**System Programming Project 1**

담당 교수 : 김영재

이름 : 오상훈

학번 : 20181654

1. **개발 목표**

본 프로젝트에서는 리눅스 시스템에서 실행되는 기본 프로그램인 shell 프로그램을 작성한다. 기존 cspro 서버에서 작동하는 bash shell 과 같은 프로그램을 작성하고 context switch, signal 등의 개념을 이해한다. 프로젝트는 3개의 세부 단계로 진행한다.

첫 번째 단계에서는 basic internal shell command 들이 실행될 수 있도록 한다. child process를 생성하고 child process 가 종료되었을 때 메모리 누수가 일어나지 않도록 관리한다.

두 번째 단계에서는 Inter process communication 의 방법 중 하나인 pipe를 구현한다. 두 개 이상의 child process들이 서로 입력을 전달 받고, 출력을 전달하는 단방향 통신을 이해한다.

세 번째 단계에서는 이전 단계에서 실행되었던 프로그램들이 모두 foreground process였던 것과 달리, background process로 프로그램을 실행할 수 있도록 한다. Terminal controlling process를 이해한다. 또한 ctrl z 시그널을 이해하고, background process 및 stopped process를 관리한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Phase 1에서는 basic internal shell command를 수행하는 shell 프로그램을 작성한다. 또한 builtin command 중 cd(change directory), exit 명령어가 수행할 수 있도록 한다. 아래 사진은 몇 가지 basic internal shell command와 cd명령어, exit명령어 수행 결과를 보여준다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 1>

또한 ctrl c가 입력되었을 때 발생하는 SIGINT signal을 처리할 수 있도록 sigint handler를 작성한다.

1. Phase 2

Phase 2에서는 pipeline 처리를 수행한다. Pipeline 입력이 들어오지 않든 몇 개가 들어오든 정상적인 pipeline처리가 수행되어야 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 2>

1. Phase 3

Phase3 에서는 프로세스가 background로 실행될 수 있도록 해야한다. 또한, background process 또는 stopped process에 대한 job control을 수행해야한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 3>

* 1. **개발 내용Phase1 (fork & signal)**

Parent process에서 fork를 이용해 child process를 생성한다. child process에서는 /bin/ 폴더에 있는 명렁어를 execve함수를 통해 실행한다. 만약 이 명령어가 실패한다면 /usr/bin/ 폴더에 있는 명령어를 execve함수를 통해 실행한다. 다시 실패한다면 Command not found 문구를 출력하고 child process를 종료한다.

Parent process에서는 child process가 종료되거나 stopped 상태가 될 때 생기는 SIGCHLD signal handler 함수를 등록한다. child process가 종료될 때까지 기다린 후, sigchld signal을 확인하면 sigchld handler 함수에서 waitpid 함수를 호출해 child process가 zombie process가 되지 않도록 reaping해준다. 또한 sigchild handler 함수에서는 SIGINT, SIGTSTP signal이 발생했을 때 ‘\n’을 출력해주어, 다음 shell command를 정상적으로 입력받을 수 있도록 한다.

**Phase2 (pipelining)**

두 child process간 communication 을 위해 pipe를 생성한다. 파이프는 pipe()함수를 통해 file descriptor를 할당 받는데, 주요 알고리즘을 수행하는 eval\_with\_pipe함수에서는 최대 2개의 pipe 를 생성한다. File descriptor 배열을 2차원 배열로 만들고, 처음에는 fd[0]에 pipe함수를 통해 file descriptor를 할당받는다. While 문의 조건으로 입력받은 문자열에서 ‘|’ 문자를 검색하여, ‘|’ 문자가 없을 때까지 반복문을 수행한다. 입력 받은 command가 A | B | C | … | Z 라면 while 반복문에서 A, B, … , Y 명령어까지 수행한다. 마지막 Z명령어는 while 반복문 밖에서 수행하는데 while 반복문 내의 내용과 거의 유사하다. Pipeline 없이 명령어가 입력되었다면 바로 마지막 명령어 Z를 수행한다. Pipe\_idx 변수는 0 또는 1 값을 갖는데, 이 값에 따라서 어떤 file descriptor에 stdin과 stdout을 redirection 할지 결정한다. 모든 file descriptor는 close하여 EOF를 전달하고, memory leak이 발생하지 않도록 한다.

**Phase3 (background process)**

먼저 phase3에서는 background process가 가능하도록 하기 위해 명령어에 &가 있는지 확인하고 있다면 bg를 1로 설정한다. bg가 1이라면 A | B | … | Z 구조의 명령어에서 마지막 명령어 Z를 수행할 때, wait를 하지 않도록하여 background 로 프로세스를 수행시킨다. 이때, jobs control을 위해 이 job을 job list에 추가한다. Sigchld handler에서는 child process가 정상적으로 종료하였을 때, child process가 signal을 받아서 비정상적으로 종료하였을 때, stop상태일 때에 대해 각각 다른 job control을 수행한다.

