epoll

select() 함수와 마찬가지로 다수의 fd를 관찰하며 요청이 온 fd를 발견하면 작업 수행
리눅스에만 존재한다

- 과정

1. epoll 객체 생성
2. epoll 객체 제어
3. 모니터링 시작
4. 조건에 맞는 fd의 I/O 수행

epoll 객체 생성

```
epoll_create(int size);
epoll_create1(int flags);
```

size: 관리 fd 개수 flag: EPOLL_CLOEXEC 지정시 해당 옵션이 지정된 다른 fd를 닫고 자신은 열기

epoll 객체 제어

```
struct epoll_event{
    int events;
    epoll_data_t data;
}
typedef union epoll_data{
    void *ptr;
    int fdl
    int u32;
    int u64;
} epoll_data_t;
```

```
epoll_ctl(epfd, op, fd, epoll_event);
//epoll fd, 함수를 통해 하려는 작업, 모니터링 fd, epoll evnet 종류와 정보 전달
```

epoll 모니터링

```
epoll_wait(epfd, events, maxevents, timeout);
```

epolltcpsrv.c

```
#include <stdio.h>
#include <stdlib.h>
```

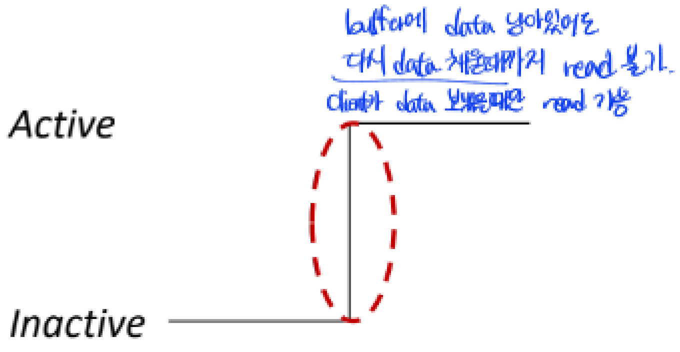
```c
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/epoll.h>
#include <errno.h>

#define MAX_EVENTS    10

void errProc(const char*);

int main(int argc, char** argv)
{
    int listenSd, connectSd;
    struct sockaddr_in srvAddr, clntAddr;
    int clntAddrLen, readLen;
    char rBuff[BUFSIZ];
    int i;

    int epfd, ready, readfd;
    struct epoll_event ev;
    struct epoll_event events[MAX_EVENTS];

    if(argc != 2)
    {
        printf("Usage: %s [Port Number]\n", argv[0]);
        return -1;
    }

    printf("Server start...\n");

    //epoll 생성
    epfd = epoll_create(1);
    if(epfd == -1) errProc("epoll_create");

    //듣기 소켓 생성
    listenSd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(listenSd == -1 ) errProc("socket");

    memset(&srvAddr, 0, sizeof(srvAddr));
    srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    srvAddr.sin_family = AF_INET;
    srvAddr.sin_port = htons(atoi(argv[1]));

    //port 할당
    if(bind(listenSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr)) == -1)
        errProc("bind");

    //듣기
    if(listen(listenSd, 5) < 0) errProc("listen");


    ev.events = EPOLLIN; //읽기 동작
    ev.data.fd = listenSd; //듣기 소켓

    //객체 제어
    //EPOLL_CTL_ADDL: 관심 리스트(epfd)에 listenSd 넣기
    if(epoll_ctl(epfd, EPOLL_CTL_ADD, listenSd, &ev) == -1)
        errProc("epoll_ctl");

    clntAddrLen = sizeof(clntAddr);

    while(1) {
        printf("Monitoring ... \n");

        //epfd에서 events가 올때까지 -1 무한대기
        ready = epoll_wait(epfd, events, MAX_EVENTS, -1);
        printf("ready: %d\n", ready);
        //ready: 이벤트 발생한 fd 개수
        if(ready == -1) {
            if(errno == EINTR) continue;
            else errProc("epoll_wait");
        }

        //이벤트 발생시
        //events 배열에는 ready된 만큼의 fd개수가 순서대로 저장되어있음
        //따라서 for(0~ready-1)로 접근하는 것.
        for(i=0; i<ready; i++)    {
            //listen socket이면 새로운 client 연결
            printf("fd: %d\n",events[i].data.fd);
            if(events[i].data.fd == listenSd) {

                //client 연결
                connectSd = accept(listenSd, (struct sockaddr *) &clntAddr, &clntAd
                if(connectSd == -1)    {
                    fprintf(stderr,"Accept Error");
                    continue;
                }
                fprintf(stderr,"A client is connected...\n");

                ev.data.fd = connectSd;
                //연결한 client를 관심있게 설정
                if(epoll_ctl(epfd, EPOLL_CTL_ADD, connectSd, &ev) == -1)
                    errProc("epoll_ctl");
            }
            //기존 client의 요청
            else {//IO
                readfd = events[i].data.fd;
                //읽기
                readLen = read(readfd, rBuff, sizeof(rBuff)-1);
                if(readLen == 0)
                {
                    fprintf(stderr,"A client is disconnected...\n");
                    //관심 list에서 제거
                    if(epoll_ctl(epfd, EPOLL_CTL_DEL, readfd, &ev) == -1)
                        errProc("epoll_ctl");
                    close(readfd);
                    continue;
                }
                rBuff[readLen] = '\0';
                printf("Client(%d): %s\n", events[i].data.fd,rBuff);
                write(events[i].data.fd, rBuff, strlen(rBuff));
```

```c
                }
            }
        }
        close(listenSd);
        close(epfd);
        return 0;
}

void errProc(const char * str)
{
        fprintf(stderr,"%s: %s", str, strerror(errno));
        exit(1);
}
```

