
select 함수

System Programming

2019 여름 계절학기

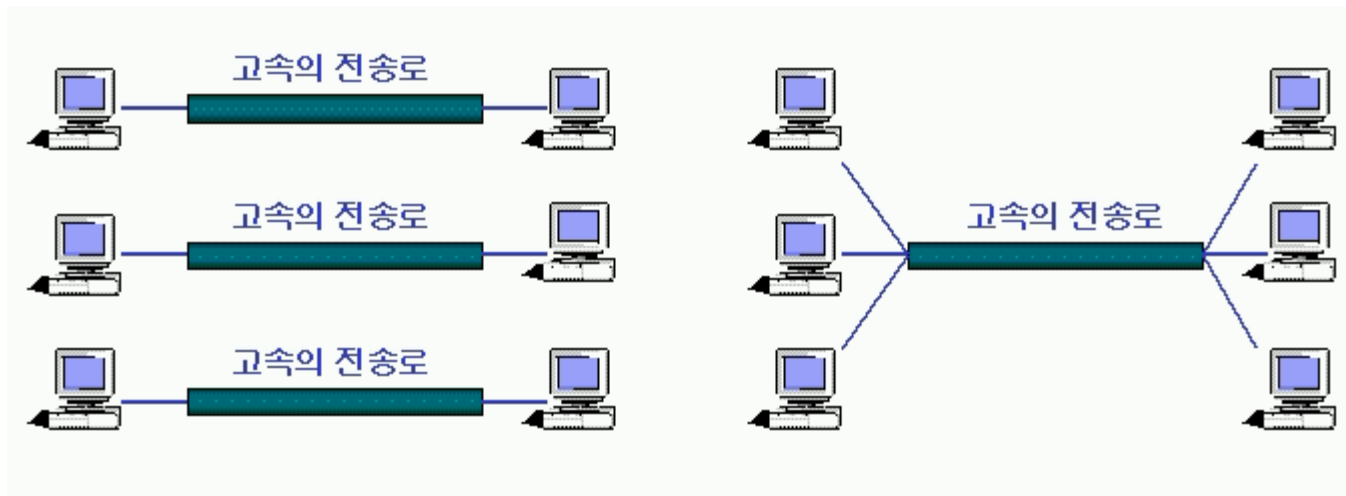
한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

멀티플렉싱이란?

멀티플렉싱

□ 멀티 플렉싱

- 하나의 전송로를 여러 사용자가 동시에 사용해서 효율성을 극대화하는 것



I/O 멀티플렉싱 기반의 서버

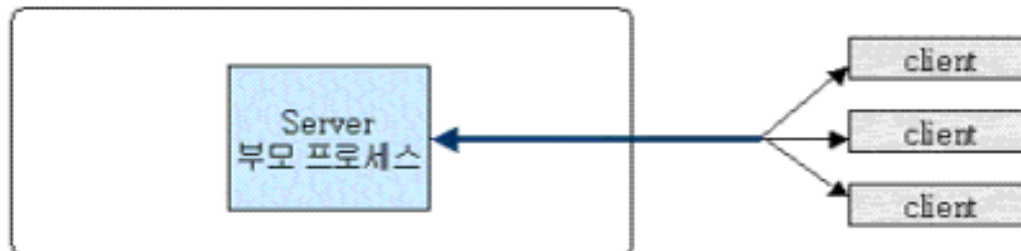
I/O 멀티플렉싱

□ I/O 멀티 플렉싱이란?

- 클라이언트와의 입/출력을 담당하는 프로세스를 하나로 묶어버리는 형식
- 프로세스가 고속의 전송로에 해당한다.



[그림 12-1]



[그림 12-2]

