

보고서

제목: 유닉스 프로그래밍 실습 프로젝트



과 목 명: 유닉스프로그래밍

제출일자: 2020.11.29

학 과: 컴퓨터공학과

학 번: 12161597

이 름: 부성준

내용

1. chdir() 처리.....	4
(1) 요구사항 정의.....	4
(2) 구현 방법.....	4
(3) 구현 시 문제점.....	4
(4) 테스트 사항.....	5
2. exit() 처리.....	5
(1) 요구사항 정의.....	5
(2) 구현 방법.....	5
(3) 구현 시 문제점.....	6
(4) 테스트 사항.....	6
3. 백그라운드 실행.....	6
(1) 요구사항 정의.....	6
(2) 구현 방법.....	6
(3) 구현 시 문제점.....	7
(4) 테스트 사항.....	7
4. 시그널 처리.....	12
(1) 요구사항 정의.....	12
(2) 구현 방법.....	12
(3) 구현 시 문제점.....	13
(4) 테스트 사항.....	13
5. Pipe 처리.....	14
(1) 요구사항 정의.....	14

(2)	구현 방법	15
(3)	구현 시 문제점	20
(4)	테스트 사항	21
6.	Pseudo Code	23

1. chdir() 처리

(1) 요구사항 정의

'cd' 명령이 제대로 먹히지 않는 버그 수정한다.

(2) 구현 방법

cd 명령어가 들어온 경우 child 프로세스를 생성하지 않고 parent 프로세스에서 chdir()을 실행하도록 하였다. 기존에 cd 명령이 제대로 먹히지 않았던 이유는 프로그램이 명령을 받았을 때 child process를 생성한 후 child process의 current working directory만 변경했기 때문에 parent process의 current working directory는 그대로였기 때문이다. 따라서 child process가 종료된 이후에는 current working directory가 다시 기존의 것으로 돌아왔다. 따라서 코드의 makelist() 실행 위치를 fork() 실행 전으로 옮겨서 child process가 생성되기 전에 명령어가 'cd'인지 확인하고, 만약 그렇다면 child process를 생성하지 않고 current working directory를 변경하도록 하였다.

```
43 while (1) {
44     fputs(prompt, stdout);
45     fgets(cmdline, BUFSIZ, stdin);
46     cmdline[strlen(cmdline) - 1] = '\0';
47
48     /* 명령을 받은 직후 makelist 실행 */
49     /* numtokens: command line 단어 개수 */
50     numtokens = makelist(cmdline, " \t", cmdvector, MAX_CMD_ARG);
51
52     /* 명령 내용이 없는 경우 */
53     if(numtokens == 0){
54         continue;
55     }
56     /* cd 명령어가 입력된 경우 */
57     else if(strcmp(cmdvector[0], "cd") == 0 ){
58         if(chdir(cmdvector[1]) == -1){
59             fatal("main()");
60         }
61     }
```

(3) 구현 시 문제점

'cd \$HOME' 명령어를 입력했을 때 사용자의 홈 디렉토리로 이동해야 하지만 작동하지 않았다. 이는 프로그램의 홈 디렉토리가 따로 저장되지 않았기 때문인 것으로 보인다.

(4) 테스트 사항

```
myshell> pwd
/home/usr1
myshell> mkdir dirA
myshell> cd dirA
myshell> pwd
/home/usr1/dirA
myshell> |
```

2. exit() 처리

(1) 요구사항 정의

'exit' 명령을 구현한다.

(2) 구현 방법

cd 명령어를 구현할 때와 마찬가지로 exit 명령어가 들어온 경우 child 프로세스를 생성하지 않고 parent 프로세스에서 exit()을 실행하도록 하였다. 기존의 코드대로 execvp()을 이용하여 exit()을 실행하면 'No such file or directory' 오류가 발생한다. 따라서 child process가 생성되기 전에 명령어가 'exit'인지 확인하고, 만약 그럴 경우 exit() 함수를 이용하여 프로그램을 종료한다. 그리고 command line에서 exit status가 존재할 경우 exit(exit status)로 종료하고, 그렇지 않은 경우 exit(0)로 종료한다. 이를 위해서 makelist() 함수의 리턴 값인 command line의 단어 개수를 이용한다. 만약 command line의 단어 개수가 한 개라면 exit status가 없다는 의미이므로 exit(0)을 실행하고 단어의 개수의 두 개라면 exit status가 존재한다는 의미이므로 두 번째 단어를 exit()함수의 인자로 넣는다.

```
62      /* exit 명령어가 입력된 경우 */
63      else if(strcmp(cmdvector[0], "exit") == 0 ){
64          /* 인자 존재 여부에 따라 exit status 설정 */
65          if(numtokens == 1)
66              exit(0);
67          else
68              exit(atoi(cmdvector[1])); /* exit status 와 함께 종료 */
69      }
```

