6주차_
입출력모델

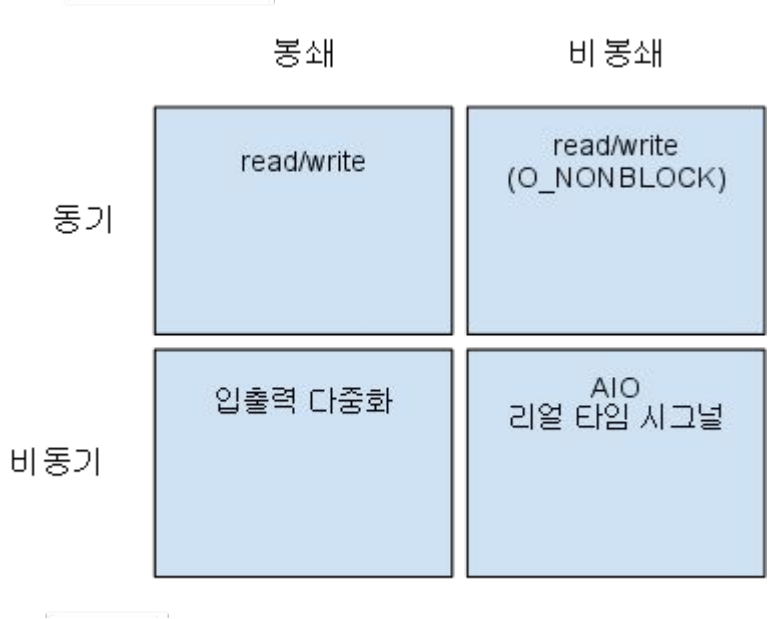
데이터 네트워크연구실
이현호
leeo75@cs-cnu.org

Goals

- 입출력모델에 대해 이해한다
- epoll에 대해 이해하고 실습한다.

