ls

```
static void
print_attr(char *_____, char *_____)
{
    struct passwd *___;
    struct group  *___;
    struct stat ___;
    char ________[SZ_STR_BUF], ________[SZ_STR_BUF], _;
    char ________[13];
    struct tm *___;

    sprintf(________, "%s/%s", _____, _____);
    if (lstat(________, &___) < 0)
        PRINT_ERR_RET();
    if    (S_IS___(___.st_mode)) c = '_';
    else if (S_IS___(___.st_mode)) c = '_';
    else if (S_IS___(___.st_mode)) c = '_';
    else if (S_IS___(___.st_mode)) c = '_';
    else if (S_IS____(___.st_mode)) c = '_';
    else if (S_IS____(___.st_mode)) c = '_';
    else if (S_IS____(___.st_mode)) c = '_';
    ________[0] = _;
    ________[1] = (___ .st_mode & S_IRUSR)? 'r': '-';
    ________[2] = (___ .st_mode & S_IWUSR)? 'w': '-';
    ________[3] = (___ .st_mode & S_IXUSR)? 'x': '-';
    ________[4] = (___ .st_mode & S_IRGRP)? 'r': '-';
    ________[5] = (___ .st_mode & S_IWGRP)? 'w': '-';
    ________[6] = (___ .st_mode & S_IXGRP)? 'x': '-';
    ________[7] = (___ .st_mode & S_IROTH)? 'r': '-';
    ________[8] = (___ .st_mode & S_IWOTH)? 'w': '-';
    ________[9] = (___ .st_mode & S_IXOTH)? 'x': '-';
    ________[10] = '\0';
    ___ = getpwuid(___.st_uid);
    ___ = getgrgid(___.st_gid);
    ___ = localtime(&___.st_mtime);
    strftime(________, 13, "%b %d %H:%M", ___);
    sprintf(________ + 10, " %3ld %-8s %-8s %8ld %s %s",
            ___.st_nlink, ___->pw_name, ___->gr_name,
            ___.st_size, ________, _____);
    if (S_ISLNK(___.st_mode)) {
        int ____, _____;
        strcat(________, " -> ");
        ____ = strlen(________);
        _____ = readlink(________, ________ + ____, SZ_STR_BUF - ____ - 1);
        ________[____ + _____] = '\0';
    }
    printf("%s\n", ________);
}

static void
print_detail(DIR *___, char *_____)
{
```

```
    struct dirent *___;

    while ((___ = readdir(___)) != NULL)
        print_attr(_____, ___->d_name);
}

static void
get_max_name_len(DIR *___, int *_________, int *__________)
{
    struct dirent *___;
    int ________ = 0;

    while ((___ = readdir(___)) != NULL) {
        int _______ = strlen(___->d_name);
        if (_______ > ________)
            ________ = _______;
    }
    rewinddir(___);
    ________ += 4;
    *__________ = 80 / ________;
    *_________ = ________;
}

static void
print_name(DIR *___)
{
    struct dirent *___;
    int ________, ____________, ___ = 0;

    get_max_name_len(___, &________, &____________);

    while ((___ = readdir(___)) != NULL) {
        printf("%-*s", ________, ___->d_name);
        if ((++___ % ____________) == 0)
            printf("\n");
    }
    if ((___ % ____________) != 0)
        printf("\n");
}

void ls(void)
{
    char *_____;
    DIR  *___;

    _____ = (argc == 0)? ".": argv[0];

    if ((___ = opendir(_____)) == NULL)
        PRINT_ERR_RET();
    if (optc == 0)
        print_name(___);
    else
```

```
            print_detail(___, _____);
        closedir(___);
}
```

