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

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

```c
  3  #include <sys/socket.h>
  4  #include <sys/types.h>
  5  #include <netinet/in.h>
  6  #include <string.h>
  7  #include <unistd.h>
  8  #include <sys/time.h>
  9  #include <sys/epoll.h>
 10  #include <errno.h>
 11
 12  #define MAX_EVENTS    10
 13
 14  void errProc(const char*);
 15
 16  int main(int argc, char** argv)
 17  {
 18      int listenSd, connectSd;
 19      struct sockaddr_in srvAddr, clntAddr;
 20      int clntAddrLen, readLen;
 21      char rBuff[BUFSIZ];
 22      int i;
 23
 24      int epfd, ready, readfd;
 25      struct epoll_event ev;
 26      struct epoll_event events[MAX_EVENTS];
 27
 28      if(argc != 2)
 29      {
 30          printf("Usage: %s [Port Number]\n", argv[0]);
 31          return -1;
 32      }
 33
 34      printf("Server start...\n");
 35
 36      //epoll 생성
 37      epfd = epoll_create(1);
 38      if(epfd == -1) errProc("epoll_create");
 39
 40      //듣기 소켓 생성
 41      listenSd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
 42      if(listenSd == -1 ) errProc("socket");
 43
 44      memset(&srvAddr, 0, sizeof(srvAddr));
 45      srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
 46      srvAddr.sin_family = AF_INET;
 47      srvAddr.sin_port = htons(atoi(argv[1]));
 48
 49      //port 할당
 50      if(bind(listenSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr)) == -1)
 51          errProc("bind");
 52
 53      //듣기
 54      if(listen(listenSd, 5) < 0) errProc("listen");
 55
 56
 57      ev.events = EPOLLIN; //읽기 동작
 58      ev.data.fd = listenSd; //듣기 소켓
 59
```

```
60        //객체 제어
61        //EPOLL_CTL_ADDL: 관심 리스트(epfd)에 listenSd 넣기
62        if(epoll_ctl(epfd, EPOLL_CTL_ADD, listenSd, &ev) == -1)
63            errProc("epoll_ctl");
64
65        clntAddrLen = sizeof(clntAddr);
66
67        while(1) {
68            printf("Monitoring ... \n");
69
70            //epfd에서 events가 올때까지 -1 무한대기
71            ready = epoll_wait(epfd, events, MAX_EVENTS, -1);
72            printf("ready: %d\n", ready);
73            //ready: 이벤트 발생한 fd 개수
74            if(ready == -1) {
75                if(errno == EINTR) continue;
76                else errProc("epoll_wait");
77            }
78
79            //이벤트 발생시
80            //events 배열에는 ready된 만큼의 fd개수가 순서대로 저장되어있음
81            //따라서 for(0~ready-1)로 접근하는 것.
82            for(i=0; i<ready; i++)    {
83                //listen socket이면 새로운 client 연결
84                printf("fd: %d\n",events[i].data.fd);
85                if(events[i].data.fd == listenSd) {
86
87                    //client 연결
88                    connectSd = accept(listenSd, (struct sockaddr *) &clntAddr, &clntAd
89                    if(connectSd == -1)    {
90                        fprintf(stderr,"Accept Error");
91                        continue;
92                    }
93                    fprintf(stderr,"A client is connected...\n");
94
95                    ev.data.fd = connectSd;
96                    //연결한 client를 관심있게 설정
97                    if(epoll_ctl(epfd, EPOLL_CTL_ADD, connectSd, &ev) == -1)
98                        errProc("epoll_ctl");
99                }
100                //기존 client의 요청
101                else {//IO
102                    readfd = events[i].data.fd;
103                    //읽기
104                    readLen = read(readfd, rBuff, sizeof(rBuff)-1);
105                    if(readLen == 0)
106                    {
107                        fprintf(stderr,"A client is disconnected...\n");
108                        //관심 list에서 제거
109                        if(epoll_ctl(epfd, EPOLL_CTL_DEL, readfd, &ev) == -1)
110                            errProc("epoll_ctl");
111                        close(readfd);
112                        continue;
113                    }
114                    rBuff[readLen] = '\0';
115                    printf("Client(%d): %s\n", events[i].data.fd,rBuff);
116                    write(events[i].data.fd, rBuff, strlen(rBuff));
```

```
117                }
118            }
119        }
120        close(listenSd);
121        close(epfd);
122        return 0;
123  }
124
125  void errProc(const char * str)
126  {
127        fprintf(stderr,"%s: %s", str, strerror(errno));
128        exit(1);
129  }
```
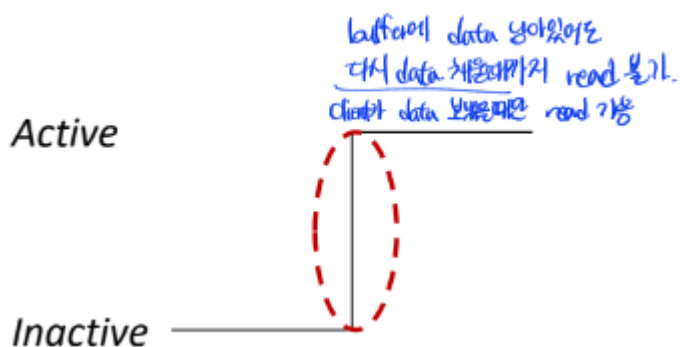
epoll을 이용해서 thread, multiplex 없이 다중통신을 구현할 수 있다. 하지만 epoll은 리눅스 only