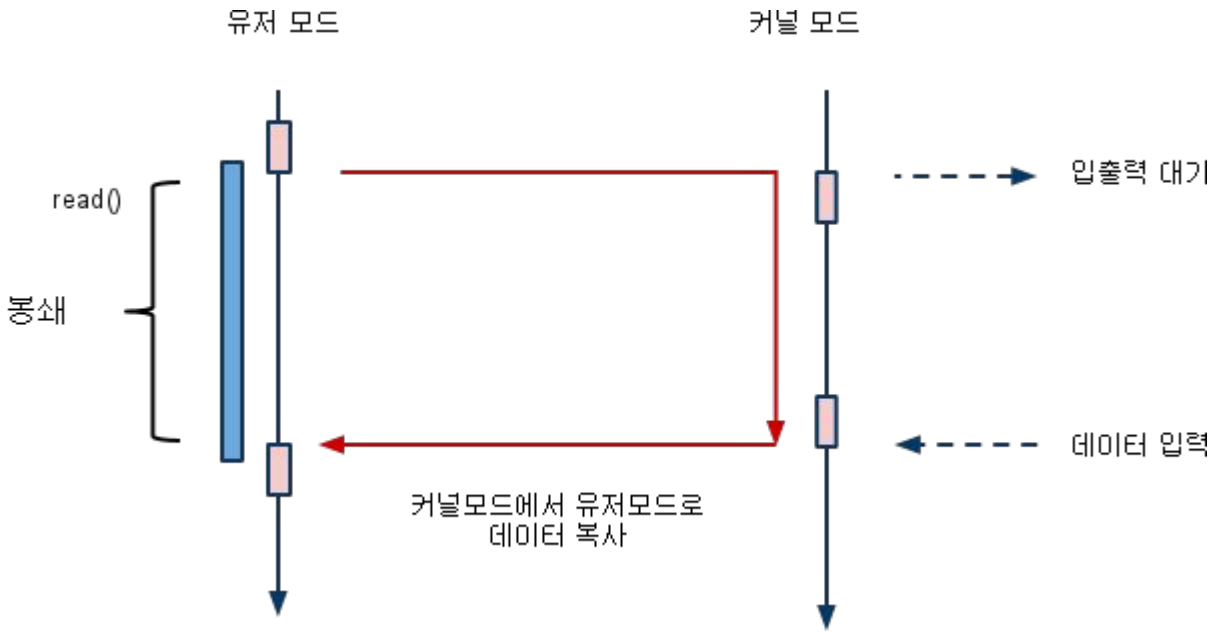
리눅스의 입출력 모델

- 네가지의 모델
- 봉쇄 / 비 봉쇄
  - 프로그램의 상태
  - 함수호출을 한 영역에서 프로그램이 (반환 될 때까지)대기
- 동기 / 비 동기
  - 데이터 상태와 관련
  - 데이터의 입출력 상태를 서로가 알면 동기
  - 그렇지 않으면 비 동기
  - 언제 메시지가 도착할지 모름



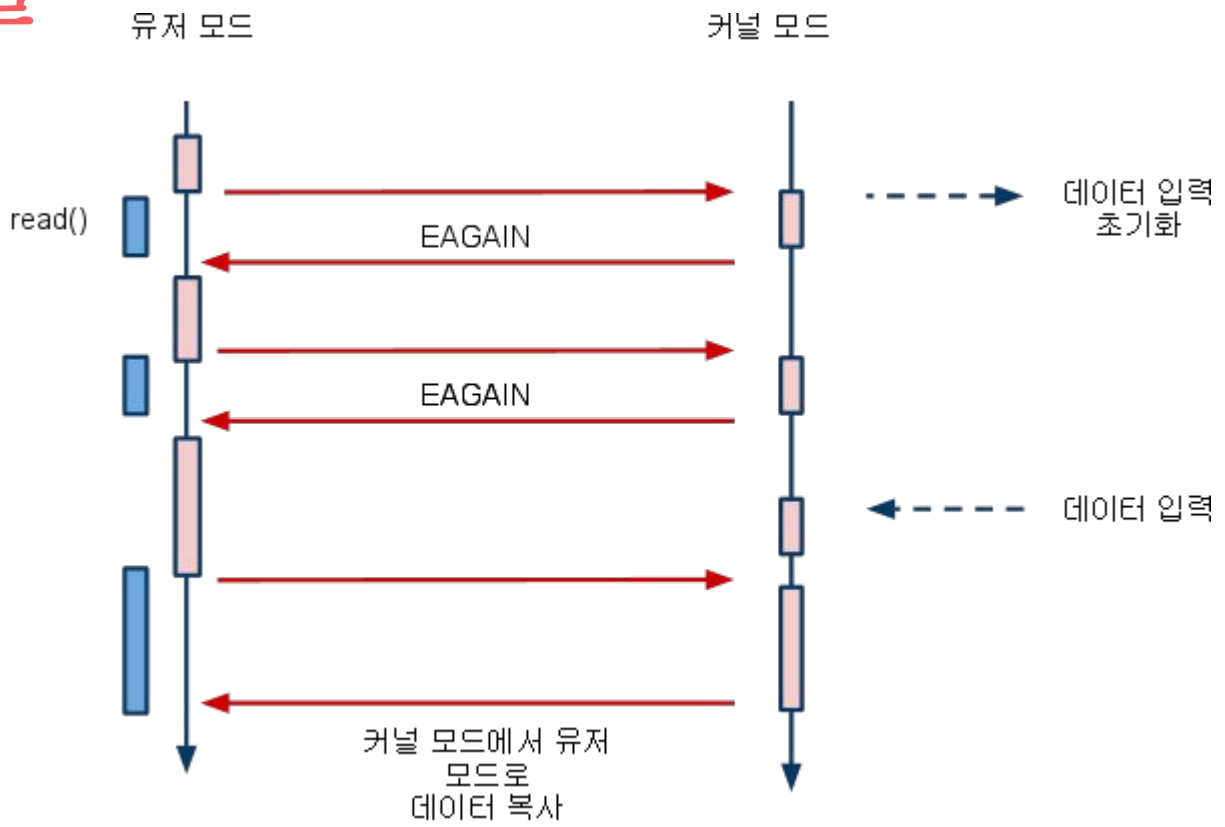
동기 봉쇄 모델

- read함수를 호출시 커널 모드로 요청이 가고 입력대기
- 애플리케이션은 데이터 입력이 있기 까지 봉쇄
- 데이터가 입력되면, 커널 모드에서 유저모드로 데이터가 복사
- 소켓통신은 기본적으로 동기 봉쇄모델

