

|  |
| --- |
| **운영체제 Report**  **System Call** |

|  |  |
| --- | --- |
| **과 목:** | **운영체제** |
| **교 수 명:** | **이 강 우 교수님** |
| **제 출 일:** | **2019.04.17** |
| **학 번:** | **2015112093** |
| **성 명:** | **이 구 협** |

**Process & Job Manipulation (or Control)**

**● load, execute, end, abort**  
 - execve() : 프로그램을 실행한다. int execve(const char \*filename, char \*const argv[], char \*const envp[])의 형태이다. filename을 file name으로 가지는 file을 실행한다. File 실행에 성공한 경우 return은 없고, File실행에 있어 error발생 시 -1을 return한다.  
code :  
#include <stdio.h>

#include <unistd.h>

main() {

char \*arg[] = {"ls", (char \*)0};

printf("Running ls with execv\n");

/\* 첫 번째 인수가 경로 이름, 명령라인 인수는 인수들의 배열 \*/

execv("/bin/ls", arg);

printf("execv failed to run ls\n");

exit(0);

}

- exit() : 현재 process를 제거한다. void \_exit(int status)의 형태이다. return하지 않는다.

code :

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

int pid;

int status;

int spid;

pid = fork();

if (pid == 0) {

sleep(5);

printf("I will be back %d\n", getpid());

return 1;

}

else if(pid > 0) {

printf("Im parent %d\n", getpid());

printf("Press any key and wait\n");

getchar();

spid = wait(&status);

printf("자식프로세스 wait 성공 \n");

//종료상태(정상종료혹은 비정상종료)를 얻어온다.

printf("PID : %d\n", spid);

printf("Exit Value : %d\n", WEXITSTATUS(status));

printf("Exit Stat : %d\n", WIFEXITED(status));

}

else {

perror("fork error :");

}

}

**● create & kill processes**

- create() : process를 생성한다. int create(const char \*pathname, mode\_t mode)의 형태이다. process 생성에 성공한 경우 File Descriptor을 return하며, error가 발생한 경우 error message와 함께 -1을 return한다.

code :

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main (int argc, char \*\* argv) {

int fd;

// 파일을 생성한다.

fd = creat ("/root/System\_Programming/Chaper\_01/creat\_test.txt", 0644);

// 에러 시 에러 메시지를 출력한다.

if (fd == -1)

printf ("Creat function failed\n");

}

- kill() : process에 signal을 보낸다. Int kill(pid\_t, int sig)의 형태이다. 성공한 경우 0을 return하고, error발생 시 error message와 함께 -1을 return한다.

code :  
#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <signal.h>

void sigusr\_proc(int signo);

int main(int argc, char\* argv[], char\* env[]) {

pid\_t cpid = fork();

printf("parent: create child.\n");

if (cpid < 0) {

// fail인 경우

printf("error: cannot create child.\n");

exit(EXIT\_FAILURE);

}

else if (cpid > 0) { // parent

printf("parent: wait a 1sec.\n");

sleep(1);

printf("parent: signal to child\n");

// child process에게 시그널을 보내기 위한 kill

kill(cpid, SIGUSR1);

printf("parent: wait a 1sec.\n");

sleep(1);

printf("parent: end!\n");

exit(EXIT\_SUCCESS);

}

else if (cpid == 0) { // child

printf("child: set signal receiver.\n");

signal(SIGUSR1, sigusr\_proc);

printf("child: wait...\n");

pause();

printf("child: end!\n");

}

return 0;

}

void sigusr\_proc(int signo) {

printf("child: i caught [%d] signal\n", signo);

}

**● get & set process attributes**

- getpid() : process id를 알려준다. pid\_t getpid(void)의 형태이다. 현재 process의 process ID를 return한다.

code :

#include <stdio.h>

#include <unistd.h>

int main() {

//현재 프로세스와 부모프로세스 ID를 출력한다.

printf("pid: %d\n",getpid());

printf("ppid: %d\n",getppid());

return 0;

}

- getppid() : 부모 process를 얻는다. pid\_t getppid(void)의 형태이다. 부모 process의 process ID를 return한다.

code :

#include <stdio.h>

#include <unistd.h>

int main() {

//현재 프로세스와 부모프로세스 ID를 출력한다.

printf("pid: %d\n",getpid());

printf("ppid: %d\n",getppid());

return 0;

}

**● wait for a certain time**

- wait() : process가 status를 바꿀 때가지 기다린다. pid\_t wait(int \*status)의 형태이다.

code :

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

int pid;

int status;

int spid;

pid = fork();

if (pid == 0) {

sleep(5);

printf("I will be back %d\n", getpid());

return 1;

}

else if(pid > 0) {

printf("Im parent %d\n", getpid());

printf("Press any key and wait\n");

getchar();

// 자식프로세스를 wait 한다.

