

UNIX 시스템 프로그래밍

» 9장. 파이프

pipe

▶ pipe :

- 한 프로세스에서 다른 프로세스로의 단방향 통신 채널
- write와 read로 data 통신 가능

pipe (2)

▶ pipe 만들기 :

```
#include <unistd.h>  
int pipe(int filedes[2]);
```

- filedes[0] : 읽기용
- filedes[1] : 쓰기용
- 성공시 0, 실패 시 -1 return
- process당 open file수, 시스템내의 file수 제한

pipe (3)

- ▶ pipe의 특성 :
 - FIFO 처리
 - lseek은 작동하지 않음 (왜? 읽은 데이터는 사라짐.)
 - pipe는 fork()에 의해 상속 가능
- ▶ pipe를 이용한 단방향 통신 (부모 → 자식)
 - pipe 생성
 - fork()에 의해 자식 생성 & pipe 복사
 - 부모는 읽기용, 자식은 쓰기용 pipe를 close

```
main() {  
    char ch[10];  
    int  pid, p[2];  
  
    if (pipe(p)==-1) {  
        perror("pipe call");  
        exit(1);  
    }  
  
    pid=fork();  
    if (pid==0){  
        close(p[1]);  
        read(p[0], ch, 10);  
        printf("%s\n", ch);  
        exit(0);  
    }  
  
    close(p[0]);  
    scanf("%s", ch);  
    write(p[1], ch, 10);  
    wait(0);  
    exit(0);  
}
```

pipe (4)

- ▶ pipe를 이용한 양방향 통신
 - pipe 2개 생성
 - fork()에 의해 자식 생성 & pipe 2개 복사
 - pipe1 : 부모는 읽기용, 자식은 쓰기용 pipe를 close
 - pipe2 : 부모는 쓰기용, 자식은 읽기용 pipe를 close
- ▶ blocking read / blocking write
 - read가 blocking 되는 경우 : pipe가 비어 있는 경우
 - write가 blocking 되는 경우 : pipe가 가득 찬 경우

```
main() {  
    int i, in, pid, p[2][2];  
  
    for(i=0;i<2;i++)  
        pipe(p[i]);  
    pid=fork();  
    if (pid==0){  
        close(p[0][1]);  
        close(p[1][0]);  
        read(p[0][0], &in, sizeof(int));  
        in++;  
        write(p[1][1], &in, sizeof(int));  
        exit(0);  
    }  
    close(p[0][0]);  
    close(p[1][1]);  
    scanf("%d", &in);  
    write(p[0][1], &in, sizeof(int));  
    read(p[1][0], &in, sizeof(int));  
    printf("%d\n", in);  
    wait(0);  
    exit(0);  
}
```

pipe (5)


▶ pipe 닫기

- 쓰기 전용 pipe 닫기 : 다른 writer가 없는 경우, read를 위해 기다리던 process들에게 0을 return (EOF과 같은 효과)
- 읽기 전용 pipe 닫기 : 더 이상 reader가 없으면, writer 들은 SIGPIPE signal을 받는다. Signal handling이 되지 않으면 process는 종료; signal handling이 되면, signal 처리 후 write는 -1을 return;

pipe (6)

- ▶ non-Blocking read / non-blocking write:
 - 여러 pipe를 차례로 polling 하는 경우;
- ▶ 사용법 :
 - `#include <fcntl.h>`
 - `fcntl(filedes, F_SETFL, O_NONBLOCK);`
 - `filedes`가 쓰기 전용이고, pipe가 차면 blocking 없이 즉시 -1을 return;
 - 읽기 전용인 경우에는, pipe가 비어 있으면, 즉시 -1을 return;
 - 이 경우, `errno`는 `EAGAIN`;

pipe를 이용한 client-server

- ▶ Client는 하나의 pipe로 request를 write
 - ▶ Server는 여러 개의 pipe로부터 request 를 read
 - no request from any client → server는 blocking
 - a request from any child → read the request
 - more than one request → read them in the order
- 

select 시스템 호출

▶ select system call

- 지정된 file descriptor 집합 중 어느것이 읽기/쓰기가 가능한지 표시
- 읽기, 쓰기가 가능한 file descriptor가 없으면 blocking
- 영구적 blocking을 막기 위해 time out 사용 가능

select 시스템 호출 (2)