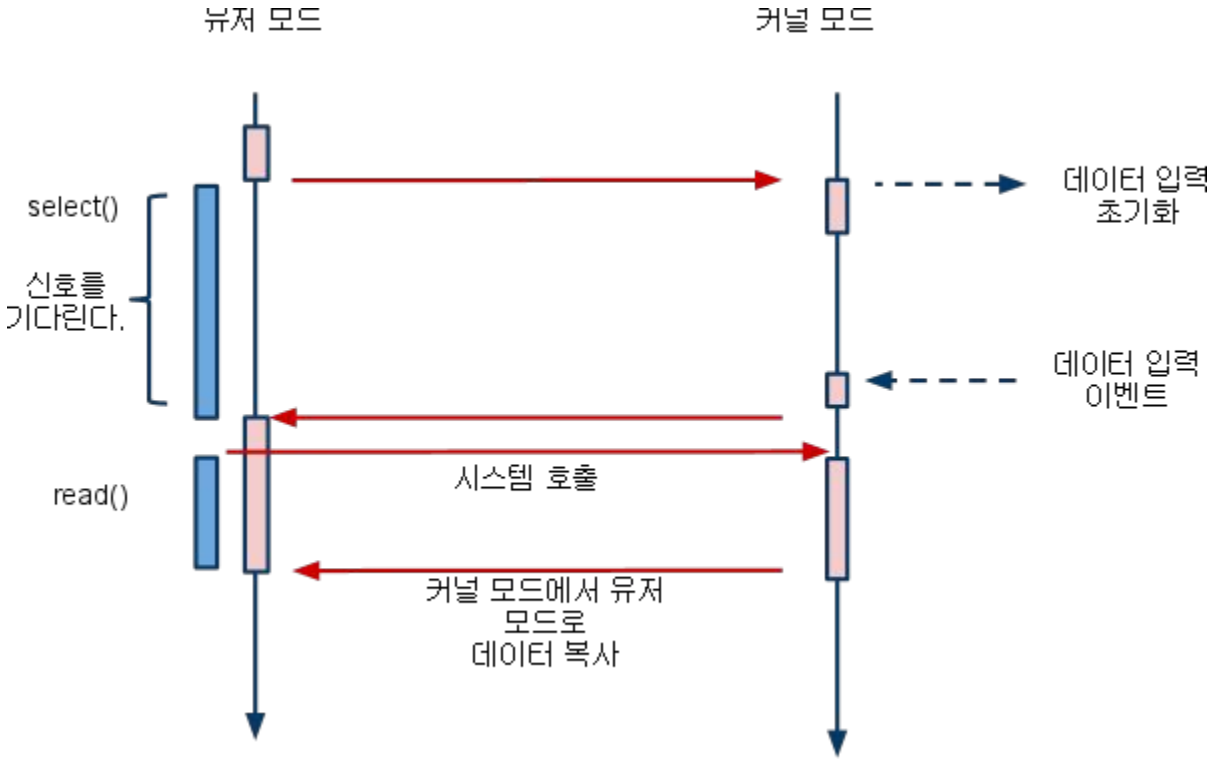


동기 비봉쇄 모델

- read함수는 바로 반환
- 데이터가 준비되지 않았다면, errno는 EAGAIN으로 설정
- 만약 데이터가 준비되어 있다면, 데이터를 읽음
- 데이터가 준비되기 전까지 바쁘게 순환해야 하는 busy wait 상태