멀티 프로세스 vs. 멀티플렉싱

멀티 프로세스 vs 멀티플렉싱

1. 멀티 프로세스 기반의 서버

- 클라이언트와 서버간의 송수신 데이터 용량이 큰 경우
- 송수신이 연속적으로 발생하는 경우에 적합

2. 멀티플렉싱 기반의 서버

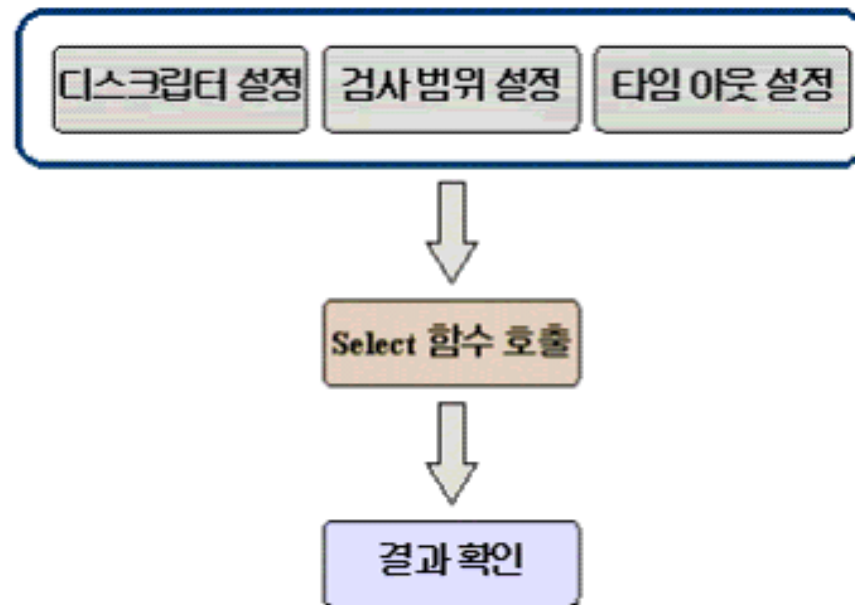
- 클라이언트와 서버간의 송수신 데이터 용량이 작은 경우
- 송수신이 연속적이지 않은 경우에 적합
- 멀티 프로세스 기반의 서버에 비해 많은 수의 클라이언트 처리에 적합

select 함수

select 함수의 기능과 호출 순서

1. 지정된 파일 디스크립터의 변화를 확인한다.

- 파일 디스크립터 변화 : 파일 디스크립터를 통해 데이터를 송수신 가능한 상태



[그림 12-3]

파일 디스크립터의 변화와 설정 1

1. 파일 디스크립터의 설정

- 변화를 확인할 파일 디스크립터를 구분지어 모아두는 것(총 3 묶음)

2. 파일 디스크립터의 변화

- 수신할 데이터가 존재하는가? (입력 버퍼에 데이터 존재)
- 데이터 전송이 가능한 상태인가? (출력 버퍼에 충분한 여유 공간 존재)
- 소켓에서 예외 상황이 발생하였는가? (OOB 메시지 전송)

3. fd_set 자료형

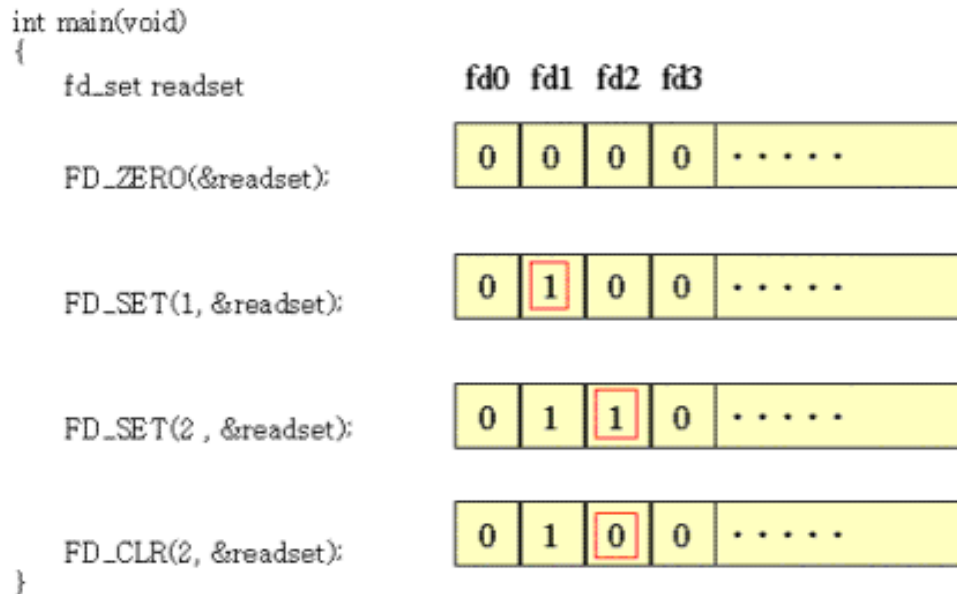
- 파일 디스크립터를 구분지어 모아두기 위한 자료형 (비트단위 배열)

fd0	fd1	fd2	fd3	
0	1	0	1

[그림 12-4]

select 함수

파일 디스크립터의 변화와 설정 2



[그림 12-5]