// 자식프로세스의 종료상태는 status 를 통해 받아온다.

spid = wait(&status);

printf("자식프로세스 wait 성공 \n");

printf("PID : %d\n", spid);

printf("Exit Value : %d\n", WEXITSTATUS(status));

printf("Exit Stat : %d\n", WIFEXITED(status));

}

else {

perror("fork error :");

}

}

- nanosleep() : 특정 시간동안 실행을 중지한다. Int nanosleep(const struct timespec \*req, struct timespec \*rem)의 형태이다.

code :

struct timespec reqtime;

reqtime.tv\_sec = 0;

reqtime.tv\_nsec = 100000000;

// 0.1초동안 sleep한다.

nanosleep(&reqtime, NULL);

**● wait & signal events**

- signal() : signal을 위한 새로운 signal handler을 설치한다. Sighandler\_t\_signal(int signum, sighandler\_t handler)의 형태이다.

code :

#include <signal.h>

#include <unistd.h>

void sig\_handler(int signo); // 비프음을 발생시키는 함수이다.

int main() {

int i = 0;

signal(SIGINT, (void \*)sig\_handler);

while(1) {

printf("%d\n", i);

i++;

sleep(1);

}

return 1;

}

void sig\_handler(int signo) {

printf("SIGINT 발생\n");

}

- sigtimedwait() : set에 있는 signal중 하나가 전달될 때까지 process 호출의 실행을 지연한다. Int sigtimedwait(const sigset\_t \*set, siginfo\_t \*info, const struct timespec \*timeout)의 형태이다.

code :

int main( int argc, char \*argv[] ) {

int result = 0;

struct sigaction sigact;

struct sigset\_t waitset;

siginfo\_t info;

struct timespec timeout;

sigemptyset( &sigact.sa\_mask );

sigact.sa\_flags = 0;

sigact.sa\_handler = catcher;

sigaction( SIGALRM, &sigact, NULL );

sigemptyset( &waitset );

sigaddset( &waitset, SIGALRM );

sigprocmask( SIG\_BLOCK, &waitset, NULL );

timeout.tv\_sec = 10; /\* Number of seconds to wait \*/

timeout.tv\_nsec = 1000; /\* Number of nanoseconds to wait \*/

alarm( 10 );

timestamp( "before sigtimedwait()" );

// 특정 시그널을 기다렸다가 waitset에서 득정 시그널이 오면 실행한다.

result = sigtimedwait( &waitset, &info, &timeout );

printf("sigtimedwait() returned for signal %d\n", info.si\_signo );

timestamp( "after sigtimedwait()" );

return( result );

}

**● allocate & free(deallocate) memory**

- alloc\_hugepages() : memory를 할당한다. 할당에 성공한 경우 할당된 가상 주소를 return한다. void \*alloc\_hugepages(int key, void \*addr, size\_t len, int prot, int flag)의 형태이다.

**File Manipulation**

**● create & delete file**

- mkdir() : pathname을 name으로 가지는 directory를 생성한다. Int mkdir(const char \*pathname, mode\_t mode)의 형태이다. 성공한 경우 0을 return하고, error가 발생한 경우 -1을 return한다.

code :

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

int main(int argc, char \*\* argv) {

if (argc != 2) {

fprintf(stderr, "Usage : jmkdir dirname\n");

return 1;

}

// 0755모드에 argv[1]의 이름을 갖는 파일을 생성한다.

if(mkdir(argv[1], 0755)) {

perror("mkdir error");

return 1;

}

return 0;

}

- rmdir() : pathname을 name으로 가지는 directory를 삭제한다. Int rmdir(const char \*pathname)의 형태이다. 성공한 경우 0을 return하고, error가 발생한 경우 -1을 return한다.

code :

// demo file을 삭제한다.

rmdir("demo");

char buf[10]="Hello";

int re = write(fd, buf, 10);

if(re == 10) {

printf("write ok\n");

}

else {

printf("write fail\n");