epoll mode

- Edge-triggered, 이벤트 일어나면 알리기

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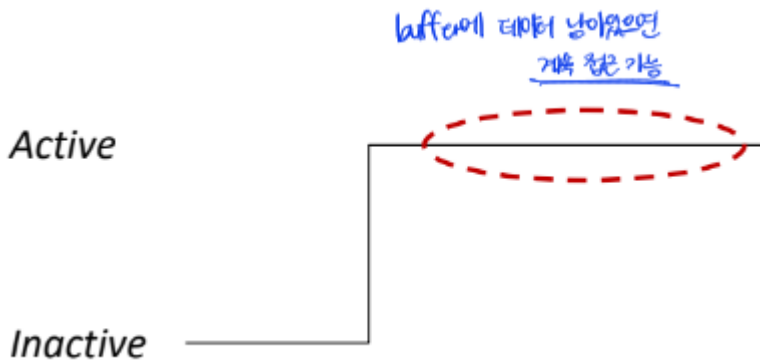이벤트가 발생했을때 알리고, 그것을 처리하였으므로 다시 이벤트 발생까지 block되는 형식

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.
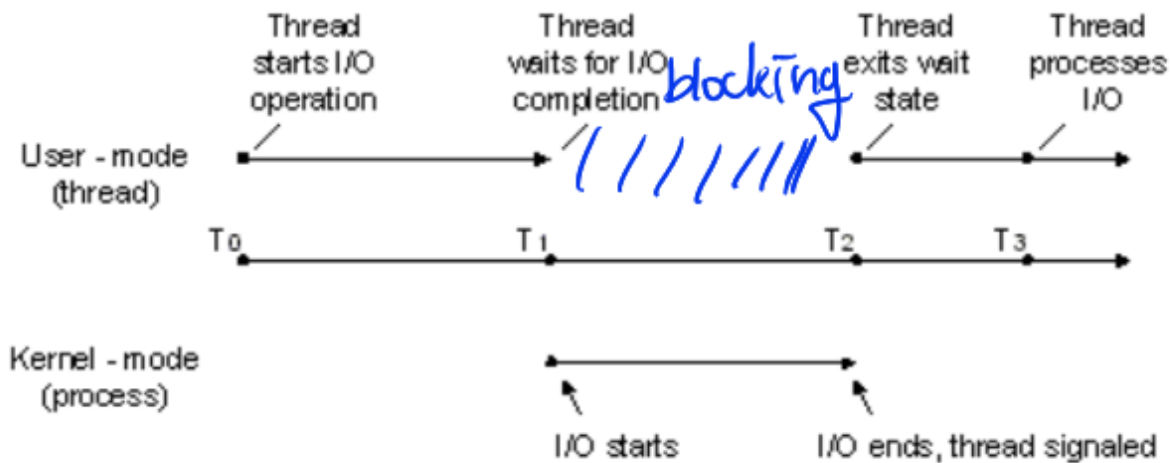**non-blocking 소켓일때, edge의 data 왔을때만 읽는 게 중요**
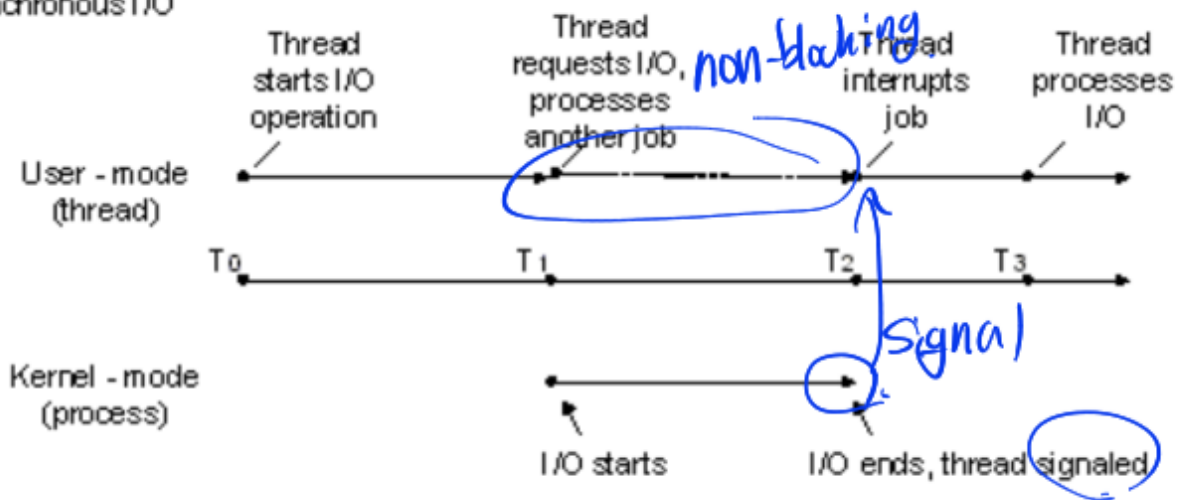
- Level-triggered, 무언가가 이용 가능하면 알리기

