쉘 인터프리터 (9장, 11장, 12장 실습문제)

```
#include <stdlib.h>
#include <fcntl.h>
#include <signal.h>
#define MAXARG 7
#define READ 0
#define WRITE 1
#define MAXCMD 10
void alarmHandler();
int pid;
int main(){
        char buf0[256], buf[256];
        char *args[MAXARG];
        char **cmd[MAXCMD];
        char *s, *d;
        char *save;
        int argn, cmdn, fd, i, ch;
        int limit;
        const char delim[] = " \t\n";
        int shpipe[2];
        int pid1, pid2, pid3, status, child;
```

(1) 명령어 실행

[shell] cmd

(2) 명령어 순차적 실행

[shell] cmd1; cmd2; cmd3

(3) 후면 실행

[shell] cmd &

(4) (5) 입출력 리디렉션

[shell] cmd > outfile

[shell] cmd < infile

(6) n초 내 실행이 끝나지 않으면 강제종료

[shell] n cmd1

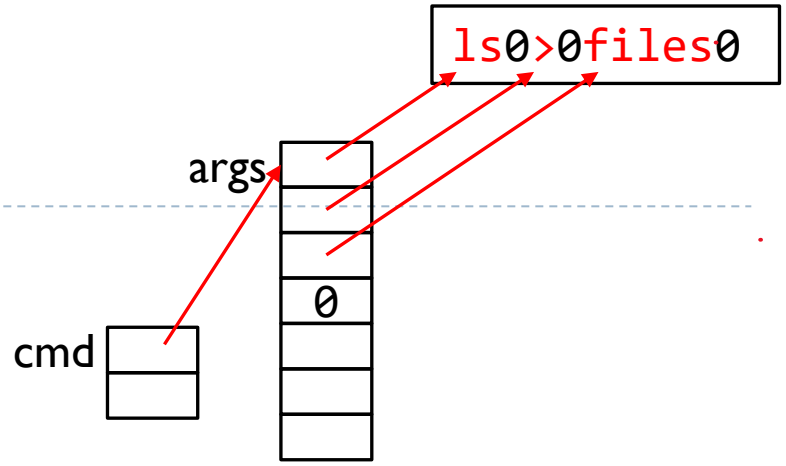
(7) 파이프 기능

[shell] cmd1 | cmd2

```
while(1){
    printf("[shell] ");
    gets(buf0);
    argn = 0;
    cmdn = 0;
    s = buf0;
    d = buf;
    while (*s != '\0') {
      switch (*s) {
        case '>':
        case '<':
        case '|':
        case '&':
        case ';': *d++ = ' '; *d++ = *s++; *d++ = ' ';
        default : *d++ = *s++;
      }
    }
    *d = '\0';
```