}

close(fd);

return 0;

}

**● open & close file**

- open() : pathname을 name으로 가지는 file을 연다. Int open(const char \*pathname, mode\_t mode)의 형태이다. File의 pathname이 주어지면 file descriptor을 return한다.

code :

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main (int \*argc , char \*\* argv) {

int fd;

// fd 파일을 오픈한다.

fd = open("/root/System\_Programming/test.txt", O\_RDONLY);

if (fd == -1) {

printf ("FD Error.... \n");

}

else {

printf ("FD Success...!!! FD is : %d \n" , fd);

}

close(fd);

}

- close() : file descriptor을 닫아서 file descriptor가 어떠한 파일을 참조할 수 없도록 하고, 다시 사용되지 못하게 한다. Int close(int fd)의 형태이다.

close :

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main (int \*argc , char \*\* argv) {

int fd;

fd = open("/root/System\_Programming/test.txt", O\_RDONLY);

if (fd == -1) {

printf ("FD Error.... \n");

}

else {

printf ("FD Success...!!! FD is : %d \n" , fd);

}

// 오픈한 파일을 닫아준다.

close(fd);

}

**● read, write & reposition**

- read() : file descriptor로부터 buffer에서 시작하는 data를 지정한 byte만큼 읽어온다. ssize\_t read(int fd, void \*buf, size\_t count)의 형태이다.

code :

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#define MAX 1024

int main (int argc, char \*\* argv[]) {

int fd;

int readn = 0;

int writen = 0;

char buf[MAX];

int i ;

// Open File and Get File Descriptor

fd = open ("/root/System\_Programming/Chapter\_01/text.txt" ,O\_RDWR, 0644);

printf ("FD is .... %d \n" , fd);

if (fd == -1) {

printf ("File open failed.....\n");

return 1;

}

memset (buf, 0x00, MAX);

// fd를 buf에서 MAX-1byte 만큼 읽어온다.

readn = read(fd, buf, MAX-1);

printf ("Buffer Size : %d\n",sizeof(buf));

printf ("Readn is ..... %d\n", readn);

printf ("%s\n",buf);

close (fd);

}

- write() : file descriptor의 buffer에 data를 지정한 byte만큼 쓴다. ssize\_t write(int fd, const void \*buf, size-t count)의 형태이다.

code :

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#define MAX 1024

int main (int argc, char \*\* argv[]) {

int fd;

int readn;

int writen;

char buf[MAX];

char \*buf2 = "XXXXXXXXXXXX";

fd = open ("/home/june/test.txt" , O\_RDWR);

if (fd == -1) {

printf ("File open failed.....\n");

return 1;

}

// buf2를 fd 파일에 쓴다.

writen = write(fd, buf2, strlen(buf2));

printf ("fd is written with buf \n");

// Set FD cursor to the starting point

int current = lseek(fd, 0, SEEK\_SET);

memset (buf, 0x00, MAX);

readn = read(fd, buf, sizeof(buf2));

printf ("Readn is ..... %d\n", readn);

printf ("buf contents is ... %s\n", buf);

close (fd);

return 1;

}

**● get & set file attributes**

- chdir() : directory를 변경한다. 현재의 directory를 특정한 경로로 변경한다. 성공한 경우 0을 return하고, error가 발생한 경우 -1을 return한다. int chdir(const char \*path)의 형태이다.

code :

#include <stdio.h>

#include <direct.h>

#ifndef \_MAX\_PATH

#define \_MAX\_PATH 260

#endif

int main( ) {

char strBuffer[\_MAX\_PATH] = { 0, };

char strChangeDir[\_MAX\_PATH] = { "C:\\Windows" };

// directory를 strChangeDir로 이동한다.

int nResult = chdir( strChangeDir );

if( nResult == 0 ) {

printf( "이동 성공" );

}

else if( nResult == -1 ) {

perror( "이동 실패 - " );

}

return 0;

}

- chmod() : file의 permission을 변경한다. int chmod(const char \*path, mode\_t mode)의 형태이다. Path로 받은 file의 mode를 변경한다.

code :

#include <stdio.h>

#include <sys/stat.h>

int main() {

// “a.out”파일의 모드를 변경한다.

if ( -1 == chmod( "a.out", 0111)) {

printf( "접근권한 변경 실패n");

}

}

**Device Manipulation**

**● request & release device**

Device가 system call을 이용할 때, device는 file과 같은 추상적인 형태를 띠고 있다. 따라서 file과 같은 system call을 사용한다.

