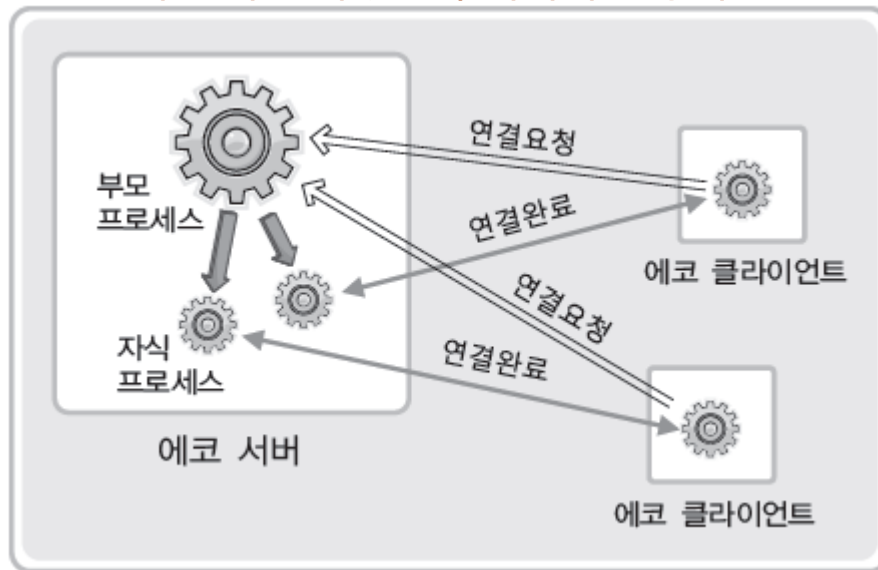


네트워크 프로그래밍

14. 멀티프로세스 기반의 서버구현 2

프로세스 기반 다중접속 서버 모델

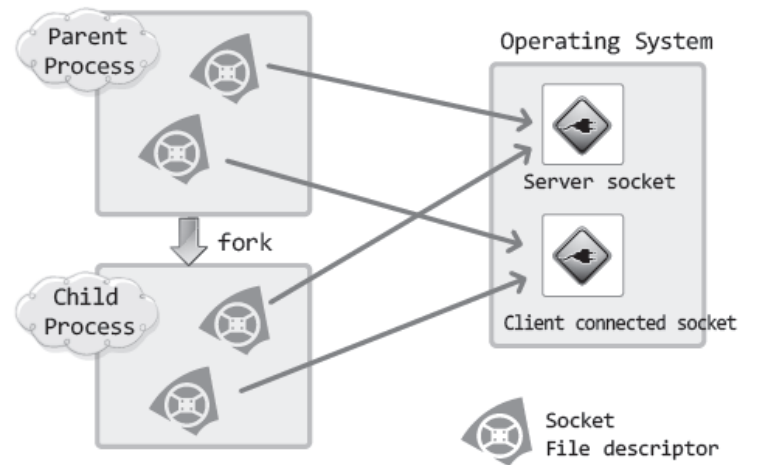
프로세스 기반 다중접속 서버의 전형적인 모델



핵심은 연결이 하나 생성될 때마다 프로세스를 생성해서 해당 클라이언트에 대해 서비스를 제공하는 것이다.