▶ 사용법 :

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

```
int select(int nfd, fd_set *readfds, fd_set *writefds,  
          fd_set *errorfds, struct timeval *timeout);
```

- nfd : server가 관심이 있는 file descriptor의 최대 번호
- readfds : 읽기 가능한 file descriptor,
- writefds : 쓰기 가능한 file descriptor,
- errorfds : 오류 발생한 file descriptor를 bit pattern으로 표현
- timeout : timeout 값 설정

fd_set 관련 매크로

- ▶ `void FD_ZERO(fd_set *fdset);`
 - fdset 초기화;
- ▶ `void FD_SET(int fd, fd_set *fdset);`
 - fdset의 fd bit를 1로 설정;
- ▶ `int FD_ISSET(int fd, fd_set *fset);`
 - fdset의 fd bit가 1인지 검사;
- ▶ `void FD_CLR(int fd, fd_set *fset);`
 - fdset의 fd bit를 0으로 설정;

timeval의 구조

▶ timeval의 구조 :

```
struct timeval {  
    long tv_sec;  
    long tv_usec;  
};
```

▶ timeout이

- NULL이면 해당 event가 발생 시까지 blocking.
- 0이면, non-blocking.
- 0이 아닌 값이면, read/write가 없는 경우 정해진 시간 후에 return

select 시스템 호출 (4)

- ▶ select의 return 값은 :
 - -1 : 오류 발생 시
 - 0 : timeout 발생 시
 - 0 보다 큰 정수 : 읽기/쓰기 가능한 file descriptor의 수
- ▶ 주의 사항 : return시 mask를 지우고, 재설정

```
#define MSGSIZE 6
```

```
char *msg1 = "hello";
```

```
char *msg2 = "bye!!";
```

```
void parent(int[][2]);
```

```
int child(int[]);
```

```
main() {
```

```
    int pip[3][2];
```

```
    int i;
```

```
    for (i=0; i<3; i++) {
```

```
        pipe(pip[i]);
```

```
        if(fork()==0)
```

```
            child(pip[i]);
```

```
    }
```

```
    parent(pip);
```

```
    for (i=0; i<3; i++) {
```

```
        wait(0);
```

```
    }
```

```
    exit(0);
```

```
}
```



```

void parent(int p[3][2]){
    char buf[MSGSIZE], ch;
    fd_set set, master;
    int i, j, k;

    for (i=0; i<3; i++)
        close(p[i][1]);

    FD_ZERO(&master);

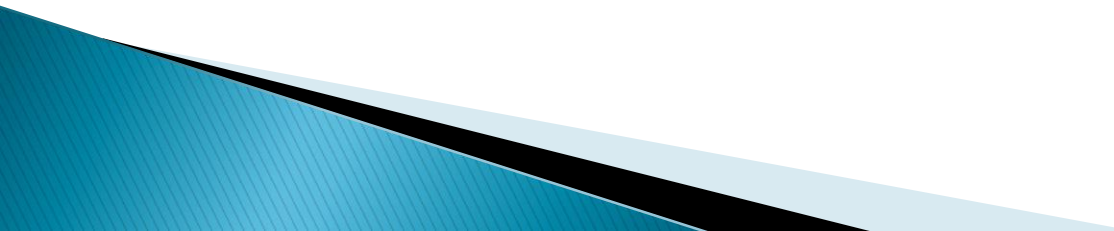
    for (i=0; i<3; i++)
        FD_SET(p[i][0], &master);
    while (set=master, select(p[2][0]+1, &set, NULL, NULL, NULL) > 0) {

        for (i=0; i<3; i++){
            if (FD_ISSET(p[i][0], &set)){
                if (read(p[i][0], buf, MSGSIZE) > 0)
                    printf("MSG from %d=%s\n", i, buf);
            }
        }

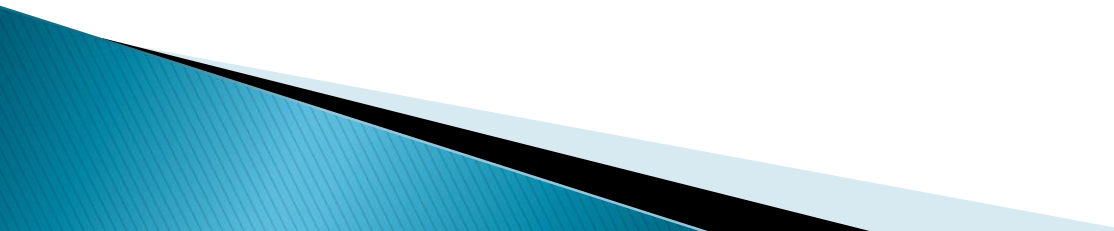
        if (waitpid(-1, NULL, WNOHANG) == -1)
            return;
    }
}