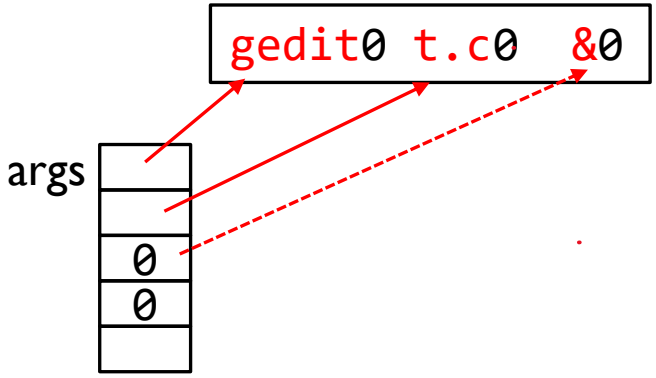
입력 [shell] ls >files

buf0	ls >files0
buf	ls > files0
나중에	ls0 >0files0

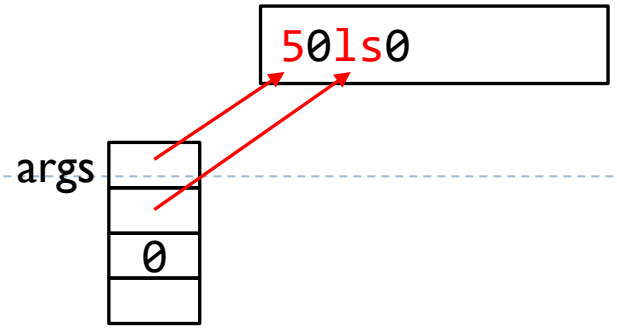


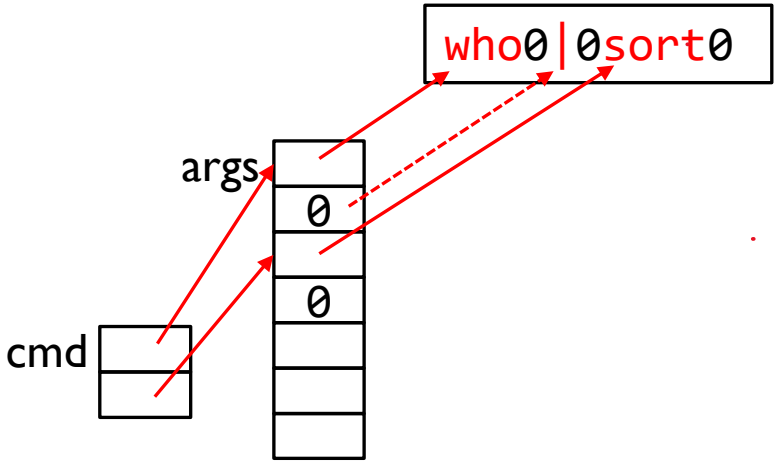

```
s = strtok_r(buf, delim, &save);
while(s){
        args[argn++] = s;
        s = strtok_r(NULL, delim, &save);
}
args[argn] = NULL;
cmd[cmdn++] = &args[0];
if (argn == 0)
        continue;

if (!strcmp(args[0], "quit")) /* ‘quit’이면 while 문 벗어남 */
        break;
```

```
ch = 1;
for(i = 0; i < argn; i++) {
        if(strcmp(args[i], "&") == 0){
                ch = 2;
                args[i] = NULL;
                break;
        }
        else if(strcmp(args[i], ">") == 0){
                ch = 4;
                args[i] = NULL;
                break;
        }
        else if(strcmp(args[i], "<") == 0){
                ch = 5;
                args[i] = NULL;
                break;
        }
```

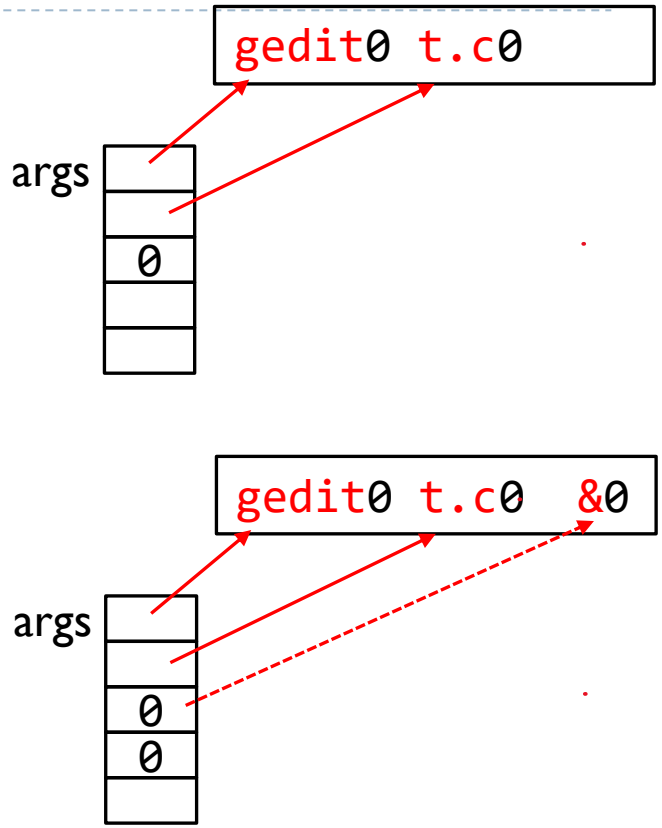


```
        else if(isdigit(*args[i])){
                ch = 6;
                limit = atoi(args[i]);
                break;
        }
        else if(strcmp(args[i], ";") == 0){
                ch = 3;
                args[i] = NULL;
                cmd[cmdn++] = &args[i+1];
        }
        else if(strcmp(args[i], "|") == 0){
                ch = 7;
                args[i] = NULL;
                cmd[cmdn++] = &args[i+1];
        }
} // end for
```

