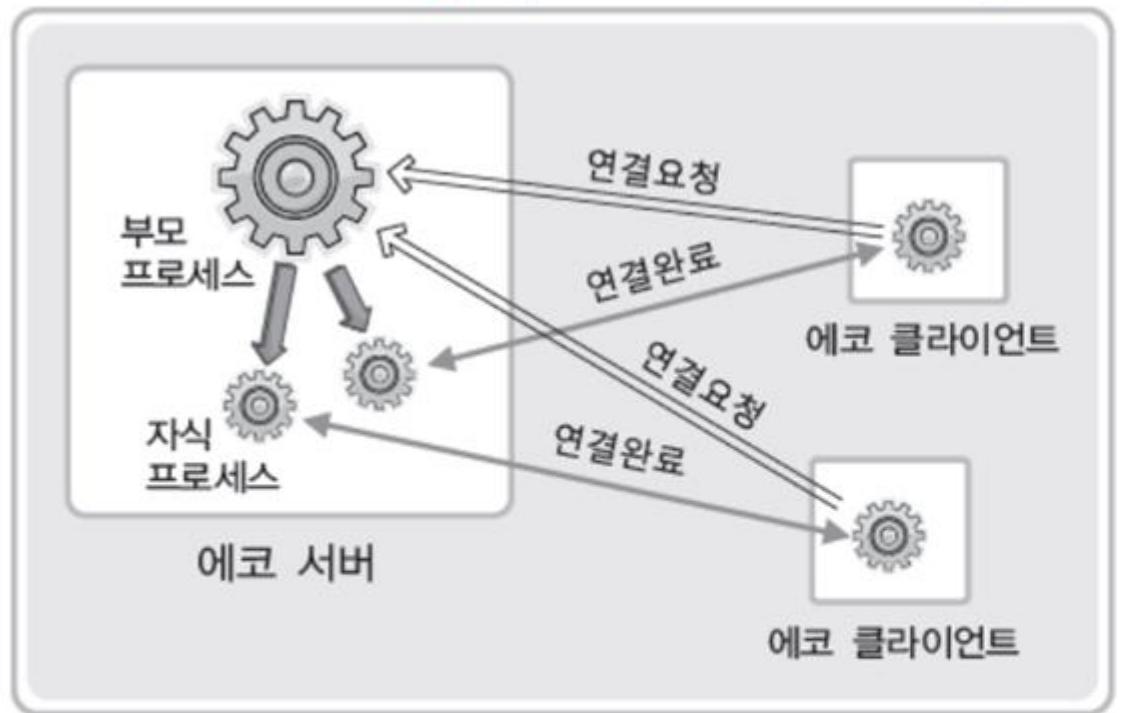


# Process 기반 다중 접속 서버

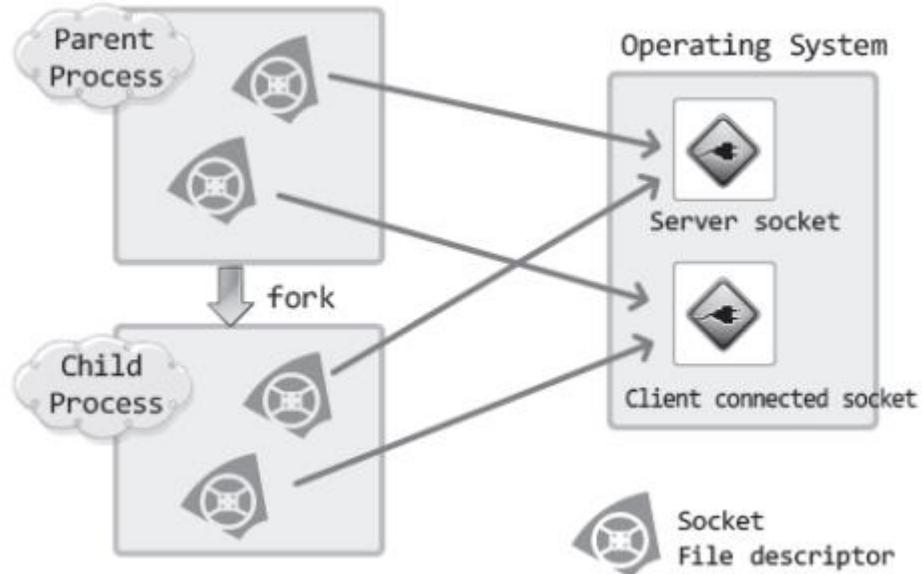
## 프로세스 기반 다중접속 서버의 전형적인 모델



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.

# Process 기반 다중 접속 서버



## 문제

이와 같은 상황에서,

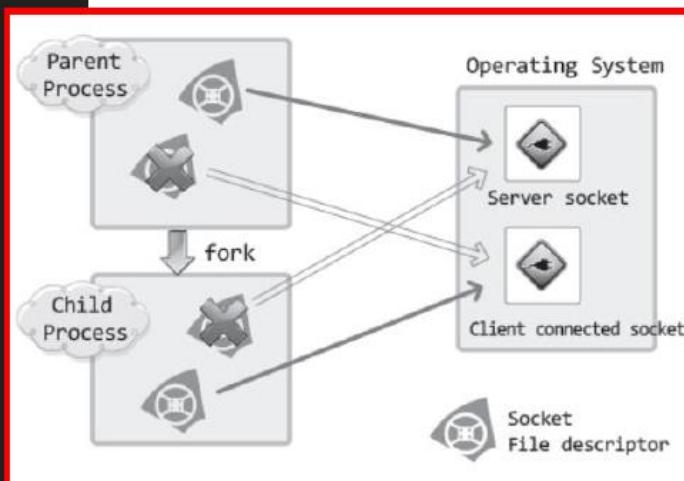
**Parent Process**가 Server socket을 종료한다고 하더라도

**Child Process**가 Server Socket discriptor를 가지고 있기 때문에 종료되지 않는다.

```
pid=fork();
if(pid== -1)
{
    close(clnt_sock);
    continue;
}
if(pid==0)
{
    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);
}
close(serv_sock);
return 0;
```

따라서 연결 이후 필요 없는 소켓은 바로 종료 해준다.