```

```
int child(int p[2]){  
    int count;  
  
    close(p[0]);  
  
    for (count=0; count<2; count++){  
        write(p[1], msg1, MSGSIZE);  
        sleep(getpid()%4);  
    }  
  
    write(p[1], msg2, MSGSIZE);  
    exit(0);  
}
```



FIFO

- ▶ pipe는 동일 ancestor를 갖는 프로세스들만 연결 가능, fifo는 모든 프로세스들을 연결 가능.
 - ▶ UNIX의 file 이름을 부여 받는다.
 - ▶ 소유자, 크기, 연관된 접근 허가를 가진다.
 - ▶ 일반 file 처럼, open, close, read, write, remove가 가능하다.
- 

FIFO (2)

- ▶ 사용법 :
 - fifo 만들기
 - → fifo open (O_RDONLY 또는 O_WRONLY)
 - O_RDWR를 쓰는 경우는?
 - → file에 read 또는 write

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

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

```
int mkfifo(const char *pathname, mode_t mode);
```

FIFO (3)

▶ 예 :

```
mkfifo("/tmp/fifo", 0666);
```

```
fd=open("/tmp/fifo", O_WRONLY);
```

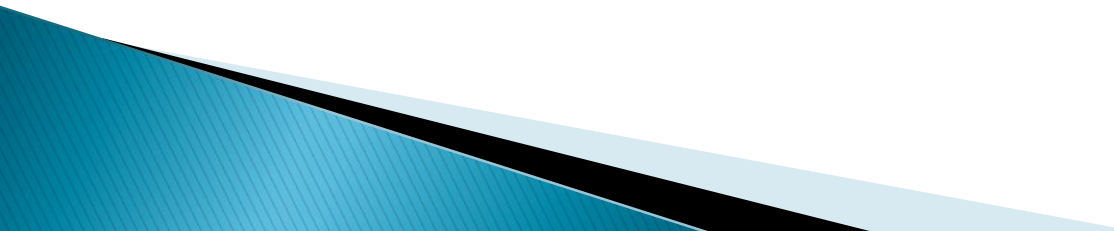
또는

```
fd=open("/tmp/fifo", O_WRONLY|O_NONBLOCK);
```

FIFO (4)

- ▶ 일반 open 호출은 다른 프로세스가 읽기 또는 쓰기를 위해 open될 때 까지 blocking
- ▶ Non-blocking open의 경우, 상대 프로세스가 준비되지 않으면, -1 return. 이때, errno=ENXIO
- ▶ fifo를 이용한 통신의 예제 :
 - reader가 O_RDWR로 fifo를 open한 이유는?
 - writer 종료 시 blocking 된 채로 기다리기 위해, 아니면, 무한 0 return.

```
int main(void){  
    int fd, n;  
    char buf[512];  
  
    mkfifo("fifo", 0600);  
  
    fd=open("fifo", O_RDWR);  
  
    for (;;) {  
        n=read(fd, buf, 512);  
        write(1, buf, n);  
    }  
}
```



```
int main(void){
    int fd, j, nread;
    char buf[512];

    if ((fd=open("fifo", O_WRONLY|O_NONBLOCK)) < 0){
        printf("fifo open failed");
        exit(1);
    }

    for (j=0; j<3; j++){
        nread=read(0, buf, 512);
        write(fd, buf, nread);
    }

    exit(0);
}
```