cmdjmp.c

```
void
run_cmd(void)
{
    pid_t __;

    if ( ( __ = fork() ) < 0 )
        longjmp(jump, -1);

    else if ( __ == 0) {
        int __, __ = 0;
        char *nargv[100];

        nargv[__++] = cmd;
        for (__ = 0; __ < optc; ++__)
            nargv[__++] = optv[__];
        for (__ = 0; __ < argc; ++__)
            nargv[__++] = argv[__];
        nargv[__++] = NULL;

        if (execvp(cmd, nargv) < 0) {
            perror(cmd);
            exit(1);
        }
    }
    else {
        if (waitpid(__, NULL, 0) < 0)
            longjmp(jump, -1);
    }
}

int
main(int __, char *____[])
{
    int ____ = 1;
    int ____;
    char ________[SZ_STR_BUF];

    setbuf(stdout, NULL);
    setbuf(stderr, NULL);

    help();
    getcwd(cur_work_dir, SZ_STR_BUF);
    ____ = setjmp(jump);

    if(____ != 0){
        if(____ == -1)
            perror(cmd);
        else if(____ == -2)
            print_usage("사용법: ", cmd_tbl[cmd_idx].cmd,
                        cmd_tbl[cmd_idx].opt, cmd_tbl[cmd_idx].arg);
    }
```

```
    for (;;) {
        printf("<%s> %d: ", cur_work_dir, ____);
        if (fgets(________, SZ_STR_BUF, stdin) == NULL)
            break;

        if (get_argv_optv(________) != NULL) {
            ____++;
            proc_cmd();
        }
    }
}
```

cmdc1.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#define _________ __________

char *__________ = "__________";
char *__________ = "__________";

int ________, ________;
int ________;
char __________[__________];

void __________(char *__________)
{
    __________(__________);
    __________;
}

void __________()
{
    ________ = open(__________, __________);
    if (________ < 0)
        __________(__________);

    ________ = open(__________, __________);
    if (________ < 0)
        __________(__________);
}

void __________()
{
    ________(________);
    if (________ != ________)
        ________(________);
}

int __________()
{
    if ((________ = read(____, __________, ________)) <= 0)
        return ________;

    if (write(________, __________, ________) != ________)
        return -1;

    return ________;
```

```
}

int __________()
{
    ________ = read(________, __________, ________);
    if (________ < 0)
        return ________;

    if (write(____, __________, ________) != ________)
        return -1;

    return ________;
}

void __________(void)
{
    while (1) {
        if (__________() <= 0)
            break;
        if (__________() <= 0)
            break;
    }
}

int main(int ________, char *________[])
{
    __________();
    __________();
    __________();
}
```

srv1.c

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

#define _________ __________

char *_________ = "__________";
char *_________ = "__________";

int ________, ________;
int ________;
char __________[__________];

void __________(char *__________)
{
    __________(__________);
    __________;
}

void __________()
{
    ________ = open(__________, __________);
    if (________ < 0)
        __________(__________);

    ________ = open(__________, __________);
    if (________ < 0)
        __________(__________);
}

void __________()
{
    ________(________);
    if (________ != ________)
        ________(________);
}

int main(int ________, char *________[])
{
    char __________[__________];

    __________();

    while (1) {
        ________ = read(________, __________, ________);
        if (________ <= 0)
            break;

        __________[________] = '___';
```

```
        if (strncmp(__________, "____", ____) == 0)
            break;

        printf("%s", __________);

        sprintf(__________, "_______: %s", __________);
        ________ = strlen(__________);

        if (write(________, __________, ________) != ________)
            break;
    }

    __________();
}
```

cmdc2.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#define SZ_STR_BUF 256

char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(msg);
    exit(1);
}

void connect_to_server()
{
    out_fd = open(c_to_s, O_WRONLY);
    if (out_fd < 0)
        print_err_exit(c_to_s);

    in_fd = open(s_to_c, O_RDONLY);
    if (in_fd < 0)
        print_err_exit(s_to_c);
}

void dis_connect()
{
    close(in_fd);
    if (in_fd != out_fd)
        close(out_fd);
}

int input_send()
{
    if ((len = read(0, cmd_line, SZ_STR_BUF)) <= 0)
        return len;

    if (write(out_fd, cmd_line, len) != len)
        return -1;

    return len;
```

```
}

int recv_output()
{
    len = read(in_fd, cmd_line, SZ_STR_BUF);
    if (len < 0)
        return len;

    if (write(1, cmd_line, len) != len)
        return -1;

    return len;
}

void single_process(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
        if (recv_output() <= 0)
            break;
    }
}

void input_send_loop(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
    }
}

void recv_output_loop(void)
{
    while (1) {
        if (recv_output() <= 0)
            break;
    }
}

void dual_process(void)
{
    pid_t _____;

    if ((_____ = fork()) < 0)
        perror("fork");
    else if (_____ > 0) {
        close(in_fd);
        input_send_loop();
        wait(NULL);
    }
    else {
```