비동기 봉쇄 모델

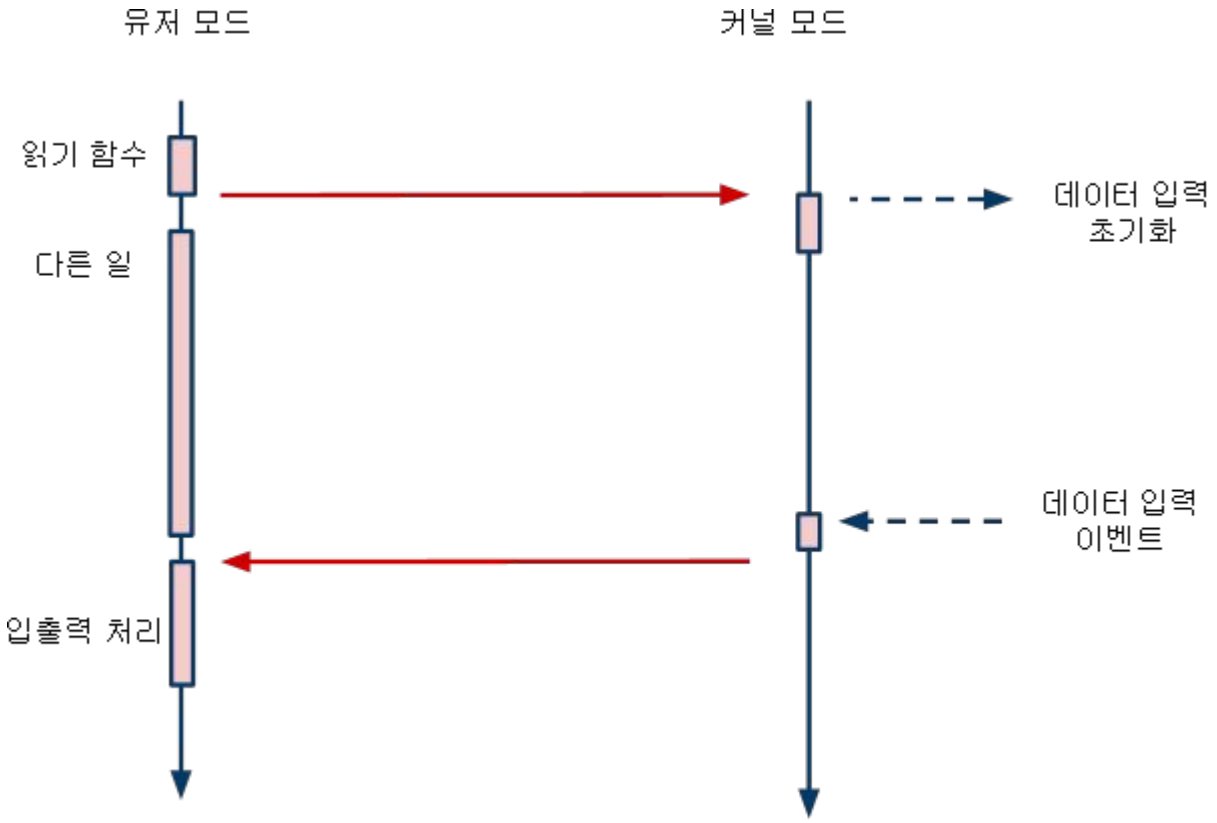

유저 모드
커널 모드
select()
신호를
기다린다.
데이터 입력
초기화
데이터 입력
이벤트
read()
시스템 호출
커널 모드에서 유저
모드로
데이터 복사

비동기 봉쇄 모델

- 동기 비 봉쇄 모델은 계속 read함수를 호출하기 때문에 busy wait 상태에 놓일 수 있다는 *단점* 존재
- 비 동기 봉쇄 모델은 입출력 함수 호출전에 *입출력 데이터가 있는지를 검사하는 함수*(select 혹은 poll)를 미리 배치
- 입출력 데이터가 없을 때는 봉쇄
- 입출력 데이터가 있으면, 비로서 입출력 함수를 호출
- 입출력 함수는 봉쇄 모드로 작동
- 모델중 epoll이 있으며, 현재 리눅스에서 가장 효율적인 네트워크 프로그래밍 도구로 알려져 있음