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, kernel으 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)
{
```

```
19      int listenSd, connectSd;
20      struct sockaddr_in srvAddr, clntAddr;
21      int clntAddrLen, readLen;
22      char rBuff[20];
23      int i, completed = 0;
24
25      int epfd, ready, readfd;
26      struct epoll_event ev;
27      struct epoll_event events[MAX_EVENTS];
28
29      if(argc != 2)
30      {
31          printf("Usage: %s [Port Number]\n", argv[0]);
32          return -1;
33      }
34
35      printf("Server start...\n");
36
37      epfd = epoll_create(1);
38      if(epfd == -1) errProc("epoll_create");
39
40
41      listenSd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
42      if(listenSd == -1 ) errProc("socket");
43
44      memset(&srvAddr, 0, sizeof(srvAddr));
45      srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
46      srvAddr.sin_family = AF_INET;
47      srvAddr.sin_port = htons(atoi(argv[1]));
48      if(bind(listenSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr)) == -1)
49          errProc("bind");
50
51      //non-blocking 소켓으로 지정
52      makeNbSocket(listenSd);
53      if(listen(listenSd, 5) < 0) errProc("listen");
54
55      //epoll mode = edge trigger
56      ev.events = EPOLLIN | EPOLLET;
57      ev.data.fd = listenSd;
58      if(epoll_ctl(epfd, EPOLL_CTL_ADD, listenSd, &ev) == -1)
59          errProc("epoll_ctl");
60
61      clntAddrLen = sizeof(clntAddr);
62      while(1)
63      {
64          printf("Monitoring ... \n");
65          ready = epoll_wait(epfd, events, MAX_EVENTS, -1);
66          if(ready == -1)
67          {
68              if(errno == EINTR)      continue;
69              else errProc("epoll_wait");
70          }
71
72          for(i=0; i<ready; i++)
73          {
74              if(events[i].data.fd == listenSd) // accept a client
75              {
```

```
 76                    while(1)
 77                    {
 78                        connectSd = accept(listenSd, (struct sockaddr *) &clntAddr, &cl
 79                        if(connectSd == -1)
 80                        {
 81                            //signal 왔는데 data 바로 안 넘어왔을 때 에러내기
 82                            if((errno == EAGAIN) || (errno == EWOULDBLOCK))
 83                                break;
 84                            else {
 85                                fprintf(stderr,"Accept Error");
 86                                continue;
 87                            }
 88                        }
 89                        fprintf(stderr,"A client is connected...\n");
 90
 91                        makeNbSocket(connectSd);
 92                        ev.data.fd = connectSd;
 93                        //epoll mode = edge trigger
 94                        ev.events = EPOLLIN | EPOLLET;
 95                        if(epoll_ctl(epfd, EPOLL_CTL_ADD, connectSd, &ev) == -1)
 96                            errProc("epoll_ctl");
 97
 98                    }
 99                    //fprintf(stderr,"There is no client in the queue...\n");
100                    continue;
101                }
102                else //IO
103                {
104                    completed = 0;
105                    //edge trigger -> 읽기 -> 메세지 출력 -> 다시 읽기 -> 읽을 거 없으면 d
106                    while(1)
107                    {
108                        readfd = events[i].data.fd;
109                        readLen = read(readfd, rBuff, sizeof(rBuff));
110
111                        //읽을 거 없으면
112                        if(readLen == -1)
113                        {
114                            if(errno != EAGAIN)
115                            {
116                                fprintf(stderr,"Read Error \n");
117                                completed = 1;
118                            }
119                            //break하기
120                            printf("data unavailable\n");
121                            break;
122
123                        }
124                        //client 제거
125                        if(readLen == 0)
126                        {
127                            printf("A client is disconnected...\n");
128                            if(epoll_ctl(epfd, EPOLL_CTL_DEL, readfd, &ev) == -1)
129                                errProc("epoll_ctl");
130                            close(events[i].data.fd);
131
132                        }
```