```
            close(out_fd);
            recv_output_loop();
            printf("_____");
        }
}

int main(int argc, char *argv[])
{
    connect_to_server();
    dual_process();
    dis_connect();
}
```

src2.c

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

#define SZ_STR_BUF 256

char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(msg);
    exit(1);
}

void connect_to_client()
{
    in_fd = open(c_to_s, O_RDONLY);
    if (in_fd < 0)
        print_err_exit(c_to_s);

    out_fd = open(s_to_c, O_WRONLY);
    if (out_fd < 0)
        print_err_exit(s_to_c);
}

void dis_connect()
{
    close(in_fd);
    if (in_fd != out_fd)
        close(out_fd);
}

void duplicate_IO()
{
    ----------;
    ----------;
    ----------;

    ----------;
    ----------;

    ----------;
}
```

```
int main(int argc, char *argv[])
{
    char ret_buf[SZ_STR_BUF];

    connect_to_client();
    duplicate_IO();

    while (1) {
        __________;
        if (len <= 0)
            break;

        cmd_line[len] = '\0';
        if (strncmp(cmd_line, "exit", 4) == 0)
            break;

        printf("%s", cmd_line);

        sprintf(ret_buf, "server: %s", cmd_line);
        len = strlen(ret_buf);

        if (__________)
            break;
    }

    dis_connect();
}
```

cmdc3.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#define SZ_STR_BUF 256

char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(msg);
    exit(1);
}

void connect_to_server()
{
    out_fd = open(c_to_s, O_WRONLY);
    if (out_fd < 0)
        print_err_exit(c_to_s);

    in_fd = open(s_to_c, O_RDONLY);
    if (in_fd < 0)
        print_err_exit(s_to_c);
}

void dis_connect()
{
    close(in_fd);
    if (in_fd != out_fd)
        close(out_fd);
}

int input_send()
{
    if ((len = read(0, cmd_line, SZ_STR_BUF)) <= 0)
        return len;

    if (write(out_fd, cmd_line, len) != len)
        return -1;

    return len;
```

```
}

int recv_output()
{
    len = read(in_fd, cmd_line, SZ_STR_BUF);
    if (len < 0)
        return len;

    if (write(1, cmd_line, len) != len)
        return -1;

    return len;
}

void single_process(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
        if (recv_output() <= 0)
            break;
    }
}

void input_send_loop(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
    }
}

void recv_output_loop(void)
{
    while (1) {
        if (recv_output() <= 0)
            break;
    }
}

static void sig_child(int sig)
{
    printf("_____");   // ← 예: "parent: sig_child: exit\n"
    exit(0);
}

void dual_process(void)
{
    pid_t pid;

    if ((pid = fork()) < 0)
        perror("fork");
```

```
    else if (pid > 0) {
        signal(_____, _____);  // ← 예: SIGCHLD, sig_child
        close(in_fd);
        input_send_loop();
        wait(NULL);
    }
    else {
        close(out_fd);
        recv_output_loop();
        printf("child: exit\n");
    }
}

int main(int argc, char *argv[])
{
    connect_to_server();
    dual_process();
    dis_connect();
}
```

src3.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define SZ_STR_BUF 256

char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(msg);
    exit(1);
}

void connect_to_client()
{
    __________;  // 클라이언트 → 서버용 FIFO 생성
    __________;  // 서버 → 클라이언트용 FIFO 생성

    in_fd = open(c_to_s, O_RDONLY);
    if (in_fd < 0)
        print_err_exit(c_to_s);

    out_fd = open(s_to_c, O_WRONLY);
    if (out_fd < 0)
        print_err_exit(s_to_c);
}

void dis_connect()
{
    close(in_fd);
    if (in_fd != out_fd)
        close(out_fd);
}

void duplicate_IO()
{
    dup2(in_fd, 0);
    dup2(out_fd, 1);
    dup2(out_fd, 2);

    setbuf(stdin, NULL);
```

```
    setbuf(stdout, NULL);

    dis_connect();
}

int main(int argc, char *argv[])
{
    // char ret_buf[SZ_STR_BUF];

    connect_to_client();
    duplicate_IO();

    while (1) {
        if (__________ == NULL)  // 클라이언트로부터 fgets로 입력
            break;

        if (strncmp(cmd_line, "exit", 4) == 0)
            break;

        __________;  // 클라이언트 메시지를 출력
        __________;  // 1초 대기
        __________;  // 동일 메시지 재출력
    }

    dis_connect();
}
```