(3) 구현 시 문제점

구현할 때 문제점은 발견되지 않았다.

(4) 테스트 사항

```
myshell> exit
```

3. 백그라운드 실행

(1) 요구사항 정의

백그라운드 실행을 구현한다. 명령 뒤에 '&' 가 붙으면 백그라운드를 실행한다.

(2) 구현 방법

명령 마지막에 '&' 기호가 있다면 parent process는 child process를 생성하고 wait()을 하지 않는다. 그렇게 하면 parent process는 child process를 기다리지 않고 계속 수행되므로 다음 명령을 입력 받을 수 있으며, child process는 백그라운드에서 실행된다.

백그라운드 실행 여부를 알아낸 뒤에는 명령을 실행하기 전에 '&'를 제거해 주어야 한다. 만약 제거하지 않고 그대로 진행한다면 에러가 발생한다. 이를 위하여 '&'와 마지막 단어가 띄어쓰기로 구분되어 있는지 확인한다. 만약 띄어쓰기가 존재하지 않는다면 cmdvector[numtokens - 1][strlen(cmdvector[numtokens - 1]) - 1]를 지운다. 여기서 'numtokens'는 command line의 단어 개수이다. 그리고 띄어쓰기가 존재한다면 cmdvector[numtokens - 1]를 NULL로 바꾼다.

```

70      /* 백그라운드로 실행 */
71      else if(cmdvector[numtokens - 1][strlen(cmdvector[numtokens - 1]) - 1] == '&'){
72          /* 명령 수행을 위해 '&'기호 제거 */
73          if(strlen(cmdvector[numtokens - 1]) == 1)
74              cmdvector[numtokens - 1] = NULL;
75          else
76              cmdvector[numtokens - 1][strlen(cmdvector[numtokens - 1]) - 1] = '\0';
77
78          /* child process 생성 후 명령 실행 */
79          /* (parent process는 wait()을 호출하지 않는다) */
80          switch(pid=fork()){
81              case 0:
82                  execvp(cmdvector[0], cmdvector);
83                  fatal("main()");
84              case -1:
85                  fatal("main()");
86          }
87      }

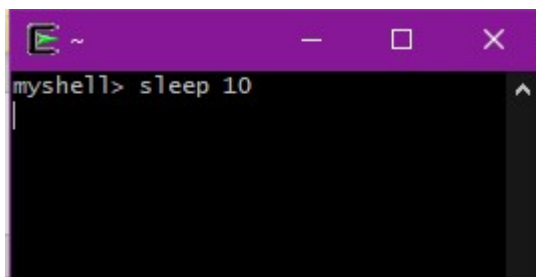
```

(3) 구현 시 문제점

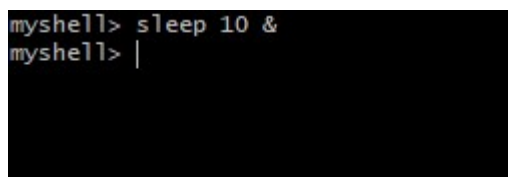
위와 같은 방법으로 백그라운드 프로세스를 실행할 경우, child process가 수행을 끝마친 후 zombie process가 된다.

(4) 테스트 사항

1)



포그라운드 이므로 셸 프롬프트가 바로 출력되지 않고 10초 뒤에 출력된다.



백그라운드 이므로 쉘 프롬프트가 바로 출력된다.