```
133                            //terminal 출력, fd=1
134                            write(1, rBuff, readLen);
135                            printf("\n");
136                            //break;
137                        }
138
139                    }
140                }
141            }
142        close(listenSd);
143        close(epfd);
144        return 0;
145    }
146
147    void errProc(const char* str)
148    {
149        fprintf(stderr,"%s: %s", str, strerror(errno));
150        exit(errno);
151    }
152
153    int makeNbSocket(int socket)
154    {
155      int res;
156
157      res = fcntl(socket, F_GETFL, 0);
158      if (res == -1) errProc("fcntl");
159      res |= O_NONBLOCK;
160      res = fcntl(socket, F_SETFL, res);
161      if (res == -1) errProc("fcntl");
162
163      return 0;
164
165    }
166
```

```
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)
```

```c
38  };
39
40  struct tcp_hdr
41  {
42      uint16_t tcp_src; //(Source Port)
43      uint16_t tcp_dst; //(Destination Port)
44
45      uint32_t tcp_seq; //(Sequence Number Field)
46
47      uint32_t tcp_ackno; //(Acknowledgment Number Field)
48
49  #if __BYTE_ORDER__ == __LITTLE_ENDIAN
50      uint8_t tcp_rsv1:4; //(Reserved 4bits)
51      uint8_t tcp_hdr_len:4;//(Header Length)
52      uint8_t tcp_fin:1;//(6bit flags = U/A/P/R/S/F)
53      uint8_t tcp_syn:1;
54      uint8_t tcp_rst:1;
55      uint8_t tcp_psh:1;
56      uint8_t tcp_ack:1;
57      uint8_t tcp_urg:1;
58      uint8_t tcp_rsv2:2;//(Reserved 2bits)
59  #else
60      uint8_t tcp_hdr_len:4;
61      uint8_t tcp_rsv1:4; //(Reserved 4bit)
62      uint8_t tcp_rsv2:2; //(Reserved 4bit)
63      uint8_t tcp_urg:1; //(6bit flags = U/A/P/R/S/F)
64      uint8_t tcp_ack:1;
65      uint8_t tcp_psh:1;
66      uint8_t tcp_rst:1;
67      uint8_t tcp_syn:1;
68      uint8_t tcp_fin:1;
69  #endif
70      uint16_t tcp_win_size; //(Window Size)
71
72      uint16_t tcp_check; //(TCP Checksum)
73      uint16_t tcp_urg_ptr; //(Urgent Pointer)
74  };
75
76  struct udp_hdr
77  {
78      uint16_t udp_src; //(Source Port)
79      uint16_t udp_dst; //(Destination Port)
80
81      uint16_t udp_len; //(Payload Length: header + SDU)
82      uint16_t udp_check; //(UDP Checksum)
83  };
84
85  struct usr_data
86  {
87      uint16_t usr_id;
88      uint16_t usr_len;
89
90      uint32_t usr_data;
91  };
92
93
94
```