cmdc.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#define SZ_STR_BUF 256

char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(msg);
    exit(1);
}

void connect_to_server()
{
    __________;  // FIFO 파일을 서버에 쓰기용으로 연다
    if (out_fd < 0)
        print_err_exit(c_to_s);

    __________;  // FIFO 파일을 서버에서 읽기용으로 연다
    if (in_fd < 0)
        print_err_exit(s_to_c);
}

void dis_connect()
{
    close(in_fd);
    if (in_fd != out_fd)
        close(out_fd);
}

int input_send()
{
    if ((len = read(0, cmd_line, SZ_STR_BUF)) <= 0)
        return len;

    if (__________ != len)   // 서버에 입력 전달
        return -1;
    return len;
}
```

```
int recv_output()
{
    __________;  // 서버로부터 메시지 읽기
    if (len < 0) return len;

    if (__________ != len)   // 화면에 출력
        return -1;
    return len;
}

void single_process(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
        if (recv_output() <= 0)
            break;
    }
}

void input_send_loop(void)
{
    while (1) {
        if (input_send() <= 0)
            break;
    }
}

void recv_output_loop(void)
{
    while (1) {
        if (recv_output() <= 0)
            break;
    }
}

static void sig_child(int sig)
{
    exit(0);
}

void dual_process(void)
{
    pid_t pid;
    if ((pid = fork()) < 0)
        perror("fork");
    else if (pid > 0) {
        signal(SIGCHLD, sig_child);
        __________;   // 부모는 in_fd 닫기
        input_send_loop();
        wait(NULL);
```

```
    }
    else {
        __________;    // 자식은 out_fd 닫기
        recv_output_loop();
    }
}

int main(int argc, char *argv[])
{
    connect_to_server();
    dual_process();
    dis_connect();
}
```

cmds.c

```
#include <__________>
#include <__________>
#include <__________>
#include <__________>
#include <__________>
#include <__________>
#include <__________>

#define SZ_STR_BUF __________

char *s_to_c = "__________";
char *c_to_s = "__________";

int in_fd, out_fd;
int len;
char cmd_line[__________];

void print_err_exit(char *__________)
{
    perror(__________);
    exit(__________);
}

void connect_to_client()
{
    mkfifo(__________, __________);
    mkfifo(__________, __________);

    in_fd = open(__________, __________);
    if (in_fd < 0)
        print_err_exit(__________);

    out_fd = open(__________, __________);
    if (out_fd < 0)
        print_err_exit(__________);
}

void dis_connect()
{
    close(__________);
    if (__________ != __________)
        close(__________);
}

void duplicate_IO()
{
    dup2(__________, __________);
    dup2(__________, __________);
    dup2(__________, __________);

    setbuf(__________, __________);
```

```
    setbuf(__________, __________);

    dis_connect();
}

int main(int __________, char *__________[])
{
    printf("cmds: Server is Ready.\n");

    while (__________) {
        connect_to_client();

        pid_t pid = fork();