2)

```
myshell> id
uid=1000(boo) gid=1000(boo) groups=1000(boo),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120
(lpadmin),131(lxd),132(sambashare)
myshell> ps -u 1000 &
myshell>
  PID TTY          TIME CMD
 1263 ?            00:00:00 systemd
 1264 ?            00:00:00 (sd-pam)
 1272 ?            00:00:00 pulseaudio
 1273 ?            00:00:00 gnome-keyring-d
 1277 ?            00:00:00 tracker-miner-f
 1279 ?            00:00:00 dbus-daemon
 1281 ?            00:00:00 gvfsd
 1287 tty2        00:00:00 gdm-x-session
 1291 ?            00:00:00 gvfsd-fuse
 1304 ?            00:00:00 gvfs-udisks2-vo
 1308 tty2        00:00:03 Xorg
 1316 ?            00:00:00 gvfs-afc-volume
 1321 ?            00:00:00 gvfs-gphoto2-vo
 1325 ?            00:00:00 gvfs-goa-volume
 1332 ?            00:00:00 goa-daemon
```

```
 2270 ?            00:00:00 gnome-terminal-
 2280 pts/0        00:00:00 bash
 2316 ?            00:00:00 deja-dup-monito
 2324 pts/1        00:00:00 bash
 2356 pts/0        00:00:00 a.out
 2363 pts/0        00:00:00 ps
ps -u 1000&
myshell>
  PID TTY          TIME CMD
 1263 ?            00:00:00 systemd
 1264 ?            00:00:00 (sd-pam)
 1272 ?            00:00:00 pulseaudio
 1273 ?            00:00:00 gnome-keyring-d
 1277 ?            00:00:00 tracker-miner-f
```



```

2280 pts/0      00:00:00 bash
2316 ?          00:00:00 deja-dup-monito
2324 pts/1      00:00:00 bash
2356 pts/0      00:00:00 a.out
2363 pts/0      00:00:00 ps <defunct>
2364 pts/0      00:00:00 ps <defunct>
2365 pts/0      00:00:00 ps

```

```
ps -u 1000
```

```

myshell>      PID TTY          TIME CMD
1263 ?          00:00:00 systemd
1264 ?          00:00:00 (sd-pam)
1272 ?          00:00:00 pulseaudio
1273 ?          00:00:00 gnome-keyring-d
1277 ?          00:00:00 tracker-miner-f

```

```

2270 ?          00:00:01 gnome-terminal-
2280 pts/0      00:00:00 bash
2316 ?          00:00:00 deja-dup-monito
2324 pts/1      00:00:00 bash
2356 pts/0      00:00:00 a.out
2363 pts/0      00:00:00 ps <defunct>
2364 pts/0      00:00:00 ps

```

```
ps -u 1000 &
```

```

myshell>      PID TTY          TIME CMD
1263 ?          00:00:00 systemd
1264 ?          00:00:00 (sd-pam)
1272 ?          00:00:00 pulseaudio
1273 ?          00:00:00 gnome-keyring-d
1277 ?          00:00:00 tracker-miner-f
1279 ?          00:00:00 dbus-daemon

```

```

2270 ?          00:00:01 gnome-terminal-
2280 pts/0      00:00:00 bash
2316 ?          00:00:00 deja-dup-monito
2324 pts/1      00:00:00 bash
2356 pts/0      00:00:00 a.out
2363 pts/0      00:00:00 ps <defunct>
2364 pts/0      00:00:00 ps <defunct>
2365 pts/0      00:00:00 ps
ps -u 1000
myshell>      PID TTY          TIME CMD
1263 ?          00:00:00 systemd
1264 ?          00:00:00 (sd-pam)
1272 ?          00:00:00 pulseaudio
1273 ?          00:00:00 gnome-keyring-d
1277 ?          00:00:00 tracker-miner-f
1279 ?          00:00:00 dbus-daemon

```

```

2270 ?          00:00:01 gnome-terminal-
2280 pts/0      00:00:00 bash
2316 ?          00:00:00 deja-dup-monito
2324 pts/1      00:00:00 bash
2356 pts/0      00:00:00 a.out
2364 pts/0      00:00:00 ps <defunct>
2365 pts/0      00:00:00 ps <defunct>
2366 pts/0      00:00:00 ps

```

[고찰] 좀비 프로세스가 생기는 이유

parent process 가 child process 를 생성한 뒤에 wait()하지 않았기 때문에 child process 가 작업을 모두 수행하면 좀비 프로세스가 됩니다.