epoll을 이용해서 thread, multiplex 없이 다중통신을 구현할 수 있다. 하지만 epoll은 리눅스 only

epoll mode

- Edge-triggered, 이벤트 일어나면 알리기

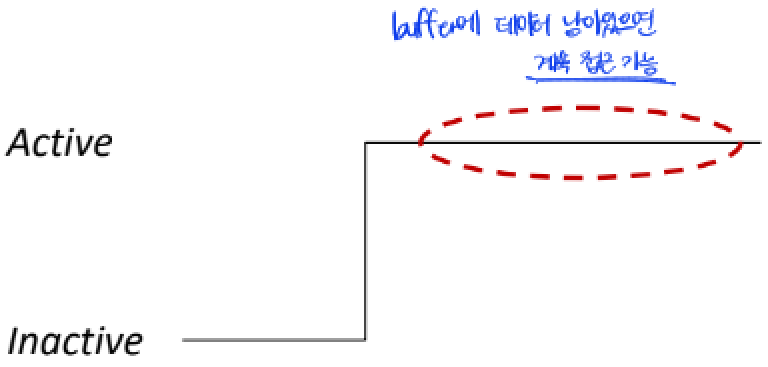

buffer에 data 남아있어도
다시 data 채울때까지 read 불가.
client가 data 보냈을때만 read 가능
Active
Inactive

버퍼에 데이터 남아있어도 다시 data 채울 때까지 read하지 못한다.
이벤트가 발생했을때 알리고, 그것을 처리하였으므로 다시 이벤트 발생까지 block되는 형식

epoll_event 구조체의 event 변수에 EPOLLET를 설정한다.
-> epoll_wait 호출시 무한대기 X, 값이 있는지 확인 후 없으면 error

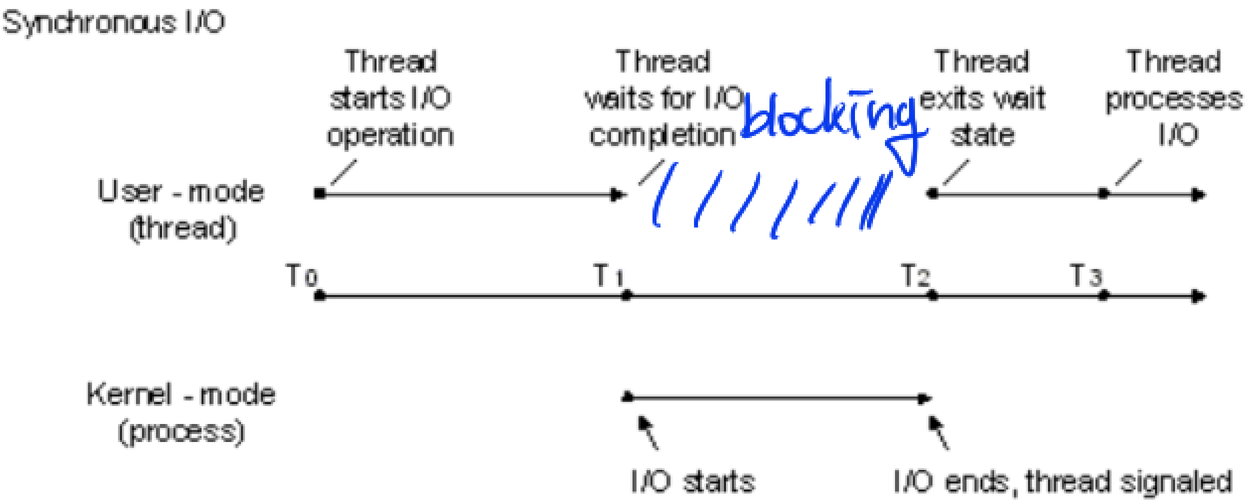
level trigger에 비해 edge를 이용하면 필요 이상의 epoll_wait 함수 호출을 줄일 수 있다.
non-blocking 소켓일때, edge의 data 왔을때만 읽는 게 중요