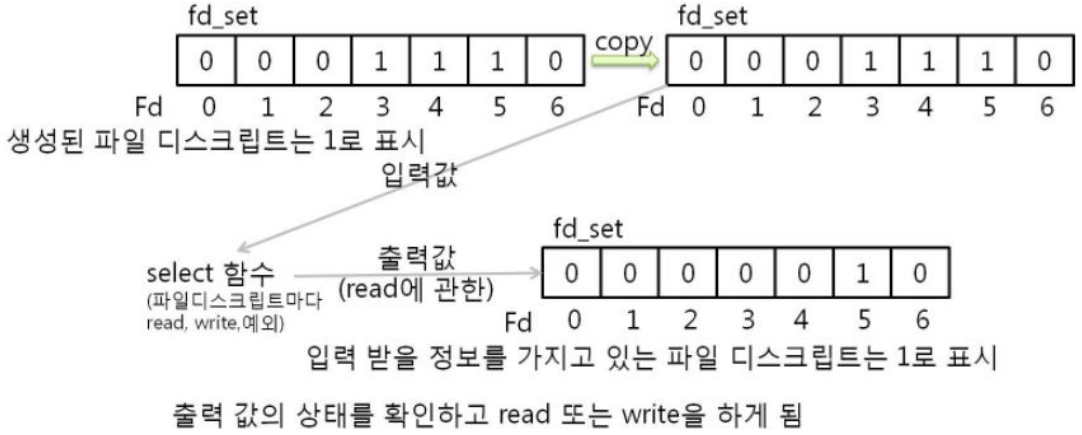
비동기 비 봉쇄

- aio_read함수를 호출한
- 다른 일을 진행
- 읽을 데이터가 발생하면, 콜백 함수 혹은 시그널 핸들러로 데이터를 처리



select 의 단점

- 고정 비트 테이블인 fd_set을 사용하는데 크기가 고정(bitmask. 1024)
- 이벤트가 발생을 감지하기 위해서는 순차검색 fd를 처음부터 끝까지 하나씩 조사한다. O(n)
- 데이터가 오면 기존 fd_set을 모두 변경



대용량 처리기술 epoll

-
- epoll_create
  - 만들자
- epoll_ctl
  - 어떤 것의 어떤 상태 확인
- epoll_wait
  - 상태에 변화가 있을 때까지 대기

epoll_create

- int epoll_create(int size);
- epoll_create는 size 만큼의 커널 폴링 공간을 만드는 함수
- 리턴 값은 정수
  - fd_epoll
- fd_epoll를 이용해서 앞으로 다른 조작

epoll_ctl

- int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
- epoll_ctl은 epoll이 관심을 가져주길 바라는 fd와 그 fd에서 발생하는 관심 있는 사건의 종류를 등록하는 인터페이스

```
epoll_event 구조체
typedef union epoll_data {
    void *ptr;
    int fd;
    __uint32_t u32;
    __uint64_t u64;
} epoll_data_t;

struct epoll_event {
    __uint32_t events;  /* Epoll events */
    epoll_data_t data;  /* User data variable */
};
```

```
int EpollAdd(const int fd)
{
        struct epoll_event ev;

        ev.events  = EPOLLIN | EPOLLOUT | EPOLLERR;
        ev.data.fd = fd;

        return epoll_ctl(fd_epoll, EPOLL_CTL_ADD, fd, &ev);
}
```

epoll_wait

- int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);
- epoll_wait 함수는 관심 있는 fd들에 무슨 일이 일어났는지 조사
- 사건들의 리스트를 (epoll_event).events[] 의 배열로 전달
- 실제 동시 접속수와는 상관없이 maxevents 파라미터로 최대 몇 개까지의 event만 처리할 것임을 지정

