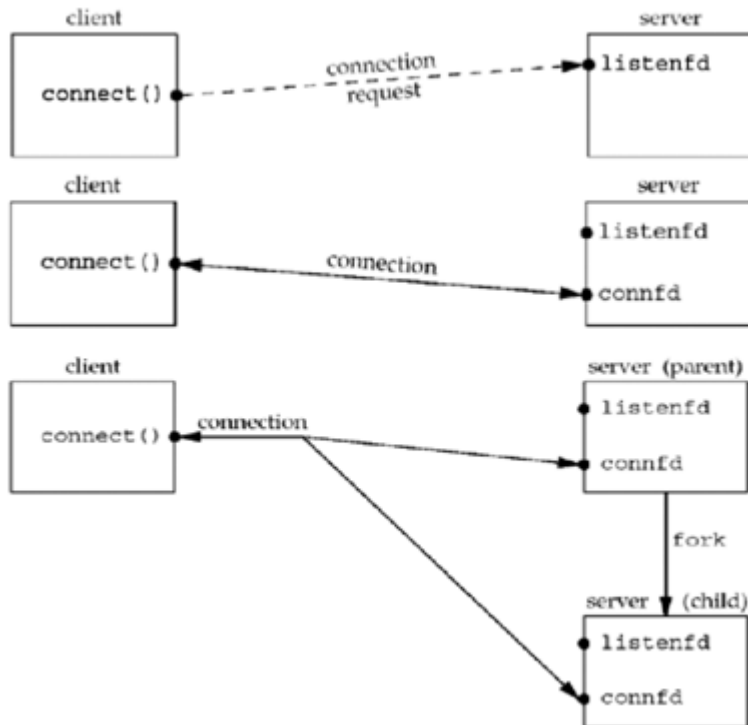
RST란?

재설정 과정. 양방향에서 동시에 발생하는 중단 작업.
비정상적인 세션 연결 끊기.

RST 발생 이유

- 연결을 맺고 있지 않은 호스트로부터 TCP 패킷이 온 경우.
- 부정확한 순서 번호나 승인 번호 필드를 가지는 메시지 수신
- 연결을 기다리는 프로세스가 없는 포트로 SYN메세지 수신

Connection Abort before accept Returns



<정상적인 연결 과정>



<문제 발생 시>

-> 의미없는 프로세스가 계속해서 생성.

Termination of Server Process

1. 클라이언트와 자식 프로세스가 연결
2. 자식 프로세스가 종료되면 클라이언트는 FIN을 받고 ACK를 전달
3. 부모 프로세스는 종료된 프로세스 정보를 전달 받음
4. 자식 프로세스 소멸
5. 클라이언트 fgets()가 대기상태

```
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
```

6. 클라이언트는 Error message를 받는다.

```
linux % tcpcli01 127.0.0.1      start client

another line                    we then type a second line to the client

str_cli : server terminated
prematurely
```

7. 클라이언트 종료 시 모든 디스크립터 종료

Termination of Server Process

해결 방안

-> select 함수와 poll 함수

```
int select(int nfds, fd_set* readSet, fd_set* writeSet, fd_set* exceptSet, const struct timeval* timeout);
```

지정한 소켓의 변화를 확인하는 함수.

처음에는 block되어있다가 특정 이벤트 발생시 작동한다.

int nfds : 디스크립터 마지막 번호 + 1 (0부터 시작이므로)

fd_set* readSet : 입력 스트림에 변화 확인 할 때 이용하는 소켓 정보 전달

fd_set* writeSet : 데이터 전송 시 블로킹 되지 않고 바로 전송이 가능 여부 확인 소켓 정보 전달

fd_set* exceptSet : 예외 발생 확인 소켓의 정보 전달

Const struct timeval* timeout : 함수 호출 후, 무한 대기 상태에 빠지지 않게 시간 제한 설정

return : 성공 (디스크립터 수), 타임아웃 (0), 실패 (-1)

Termination of Server Process

```
int poll(struct pollfd *fds, nfds_t nfd, int timeout);
```

select()와 같은 기능.

차이점 : 파일 디스크립터가 무제한적.

low level의 처리 -> system call의 호출이 더 적고, 이식성이 나쁨

struct pollfd : struct pollfd{

```
int fd; // file descriptor
```

```
short events; // request events
```

```
short revents; // returned events
```

```
};
```

short events :

POLLIN : 읽을 데이터가 있을 때

POLLPRI : 긴급 데이터를 읽을 것이 있을 때

POLLOUT : 바로 쓸 수 있는 상태

POLLWRBAND : 긴급 데이터를 쓸 수 있을 때

Termination of Server Process

```
int poll(struct pollfd *fds, nfd_t nfd, int timeout);
```

nfd_t nfd : 파일 디스크립터 개수

int timeout : ms단위로 입력.

-1 : 무한 대기

0 : 대기 없음

return

1 이상 : event발생한 파일 디스크립터 개수

0 : timeout 발생

-1 : 오류 발생

SIGPIPE Signal

클라이언트에서 read에서 오는 오류 무시하고 write 할 시
-> SIGPIPE signal 발생

연결이 끊어진 소켓에 send(),write() 등을 사용시에도 발생한다.

```
#include "unp.h"

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

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

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

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

        Fputs(recvline, stdout);
    }
}
```

SIGPIPE Signal

```
linux %tcpkill 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

Crashing of Server Host

문제 :

crashing 발생 시 클라이언트는 crashing 여부 모름.
-> 시간이 오래 걸림.

방법 :

fgets() -> write() -> read() (FIN,RST 오지 않는다면)
여러 번 재전송 (12번, 9분). client 커널 타임아웃.
read() 시스템 콜 -1 리턴. errno ETIMEDOUT설정

라우터를 통해 목적지로 통신중, 해당 ip로 통신 불가라는 정보를 가진 라우터를 통과하면, 그 라우터에서 ICMP메세지를 반송

*ICMP Message :

네트워크 오류 보고용, 상태 조사 메시지.

Crashing and Rebooting of Server Host

문제 :

서버가 다운되고 커짐

-> 서버에 꺼지기 전 설정된 소켓 정보 소멸

방법 :

fgets() -> write() -> read() 과정 후

서버에서 RTS 메시지 리턴.

read() 시스템 콜 -1 리턴. errno ETIMEDOUT 설정

Shutdown of Server Host

문제 :

서버 프로세스가 동작 중 서버 호스트 종료.
-> SIGTERM 신호를 모든 프로세스에게 보냄.

방법 :

실행 중인 프로세스 정리 및 시간 부여
SIGTERM으로 종료가 안되면 SIGKILL신호로 종료.
Open 디스크립터들을 닫는다.

해결 :

select와 poll함수 사용.

Thank you