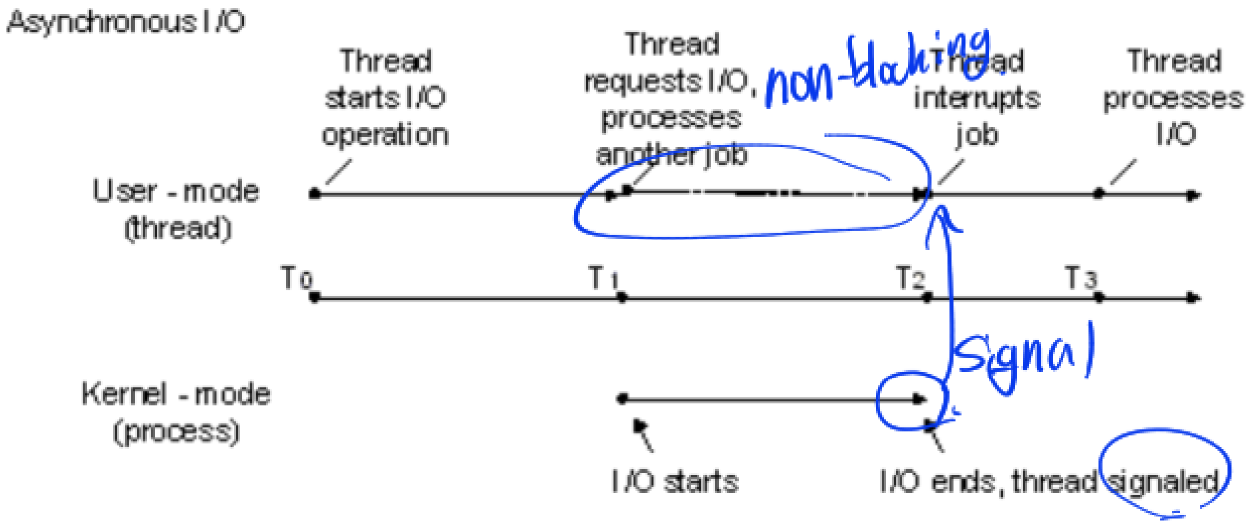
- Level-triggered, 무언가가 이용 가능하면 알리기



버퍼에 데이터 남아있으면 계속 접근 가능하다

epoll I/O way





- blocking read/write 기다림

- non-blockding 읽어올 때까지 기다리지 않고 동작한다.
  다른 thread 수행 끝나면 signal해주는 형식

1. synchronous blocking

- 가장 흔한 모델
- 자원 읽는 동시에 App은 block, kernel이 끝날 때까지 기다림

2. synchronous non-blocking I/O

- I/O작업 끝난 걸 app이 확인 불가하다
- 계속 read 요청, kernel은 읽을 거 없다고 대답, 반복
- 다 읽으면 read 요청시 리턴

3. asynchronous blocking

- select 모듈을 이용함
- read() 자체는 non-blocking하지만 select때문에 block됨

4. asynchronous non-blocking

- read 요청 후 다른 작업 하기
- 계속 read 요청이 아니라 signal 받으면 read하는 형태로..