epoll_wait

- 만약 현재 접속수가 1만이라면 최악의 경우 1만개의 연결에서 사건이 발생할 가능성도 있기 때문에 1만개의 events[] 배열을 위해 메모리를 확보해 놓아야 하지만, 이 maxevents 파라미터를 통해 한번에 처리하길 희망하는 숫자를 제한가능
- timeout은 epoll_wait의 동작특성을 지정해주는 중요한 요소(밀리세컨드 단위)
- 시간만큼 사건발생을 기다리라는 의미, 기다리는 도중에 사건이 발생하면 즉시 리턴에 (-1)을 지정해주면 영원히 사건을 기다리고(blocking), 0을 지정해주면, 사건이 있건 없건 조사만 하고 즉 시 리턴

```c
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/epoll.h>

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

#define PORT_NUM 6292
#define EPOLL_SIZE 20
#define MAXLINE 1024

struct udata
{
        int fd;
        char name[80];
};
```

```
20 int user_fds[1024];
21 void send_msg(struct epoll_event ev, char *msg);
22 int main(int argc, char **argv)
23 {
24         struct sockaddr_in addr, clientaddr;
25         struct epoll_event ev, *events;
26         struct udata *user_data;
27         int listenfd;
28         int clientfd;
29         int i;
30         socklen_t addrlen, clilen;
31         int readn;
32         int eventn;
33         int epollfd;
34         char buf[MAXLINE];
35
36         events = (struct epoll_event *)malloc(sizeof(struct epoll_event) * EPOLL_SIZE);
37        if((epollfd = epoll_          )) == -1)
```

```
38              return 1;
39
40      addrlen = sizeof(addr);
41      if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
42              return 1;
43      addr.sin_family =
44      addr.sin_port =
45      addr.sin_addr.s_addr = htonl(INADDR_ANY);
46      if(bind (listenfd, (struct sockaddr *)&addr, addrlen) == -1)
47              return 1;
48      listen(listenfd, 5);
49      ev.events =
50      ev.data.fd =
51      epoll_ctl(                                                    );
52      memset(user_fds, -1, sizeof(int) * 1024);
53
54      while(1)
55      {
```

```
56                 eventn = epoll_wait(                              );
57                 if(eventn == -1)
58                 {
59                         return 1;
60                 }
61                 for(i = 0; i < eventn ; i++)
62                 {
63                         if(events[i].data.fd == listenfd)
64                         {
65                                 clilen = sizeof(struct sockaddr);
66                                 clientfd = accept(listenfd, (struct sockaddr *)&clientaddr, &cl
   ilen);
67                                 user_fds[clientfd] = 1;
68
69                                 user_data = malloc(sizeof(user_data));
70                                 user_data->fd = clientfd;
71                                 sprintf(user_data->name, "user(%d)",clientfd);
72
```

```
73                      ev.events = EPOLLIN;
74                      ev.data.ptr = user_data;
75
76                      epoll_ctl(epollfd, EPOLL_CTL_ADD, clientfd, &ev);
77              }
78              else
79              {
80                      user_data = events[i].data.ptr;
81                      memset(buf, 0x00, MAXLINE);
82                      readn = read(user_data->fd, buf, MAXLINE);
83                      if(readn <= 0)
84                      {
85                              epoll_ctl(epollfd, EPOLL_CTL_DEL, user_data->fd, events
   );
86                              close(user_data->fd);
87                              user_fds[user_data->fd] = -1;
88                              free(user_data);
89                      }
```

```
89                                      }
90                                      else
91                                      {
92                                              send_msg(events[i], buf);
93                                      }
94                              }
95                      }
96              }
97 }
```

```
 99 void send_msg(struct epoll_event ev, char *msg)
100 {
101         int i;
102         char buf[MAXLINE+24];
103         struct udata *user_data;
104         user_data = ev.data.ptr;
105         for(i =0; i < 1024; i++)
106         {
107                 memset(buf, 0x00, MAXLINE+24);
108                 sprintf(buf, "%s %s", user_data->name, msg);
109                 if((user_fds[i] == 1))
110                 {
111                         write(i, buf, MAXLINE+24);
112                 }
113         }
114 }
```

```
hyunholee@DNLAB:~/tem
p/Multi_process$ ./ec
ho_client
hi
send : hi
read : user(5) hi
hello
send : hello
my name is
read : send : my name is
read : user(5) hello
leo
send : leo
```