함수 선언
FD_ZERO(fd_set * fdset);
FD_SET(int fd, fd_set * fdset);
FD_CLR(int fd, fd_set * fdset);
FD_ISSET(int fd, fd_set * fdset);

[표 12-1]

select 함수

검사 범위와 타임 아웃의 설정

1. 검사해야할 파일 디스크립터의 범위를 지정해준다.
 - 실제로는 검사해야할 파일 디스크립터의 개수를 인자로 전달
 - 가장 큰 파일 디스크립터 값에 1을 더해서 인자로 전달한다
2. 타임 아웃을 설정한다.

```
struct timeval
{
    long        tv_sec;           /* seconds */
    long        tv_usec;         /* microseconds */
}
```


select 함수

select 함수의 호출 및 결과 확인 1

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int n, fd_set * readfds,
           fd_set * writefds, fd_set * exceptfds, struct timeval * timeout);
```

리턴 값	의미
-1	오류 발생
0	타임 아웃
0보다 큰 수	변화 발생 파일 디스크립터 수

select 함수

select 함수의 호출 및 결과 확인 2

□ select 함수 호출 전

- 변화에 관심이 있는 파일 디스크립터값을 셋팅(1)해서 호출

□ select 함수 호출 후

- 실제로 파일 디스크립터에 변화가 생긴 파일디스크립터만 셋팅되어서 리턴

fd0	fd1	fd2	fd3	
1	0	0	1

select 호출 전 readfds

fd0	fd1	fd2	fd3	
0	0	0	1

select 호출 후 readfds

[그림 12-6]

select 함수

select 함수 예제 코드 1

```
int main(int argc, char **argv)
{
    fd_set reads, temps;
    int result;

    char message[BUFSIZE];
    int str_len;
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads); /* standard input 설정 */

    /*
    timeout.tv_sec = 5;
    timeout.tv_usec = 100000;
    */ /* 이곳에 설정할 경우 문제 발생 */
    while(1)
    {
        temps=reads;
        timeout.tv_sec = 5;
        timeout.tv_usec = 0;

        result = select(1, &temps, 0, 0, &timeout);
        if(result == -1){ /*select 함수 오류 발생 */
            puts("select 오류 발생");
            exit(1);
        }
        else if(result == 0){ /* time-out에 의한 리턴 */
            puts("시간이 초과 되었습니다 : select ");
        }
        else { /* 파일 디스크립터 변화에 의한 리턴 */
            if(FD_ISSET(0, &temps)) {
                str_len = read(0, message, BUFSIZE);
                message[str_len]=0;
                fputs(message, stdout);
            }
        }
    }
} /* while(1) end */
```

이 코드를 바꾸기!

select 함수

select 함수 예제 코드 1

□ Select.c 코드 작성

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

#define BUFSIZE 30

int main(int argc, char **argv)
{
    fd_set reads, temps;
    int result;

    char message[BUFSIZE];
    int str_len;
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads);
```

(계속)

select 함수

select 함수 예제 코드 1

□ Select.c 코드 작성

```
while(1){
    temps = reads;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;

    result = select(1, &temps, 0, 0, &timeout);
    if(result == -1) {
        puts("select error");
        exit(1);
    }
    else if(result == 0){
        puts("time expired : select ");
    }
    else{
        if(FD_ISSET(0, &temps)){
            str_len = read(0, message, BUFSIZE);
            message[str_len]=0;
            fputs(message, stdout);
        }
    }
}
```

(끝)

select 함수

select 함수 예제 코드 1

□ Select.c 실행 화면

- 5초동안 입력이 있으면 “time expired : select”출력
- 5초전에 입력이 있으면 그대로 출력

```
sjhong@ubuntu: ~/sysprog
sjhong@ubuntu:~/sysprog$ ./select
time expired : select
sdf
sdf
qq
qq
ddd
ddd
time expired : select
^C
sjhong@ubuntu:~/sysprog$
```

select 함수

select 함수 예제 코드 1

□ Select2.c 실행 화면

- Select1.c를 수정해서 select2.c 작성
- FD_SET(0, &read); 바로 아래에 FD_CLR(0, &reads);코드 넣고 실행해보기
- 입력을 해도 반응이 없음을 확인하기

```
sjhong@ubuntu: ~/sysprog
sjhong@ubuntu:~/sysprog$ ./select2
time expired : select
sdf
time expired : select
dfewq
time expired : select
^C
sjhong@ubuntu:~/sysprog$
```