        if (pid < 0)
            print_err_exit("__________");
        else if (pid == __________) {
            printf("client connected: cmd(pid %d) exec\n", getpid());
            duplicate_IO();
            execl("__________", "__________", __________);
        } else {
            dis_connect();
            waitpid(__________, __________, __________);
            printf("disconnected: cmd(pid %d) terminated\n", pid);
            sleep(__________);
        }
    }
}
```

cmdsd.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <signal.h>

#define SZ_STR_BUF __________

char *s_to_c = "________";
char *c_to_s = "________";

int in_fd, out_fd;
int len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
    perror(________);
    exit(________);
}

void connect_to_client()
{
    mkfifo(________, ________);
    mkfifo(________, ________);

    in_fd = open(________, O_RDONLY);
    if (in_fd < 0)
        print_err_exit(________);

    out_fd = open(________, O_WRONLY);
    if (out_fd < 0)
        print_err_exit(________);
}

void dis_connect()
{
    close(________);
    if (in_fd != out_fd)
        close(________);
}

void duplicate_IO()
{
    // dup2(in_fd, 0);
```

```
    // dup2(out_fd, 1);
    dup2(________, 2);

    setbuf(________, NULL);
    setbuf(________, NULL);

    // dis_connect(); // 실제 연결은 표준 입출력으로 대체됨
}

void daemonize(void)
{
    int i;
    pid_t pid;
    struct rlimit rl;

    umask(________);

    if ((pid = fork()) < 0)
        print_err_exit("________");
    else if (pid > 0)
        exit(0);

    setsid();

    signal(________, SIG_IGN);

    if ((pid = fork()) < 0)
        print_err_exit("________");
    else if (pid > 0)
        exit(0);

    // chdir("/");

    if (getrlimit(RLIMIT_NOFILE, &rl) < 0)
        print_err_exit("________");

    if (rl.rlim_max == RLIM_INFINITY)
        rl.rlim_max = ________;

    for (i = 0; i < rl.rlim_max; ++i)
        close(i);
}

int main(int argc, char *argv[])
{
    printf("cmdsd: Daemon is Ready.\n");

    daemonize();

    while (1) {
        connect_to_client();
        pid_t pid = fork();
```

```
        if (pid < 0)
            print_err_exit("________");
        else if (pid == 0) {
            duplicate_IO();
            execl("________", "________", NULL);
        } else {
            dis_connect();
            waitpid(________, NULL, 0);
            sleep(________);
        }
    }
}
```

cmdc_thread.c

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

#define SZ_STR_BUF              ________

char *s_to_c = "________";
char *c_to_s = "________";

int   in_fd, out_fd;
int   len;
char cmd_line[SZ_STR_BUF];

void print_err_exit(char *msg)
{
        perror(________);
        exit(________);
}

void connect_to_server()
{
        out_fd = open(________, O_WRONLY);
        if (out_fd < 0)
                print_err_exit(________);

        in_fd = open(________, O_RDONLY);
        if (in_fd < 0)
                print_err_exit(________);
}

void dis_connect()
{
        close(________);
        if (in_fd != out_fd)
                close(________);
}

int input_send()
{
        if ((len = read(________, cmd_line, SZ_STR_BUF)) <= 0)
                return len;

        if (write(________, cmd_line, len) != len)
                return -1;

        return len;
}

int recv_output()
```

```
{
        len = read(________, cmd_line, SZ_STR_BUF);
        if (len < 0) return len;

        if (write(________, cmd_line, len) != len)
                return -1;

        return len;
}

void input_send_loop(void)
{
        while (1) {
                if (input_send() <= 0)
                        break;
        }
}

void recv_output_loop(void)
{
        while(1) {
                if (recv_output() <= 0)
                        break;
        }
}

#include <pthread.h>

void *InputSendThread(void *arg)
{
        input_send_loop();
        return(NULL);
}

void *RecvOutputThread(void *arg)
{
        recv_output_loop();
        return(NULL);
}

void thread_err_exit(int err, char *msg)
{
        printf("%s: %s\n", msg, strerror(err));
        exit(1);
}

void dual_threads(void)
{
        int ret;
        pthread_t tid1, tid2;
        if ((ret = pthread_create(&tid1, NULL, InputSendThread, NULL)) != 0)
                thread_err_exit(ret, "pthread_create");
```

```
        if ((ret = pthread_create(&tid2, NULL, RecvOutputThread, NULL)) != 0)
                thread_err_exit(ret, "pthread_create");
        pthread_join(tid2, NULL);
}

int main(int argc, char *argv[])
{
        connect_to_server();
        dual_threads();
        dis_connect();
}
```