3)

```

myshell> sleep 10 &
myshell> sleep 20 &
myshell> ps
  PID TTY          TIME CMD
 2280 pts/0        00:00:00 bash
 2592 pts/0        00:00:00 a.out
 2594 pts/0        00:00:00 sleep
 2595 pts/0        00:00:00 sleep
 2596 pts/0        00:00:00 ps
myshell>

```

백그라운드를 기다리지 않고 바로 쉘 프롬프트가 출력된다.

[고찰] 이 테스트의 문제점과 문제점을 해결하기 위한 방법

sleep 명령을 실행하는 프로세스가 작업을 마친 뒤에 좀비 프로세스가 됩니다. 이를 해결하기 위해서는 parent process 가 wait() 함수를 통해 child process 의 수행을 기다리거나 좀비 프로세스를 제거해야 합니다.

4)

```

boo@boo-VirtualBox: ~
myshell> sleep 1000 &
myshell> sleep 10
myshell>

```

```

boo@boo-VirtualBox:~$ ps -ef | grep sleep
boo      2648      2644  0 16:38 pts/0    00:00:00 sleep 1000
boo      2649      2644  0 16:38 pts/0    00:00:00 sleep 10
boo      2651      2324  0 16:39 pts/1    00:00:00 grep --color=auto sleep
boo@boo-VirtualBox:~$

```

만든 쉘에서 sleep 을 실행시켰을 때, 또 다른 쉘에서 sleep 이 잘 실행되는 것을 확인 할 수 있습니다.

4. 시그널 처리

(1) 요구사항 정의

- SIGCHLD로 자식 프로세스 WAIT() 시 프로세스가 온전하게 수행되도록 구현
- ^C(SIGINT), ^W(SIGQUIT) 사용시 쉘이 종료되지 않도록, Foreground 프로세스 실행 시 SIGINT를 받으면 프로세스가 끝나는 것을 구현

(2) 구현 방법

백그라운드 프로세스를 실행시켰을 때 좀비 프로세스가 생성되는 이유는 해당 명령을 실행하는 child 프로세스가 작업을 마쳐도 부모 프로세스에서 wait()을 하지 않기 때문이다. 따라서 자식 프로세스가 작업을 끝내면 부모 프로세스는 wait()을 호출하여 자식 프로세스가 좀비 프로세스가 되는 것 막아야 한다.

자식 프로세스는 작업을 끝내면 SIGCHLD 시그널을 보내므로 부모 프로세스는 SIGCHLD를 받은 경우 wait()을 호출하면 자식 프로세스가 좀비 프로세스가 변하는 것을 막을 수 있다. 따라서 wait()을 실행하는 'sigchldhandler(int chld)'함수를 생성하고, 백그라운드 명령을 실행할 때 부모 프로세스에서 'signal(SIGCHLD, sigchldhandler)'를 실행하여 좀비 프로세스 생성을 막을 수 있다.

```
/* SIGCHLD 시그널 핸들러 */
void sigchldhandler(int signal){
    wait(NULL);
}

switch(pid=fork()){
    case 0:
        execvp(cmdvector[0], cmdvector);
        fatal("main()");
    case -1:
        fatal("main()");
    default:
        /* 자식 프로세스 종료 시 wait() 실행되도록 설정 */
        signal(SIGCHLD, sigchldhandler);
}
```

^C(SIGINT), ^W(SIGQUIT)를 입력 받았을 때 쉘이 종료되지 않도록 하기 위해서는 해당 시그널에 대해 signal handler를 추가해 줘야한다. 따라서 현재 pid가 0인 경우에만 exit()을 실행하고