- open() : pathname을 name으로 가지는 file을 연다. int open(const char \*pathname, mode\_t mode)의 형태이다. File의 pathname이 주어지면 file descriptor을 return한다.

code :

// /usr/my.temp 파일을 읽기 전용으로 열고자 할때

fd = open("/usr/my.temp", O\_RDONLY);

...

close(fd);

- close() : file descriptor을 닫아서 file descriptor가 어떠한 파일을 참조하지 못하도록 하고, 다시 사용되지 못하게 한다. int close(int fd)의 형태이다.

code :

// /usr/my.temp 파일을 읽기 전용으로 열고자 할때

fd = open("/usr/my.temp", O\_RDONLY);

...

// 파일 닫기

close(fd);

**● read, write & reposition**

- read() : file descriptor로부터 buffer에서 data를 지정한 byte만큼 읽어온다. ssize\_t read(int fd, void \*buf, size\_t count)의 형태이다.

code :

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#define STDIN 1

int main() {

char buf[80];

memset(buf, 0x00, 80);

// buf에서 80바이트만큼 STDIN에서 읽어온다.

if (read(STDIN, buf, 80) < 0) {

perror("read erro : ");

exit(0);

}

printf("%s", buf);

}

- write() : file descriptor의 buffer에 data를 지정한 byte만큼 쓴다. ssize\_t write(int fd, const void \*buf, size\_t count)의 형태이다.

code :

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#define MAX 1024

int main (int argc, char \*\* argv[]) {

int fd;

int readn;

int writen;

char buf[MAX];

char \*buf2 = "XXXXXXXXXXXX";

fd = open ("/home/june/test.txt" , O\_RDWR);

if (fd == -1) {

printf ("File open failed.....\n");

return 1;

}

// buf2를 fd 파일에 쓴다.

writen = write(fd, buf2, strlen(buf2));

printf ("fd is written with buf \n");

// Set FD cursor to the starting point

int current = lseek(fd, 0, SEEK\_SET);

memset (buf, 0x00, MAX);

readn = read(fd, buf, sizeof(buf2));

printf ("Readn is ..... %d\n", readn);

printf ("buf contents is ... %s\n", buf);

close (fd);

return 1;

}

**● get & set device attributes**

- link() : file에 새로운 filename을 만들어 준다. int link(const char \*oldpath, const char \*newpath)의 형태이다. 이 system call은 존재하는 file에 새로운 link을 만들어 준다.

code :

#include <stdio.h>

#include <unistd.h>

int main() {

//test.c와 sample.c 사이의 링크 생성

if ( -1 == link( "test.c", "sample.c"))

printf( "링크 생성 실패n");

}

- chown() : file의 ownership을 변경한다. int chown(const char \*path, uid\_t owner, gid\_t group)의 형태이다. 이 system call은 path에 의해 특정된 file의 group와 owner을 변경한다.

code :

#include <stdio.h>

#include <unistd.h>

#include <pwd.h>

#include <sys/stat.h>

int main() {

struct passwd \*root\_pw;

if ( 0 != getuid()) {

printf( "수퍼유저만 사용할 수 있습니다.n");

return -1;

}

root\_pw = getpwnam( "jwjw");

// a.out의 소유권을 root\_pw->pw\_uid 로 변경한다.

if ( -1 == chown( "./a.out", root\_pw->pw\_uid, root\_pw->pw\_gid)) {

printf( "소유권 변경 실패n");

}

}

**● logically attach & detach devices**

- setup() : device와 filesystem을 setup한다. void setup()의 형태를 가지며 항상 -1을 return한다.

code :

#include <sys/types.h>

#include <sys/uio.h>

#include <sys/socket.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

// 디바이스 및 파일시스템을 setup

int setup(char \*device, char \*initprog, char \*args[]) {

int fds[2];

int I,nfd, narg,cstat;

pid\_t pid;

struct iovec iov;

}

**Information Maintenance**

**● get & set time & date**

- time() : 1970년 1월 1일 00:00:00 이후의 time값을 받아온다. time\_t time(time\_t \*t)의 형태이다.

code :

#include <stdio.h>

#include <time.h>

int main(void) {

time\_t current\_time;

// 현재 시간 받아오기

time( &current\_time);

printf( "%ldn", current\_time);

printf( ctime( &current\_time));

return 0;

