

Chapter 7: I/O multiplexing

작성자: 김선영

메 일: sunyzero@gmail(dot)com

버 전: 1.01

I/O multiplexing

❖ 다중 입출력 통신

- 지체되어진 작업만큼 대기시간(**latency time**)이 늘어남
- 서비스 받을 준비가 되어진 채널부터 처리하도록 준비 이벤트를 감지하는 메커니즘
- **select(2)**, **poll(2)** 혹은 **epoll** 을 사용
 - » cf. IOCP(windows), BSD(kqueue)

C10K problem

❖ Client 10,000 개에 대한 처리

- fork 를 이용한 복제
- I/O multiplexer
- threading
- Realtime Extensions

❖ <http://www.kegel.com/c10k.html>

stateless vs stateful

❖ stateless system call

- kernel 내부에 **상태(state)**를 저장하지 않음
- 즉 **system call** 호출시마다 필요한 모든 정보가 커널로 복사되고, 리턴시에는 다시 유저영역으로 복사됨: **overhead !!**
- 과거 메모리가 매우 비싸고, 시스템에 작은 메모리가 장착되던 시절에는 메모리를 아끼는 것이 가장 최우선이었다는 점에서 과거의 **system call**은 **stateless** 방식으로 디자인 됨
- **select/poll**이 **stateless system call**의 대표적 함수

stateless vs stateful (con't)

❖ stateful system call

- 메모리의 가격이 떨어지고, 오히려 커널 내부의 메모리를 소모하더라도 메모리 복사(I/O관련)를 줄이는 것이 효과적!
 - » Jonathan Lemon, 2001 USENIX annual Technical Conference
- select/poll 을 대체하여 epoll, kqueue가 나오기 시작

select/pselect

❖ select(2)

- level trigger 사용
- 전수적 loop를 통해서 이벤트를 검사하는 오버헤드 존재
- 1024개의 fd 검사 가능

❖ pselect(2)

- timeout 지정 구조체의 변화
- 시그널 마스크 지정 가능
 - » sigprocmask()의 사용을 피할 수 있음

+ select/pselect (con't)

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

int pselect(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
            const struct timespec *timeout, const sigset_t *sigmask);
```

```
struct timeval {
    long    tv_sec;        /* Seconds */
    long    tv_usec;     /* Microseconds */
};

struct timespec {
    time_t  tv_sec;       /* Seconds. */
    long int tv_nsec;     /* Nanoseconds. */
};
```

- fd_set : 파일기술키 세트 마스크

select/pselect (con't)

읽기가능	readfds -소켓 수신버퍼에 데이터가 도착한 경우 리턴함 -소켓 수신버퍼에 접속연결해제요청(FIN)이 발생한 경우 리턴함 -리스너 소켓의 경우에 새로운 접속(SYN요청)이 있는 경우 리턴함 -스트림으로부터 데이터가 도착하였을 경우에 리턴함(e.g. stdin)
쓰기가능	writefds -송신소켓버퍼에 빈 공간이 생긴 경우에 리턴함 -반대편 peer가 끊어진 경우에 리턴함 -스트림에 데이터를 전송가능한 경우에 리턴함(e.g. non블록connect)
예외상황	TCP의 OOB데이터 (URG 플래그 지정됨)가 수신된 경우에 리턴함

FD_ZERO(fd_set *set)	set 을 초기화 한다
FD_SET(int fd, fd_set *set)	set 에 파일기술자 fd 를 더한다
FD_CLR(int fd, fd_set *set)	set 에서 파일기술자 fd 를 뺀다
FD_ISSET(int fd, fd_set *set)	set 에서 fd 에 이벤트가 발생했는지를 체크한다

select example

```
fd_set  fds_read;
int      fd_biggest;
int      fd_socket[MAX_FD_SOCKET];
int      cnt_fd_socket;
/* ... 생략 ... */
FD_ZERO(&fds_read);
FD_SET( ... , &fds_read);
/* ... 생략 ... */
ret_select = select(fd_biggest+1, &fds_read, NULL, NULL, NULL);
/* ... 생략 ... */
    for (i=1; i<cnt_fd_socket; i++) { /* 읽을 데이터가 존재하는 경우 */
        if (FD_ISSET(fd_socket[i], &fds_read)) {
            if ((n_recv = recv(fd_socket[i], buf_recv, sizeof(buf_recv), 0))
                == -1) {
                /* error */
            } else {
```

fd_set structure

```
#define __NFDBITS      (8 * sizeof (__fd_mask))
typedef long int __fd_mask;
typedef struct
{
    /* XPG4.2 requires this member name.  Otherwise avoid the name
       from the global namespace.  */
#ifdef __USE_XOPEN
    __fd_mask  fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->fds_bits)
#else
    __fd_mask  __fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->__fds_bits)
#endif
} fd_set;
#define FD_SETSIZE    __FD_SETSIZE
```

select example

❖ io_select.c (1/8)

```
#define USAGE    "%s <listen port>\n"
#define LISTEN_BACKLOG  20
#define MAX_FD_SOCKET  0xff
#define MAX(a, b)      (a >= b ? a : b)

int    fd_socket[MAX_FD_SOCKET];
int    cnt_fd_socket;