pid가 0이 아닌 경우에는 아무것도 하지 않는 'sighandler(int signal)' 함수를 생성한다. 그리고 main 함수 시작 부분에 'signal(SIGINT, sighandler)', 'signal(SIGQUIT, sighandler)'을 실행시켜 시그널 핸들러를 등록한다. 이렇게 하면 자식 프로세스는 시그널을 받았을 때 종료되지만, 부모 프로세스는 시그널을 받아도 종료되지 않으므로 셸이 유지된다.

```
/* 시그널 핸들러 */
void sighandler(int signal){
    /* 자식 프로세스인 경우에만 exit */
    if(pid == 0){
        exit(0);
    }
}

int main(int argc, char**argv){
    int i=0;

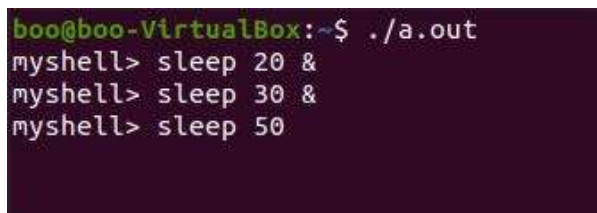
    /* SIGINT, SIGQUIT에 셸 종료되지 않게 설정 */
    signal(SIGINT, sighandler);
    signal(SIGQUIT, sighandler);
}
```

(3) 구현 시 문제점

구현할 때 별다른 문제점은 발견되지 않았다.

(4) 테스트 사항

1)



```
boo@boo-VirtualBox:~$ ./a.out
mysHELL> sleep 20 &
mysHELL> sleep 30 &
mysHELL> sleep 50
```

"sleep 50"이 "sleep 20 &"과 "sleep 30 &"에 방해받지 않고 수행되는 것을 확인할 수 있다.

```
boo@boo-VirtualBox:~$ ./a.out
mysHELL> sleep 20 &
mysHELL> sleep 30 &
mysHELL> sleep 50
mysHELL> ps -u 12345
      PID TTY          TIME CMD
mysHELL> █
```

좀비 프로세스가 없음이 확인된다.

2)

```
mysHELL> ^C
mysHELL>
```

셸이 종료되지 않고 무시되는 것을 확인할 수 있다.

3)

```
mysHELL> sleep 30
^CmysHELL>
```

제어키(^C, ^W)을 받아 포그라운드 프로세스 sleep이 죽는 것을 확인할 수 있다.

5. Pipe 처리

(1) 요구사항 정의

리디렉션과 파이프를 구현한다.

(2) 구현 방법

1) in redirection

```
/* child process 생성 후 명령 실행 */
/* (parent process는 wait()으로 child process 수행을 기다린다.) */
switch(pid=fork()){
case 0:
    /* redirection 있는지 확인 */
    if(strchr(cmdline2, '<')){
        redirection_in(cmdvector);
    }
    if(strchr(cmdline2, '>')){
        redirection_out(cmdvector);
    }
}
```

먼저 main 함수에서 fork()를 실행한 뒤에 자식 프로세스에서 '<'가 있는지 확인한다. 만약 '<'가 존재한다면 in redirection을 수행하는 함수를 실행한다.

```

/* redirection in 함수 */
int redirection_in(char **cmdvector){
    int idx = 0, fd = 0;
    /* cmdvector에서 "<" 위치 찾기 */
    while(cmdvector[idx] != NULL){
        if(strcmp(cmdvector[idx], "<") == 0)
            break;
        idx++;
    }

    if(idx == 0 || !cmdvector[idx + 1])
        return -1;
    /* redirection */
    if((fd = open(cmdvector[idx+1], O_RDONLY)) == -1){
        fatal("(redirection_in)");
        return -1;
    }
    dup2(fd, 0);
    close(fd);
    /* cmdvector에서 '<' 이후 내용 정리 */
    while(cmdvector[idx] != NULL){
        cmdvector[idx] = cmdvector[idx + 2];
        idx++;
    }
    cmdvector[idx] = NULL;

    return 0;
}

```

in redirection을 수행하는 함수에서는 cmdvector에서 '<' 기호 이후의 내용을 찾은 뒤, dup2()를 이용하여 리디렉션 기능을 구현하였다.