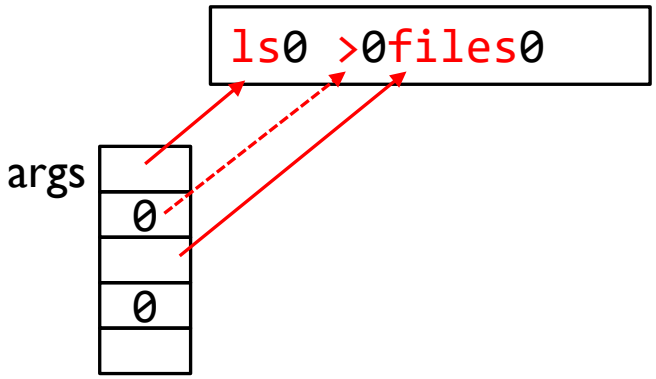




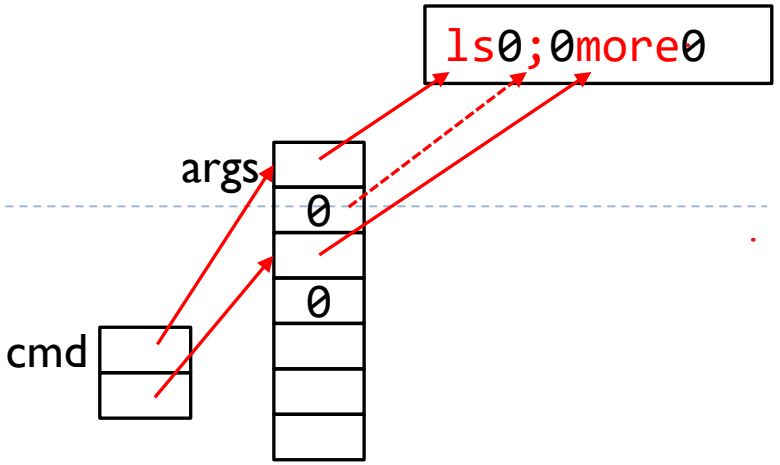
```
if(ch == 1){
        if((pid = fork()) == -1)
                perror("fork failed");
        else if(pid != 0)
                child = waitpid(pid, &status, 0);
        else
                execvp(args[0], args);
}
if(ch == 2){                    // &
        if((pid = fork()) == -1)
                perror("fork failed");
        else if(pid == 0){
                execvp(args[0], args);
        }
}
```

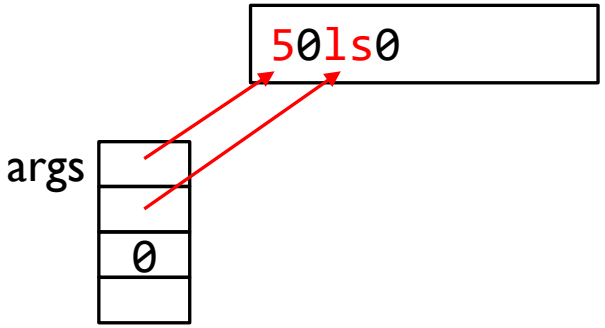


```
if(ch == 4){                        // >
    if((pid = fork()) == 0){
        fd = open(args[i+1], O_CREAT|O_TRUNC|O_WRONLY, 0600);
        dup2(fd, 1);
        close(fd);
        execvp(args[0], args);
        fprintf(stderr, "%s: 실행 불가 \n", args[0]);
    }
    else
        child = waitpid(pid, &status, 0);
}
if(ch == 5){                        // <
    if((pid = fork()) == 0){
        fd = open(args[i+1], O_RDONLY);
        dup2(fd, 0);
        close(fd);
        execvp(args[0], args);
        fprintf(stderr, "%s: 실행 불가 \n", args[0]);
    }
    else    child = waitpid(pid, &status, 0);
}
```