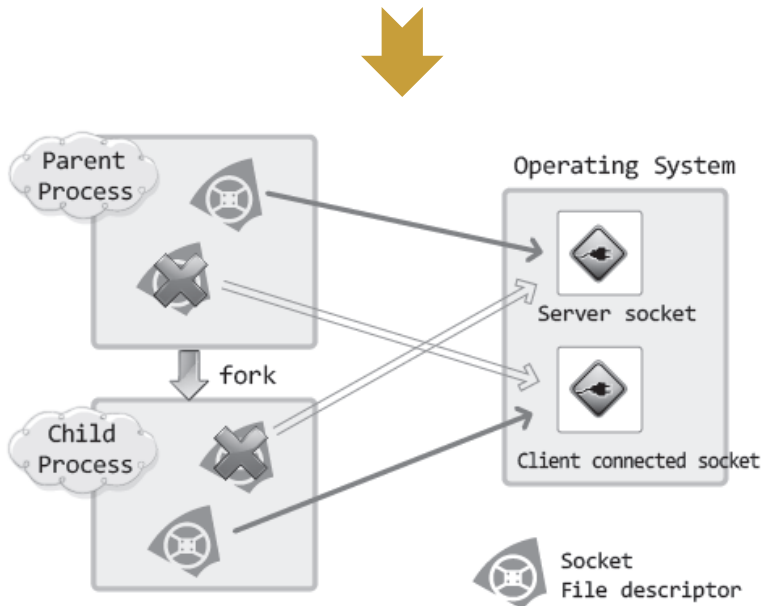
- 1단계 에코 서버(부모 프로세스)는 accept 함수호출을 통해서 연결요청을 수락한다.
- 2단계 이때 얻게 되는 소켓의 파일 디스크립터를 자식 프로세스를 생성해서 넘겨준다.
- 3단계 자식 프로세스는 전달받은 파일 디스크립터를 바탕으로 서비스를 제공한다.

fork 함수호출을 통한 디스크립터의 복사



한쪽에서 소켓을 닫을 때 소켓에대한 fd가 1개 이상일 경우
FIN이 날아가지 않는다.

프로세스가 복사되는 경우 해당 프로세스에 의해 만들어진
소켓이 복사되는 게 아니고, 파일 디스크립터가 복사된다.



하나의 소켓에 두 개의 파일 디스크립터가 존재하는 경우, 두
파일 디스크립터 모두 종료되어야 해당 소켓 소멸 그래서 **fork**
함수호출 후에는 서로에게 상관없는 파일 디스크립터를
종료한다.

다중접속 에코 서버의 구현

echo_mpserv.c의 일부

```
while(1)
{
    adr_sz=sizeof(clnt_adr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
    if(clnt_sock==-1)
        continue;
    else
        puts("new client connected...");
    pid=fork();
    if(pid==-1)
    {
        close(clnt_sock);
        continue;
    }
    if(pid==0) /* 자식 프로세스 실행영역 */
    {
        close(serv_sock);
        while((str_len=read(clnt_sock, buf, BUF_SIZE))!=0)
            write(clnt_sock, buf, str_len);

        close(clnt_sock);
        puts("client disconnected...");
        return 0;
    }
    else
        close(clnt_sock);
}
```

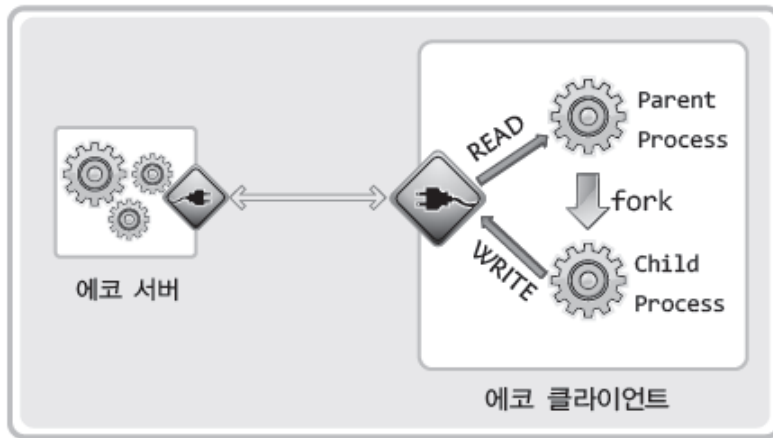
} 클라이언트와 연결되면

} 자식 프로세스를 생성해서

} 자식 프로세스가 서비스를 제공하게 한다.

