Part 03. 리눅스 소켓 프로그래밍

Chapter 06. 다중 입출력(1)

진행 순서

Chapter 06_01	다중 입출력 개요
Chapter 06_02	select
Chapter 06_03	pselect
Chapter 06_04	select 실습

Chapter 06_01 다중 입출력 개요

우리는 앞장에서 하나의 서버에 하나의 클라이언트가 접속하는 메커니즘과 그 구현에 초점을 맞추었습니다.
이제 서버에 연결된 클라이언트, 즉 소켓이 여러 개인 상황을 가정해 보겠습니다.

만약 서버를 구현하고 있는데 클라이언트 2개가 접속하여 accep된 상황을 생각해 보면,

```
listen(sfd, BACKLOG);
client1 = accept(sfd, &addr, &addr_size);

listen(sfd, BACKLOG);
client2 = accept(sfd, &addr, &addr_size);
```

대략 상기와 같이 2개의 클라이언트가 접속이 성공하였고,
2개의 클라이언트가 각각 데이터를 서버로 전송하는 경우를 생각해보면,
어느 클라이이언트, 즉 어느 소켓을 먼저 recv 처리해야할까?

만약 client2에서 send한 데이터가 먼저 서버로 도착했는데
client1 소켓에서 recv를 하면 도착한 데이터가 없기때문에 블록킹 모드라면 무한 대기 상태에 빠져들 것이고,
넌블록킹 모드라면 에러가 발생할 것입니다.

이와 같은 상황을 위해 다중 입출력(I/O Multiplexing) 메커니즘이 존재합니다.
먼저 데이터가 도착(이벤트가 발생)한 소켓을 인지하여 해당 소켓이 어떤 소켓인지를 알려주는 메커니즘 입니다.

Chapter 06_01 다중 입출력 개요

다중 입출력, 즉 I/O Multiplexing은 이벤트를 감지할 수 있습니다.

다중 입출력 지원함수들로 아래와 같은 함수들이 존재합니다.

select	기본적인 다중 입출력 함수, 소켓 파일 기술자가 클 경우(int 번호가 큰 경우) 비효율적
pselect	timeout 시 나노 초까지 설정할 수 있음, 시그널 마스크 값을 설정할 수 있음
poll	select 대비 효율성 (selec의 경우 이벤트 감지를 위해 항상 max_fd 만큼 loop, 이에 반해 poll은 원하는 만큼만 지정하여 loop)
epoll	고성능, 엣지 트리거(edge trigger) 지원 가능

Chapter 06_02 select

#include <sys/select.h>

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

select()는 프로그램이 여러 파일 디스크립터를 모니터 할 수 있게 하여 하나 이상의 파일 디스크립터가 일부 I/O 조작 클래스(예 : 입력 가능)에 대해 "준비"될 때까지 대기합니다.

세 개의 독립적 인 파일 디스크립터 세트가 감시됩니다.
readfds는 읽기를 감시하며, writefds는 쓰기를 감시하며, exceptfds는 예외를 감시합니다.

nfds는 세 세트 중 하나에 1을 더한 파일 번호입니다.

timeout 인수는 select()가 파일 디스크립터가 준비 될 때까지 대기해야 하는 최소 시간 간격을 지정합니다.
즉, select()가 이벤트 감지 중 이벤트가 감지되지 않은 상태에서 시간이 경과한 경우 select()는 리턴됩니다.
timeout이 NULL인 경우 select()는 무한 대기할 수 있습니다.

성공하면 select() 는 리턴 된 세 개의 디스크립터 세트에 포함 된 파일 디스크립터 수 (즉, readfds, writefds, exceptfds를 포함한 총 비트 수)를 리턴합니다. timeout 시간이 만료되면 0이 리턴됩니다. 오류가 발생하면 -1이 리턴되고 errno가 적절하게 설정됩니다.

Chapter 06_02 select

```
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

세트를 조작하기 위해 4개의 매크로가 제공됩니다.
FD_ZERO()는 세트를 지웁니다.
FD_SET() 및 FD_CLR()은 각각 주어진 파일 디스크립터를 세트에서 추가하고 제거합니다.
FD_ISSET()은 파일 디스크립터가 세트의 일부인지 여부를 테스트합니다. select()가 리턴 된 후 체크에 유용합니다.

Chapter 06_02 select

man select
select example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/select.h>

int main(void)
{
        fd_set rfds;
        struct timeval tv;
        int retval;

        /* Watch stdin (fd 0) to see when it has input. */
        FD_ZERO(&rfds);
        FD_SET(0, &rfds);

        /* Wait up to five seconds. */
        tv.tv_sec = 5;
        tv.tv_usec = 0;
```

```
        retval = select(1, &rfds, NULL, NULL, &tv);
        /* Don't rely on the value of tv now! */

        if (retval == -1)
                perror("select()");
        else if (retval)
                printf("Data is available now.\n");
                /* FD_ISSET(0, &rfds) will be true. */
        else
                printf("No data within five seconds.\n");

        exit(EXIT_SUCCESS);
}
```