non-blocking socket

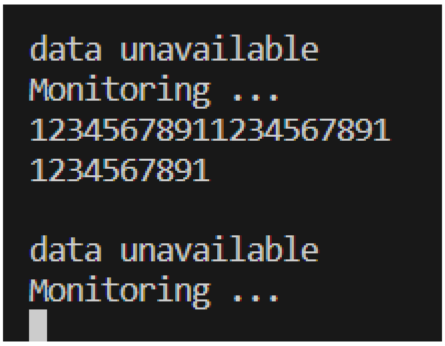
epoll edge-trigger와 함께 사용한다.
수신 버퍼에 데이터가 없을 경우, 에러코드 반환함 (=수신 데이터가 올 때까지 기다리지 않음)

non-blocking socket을 위한 fcntl()

- fcntl(fd, cmd..)

epollsrv2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/epoll.h>
#include <errno.h>
#include <fcntl.h>

#define MAX_EVENTS    10
void errProc(const char *);
int makeNbSocket(int);

int main(int argc, char** argv)
{
    int listenSd, connectSd;
    struct sockaddr_in srvAddr, clntAddr;
    int clntAddrLen, readLen;
    char rBuff[20];
    int i, completed = 0;

    int epfd, ready, readfd;
    struct epoll_event ev;
    struct epoll_event events[MAX_EVENTS];

    if(argc != 2)
    {
        printf("Usage: %s [Port Number]\n", argv[0]);
        return -1;
    }

    printf("Server start...\n");

    epfd = epoll_create(1);
    if(epfd == -1) errProc("epoll_create");


    listenSd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(listenSd == -1 ) errProc("socket");

    memset(&srvAddr, 0, sizeof(srvAddr));
    srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    srvAddr.sin_family = AF_INET;
    srvAddr.sin_port = htons(atoi(argv[1]));
    if(bind(listenSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr)) == -1)
        errProc("bind");

    //non-blocking 소켓으로 지정
    makeNbSocket(listenSd);
    if(listen(listenSd, 5) < 0) errProc("listen");

    //epoll mode = edge trigger
    ev.events = EPOLLIN | EPOLLET;
    ev.data.fd = listenSd;
    if(epoll_ctl(epfd, EPOLL_CTL_ADD, listenSd, &ev) == -1)
        errProc("epoll_ctl");

    clntAddrLen = sizeof(clntAddr);
    while(1)
    {
        printf("Monitoring ... \n");
        ready = epoll_wait(epfd, events, MAX_EVENTS, -1);
        if(ready == -1)
        {
            if(errno == EINTR)    continue;
            else errProc("epoll_wait");
        }

        for(i=0; i<ready; i++)
        {
            if(events[i].data.fd == listenSd) // accept a client
            {
                while(1)
                {
                    connectSd = accept(listenSd, (struct sockaddr *) &clntAddr, &cl
                    if(connectSd == -1)
                    {
                        //signal 왔는데 data 바로 안 넘어왔을 때 에러내기
                        if((errno == EAGAIN) || (errno == EWOULDBLOCK))
                            break;
                        else {
                            fprintf(stderr,"Accept Error");
                            continue;
                        }
                    }
                    fprintf(stderr,"A client is connected...\n");

                    makeNbSocket(connectSd);
                    ev.data.fd = connectSd;
                    //epoll mode = edge trigger
                    ev.events = EPOLLIN | EPOLLET;
                    if(epoll_ctl(epfd, EPOLL_CTL_ADD, connectSd, &ev) == -1)
                        errProc("epoll_ctl");

                }
                //fprintf(stderr,"There is no client in the queue...\n");
                continue;
            }
            else //IO
            {
                completed = 0;
                //edge trigger -> 읽기 -> 메세지 출력 -> 다시 읽기 -> 읽을 거 없으면 d
                while(1)
                {
                    readfd = events[i].data.fd;
                    readLen = read(readfd, rBuff, sizeof(rBuff));

                    //읽을 거 없으면
                    if(readLen == -1)
                    {
                        if(errno != EAGAIN)
                        {
                            fprintf(stderr,"Read Error \n");
                            completed = 1;
                        }
                        //break하기
                        printf("data unavailable\n");
                        break;

                    }
                    //client 제거
                    if(readLen == 0)
                    {
                        printf("A client is disconnected...\n");
                        if(epoll_ctl(epfd, EPOLL_CTL_DEL, readfd, &ev) == -1)
                            errProc("epoll_ctl");
                        close(events[i].data.fd);

                    }
                    //terminal 출력, fd=1
                    write(1, rBuff, readLen);
                    printf("\n");
                    //break;
                }