```c
 95  int main(int argc, char ** argv)
 96  {
 97      int socketSd;
 98      int sock_opt=1;
 99      int size_tx_packet = sizeof(struct ip_hdr)+sizeof(struct tcp_hdr)+sizeof(struct
100
101      struct ip_hdr *myIp;
102      struct tcp_hdr *myTcp;
103      struct usr_data *myData;
104
105      struct in_addr srcAddr, destAddr;
106      struct sockaddr_in sockAddr;
107
108      char *packet = (char *)malloc(size_tx_packet);
109
110      myIp = (struct ip_hdr *) (packet);
111      myTcp = (struct tcp_hdr *) (packet + sizeof(struct ip_hdr));
112      myData = (struct usr_data *) (packet + sizeof(struct ip_hdr) + sizeof(struct tc
113
114      //SOCK_RAW로 설정
115      if((socketSd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) < 0)
116      {
117          fprintf(stderr, "socket open error\n");
118          exit(0);
119      }
120
121      //IP_HDRINCL 옵션으로 IP 헤더 조작
122      if(setsockopt(socketSd, IPPROTO_IP, IP_HDRINCL, (char *)&sock_opt, sizeof(sock_
123      {
124          fprintf(stderr, "setsockopt error\n");
125          exit(0);
126      }
127
128      //헤더 조작
129      memset(packet, 0, size_tx_packet);
130
131      srcAddr.s_addr = inet_addr(IP_ADDRESS);
132      destAddr.s_addr = inet_addr(IP_ADDRESS);
133
134      myData->usr_id = 1;
135      myData->usr_len = 16;
136      myData->usr_data = 1981;
137
138      myTcp->tcp_src = htons(SPORT);
139      myTcp->tcp_dst = htons(DPORT);
140      myTcp->tcp_seq = htons(rand()%time(NULL));
141      myTcp->tcp_ackno = 0;
142      myTcp->tcp_hdr_len = 5;
143      myTcp->tcp_rsv1 = 0;
144      myTcp->tcp_rsv2 = 0;
145      myTcp->tcp_fin = 0;
146      myTcp->tcp_syn = 1;
147      myTcp->tcp_rst = 0;
148      myTcp->tcp_psh = 0;
149      myTcp->tcp_ack = 0;
150      myTcp->tcp_urg = 0;
151      myTcp->tcp_win_size = htons(1024);
```