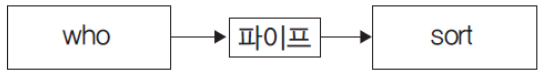


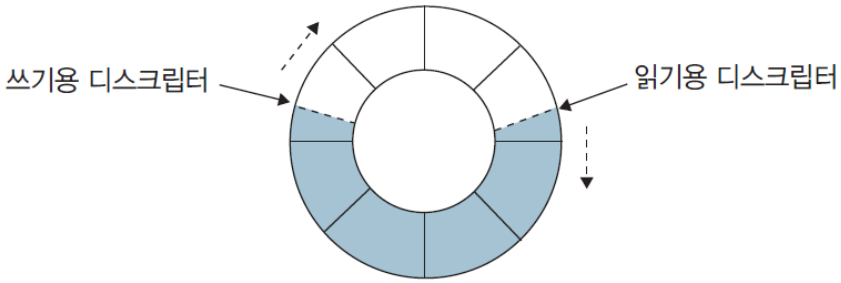
```
if(ch == 3){                        // group
    for (i=0; i<cmdn; i++) {
        if((pid = fork()) != 0) {
            pid = waitpid(pid,&status,0);
        }
        else {
            execvp(cmd[i][0], cmd[i]);
        }
    }
}
if(ch == 6){
    signal(SIGALRM, alarmHandler);
    alarm(limit);

    if((pid = fork()) == 0){
        execvp(args[1], &args[1]);
        fprintf(stderr, "%s: 실행불가\n", args[1]);
    }
    else{
        child = waitpid(pid, &status, 0);
        printf("[%d]자식프로세스 %d 종료 \n", getpid(), pid);
        alarm(0);
    }
}
```

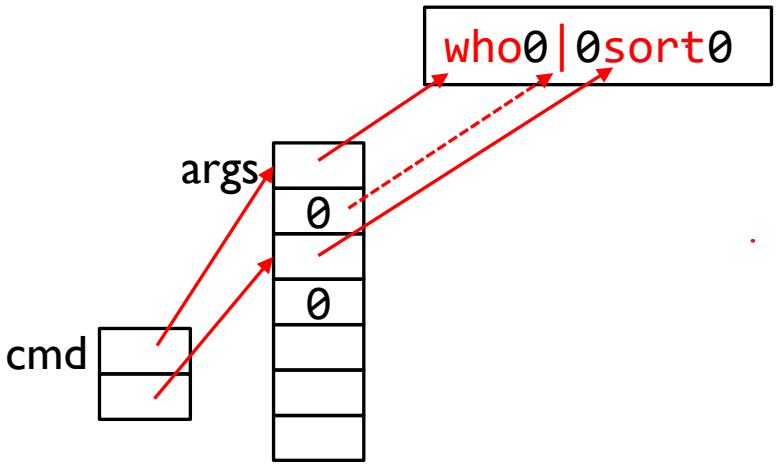




```
        if(ch == 7){
            if((pid = fork()) == 0){
                pipe(shpipe);
                if(fork() == 0){
                    close(shpipe[READ]);
                    dup2(shpipe[WRITE], 1);
                    close(shpipe[WRITE]);
                    execvp(cmd[0][0], cmd[0]);
                    perror("pipe");
                }
                else{
                    close(shpipe[WRITE]);
                    dup2(shpipe[READ], 0);
                    close(shpipe[READ]);
                    execvp(cmd[1][0], cmd[1]);
                    perror("pipe");
                }
            }
            else
                child = wait(pid, &status, 0);
        }
    } // while
    exit(0);
}
```







```
void alarmHandler(){
	printf("[알람]자식 프로세스 %d 시간 초과\n", pid);
	kill(pid, SIGINT);
}
```