2) out redirection

out redirection도 in redirection과 마찬가지로 main 함수에서 fork()를 실행한 뒤에 자식 프로세스에서 '>'가 있는지 확인한다. 만약 '>'가 존재한다면 out redirection을 수행하는 함수를 실행한

다.

```
/* redirection out 함수 */
int redirection_out(char **cmdvector){
    int idx = 0;
    int fd = 0;
    /* cmdvector에서 ">" 위치 찾기 */
    while(cmdvector[idx] != NULL){
        if(strcmp(cmdvector[idx], ">") == 0){
            break;
        }
        idx++;
    }

    if(idx == 0 || !cmdvector[idx + 1]){
        return -1;
    }
    /* redirection */
    if((fd = open(cmdvector[idx+1], O_RDWR|O_CREAT, 0644)) == -1){
        fatal("(redirection_out)");
        return -1;
    }
    dup2(fd, 1);
    close(fd);
    /* cmdvector에서 '>' 이후 내용 정리 */
    while(cmdvector[idx] != NULL){
        cmdvector[idx] = cmdvector[idx + 2];
        idx++;
    }
    cmdvector[idx] = NULL;

    return 0;
}
```

out redirection을 수행하는 함수에서는 cmdvector에서 '>' 기호 이후의 내용을 찾은 뒤, dup2()를 이용하여 리디렉션 기능을 구현하였다.

3) pipe

```

/* 파이프 있을 때 */
else if(strchr(cmdline2, '|')){
    mypipe(cmdvector);
}

```

먼저 main 함수에서 파이프 명령어가 존재하는지 확인한 뒤, 존재할 경우 mypipe 함수를 수행한다.

```

/* 파이프 수행 함수 */
int mypipe(char **cmdvector){
    int pipe_cnt = 0;
    int pipe_idx[4];
    /* 첫 번째 파이프 위치 확인 */
    for(int i = 0; cmdvector[i] != NULL; i++){
        if(strcmp(cmdvector[i], "|") == 0){
            pipe_cnt++;
            pipe_idx[pipe_cnt] = i;
        }
    }

    if(pipe_cnt == 0){
        return -1;
    }
    /* cmdvector의 첫 번째 파이프 위치 값을 NULL로 변경 */
    cmdvector[pipe_idx[1]] = NULL;

    /* cmdvector2에 첫 번째 파이프 이후 내용을 옮긴다 */
    int i = pipe_idx[1] + 1;
    int cur = 0;
    for(i = i; cmdvector[i] != NULL && strcmp(cmdvector[i], "|") != 0; i++){
        cmdvector2[cur] = cmdvector[i];
        cur++;
    }
    cmdvector2[cur] == NULL;

    if(pipe_cnt == 2){
        /* cmdvector2의 파이프 이후 내용 cmdvector3에 옮긴다 */
        i = pipe_idx[2] + 1;
        cur = 0;
        for(; cmdvector[i] != NULL; i++){
            cmdvector3[cur] = cmdvector[i];
            cur++;
        }
        cmdvector3[cur] == NULL;
    }
}

```

mypipe 함수에서는 우선 파이프의 개수를 확인하고 cmdvector의 '|' 이전 내용과 '|' 이후 내용을

각각 다른 배열로 분리시킨다.

```
/* parent process */
switch(pid=fork()){
case 0:
    break;
case -1:
    fatal("(pipe first fork) ");
default:
    wait(&status);
    if(pipe_cnt < 2){
        return(status);
    }
}
if(pipe(p) == -1){
    fatal("(pipe call) ");
}
/* child process */
switch(pid=fork()){
case 0:
    /* redirection 있는지 확인 */
    if(strchr(cmdline2, '<')){
        redirection_in(cmdvector);
    }
    dup2 (p[1], 1);
    close(p[0]);
    close(p[1]);
    execvp(cmdvector[0], cmdvector);
    fatal("(pipe child) ");
case -1:
    fatal("(pipe second fork) ");
default:
    if(pipe_cnt == 1 && strchr(cmdline2, '>')strchr(cmdline2, '>')){
        redirection_out(cmdvector);
    }
    dup2 (p[0], 0);
    close(p[0]);
    close(p[1]);
    execvp(cmdvector2[0], cmdvector2);
    fatal("(pipe parent) ");
}
```