}

- times() : process의 time을 받아온다. clock\_t times(struct tms \*buf)의 형태이다. 현재 process의 time을 buf가 가리키는 struct tms에 저장한다.

code :

#define \_POSIX\_SOURCE

#include <sys/times.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdio.h>

#include <unistd.h>

main() {

int status;

long i, j;

struct tms t;

clock\_t dub;

int tics\_per\_second;

tics\_per\_second = sysconf(\_SC\_CLK\_TCK);

if (fork() == 0) {

for (i=0, j=0; i<1000000; i++)

j += i;

exit(0);

}

if (wait(&status) == -1) {

perror("wait() error");

}

else if (!WIFEXITED(status)) {

puts("Child did not exit successfully");

}

// 현재 프로세스의 시간을 확인하여 –1이 되는지 확인

else if ((dub = times(&t)) == -1) {

perror("times() error");

}

else {

printf("process was dubbed %f seconds ago.\n\n", ((double) dub)/tics\_per\_second);

printf(" utime stime\n");

printf("parent: %f %f\n", ((double) t.tms\_utime)/tics\_per\_second, ((double) t.tms\_stime)/tics\_per\_second);

printf("child: %f %f\n", ((double) t.tms\_cutime)/tics\_per\_second, ((double) t.tms\_cstime)/tics\_per\_second);

}

}

**● get & set system data**

- getpriority() : OS의 scheduling priority를 얻는다. int gerpriority(int which, int who)의 형태이다. -1을 return할 수 있다.

code :

#include <sys/resource.h>

...

int which = PRIO\_PROCESS;

id\_t pid;

int ret;

pid = getpid();

// pid의 priority를 구한다.

ret = getpriority(which, pid);

- sysinfo() : 전체 system 통계의 정보를 얻는다. int sysinfo(struct sysinfo \*info)의 형태이다. return값은 전체 system통계의 정보이다. 성공한 경우 0을 return하며, error가 발생한 경우 -1을 return한다.

code :

#include <sys/sysinfo.h>

int main() {

struct sysinfo info;

//전체 시스템의 정보를 가져온다.

sysinfo(&info);

printf("load: %d %d %d\n", info.loads[0],

info.loads[1],

info.loads[2]);

printf("mem : %d %d %d\n", info.totalram,

info.totalram-info.freeram,

info.freeram);

}

**● get & set process, file & device attributes**

- getppid() : 현재 process의 parent process를 return한다. pid\_t getppid(void)의 형태이다.

code :

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

main() {

pid\_t pid;

if ((pid=fork()) == -1) {

perror("fork failed");

}

else if (pid != 0) {

printf("PID is %ld, child process pid is %ld\n", getpid(), pid);

}

else {

// 부모 프로세스의 프로세스 ID를 반환

printf("PID is %ld, parent process pid is %ld\n", getpid(), getppid());

}

}

- fork() : parent process와 같은 정보를 가지는 child process를 생성한다. pid\_t fork(void)의 형태이다.

code :

#include <stdio.h>

#include <unistd.h>

int main() {

int counter = 0;

pid\_t pid;

printf( "자식 프로세스 생성");

// 자식 프로세스를 생성한다.

pid = fork();

switch(pid) {

case -1 :

{

printf( "자식 프로세스 생성 실패\n");

return -1;

}

case 0 :

{

printf( "저는 자식 프로세스로 디스카운트하겠습니다.\n");

while( 1 )

{

printf( "자식: %d\n", counter--);

sleep( 1);

}

}

default :

{

printf( "저는 부모 프로세스로 카운트하겠습니다.\n");

printf( "자식 프로세스의 pid는 %d입니다.\n", pid);

while( 1 )

{

printf( "부모: %d\n", counter++);

sleep( 1);

}

}

}

}

**Communications**

**● create & delete communication connection**

- socket() : socket을 생성한다(communication의 endpoint). int socket(int domain, int type, int protocol)의 형태이다. File descriptor을 return한다.

code :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#define BUFF\_SIZE 1024

int main( int argc, char \*\*argv) {

int client\_socket;

struct sockaddr\_in server\_addr;

char buff[BUFF\_SIZE+5];

// 소켓을 생성한다.

client\_socket = socket( PF\_INET, SOCK\_STREAM, 0);

if( -1 == client\_socket) {

printf( "socket 생성 실패\n");

exit( 1);