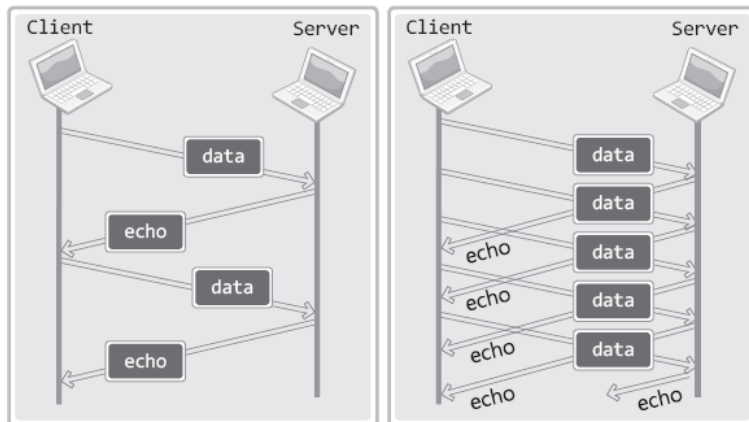
TCP의 입출력 루틴 분할

입출력 루틴 분할의 이점과 의미



프로세스 2개(혹은 스레드)만 있다면
full-duplex를 구현할 수 있다(동시 입출력)

입력을 담당하는 프로세스와 출력을 담당하는 프로세스를 각각 생성하면, 입력과 출력을 각각 별도로 진행시킬 수 있다.



입출력 루틴을 분할하면, 보내고 받는 구조가 아니라, 이 둘이 동시에 진행 가능하다.

에코 클라이언트의 입출력 분할의 예

```
if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
    error_handling("connect() error!");

pid=fork();
if(pid==0)
    write_routine(sock, buf);
else
    read_routine(sock, buf);
```

`close(sock);`

echo_mpclient.c의 일부

여기서 `read, write routine`의 `buf`가 서로 겹쳐도
>>fork로 탄생한 녀석이기 때문에 서로 다른 메모리영역을 가져서
잘 돌아감 문제 X

```
void read_routine(int sock, char *buf)
{
    while(1)
    {
        int str_len=read(sock, buf, BUF_SIZE);
        if(str_len==0)
            return;

        buf[str_len]=0;
        printf("Message from server: %s", buf);
    }
}

void write_routine(int sock, char *buf)
{
    while(1)
    {
        fgets(buf, BUF_SIZE, stdin);
        if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n"))
        {
            shutdown(sock, SHUT_WR);
            return;
        }
        write(sock, buf, strlen(buf));
    }
}
```

execl 함수로 프로그램 실행하기

```
#include <unistd.h>
int execl(const char * path, const char * arg, ...);
```

- path: 경로를 포함한 실행 프로그램의 이름
- arg: 프로그램 실행 인자

exec1 함수로 프로그램 실행하기

- fork : 새로운 프로세스의 생성이 아닌, 복사임
- exec관련 계열 함수로 프로그램을 실행
 - ▣ 새로운 프로세스를 만들지 않고, 원래의 프로세스 이미지를 덮어씀