그 다음 fork 함수를 수행한다. fork 함수는 두 번 수행하며 첫 번째 fork 함수의 자식 프로세스는 두 번째 fork 함수의 부모 프로세스가 되게 한다. 첫 번째 fork 함수를 수행한 뒤에는 pipe를 실행한다. 그 뒤, 두 번째 fork 함수를 실행하고 dup2 함수를 이용하여 부모 프로세스와 자식프로세스가 서로 데이터를 주고받을 수 있도록 한다.

```

/* 파이프가 두 개일 때 */
if(pipe_cnt == 2){
    if(pipe(p) == -1){
        fatal("(pipe call) ");
    }

    switch(pid=fork()){
    case 0:
        dup2 (p[1], 1);
        close(p[0]);
        close(p[1]);
        fatal("(pipe child) ");
    case -1:
        fatal("(pipe second fork) ");
    default:
        /* redirection 있는지 확인 */
        if(strchr(cmdline2, '>')){
            redirection_out(cmdvector);
        }
        dup2 (p[0], 0);
        close(p[0]);
        close(p[1]);
        execvp(cmdvector3[0], cmdvector3);
        fatal("(pipe parent) ");
    }
}
return 0;
}

```

파이프 기호가 두 개 일 때에는 파이프와 fork 함수를 한 번 더 수행한다.

(3) 구현 시 문제점

파이프를 두 개 이상 사용할 경우 기능이 제대로 수행되지 않고 셸이 종료된다. 프로세스 간 데이터 전송을 주고받을 때 오류가 생긴 것으로 보인다.

(4) 테스트 사항

1) 리디렉션1

```
myshell> cat > test.txt
hello!
myshell> cat < test.txt
hello!
myshell> cat < test.txt > test1.txt

myshell> cat < test1.txt
hello!
myshell> |
```

리디렉션이 정상적으로 수행됨을 확인할 수 있다.

2) 리디렉션2

```
myshell> ls -l > ls.txt
myshell> cat ls.txt
합 계 841
-rwxr-xr-x 1 booro booro 168191 11월 29 12:34 a.exe
-rwxr-xr-x 1 booro booro 966 11월 29 10:52 a.exe.stackdump
drwxr-xr-x+ 1 booro booro 0 10월 21 19:37 dir
-rw-r--r-- 1 booro booro 52 11월 29 11:55 grep
-rwxr-xr-x 1 booro booro 128 10월 23 16:55 hello.c
-rwxr-xr-x 1 booro booro 0 11월 29 12:40 ls.txt
-rwxr-xr-x 1 booro booro 1214 10월 21 18:31 orig.c
-rwxr-xr-x 1 booro booro 162316 10월 23 17:39 orig.exe
-rwxr-xr-x 1 booro booro 1092 11월 27 20:23 pipe.c
-rwxr-xr-x 1 booro booro 3860 11월 21 21:19 pipe_shell.c
-rwxr-xr-x 1 booro booro 3860 11월 21 21:19 shell.c
-rwxr-xr-x 1 booro booro 7130 11월 29 12:34 simple_myspell.c
-rwxr-xr-x 1 booro booro 430 10월 25 19:11 strotok.c
-rwxr-xr-x 1 booro booro 162316 10월 21 18:26 test.c
-rwxr-xr-x 1 booro booro 159957 10월 23 16:56 test.exe
-rw-r--r-- 1 booro booro 7 11월 29 12:37 test.txt
-rw-r--r-- 1 booro booro 7 11월 29 12:38 test1.txt
-rwxr-xr-x 1 booro booro 159960 10월 25 19:12 tok.exe
myspell> .....
```

리디렉션이 정상적으로 수행됨을 확인할 수 있다.

3) 파이프1 – 파이프 한 개가 있는 명령어

```
myshell> ls -l | grep ^d
drwxr-xr-x+ 1 booro booro      0 10월  21 19:37 dir
myshell> |
```

파이프를 통해 'ls -l' 명령 결과 중 디렉토리만 출력시킨다.

4) 파이프2 - 파이프가 두 개 있는 명령어

```
myshell> cat < ls.txt | grep ^d | wc -l > dir_num.txt
(pipe child) : No such file or directory
(pipe child) : No such file or directory

booro@DESKTOP-C85VBT1 ~
$ |
```

오류 메시지와 함께 셸이 종료된다.