}

memset( &server\_addr, 0, sizeof( server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons( 4000);

server\_addr.sin\_addr.s\_addr= inet\_addr( "127.0.0.1");

if( -1 == connect( client\_socket, (struct sockaddr\*)&server\_addr, sizeof( server\_addr) ) ) {

printf( "접속 실패\n");

exit( 1);

}

write( client\_socket, argv[1], strlen( argv[1])+1); // +1: NULL까지 포함해서 전송

read ( client\_socket, buff, BUFF\_SIZE);

printf( "%s\n", buff);

close( client\_socket);

return 0;

}

- socketpari() : 한 쌍의 연결된 socket을 생성한다. int socketpair(int d, int type, int protocol, int sv[2])의 형태이다. 이 system call은 name이 없는 socket이며, d, type, protocol의 특정 정보를 갖는다.

code :

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <sys/types.h>

int main() {

int pid;

char buf[256];

int fd;

int sv[2];

int num = 0;

// AF\_UNIX domain, SOCK\_STREAM의 자료형, AF\_LOCAL의 protocol을 갖는 socketpair 생성

socketpair(AF\_UNIX, SOCK\_STREAM, AF\_LOCAL, sv);

pid = fork();

if (pid == 0) {

dup2(sv[0], 0);

close(sv[1]);

close(sv[0]);

execl("./pipe\_cl", "pipe\_cl", NULL);

}

else if (pid > 0) {

close(sv[0]);

while(1) {

write(sv[1], (void \*)&num, 4);

printf("write %d\n", num);

read(sv[1], (void \*)&num, 4);

printf("read %d\n", num);

}

}

}

**● send & receive messages**

- send() : message를 또 다른 socket으로 보낸다. ssize\_t send(int s, const void \*buf, size\_t len, int flags)의 형태이다.

code :

#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<arpa/inet.h>

int main(int argc , char \*argv[]) {

int sock;

struct sockaddr\_in server;

char message[1000] , server\_reply[2000];

// Create socket

sock = socket(AF\_INET , SOCK\_STREAM , 0);

if (sock == -1) {

printf("Could not create socket");

}

puts("Socket created");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons( 8888 );

// Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0) {

perror("connect failed. Error");

return 1;

}

puts("Connected\n");

// keep communicating with server

while(1) {

printf("Enter message : ");

scanf("%s" , message);

// message를 socket으로 전송한다.

if( send(sock , message , strlen(message) , 0) < 0) {

puts("Send failed");

return 1;

}

if( recv(sock , server\_reply , 2000 , 0) < 0) {

puts("recv failed");

break;

}

puts("Server reply :");

puts(server\_reply);

}

close(sock);

return 0;

}

- recv() : message를 socket으로부터 읽어온다. ssize\_t recv(int s, void \*buf, size\_t len, int flags)의 형태이다.

code :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#define BUFF\_SIZE 1024

int main( int argc, char \*\*argv) {

int client\_socket;

struct sockaddr\_in server\_addr;

char buff[BUFF\_SIZE+5];

client\_socket = socket( PF\_INET, SOCK\_STREAM, 0);

if( -1 == client\_socket) {

printf( "socket 생성 실패\n");

exit( 1);

}

memset( &server\_addr, 0, sizeof( server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons( 4000);

server\_addr.sin\_addr.s\_addr= inet\_addr( "127.0.0.1");

if( -1 == connect( client\_socket, (struct sockaddr\*)&server\_addr, sizeof( server\_addr) ) ) {

printf( "접속 실패\n");

exit( 1);

}

send( client\_socket, argv[1], strlen( argv[1])+1, 0);

// buff에서 BIUFF\_SIZE만큼 data를 받아온다

recv( client\_socket, buff, BUFF\_SIZE, 0);

printf( "%s\n", buff);

close( client\_socket);

return 0;

}

**● transfer status information**

- getpeername() : 연결된 peer socket의 name을 얻는다. int getpeername(int s, struct sockaddr \*name, socklen\_t \*namelen)의 형태이다. socket s에 연결된 peer의 name을 return한다.

code :

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int main() {

int sockfd;

int client\_sockfd;

int client\_len;

int state;

struct sockaddr\_in clientaddr, serveraddr, myaddr,test;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket error : ");

exit(0);

}

bzero(&serveraddr, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serveraddr.sin\_port = htons(4444);

state = bind(sockfd , (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

if (state == -1) {

perror("bind error : ");

exit(0);

}

state = listen(sockfd, 5);

if (state == -1) {

perror("listen error : ");

exit(0);

}

client\_len = sizeof(clientaddr);

client\_sockfd = accept(sockfd, (struct sockaddr \*)&clientaddr, &client\_len);

// 로컬 소켓의 정보를 얻어온다.