            }
        }
    }
    close(listenSd);
    close(epfd);
    return 0;
}

void errProc(const char* str)
{
    fprintf(stderr,"%s: %s", str, strerror(errno));
    exit(errno);
}

int makeNbSocket(int socket)
{
  int res;

  res = fcntl(socket, F_GETFL, 0);
  if (res == -1) errProc("fcntl");
  res |= O_NONBLOCK;
  res = fcntl(socket, F_SETFL, res);
  if (res == -1) errProc("fcntl");

  return 0;

}

```



```
data unavailable
Monitoring ...
12345678911234567891
1234567891

data unavailable
Monitoring ...
```

buff 크기 20으로 설정했을 때, 20 읽고 나머지 10 다시 읽음

select	epoll
관심 fd 전부 관찰	event 일어난 fd만 관찰
윈도우, 리눅스에서 전부 동작	리눅스에서만 동작

Raw Socket

socket(AF_INET, SOCK type, 0)
SOCK type

- SOCK_STREAM: TCP 소켓
- SOCK_DGRAM: UDP 소켓
- **SOCK_RAW**: 사용자 정의 4계층 통신

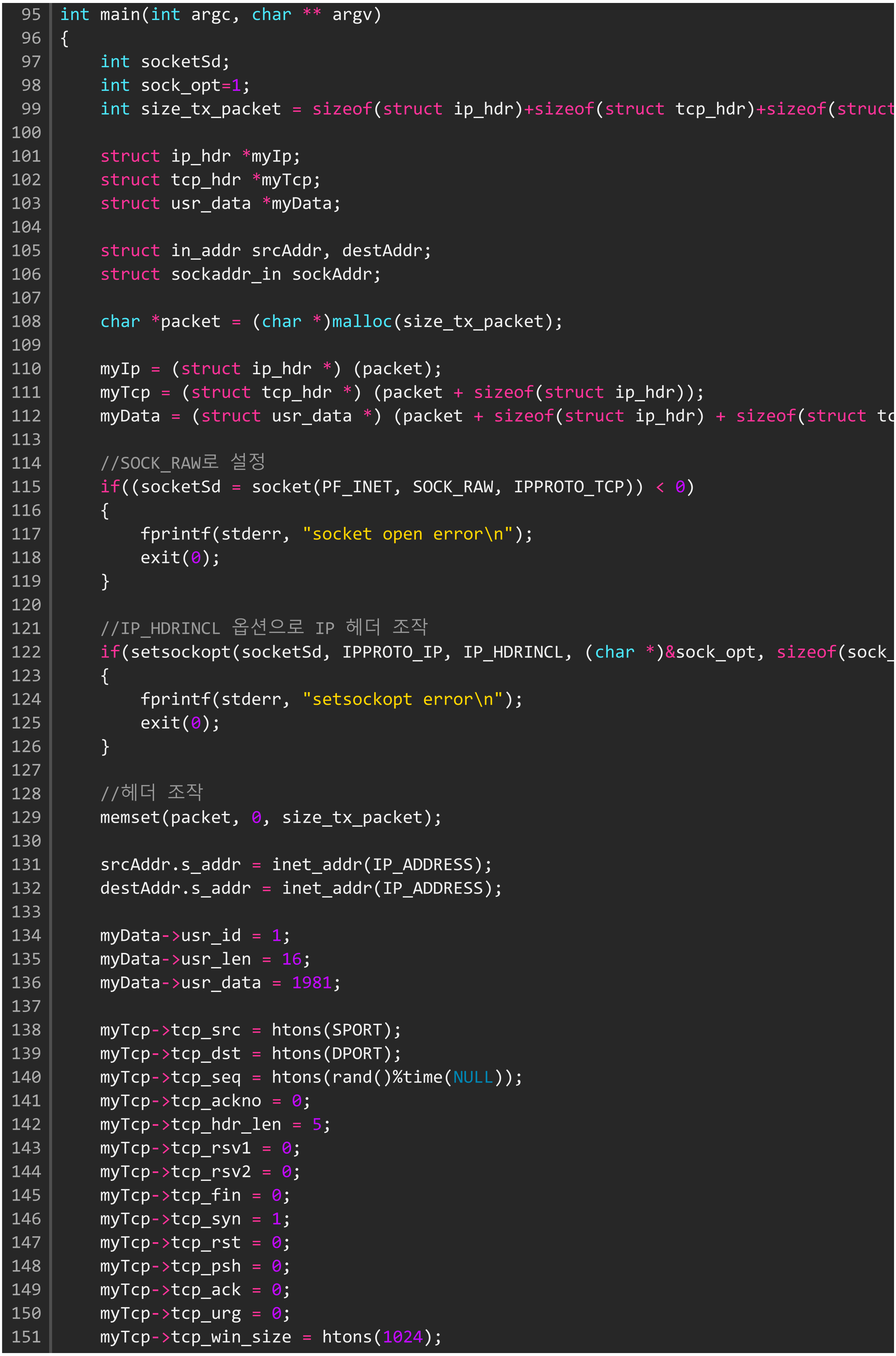
Raw Socket + IP_HDRINCL

IP 헤더정보를 사용자가 수정할 수 있다.

구조체에서 ':'를 이용하여 IP 헤더와 TCP 헤더까지 정의할 수 있다.

ip.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <sys/socket.h>
#include <sys/types.h> //uintx_t

#define SPORT 90
#define DPORT 90
#define IP_ADDRESS "127.0.0.1"


struct ip_hdr
{
#if __BYTE_ORDER__ == __LITTLE_ENDIAN
    uint8_t ip_hdr_len:4; //(IP Header Length)
    uint8_t ip_version:4; //(IP Version)
#else
    uint8_t ip_version:4;
    uint8_t ip_hdr_len:4;
#endif
    uint8_t ip_tos; // (TOS Field)
    uint16_t ip_len; // (Payload Field= header + SDU)

    uint16_t ip_id; // (Identification Field)
    uint16_t ip_off; // (Flag(DF,MF) + Fragment offset Field)