select 함수

select 함수 예제 코드 2

□ Tcp_server_select1.c 코드 작성

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/time.h>

#define BUFSIZE 1024

void error_handling(char * message);

int main()
{
    int state;
    int num=0;
    pid_t pid;

    int serv_sock;
    int clnt_sock;
    char message[BUFSIZE];
    int str_len;

    struct sockaddr_in serv_addr;
    struct sockaddr_in clnt_addr;
    int clnt_addr_size;

    int ret;
    int curr_fds;
    fd_set readset, tempset;
    int fd;
```

(계속)

select 함수

select 함수 예제 코드 2

□ Tcp_server_select1.c 코드 작성

```
FD_ZERO(&readset);

struct sigaction act;
act.sa_handler = SIG_IGN;
sigemptyset(&act.sa_mask);
act.sa_flags=0;

state = sigaction(SIGCHLD, &act, 0);
if(state != 0){
    puts("sigaction() error");
    exit(1);
}

serv_sock = socket(PF_INET, SOCK_STREAM, 0);
if(serv_sock == -1)
    error_handling("socket() error");

curr_fds = serv_sock;
FD_SET(serv_sock, &readset);

printf("fd:%d server socket\n", serv_sock);

memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_addr.sin_port=htons(4000);

if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
    error_handling("bind() error");

if(listen(serv_sock, 5) == -1)
    error_handling("listen() error");

clnt_addr_size = sizeof(clnt_addr);
```

select 함수

select 함수 예제 코드 2

□ Tcp_server_select1.c 코드 작성

```
while(1)
{
    tempset = readset;

    ret = select(curr_fds+1, &tempset, 0, 0, NULL);
    if(ret == -1)
        printf("select error\n");
    else if(ret == 0)
        printf("time out");
    else {
        for(fd=0; fd < curr_fds+1; fd++)
        {
            if(FD_ISSET(fd, &tempset))
            {
                if(fd == serv_sock){
                    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
                    printf("fd:%d client connected\n", clnt_sock);

                    FD_SET(clnt_sock, &readset);
                    if(curr_fds < clnt_sock)
                        curr_fds = clnt_sock;
                }
                /*
                 * For client sock communication, need to add some codes.
                 */
            }
        }
    }
}

return 0;
}

void error_handling(char * message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

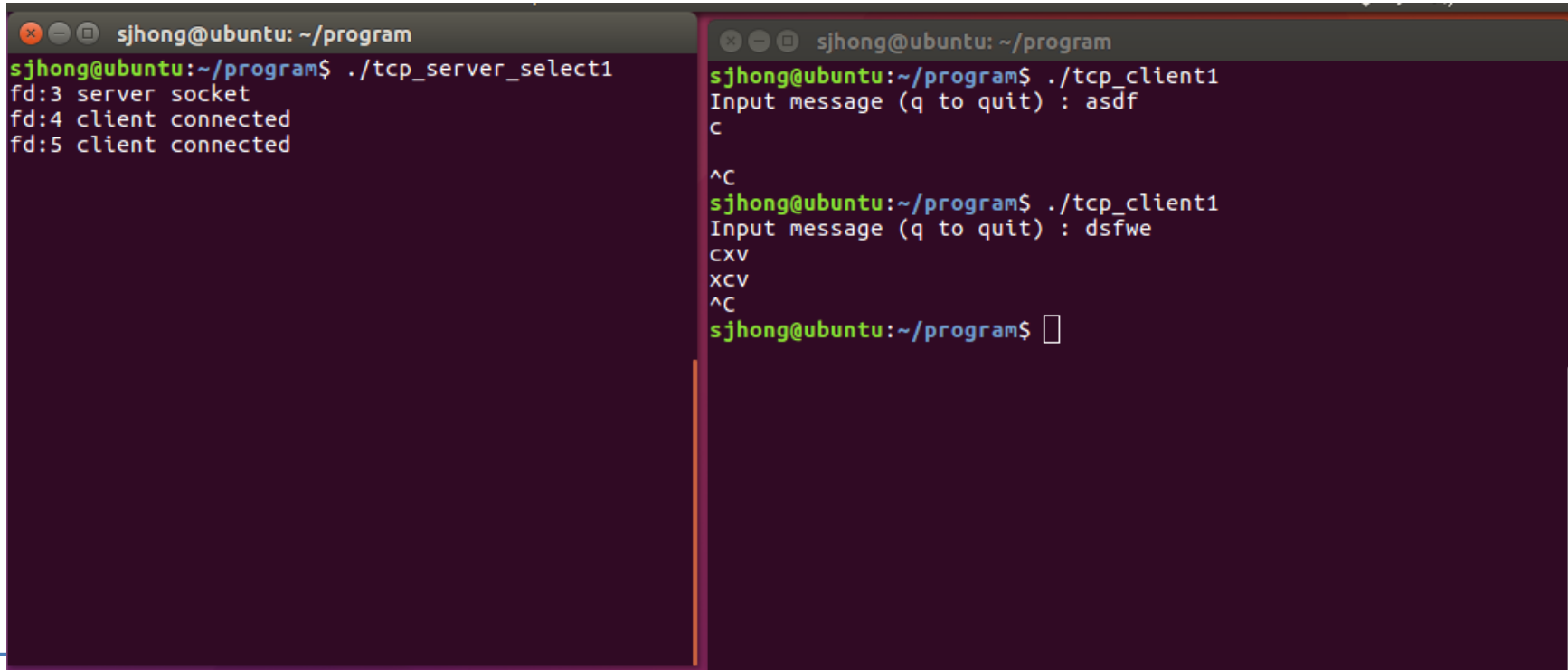
(끝)