```
152        myTcp->tcp_check = 0;
153        myTcp->tcp_urg_ptr = 0;
154
155        myIp->ip_hdr_len = 5;
156        myIp->ip_version = 4;
157        myIp->ip_tos = 0;
158        myIp->ip_len = htons(size_tx_packet);
159        myIp->ip_id = htons(2);
160        myIp->ip_off = 0;
161        myIp->ip_ttl = IPDEFTTL;
162        myIp->ip_proto = IPPROTO_TCP;
163        myIp->ip_src = srcAddr.s_addr;
164        myIp->ip_dst = destAddr.s_addr;
165        //checksum까지 조작
166        myIp->ip_check = 0x1111;
167
168
169        sockAddr.sin_family = PF_INET;
170        sockAddr.sin_addr = destAddr;
171        sockAddr.sin_port = htons(DPORT);
172
173        if( sendto(socketSd, packet, size_tx_packet, 0x0, (struct sockaddr *)&sockAddr,
174        {
175            fprintf(stderr,"send error \n");
176            exit(1);
177        }
178
179        close(socketSd);
180
181        return 0;
182 }
183
184
```

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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include <netinet/in.h>
6  #include <netinet/ip.h>
```

```c
 7  #include <netinet/tcp.h>
 8  #include <sys/socket.h>
 9  #include <sys/types.h> //uintx_t
10  #include <errno.h>
11
12  struct ip_hdr
13  {
14  #if __BYTE_ORDER__ == __LITTLE_ENDIAN
15      uint8_t ip_hdr_len:4; //(IP Header Length)
16      uint8_t ip_version:4; //(IP Version)
17  #else
18      uint8_t ip_version:4;
19      uint8_t ip_hdr_len:4;
20  #endif
21      uint8_t ip_tos; // (TOS Field)
22      uint16_t ip_len; // (Payload Field= header + SDU)
23
24      uint16_t ip_id; // (Identification Field)
25      uint16_t ip_off; // (Flag(DF,MF) + Fragment offset Field)
26
27      uint8_t ip_ttl; // (Time to Live)
28      uint8_t ip_proto; // (Upper Layer Protocol)
29      uint16_t ip_check; // (IP Checksum)
30
31      uint32_t ip_src; //(Source Address)
32
33      uint32_t ip_dst; //(Destination Address)
34  };
35
36  struct tcp_hdr
37  {
38      uint16_t tcp_src; //(Source Port)
39      uint16_t tcp_dst; //(Destination Port)
40
41      uint32_t tcp_seq; //(Sequence Number Field)
42
43      uint32_t tcp_ackno; //(Acknowledgment Number Field)
44
45  #if __BYTE_ORDER__ == __LITTLE_ENDIAN
46      uint8_t tcp_rsv1:4; //(Reserved 4bits)
47      uint8_t tcp_hdr_len:4;//(Header Length)
48      uint8_t tcp_fin:1;//(6bit flags = U/A/P/R/S/F)
49      uint8_t tcp_syn:1;
50      uint8_t tcp_rst:1;
51      uint8_t tcp_psh:1;
52      uint8_t tcp_ack:1;
53      uint8_t tcp_urg:1;
54      uint8_t tcp_rsv2:2;//(Reserved 2bits)
55  #else
56      uint8_t tcp_hdr_len:4;
57      uint8_t tcp_rsv1:4; //(Reserved 4bit)
58      uint8_t tcp_rsv2:2; //(Reserved 4bit)
59      uint8_t tcp_urg:1; //(6bit flags = U/A/P/R/S/F)
60      uint8_t tcp_ack:1;
61      uint8_t tcp_psh:1;
62      uint8_t tcp_rst:1;
63      uint8_t tcp_syn:1;
```

```
64         uint8_t tcp_fin:1;
65  #endif
66         uint16_t tcp_win_size; //(Window Size)
67
68         uint16_t tcp_check; //(TCP Checksum)
69         uint16_t tcp_urg_ptr; //(Urgent Pointer)
70  };
71
72  struct udp_hdr
73  {
74         uint16_t udp_src; //(Source Port)
75         uint16_t udp_dst; //(Destination Port)
76
77         uint16_t udp_len; //(Payload Length: header + SDU)
78         uint16_t udp_check; //(UDP Checksum)
79  };
80
81  struct usr_data
82  {
83         uint16_t usr_id;
84         uint16_t usr_len;
85
86         uint32_t usr_data;
87  };
88
89  struct pseudo_hdr
90  {
91         uint32_t src;
92         uint32_t dst;
93         uint8_t zeros;
94         uint8_t proto;
95         uint16_t len;
96  };
97
98  void errProc(const char*);
99  uint16_t checksum(const void *ptr, int len);
100 void parseTcpHeader(struct tcp_hdr * myHdr);
101
102 int main(int argc, char ** argv)
103 {
104        int socketSd;
105        int fromAddrLen;
106        char rBuff[BUFSIZ];
107
108        struct tcp_hdr *myTcp;
109        struct ip_hdr *myIp;
110
111        struct sockaddr_in fromAddr;
112
113        if((socketSd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) < 0)
114            errProc("socket");
115
116        while(1)
117        {
118            if(recvfrom(socketSd, rBuff, BUFSIZ-1, 0x0, (struct sockaddr *)&fromAddr, &
119                    errProc("Recv Error");
120
```

```
121            myIp = (struct ip_hdr *) rBuff;
122            myTcp = (struct tcp_hdr *) (rBuff + sizeof(struct ip_hdr));
123            parseTcpHeader(myTcp);
124        }
125
126        close(socketSd);
127
128        return 0;
129    }
130
131    void parseTcpHeader(struct tcp_hdr * myHdr)
132    {
133        printf("============Recv TCP Segment ===========\n");
134        printf("Source Port: %d\n",ntohs(myHdr->tcp_src));
135        printf("Destination Port: %d\n",ntohs(myHdr->tcp_dst));
136        printf("Sequence No.t: %d\n",ntohs(myHdr->tcp_seq));
137        printf("ACK No.: %d\n",ntohs(myHdr->tcp_ackno));
138        printf("Flags: %c%c%c%c%c%c\n",(myHdr->tcp_fin?'F':'X'),
139            (myHdr->tcp_syn?'S':'X'),(myHdr->tcp_rst?'R':'X'),(myHdr->tcp_psh?'P':'X'),
140            (myHdr->tcp_ack?'A':'X'),(myHdr->tcp_urg?'U':'X'));
141        printf("Checksum: %X\n",ntohs(myHdr->tcp_check));
142    }
143
144
145    void errProc(const char* str)
146    {
147        fprintf(stderr,"%s: %s \n", str, strerror(errno));
148        exit(1);
149    }
150
151
152    uint16_t checksum(const void *ptr, int len)
153    {
154        int sum = 0;
155        uint16_t answer = 0;
156        uint16_t *w = (uint16_t *) ptr;
157        int nleft = len;
158
159        while(nleft > 1){
160            sum += *w++;
161            nleft -= 2;
162        }
163
164        sum = (sum >> 16) + (sum & 0xFFFF);
165        sum += (sum >> 16);
166        answer = ~sum;
167        return(answer);
168    }
```