    uint8_t ip_ttl; // (Time to Live)
    uint8_t ip_proto; // (Upper Layer Protocol)
    uint16_t ip_check; // (IP Checksum)

    uint32_t ip_src; //(Source Address)

    uint32_t ip_dst; //(Destination Address)
};

struct tcp_hdr
{
    uint16_t tcp_src; //(Source Port)
    uint16_t tcp_dst; //(Destination Port)

    uint32_t tcp_seq; //(Sequence Number Field)

    uint32_t tcp_ackno; //(Acknowledgment Number Field)

#if __BYTE_ORDER__ == __LITTLE_ENDIAN
    uint8_t tcp_rsv1:4; //(Reserved 4bits)
    uint8_t tcp_hdr_len:4;//(Header Length)
    uint8_t tcp_fin:1;//(6bit flags = U/A/P/R/S/F)
    uint8_t tcp_syn:1;
    uint8_t tcp_rst:1;
    uint8_t tcp_psh:1;
    uint8_t tcp_ack:1;
    uint8_t tcp_urg:1;
    uint8_t tcp_rsv2:2;//(Reserved 2bits)
#else
    uint8_t tcp_hdr_len:4;
    uint8_t tcp_rsv1:4; //(Reserved 4bit)
    uint8_t tcp_rsv2:2; //(Reserved 4bit)
    uint8_t tcp_urg:1; //(6bit flags = U/A/P/R/S/F)
    uint8_t tcp_ack:1;
    uint8_t tcp_psh:1;
    uint8_t tcp_rst:1;
    uint8_t tcp_syn:1;
    uint8_t tcp_fin:1;
#endif
    uint16_t tcp_win_size; //(Window Size)

    uint16_t tcp_check; //(TCP Checksum)
    uint16_t tcp_urg_ptr; //(Urgent Pointer)
};

struct udp_hdr
{
    uint16_t udp_src; //(Source Port)
    uint16_t udp_dst; //(Destination Port)

    uint16_t udp_len; //(Payload Length: header + SDU)
    uint16_t udp_check; //(UDP Checksum)
};

struct usr_data
{
    uint16_t usr_id;
    uint16_t usr_len;

    uint32_t usr_data;
};