/* I/O multiplexing 을 위한 변수들 */
fd_set  fds_read;
int     fd_biggest;

int add_socket(int fd);
int del_socket(int fd);
int mk_fds(fd_set *fds, int *a_fd_socket);
int main(int argc, char *argv[])
{
```

+ select example (con't)

❖ io_select.c (2/8)

```
short    port;
socklen_t len_saddr;
struct sockaddr_in  saddr_s, saddr_c;
int      fd, fd_listener, n_recv, n_buffer;
char     buf_recv[1024];
int      ret_select;
int      i;

memset(&saddr_s, 0x00, sizeof(struct sockaddr_in));
memset(&saddr_c, 0x00, sizeof(struct sockaddr_in));
for (i=0; i<MAX_FD_SOCKET; i++) {
    fd_socket[i] = -1;
}
if (argc != 2)
    printf(USAGE, argv[0]);
if (argc == 2) {
    port = (short)atoi((char *)argv[1]);
} else {
    port = 0;    /* 포트번호가 지정되지 않으면 랜덤으로 */
}
```

+ select example (con't)

❖ io_select.c (3/8)

```
fd_listener = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
if (fd_listener == -1) {
    pr_err("Fail: socket()");
    exit(EXIT_FAILURE);
}
saddr_s.sin_family = AF_INET;          /* AF_INET, PF_INET 는 동일함 */
saddr_s.sin_addr.s_addr = INADDR_ANY; /* localhost의 모든주소에 대하여 적용 */
saddr_s.sin_port = htons(port);
if (bind(fd_listener, (struct sockaddr *)&saddr_s, sizeof(saddr_s)) == -1) {
    pr_err("Fail: bind()");
    exit(EXIT_FAILURE);
}
len_saddr = sizeof(saddr_s);
if (port == 0) {
    getsockname(fd_listener, (struct sockaddr *)&saddr_s, &len_saddr);
}
pr_out("Port : %#d", ntohs(saddr_s.sin_port));
listen(fd_listener, LISTEN_BACKLOG);
add_socket(fd_listener);
```

+ select example (con't)

❖ io_select.c (4/8)

```
while (1) {
    fd_biggest = mk_fds(&fds_read, fd_socket);
    ret_select = select(fd_biggest+1, &fds_read, NULL, NULL, NULL);
    if (ret_select == -1) {
        /* error */
    }
    pr_out("Select return (%d)", ret_select);
    if (FD_ISSET(fd_listener, &fds_read)) {
        fd = accept(fd_listener, (struct sockaddr *)&saddr_c, &len_saddr);
        if (fd == -1) {
            pr_err("Error get connection from listen socket");
            continue;
        }
        if (add_socket(fd) == -1) {           /* 에러처리 */
        }
        pr_out("Add socket (%d)", fd);
        continue;
    }
}
```

+ select example (con't)

❖ io_select.c (5/8)

```
for (i=1; i<cnt_fd_socket; i++) { /* 읽을 데이터가 존재하는 경우 */
    if (FD_ISSET(fd_socket[i], &fds_read)) {
        if ((n_recv = recv(fd_socket[i], buf_recv,
            sizeof(buf_recv), 0)) == -1) {
            pr_err("fd(%d) recv() error(%s)",
                fd_socket[i], strerror(errno));
        } else {
            if (n_recv == 0) { /* 연결이 닫힌 경우 */
                pr_out("fd(%d) was closed", fd_socket[i]);
                del_socket(fd_socket[i]);
            } else {
                pr_out("fd(%d) %d bytes received", fd_socket[i], n_recv);
            }
        }
    }
}
}
} /* while(1) 문의 끝 */
return 0;
}
```

select example (con't)

❖ io_select.c (6/8)

```
int add_socket(int fd)
{
    if (cnt_fd_socket < MAX_FD_SOCKET) {
        fd_socket[cnt_fd_socket] = fd;
        return ++cnt_fd_socket;
    } else {
        return -1;
    }
}
```

+ select example (con't)

❖ io_select.c (7/8)

```
int del_socket(int fd)
{
    int i, flag;
    flag = 0; /* 1:found, 0:not found */
    close(fd);
    for (i=0; i<cnt_fd_socket; i++) {
        /* 루프를 돌면서 제거할 파일기술자 번호를 찾는다 */
        if (fd_socket[i] == fd) {
            if (i != (cnt_fd_socket-1))
                fd_socket[i] = fd_socket[cnt_fd_socket-1];
            fd_socket[cnt_fd_socket-1] = -1;
            flag = 1;
        }
    }
    if (flag == 0)
        return -1; /* 못찾은 경우는 -1 로 리턴 */
    --cnt_fd_socket;
    return i;
}
```

+ select example (con't)

❖ io_select.c (8/8)

```
int mk_fds(fd_set *fds, int *a_fd_socket)
{
    int i, fd_max;
    FD_ZERO(fds); /* 파일기술자 세트 초기화 */
    for (i=0, fd_max = -1; i<cnt_fd_socket; i++) {
        fd_max = MAX(fd_max, a_fd_socket[i]);
        FD_SET(a_fd_socket[i], fds);
    }
    return fd_max;
}
```

`select()`는 리턴시 인수로 쓰인 `fd_set`을 이벤트 리스트를 저장하여 리턴해준다.
따라서 매번 `select()`호출시마다 `fd_set`을 새로 작성해주어야만 한다.