// 얻어온 정보는 myaddr 에 채워진다.

getpeername(client\_sockfd, (struct sockaddr \*)&myaddr, &client\_len);

// 얻어온 정보를 출력한다.

printf("Port : %d\n", ntohs(myaddr.sin\_port));

printf("address : %s\n", inet\_ntoa(myaddr.sin\_addr));

close(client\_sockfd);

return 1;

}

- getsocketname() : socket의 name을 얻는다. int getsocketname(int s, struct sockaddr \*name, socklen\_t \*namelen)의 형태이다. 특정 socket의 현재 name을 return한다.

code :

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int main() {

int sockfd;

int client\_sockfd;

int client\_len;

int state;

struct sockaddr\_in clientaddr, serveraddr, myaddr,test;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket error : ");

exit(0);

}

bzero(&serveraddr, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serveraddr.sin\_port = htons(4444);

state = bind(sockfd , (struct sockaddr \*)&serveraddr, sizeof(serveraddr));

if (state == -1) {

perror("bind error : ");

exit(0);

}

state = listen(sockfd, 5);

if (state == -1) {

perror("listen error : ");

exit(0);

}

client\_len = sizeof(clientaddr);

client\_sockfd = accept(sockfd, (struct sockaddr \*)&clientaddr, &client\_len);

// 로컬 소켓의 정보를 가져온다.

// 언어온 정보는 myaddr 에 채워진다.

getsockname(client\_sockfd, (struct sockaddr \*)&myaddr, &client\_len);

// 얻어온 정보를 출력한다.

printf("Port : %d\n", ntohs(myaddr.sin\_port));

printf("address : %s\n", inet\_ntoa(myaddr.sin\_addr));

close(client\_sockfd);

close>(sockfd);

return 1;

}

**● attach or detach remote devices**

- shutdown() : 이중 통신의 부분 혹은 전부를 닫는다. int shutdown(int s, int how)의 형태이다. 이 system call은 s와 관련 있는 socket을 shutdown한다.

code :

if( fork() ) {

while( gets(buffer) )

write(s,buffer,strlen(buffer));

/\* 쓰기를 위해 연결을 끊는다

서버는 EOF를 받게 된다. Note: 소켓으로부터의 읽기는 여전히 허용된다.

서버는 EOF를 받은 후에 몇 데이터를 보낼 것이다. \*/

shutdown(s,1);

exit(0);

}

- connect() : socket 사이의 connection을 시작한다. int connect(int sockfd, const struct sockaddr \*serv\_addr, socklen\_t addrlen)의 형태이다. sockfd file descriptor에 의해 참조되는 socket을 serv\_addr에 connect한다.

code :

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/socket.h>

#include <unistd.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*\*argv) {

int client\_len;

int client\_sockfd;

FILE \*fp\_in;

char buf\_in[255];

char buf\_get[255];

struct sockaddr\_in clientaddr;

if (argc != 2) {

printf("Usage : ./zipcode\_cl [port]\n");

printf("예 : ./zipcode\_cl 4444\n");

exit(0);

}

client\_sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

clientaddr.sin\_family = AF\_INET;

clientaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

clientaddr.sin\_port = htons(atoi(argv[1]));

client\_len = sizeof(clientaddr);

// client\_sockfd소켓을 clientaddr에 연결해준다.

if (connect(client\_sockfd, (struct sockaddr \*)&clientaddr, client\_len) < 0) {

perror("Connect error: ");

exit(0);

}

while(1) {

printf("지역이름 입력 : ");

fgets(buf\_in, 255,stdin);

buf\_in[strlen(buf\_in) - 1] = '0';

write(client\_sockfd, buf\_in, 255);

if (strncmp(buf\_in, "quit", 4) == 0) {

close(client\_sockfd);

exit(0);

}

while(1) {

read(client\_sockfd, buf\_get, 255);

if (strncmp(buf\_get, "end", 3) == 0) {

break;

}

printf("%s", buf\_get);

}

}

close(client\_sockfd);

exit(0);

}