Chapter 06_03 pselect

```
#include <sys/select.h>

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

select()와 pselect()의 작동은 다음 세 가지 차이점 외에는 동일합니다.

1) select()는 struct timeval (초 및 마이크로 초) 인 시간 초과를 사용하는 반면
pselect()는 struct timespec (초 및 나노초)을 사용합니다.

2) select()는 시간 초과 인수를 업데이트하여 남은 시간을 나타냅니다. pselect()는 이 인수를 변경하지 않습니다.

3) select()에는 sigmask 인수가 없으며 pselect()는 sigmask 설정이 가능합니다.

select()는 이벤트 감지 도중(블록킹 중) 시그널이 발생하면 에러를 리턴하고 빠져나가기 때문에 시그널 핸들러를 사용한다면 전역적인 시그널 차단 마스크를 설정해야 한다. (sigprocmask 사용)

pselect ()는 디스크립터를 검사하기 전에 sigmask가 가리키는 신호 세트로 호출자의 시그널 마스크를 대체하고 리턴하기 전에 호출자의 시그널 마스크를 복원합니다.
sigmask가 NULL 인 pselect()를 호출하는 것은 select ()를 호출하는 것과 같습니다.

Chapter 06_04 select 실습

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define handle_error(msg) \
   do { perror(msg); exit(EXIT_FAILURE); } while (0)

#define LISTEN_BACKLOG 50
#define MAX(a, b) ((a>b)?a:b)
#define MAX_SOCKFDS 50

int sockfds[MAX_SOCKFDS]; // store socket file descriptor
int cnt_sockfds;    // count of stored socket file descriptor

void init_sockfds()
{
   int i;
   for (i = 0; i < MAX_SOCKFDS; i++)
      sockfds[i] = -1;
   cnt_sockfds = 0;
}
```

```
int add_socket(int fd)
{
   if (cnt_sockfds >= MAX_SOCKFDS)
      return -1;

   sockfds[cnt_sockfds++] = fd;
   return cnt_sockfds;
}

void del_socket(int fd)
{
   int i, lastidx = cnt_sockfds-1;
   for (i = 1; i < cnt_sockfds; i++) {
      if (sockfds[i] == fd) {
         if (i != lastidx)
            sockfds[i] = sockfds[lastidx];
         sockfds[lastidx] = -1;
         cnt_sockfds--;
         break;
      }
   }
}
```

Chapter 06_04 select 실습

```
int main(int argc, char *argv[])
{
    int sfd, cfd;
    int portnum;
    struct sockaddr_in saddr;
    struct sockaddr_in caddr;
    socklen_t caddr_len;
    int ret;
    char buf[1024] = {0,};

    fd_set readfds;
    int ret_select;
    int i, maxfd = 0;

    if (argc < 2) {
        printf("Usage: %s <port>\n", argv[0]);
        exit(1);
    }

    portnum = atoi(argv[1]);

    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd == -1)
        handle_error("socket");
```

```
memset(&saddr, 0, sizeof(struct sockaddr_in));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(portnum);

if (bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr)) == -1)
    handle_error("bind");

if (listen(sfd, LISTEN_BACKLOG) == -1)
    handle_error("listen");

init_sockfds();

add_socket(sfd);

while(1) {
    maxfd = 0;
    FD_ZERO(&readfds);
    for (i = 0; i < cnt_sockfds; i++) {
        FD_SET(sockfds[i], &readfds);
        maxfd = MAX(maxfd, sockfds[i]);
    }
```

Chapter 06_04 select 실습

```
    ret_select = select(maxfd+1, &readfds, NULL, NULL, NULL);
    if (ret_select == -1) {
        perror("select error");
        exit(1);
    }

    if (FD_ISSET(sfd, &readfds)) { // is connect ?
        caddr_len = sizeof(caddr);
        cfd = accept(sfd, (struct sockaddr *)&caddr, &caddr_len);
        if (cfd == -1)
            handle_error("accept");

        if (add_socket(cfd) == -1) {
            printf("error: current socket count (%d) / max socket scount (%d)\n",
                cnt_sockfds, MAX_SOCKFDS);
        } else {
            printf("add socket(%d) OK! (IP: %s, Port: %d)\n", cfd,
                inet_ntoa(((struct sockaddr_in *)&caddr)->sin_addr),
                ntohs(((struct sockaddr_in *)&caddr)->sin_port));
        }
        continue;
    }
```

```
    for (i = 1; i < cnt_sockfds; i++) {
        int sockfd = sockfds[i];
        if (sockfd >= 0 && FD_ISSET(sockfd, &readfds)) {
            memset(buf, 0, sizeof(buf));
            ret = recv(sockfd, buf, sizeof(buf), 0);
            if (ret == -1)
                handle_error("recv");
            else if (ret == 0) { // close
                del_socket(sockfd);
                printf("del socket(%d) OK!\n", sockfd);
                close(sockfd);
            } else {
                printf("recv data (sock: %d): %s\n", sockfd, buf);
            }
        }
    }

    return 0;
}
```

Chapter 06_04 select 실습

CentOS

```
[root@localhost ch02]# ./client
Usage: ./client <ipaddr> <port>
[root@localhost ch02]# ./client 127.0.0.1 1234
Input message(.:quit): aaa
Input message(.:quit): hello this is aaa
Input message(.:quit): .
finish.
[root@localhost ch02]#
```

CentOS

```
[parallels@localhost ch02]$ ./client 127.0.0.1 1234
Input message(.:quit): bbb
Input message(.:quit): hello this is bbb
Input message(.:quit): anybody here?
Input message(.:quit): .
finish.
[parallels@localhost ch02]$
```

CentOS

```
[root@localhost ch05]# gcc -g select_server.c -o select_server
[root@localhost ch05]# ./select_server
Usage: ./select_server <port>
[root@localhost ch05]# ./select_server 1234
add socket(4) OK! (IP: 127.0.0.1, Port: 50422)
add socket(5) OK! (IP: 127.0.0.1, Port: 50424)
recv data (sock: 4): aaa

recv data (sock: 5): bbb

recv data (sock: 4): hello this is aaa

recv data (sock: 5): hello this is bbb

del socket(4) OK!
recv data (sock: 5): anybody here?

del socket(5) OK!
```