```



```c
int main(int argc, char ** argv)
{
    int socketSd;
    int sock_opt=1;
    int size_tx_packet = sizeof(struct ip_hdr)+sizeof(struct tcp_hdr)+sizeof(struct

    struct ip_hdr *myIp;
    struct tcp_hdr *myTcp;
    struct usr_data *myData;

    struct in_addr srcAddr, destAddr;
    struct sockaddr_in sockAddr;

    char *packet = (char *)malloc(size_tx_packet);

    myIp = (struct ip_hdr *) (packet);
    myTcp = (struct tcp_hdr *) (packet + sizeof(struct ip_hdr));
    myData = (struct usr_data *) (packet + sizeof(struct ip_hdr) + sizeof(struct tc

    //SOCK_RAW로 설정
    if((socketSd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) < 0)
    {
        fprintf(stderr, "socket open error\n");
        exit(0);
    }

    //IP_HDRINCL 옵션으로 IP 헤더 조작
    if(setsockopt(socketSd, IPPROTO_IP, IP_HDRINCL, (char *)&sock_opt, sizeof(sock_
    {
        fprintf(stderr, "setsockopt error\n");
        exit(0);
    }

    //헤더 조작
    memset(packet, 0, size_tx_packet);

    srcAddr.s_addr = inet_addr(IP_ADDRESS);
    destAddr.s_addr = inet_addr(IP_ADDRESS);

    myData->usr_id = 1;
    myData->usr_len = 16;
    myData->usr_data = 1981;

    myTcp->tcp_src = htons(SPORT);
    myTcp->tcp_dst = htons(DPORT);
    myTcp->tcp_seq = htons(rand()%time(NULL));
    myTcp->tcp_ackno = 0;
    myTcp->tcp_hdr_len = 5;
    myTcp->tcp_rsv1 = 0;
    myTcp->tcp_rsv2 = 0;
    myTcp->tcp_fin = 0;
    myTcp->tcp_syn = 1;
    myTcp->tcp_rst = 0;
    myTcp->tcp_psh = 0;
    myTcp->tcp_ack = 0;
    myTcp->tcp_urg = 0;
    myTcp->tcp_win_size = htons(1024);
    myTcp->tcp_check = 0;
    myTcp->tcp_urg_ptr = 0;

    myIp->ip_hdr_len = 5;
    myIp->ip_version = 4;
    myIp->ip_tos = 0;
    myIp->ip_len = htons(size_tx_packet);
    myIp->ip_id = htons(2);
    myIp->ip_off = 0;
    myIp->ip_ttl = IPDEFTTL;
    myIp->ip_proto = IPPROTO_TCP;
    myIp->ip_src = srcAddr.s_addr;
    myIp->ip_dst = destAddr.s_addr;
    //checksum까지 조작
    myIp->ip_check = 0x1111;


    sockAddr.sin_family = PF_INET;
    sockAddr.sin_addr = destAddr;
    sockAddr.sin_port = htons(DPORT);

    if( sendto(socketSd, packet, size_tx_packet, 0x0, (struct sockaddr *)&sockAddr,
    {
        fprintf(stderr,"send error \n");
        exit(1);
    }

    close(socketSd);

    return 0;
}


```

checksum 임의로 지정할 수 있지만 wireshark에서는 다른 값으로 나옴.
잘못된 checksum이기 때문에 os가 알아서 계산해준다.

Checksum

- IP헤더 Checksum IP헤더 전체 영역
- TCP 헤더 Checksum IP헤더에서 변하지 않는 값 + TCP 헤더 + Data를 통해 Checksum 생성

메세지에 대한 무결성을 제공하는 것이지, 공격여부는 탐지하지 못한다.

Sniffer

- Promiscuous 모드 NIC(Network Interface Card)가 MAC주소 외의 데이터도 상위 계층으로 전달함

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

#include <netinet/in.h>
#include <netinet/ip.h>
```

```
#include <netinet/tcp.h>
#include <sys/socket.h>
#include <sys/types.h> //uintx_t
#include <errno.h>

struct ip_hdr
{
#if __BYTE_ORDER__ == __LITTLE_ENDIAN
    uint8_t ip_hdr_len:4; //(IP Header Length)
    uint8_t ip_version:4; //(IP Version)
#else
    uint8_t ip_version:4;
    uint8_t ip_hdr_len:4;
#endif
    uint8_t ip_tos; // (TOS Field)
    uint16_t ip_len; // (Payload Field= header + SDU)

    uint16_t ip_id; // (Identification Field)
    uint16_t ip_off; // (Flag(DF,MF) + Fragment offset Field)

    uint8_t ip_ttl; // (Time to Live)
    uint8_t ip_proto; // (Upper Layer Protocol)
    uint16_t ip_check; // (IP Checksum)

    uint32_t ip_src; //(Source Address)

    uint32_t ip_dst; //(Destination Address)
};

struct tcp_hdr
{
    uint16_t tcp_src; //(Source Port)
    uint16_t tcp_dst; //(Destination Port)

    uint32_t tcp_seq; //(Sequence Number Field)

    uint32_t tcp_ackno; //(Acknowledgment Number Field)