5) 백그라운드로 리디렉션 수행

```
$ ./a
myshell> ls -l > ls.txt&
myshell> |
```

셸에 위와 같이 명령어를 입력한다.

```
ls.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
합계 840
-rwxr-xr-x 1 booro booro 168191 11월 29 12:23 a.exe
-rwxr-xr-x 1 booro booro 966 11월 29 10:52 a.exe.stackdump
drwxr-xr-x+ 1 booro booro 0 10월 21 19:37 dir
-rw-r--r-- 1 booro booro 52 11월 29 11:55 grep
-rwxr-xr-x 1 booro booro 128 10월 23 16:55 hello.c
-rwxr-xr-x 1 booro booro 0 11월 29 12:23 ls.txt
-rwxr-xr-x 1 booro booro 1214 10월 21 18:31 orig.c
-rwxr-xr-x 1 booro booro 162316 10월 23 17:39 orig.exe
-rwxr-xr-x 1 booro booro 1092 11월 27 20:23 pipe.c
-rwxr-xr-x 1 booro booro 3860 11월 21 21:19 pipe_shell.c
-rwxr-xr-x 1 booro booro 3860 11월 21 21:19 shell.c
-rwxr-xr-x 1 booro booro 7142 11월 29 12:20 simple_myshell.c
-rwxr-xr-x 1 booro booro 430 10월 25 19:11 strotok.c
-rwxr-xr-x 1 booro booro 162316 10월 21 18:26 test.c
-rwxr-xr-x 1 booro booro 159957 10월 23 16:56 test.exe
-rw-r--r-- 1 booro booro 47 11월 26 13:51 test.txt
-rwxr-xr-x 1 booro booro 159960 10월 25 19:12 tok.exe
```

'ls.txt'에 위와 같이 입력된 것을 확인할 수 있다.

6. Psuedo Code

```
function sighandler(int signal):
```

```
    if pid = 0:
```

```
        exit
```

```
function sigchldhandler(int signal):
```

```
    wait()
```

```
function redirection_in(char **cmdvector):
```

```

idx = location of "<" in cmdvector

fd = open(cmdvector[idx + 1], O_RDONLY)

dup2(fd, 0)

close(fd)

remove cmdvector contents after idx

```

```

function redirection_out(char **cmdvector):

```

```

    idx = location of ">" in cmdvector

    fd = open(cmdvector[idx + 1], O_RDWR|O_CREATE, 0644)

    dup2(fd, 1)

    close(fd)

    remove cmdvector contents after idx

```

```

function mypipe(char **cmdvector):

```

```

    pipe_cnt = number of '|' in command

    cmdvector = command list before first "|" in command

    cmdvector2 = command list between first '|' and second '|' in command

    cmdvector3 = command list after second '|' in command

    pid = fork

    switch pid :

        case 0:

            break

        default:

            wait()

```



```

        if pipe_cnt < 2: return

    pipe(p)

    pid = fork()

    switch pid:

        case 0:

            dup2(p[1],1)

            execvp(cmdvector[0], cmdvector)

        default:

            dup2(p[0], 0)

            execvp(cmdvector2[0], cmdvector2)

    if pipe_cnt == 2:

        pipe(p)

        pid = fork()

        switch pid:

            case 0:

                dup2(p[1], 1)

            default:

                dup2(p[0], 0)

                execvp(cmdvector3[0], cmdvector3)

while True:

    signal(SIGINT, sighandler)

    signal(SIGQUIT, sighandler)

```

print prompt

Read cmdline

```
numtokens <- makelist(cmdline, " %t", cmdvector, MAX_CMD_ARG);
```

```
if numtokens = 0:
```

```
    continue
```

```
else if cmdvector[0] = "cd":
```

```
    chdir(cmdvector[1])
```

```
else if cmdvector[0] = "exit":
```

```
    exit(cmdvector[1])
```

```
else if '|' in cmdvector:
```

```
    mypipe(cmdvector)
```

```
else if cmdvector[-1][-1] = '&':
```

```
    cmdvector[-1][-1] <- NULL
```

```
    pid <- fork()
```

```
    if pid = 0:
```

```
        if '<' in command:
```

```
            redirection_in(cmdvector)
```

```
        if '>' in command:
```

```
            redirection_out(cmdvector)
```

```
        execvp(cmdvector[0], cmdvector)
```

```
    else if pid > 0:
```

```
        sighandler(SIGCHLD, sigchldhandler)
```

```
else:

    pid <- fork()

    if pid == 0:

        if '<' in command:

            redirection_in(cmdvector)

        if '>' in command:

            redirection_out(cmdvector)

        execvp(cmdvector[0], cmdvector)

    else if pid > 0:

        wait(NULL)
```