select 함수

select 함수 예제 코드 2

□ Tcp_server_select1.c 코드 실행 및 tcp_client1으로 접속해보기

- 접속할 때마다 파일 디스크립터가 증가되면서 client 소켓이 열리는 것을 확인
- 하지만, client쪽에서 입력하는 값은 echo(다시 출력되는 것)되지 않음.
- 이것은 현재 서버 코드에서 client 소켓으로 들어오는 데이터 처리를 안하고 있기 때문임.



```
sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ ./tcp_server_select1
fd:3 server socket
fd:4 client connected
fd:5 client connected

sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ ./tcp_client1
Input message (q to quit) : asdf
c
^C
sjhong@ubuntu:~/program$ ./tcp_client1
Input message (q to quit) : dsfwe
cxv
xcv
^C
sjhong@ubuntu:~/program$
```

select 함수 예제 코드 2

□ Tcp_server_select2.c 코드 작성 및 tcp_client1으로 접속해서 실행

- 지금까지 자신이 배운 지식을 기반으로 client가 이전 다른 서버에서처럼 echo응답을 받을 수 있도록 수정해볼 것
- 이전의 멀티 프로세스 방식이나 멀티 쓰레드 방식에서처럼 여러 client의 접속 및 echo응답 수신이 가능하도록 수정할 것 (다음 슬라이드 참고)
- HINT : client가 q입력으로 접속을 종료한 경우에는 서버에서는 client로 EOF를 받게 되고, 이때의 읽은 데이터의 크기는 '0'이다.
- 또한, client에서 접속 종료시에는 서버에서 FD_CLR로 해당 소켓 디스크립터를 기존 readset에서 해제해줄 것.(그래야 다음번 검사때는 해당 디스크립터를 검사하지 않을 수 있으므로)

select 함수

select 함수 예제 코드 2

□ Tcp_server_select2.c 코드 작성 및 tcp_client1으로 접속해서 실행

- 정상적으로 접속 및 echo됨을 확인

```
sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ ./tcp_server_select2
fd:3 server socket
fd:4 client connected
fd:5 client connected
dsf;
sdf
ce
fd:5 client disconnected
fd:4 client disconnected
fd:4 client connected
dsf
c
fd:5 client connected
vsdf

```

```
sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ ./tcp_client1
Input message (q to quit) : dsf;
Message from server :dsf;

Input message (q to quit) : q
sjhong@ubuntu:~/program$ ./tcp_client1
Input message (q to quit) : dsf
Message from server :dsf

Input message (q to quit) : c
Message from server :c

Input message (q to quit) : 
```

```
sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ ./tcp_client1
Input message (q to quit) : sdf
Message from server :sdf

Input message (q to quit) : ce
Message from server :ce

Input message (q to quit) : q
sjhong@ubuntu:~/program$ ./tcp_client1 wfc
Input message (q to quit) : vsdf
Message from server :vsdf

Input message (q to quit) : 
```

Thank you for your attention !!

Q and A