## Multi-Flexing 기반 서버

# Multi-Flexing 기반 서버 – Select 함수 이해

## 목표

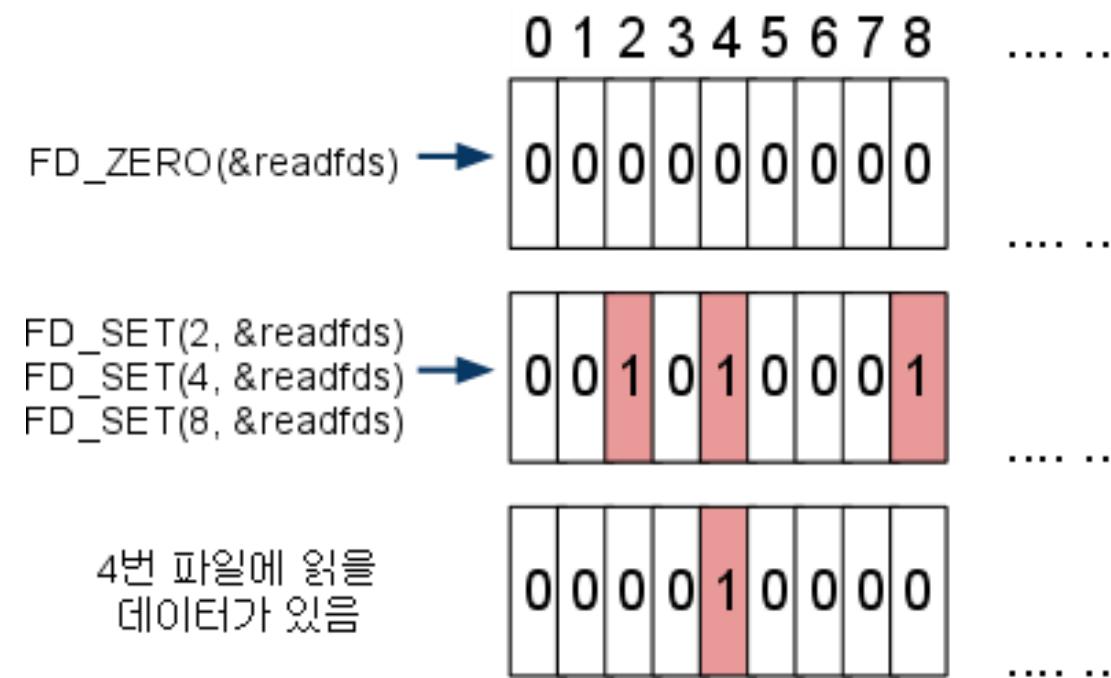
하나의 Process로 여러 요청을 처리하자.

## Point

서버 소켓에 **여러 연결 요청**이 들어오면, 그에 해당하는 **클라이언트 소켓**이 생성된다.

하나의 프로세스에서 **클라이언트 소켓들을 배열에 저장**해서 관리한다.

소켓에 R/W을 하기 위해서는 소켓을 관찰해야 한다. 따라서 배열에 **관찰 여부를 설정**해 놓는다.



2번, 4번, 8번 소켓(파일 디스크립터) Read 모니터링 설정  
→ 읽을 데이터가 존재하는가?  
※ 소켓 == 파일 디스크립터

위 배열을 복사 후 복사본을 select 함수에 전달하면,  
왼쪽과 같은 배열이 반환된다.

이를 통해 우리는 4번 배열에 READ할 데이터가 존재함을 알 수 있다.

# Multi-Flexing 기반 서버 - Select 함수 이해



```
int main(int argc, char *argv[])
{
    fd_set reads, temps;
    int result, str_len;
    char buf[BUF_SIZE];
    struct timeval timeout;
    FD_ZERO(&reads);
    FD_SET(0, &reads); // 0 is standard input(console)
    /*      검색의 대상 지정
    timeout.tv_sec=5;
    timeout.tv_usec=5000;
    */
    while(1)
    {
        /*      select 함수호출 후에는 타임아웃 발생식전
        의 시간이 담기므로 select 함수 호출 전에
        매번 초기화!
    }
```

**select 함수** temps=reads; **SELECT에 전달할 배열 복사**  
호출 후에 저 timeout.tv\_sec=5;  
장된 값이 바 timeout.tv\_usec=0;  
꺼니, reads result=select(1, &temps, 0, 0, &timeout);  
복사! if(result== -1) 데이터 입력여부 검사  
{  
 puts("select() error!");  
 break;  
}

이어짐 →

```
else if(result==0)
{
    puts("Time-out!");
}
else
{
    파일 디스크립터 0의 변화 관찰
    if(FD_ISSET(0, &temps))
    {
        str_len=read(0, buf, BUF_SIZE);
        buf[str_len]=0;
        printf("message from console: %s", buf);
    }
    return 0;
}
```

수신된 데이터가 있으므로 읽는다!

```
root@my_linux:/tcpip# gcc select.c -o select
root@my_linux:/tcpip# ./select
Hi~
message from console: Hi~
Hello~
message from console: Hello~
Time-out!
Time-out!
Good bye~
message from console: Good bye~
```

실행 결과