exec1 함수로 프로그램 실행하기

command.c

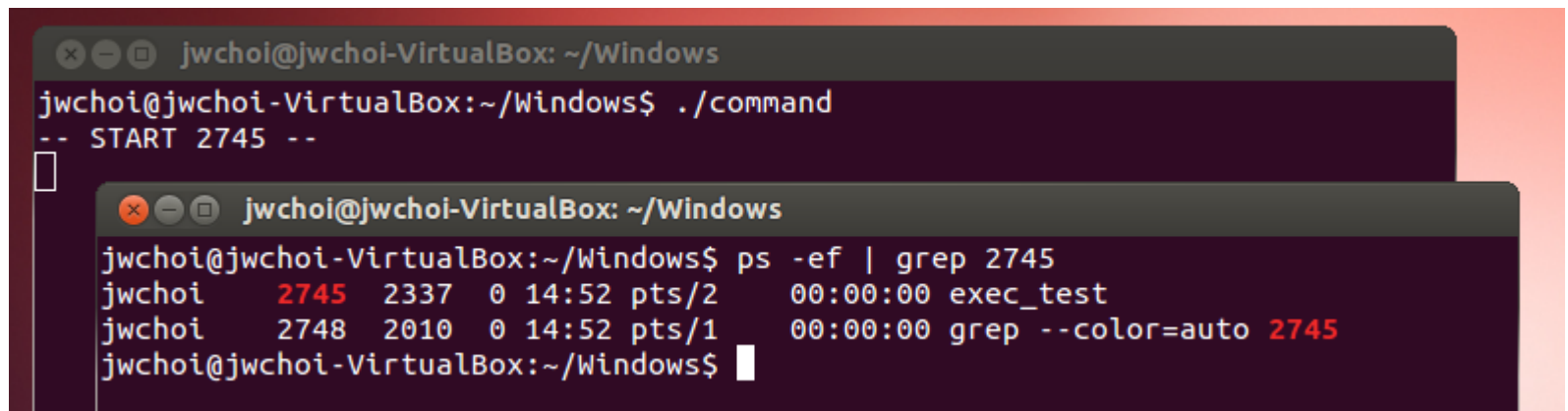
```
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
    printf("-- START %d--\n", getpid());
    execl("./exec_test", "exec_test", NULL);
    printf("-- END --\n");
    return 1;
}
```

exec_test.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
    getc(stdin);
    return 0;
}
```



The screenshot shows two terminal windows in a VirtualBox environment. The top window shows the execution of `./command`, which prints `-- START 2745 --`. The bottom window shows the output of `ps -ef | grep 2745`, which lists the `exec_test` process (PID 2745) and the `grep` process (PID 2748) that found it.

```
jwchoi@jwchoi-VirtualBox: ~/Windows
jwchoi@jwchoi-VirtualBox:~/Windows$ ./command
-- START 2745 --

jwchoi@jwchoi-VirtualBox:~/Windows
jwchoi@jwchoi-VirtualBox:~/Windows$ ps -ef | grep 2745
jwchoi      2745    2337    0 14:52 pts/2    00:00:00 exec_test
jwchoi      2748    2010    0 14:52 pts/1    00:00:00 grep --color=auto 2745
jwchoi@jwchoi-VirtualBox:~/Windows$
```

fork 함수와 execl 함수로 새로운 프로그램 실행하기



```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>

#define MAX_LINE 256
#define PROMPT "# "
#define chop(str) str[strlen(str) - 1] = 0x00;

int main(int argc, char **argv)
{
    char buf[MAX_LINE];
    int proc_status;
    pid_t pid;
    printf("My Shell Ver 1.0\n");
    while(1)
    {
        printf("%s", PROMPT);
        memset(buf, 0x00, MAX_LINE);
        fgets(buf, MAX_LINE - 1, stdin);
        if (strncmp(buf, "quit\n", 5) == 0)
        {
            break;
        }
        chop(buf);
        pid = fork();
        if(pid == 0)
        {
            if(execl(buf, buf, NULL) == -1)
            {
                printf("Execl failure\n");
                exit(0);
            }
        }
        if (pid > 0)
        {
            printf("Child wait\n");
            wait(&proc_status);
            printf("Child exit\n");
        }
    }
    return 0;
}
```

```
jwchoi@jwchoi-VirtualBox: ~/Windows
jwchoi@jwchoi-VirtualBox:~/Windows$ ./myshell
My Shell Ver 1.0
# /bin/ps
Child wait
  PID TTY          TIME CMD
 2010 pts/1        00:00:00 bash
 2875 pts/1        00:00:00 myshell
 2876 pts/1        00:00:00 ps
Child exit
# /bin/date
Child wait
Thu Apr 18 14:57:43 KST 2013
Child exit
# quit
jwchoi@jwchoi-VirtualBox:~/Windows$
```