#if __BYTE_ORDER__ == __LITTLE_ENDIAN
    uint8_t tcp_rsv1:4; //(Reserved 4bits)
    uint8_t tcp_hdr_len:4;//(Header Length)
    uint8_t tcp_fin:1;//(6bit flags = U/A/P/R/S/F)
    uint8_t tcp_syn:1;
    uint8_t tcp_rst:1;
    uint8_t tcp_psh:1;
    uint8_t tcp_ack:1;
    uint8_t tcp_urg:1;
    uint8_t tcp_rsv2:2;//(Reserved 2bits)
#else
    uint8_t tcp_hdr_len:4;
    uint8_t tcp_rsv1:4; //(Reserved 4bit)
    uint8_t tcp_rsv2:2; //(Reserved 4bit)
    uint8_t tcp_urg:1; //(6bit flags = U/A/P/R/S/F)
    uint8_t tcp_ack:1;
    uint8_t tcp_psh:1;
    uint8_t tcp_rst:1;
    uint8_t tcp_syn:1;
    uint8_t tcp_fin:1;
#endif
    uint16_t tcp_win_size; //(Window Size)

    uint16_t tcp_check; //(TCP Checksum)
    uint16_t tcp_urg_ptr; //(Urgent Pointer)
};

struct udp_hdr
{
    uint16_t udp_src; //(Source Port)
    uint16_t udp_dst; //(Destination Port)

    uint16_t udp_len; //(Payload Length: header + SDU)
    uint16_t udp_check; //(UDP Checksum)
};

struct usr_data
{
    uint16_t usr_id;
    uint16_t usr_len;

    uint32_t usr_data;
};

struct pseudo_hdr
{
    uint32_t src;
    uint32_t dst;
    uint8_t zeros;
    uint8_t proto;
    uint16_t len;
};

void errProc(const char*);
uint16_t checksum(const void *ptr, int len);
void parseTcpHeader(struct tcp_hdr * myHdr);

int main(int argc, char ** argv)
{
    int socketSd;
    int fromAddrLen;
    char rBuff[BUFSIZ];

    struct tcp_hdr *myTcp;
    struct ip_hdr *myIp;

    struct sockaddr_in fromAddr;

    if((socketSd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) < 0)
        errProc("socket");

    while(1)
    {
        if(recvfrom(socketSd, rBuff, BUFSIZ-1, 0x0, (struct sockaddr *)&fromAddr, &
                errProc("Recv Error");

        myIp = (struct ip_hdr *) rBuff;
        myTcp = (struct tcp_hdr *) (rBuff + sizeof(struct ip_hdr));
        parseTcpHeader(myTcp);
    }

    close(socketSd);

    return 0;
}

void parseTcpHeader(struct tcp_hdr * myHdr)
{
    printf("===========Recv TCP Segment ===========\n");
    printf("Source Port: %d\n",ntohs(myHdr->tcp_src));
    printf("Destination Port: %d\n",ntohs(myHdr->tcp_dst));
    printf("Sequence No.t: %d\n",ntohs(myHdr->tcp_seq));
    printf("ACK No.: %d\n",ntohs(myHdr->tcp_ackno));
    printf("Flags: %c%c%c%c%c%c\n",(myHdr->tcp_fin?'F':'X'),
        (myHdr->tcp_syn?'S':'X'),(myHdr->tcp_rst?'R':'X'),(myHdr->tcp_psh?'P':'X'),
        (myHdr->tcp_ack?'A':'X'),(myHdr->tcp_urg?'U':'X'));
    printf("Checksum: %X\n",ntohs(myHdr->tcp_check));
}


void errProc(const char* str)
{
    fprintf(stderr,"%s: %s \n", str, strerror(errno));
    exit(1);
}


uint16_t checksum(const void *ptr, int len)
{
    int sum = 0;
    uint16_t answer = 0;
    uint16_t *w = (uint16_t *) ptr;
    int nleft = len;

    while(nleft > 1){
        sum += *w++;
        nleft -= 2;
    }

    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    answer = ~sum;
    return(answer);
}
```