* 1. **개발 방법**
* Phase1
  + 1. parse\_input(char\* buf, char\*\* argv)

parse\_input 함수는 하나의 명렁어 단위를 buf로 입력 받아 argv에 저장하는 함수이다. ls -al 같은 명령어를 ‘ls’, ‘-al’ 과 같이 명령어 및 옵션 단위로 parsing하여 argv배열에 저장한다. argv의 마지막 원소 뒤에는 NULL을 저장한다. 이때, echo 또는 grep과 같은 명령어에서 echo “hello world”와 같이 공백을 포함하는 인자를 입력 받기 위해 ‘, “ 존재 여부를 찾고 이에 대한 처리를 수행한다.

* + 1. Chdir(char\* argv)

chdir에서는 parsing된 입력에 대해 여러가지 조건들에 따라 다른 처리를 한다. cd, cd ~, cd ~/ 과 같은 문자열이 입력되었을 때 home directory로 이동하도록 추가 구현하였다. stat() system call 함수를 사용하여 파일 또는 directory의 정보를 읽고 해당 파일 또는 디렉토리가 존재하는지, 디렉토리인지, 접근할 수 있는지 등을 검사한 후, chdir()함수를 통해 현재 디렉토리를 변경한다.

* + 1. eval(char\* cmdline)

eval 함수에서는 입력받은 문자열을 parse\_input함수를 호출하여 execve 인자에 전달할 수 있는 형태로 바꾼다. 해당 입력이 builtin command가 아니라면 다음과 같이 fork 직전에 SIGHLD signal을 block 하고 SIGINT, SIGTSTP signal을 ignore하도록 signal handler를 등록한 후, fork 이후 child process에서는 Ctrl+c, Ctrl +z 를 입력받을 수 있도록 다시 SIGINT, SIGTSTP signal에 대해서 SIG\_DFL 함수로 바꾸어 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 4>

이후 child process는 2.B에서 설명한 것과 같이 execve함수를 통해 프로그램을 실행하고, parent process는 volatile sig\_atomic\_t 로 선언된 sig\_pid 값이 1이 될 때까지 기다린다.

텍스트, 장치, 측정기, 게이지이(가) 표시된 사진

자동 생성된 설명<그림 5>

Sigchld signal이 전달되고 child process를 reaping 한 후에 다시 parent process에서는 다음과 같이 SIGCHLD, SIGINT, SIGTSTP를 입력받을 수 있도록 signal 함수를 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 6>

* + 1. sigint\_handler

sigint\_handler는 shell 입력에서 Ctrl+c가 입력되더라도 myshell 프로그램이 종료되지 않도록 main 으로 siglongjmp함수를 통해 nonlocaljmp를 수행한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 7>

* + 1. sigchld\_handler

sigchld\_handler함수는 errno 전역변수를 저장해두었다가 마지막에 복원한다. 반복문에서 waitpid함수의 첫 번째 인자로 -1을 전달하여 임의의 자식프로세스를 기다리고, 세 번째 인자로 WNOHANG | WUNTRACED를 전달한다. SIGINT 또는 SIGTSTP 에 의해 waitpid가 값을 return 했을 때 ‘\n’을 출력한다.

* Phase2

1. eval\_with\_pipe(char\* cmdline)

eval\_with\_pipe함수는 크게 2 부분으로 나눌 수 있다. 위의 B에서 설명한 것과 같이 A, B, … , Y를 수행하고 pipeline을 처리하는 while문과 마지막 명령어를 수행하는 Z부분 while 문 이후에 있다. 이 함수에서 사용되는 변수 pipe\_idx는 어떤 file descriptor를 redirection 할지 결정하고, nopipe는 pipeline이 하나라도 존재한다면 0이고, 아니라면 1을 갖는다.

While문 전에 다음과 같이 fd[0]에 pipe 함수를 사용하여 두 개의 file descriptor를 할당 받고, pipe\_idx를 1로 설정한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 8>

이후 while문에서 ‘|’ 문자의 위치를 찾고 입력 받은 명령어 하나에 대해서 phase1의 함수와 동일한 parse\_input함수를 호출한다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명<그림 9>

이후 다음과 같이 pipe\_idx에 따라서 다음 pipe에 사용할 file descriptor를 생성한다.

텍스트, 장치, 게이지, 측정기이(가) 표시된 사진

자동 생성된 설명<그림 10>

이후 fork를 하고 child process에서 다음과 같이 pipe\_idx가 0이라면 첫 번째 pipe에 출력을 하고, 두 번째 pipe에서 입력받는다. Pipe\_idx가 1이라면 두 번째 pipe에 출력을 하고, 첫 번째 pipe에서 입력받는다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 11>

이후 phase1과 동일하게 execve함수를 호출한 후 child process는 종료된다.

Parent process는 또한 동일하게 SIGCHLD signal을 기다리고 child process를 reaping 한다.

While 반복문 조건에서 ‘|’ 문자를 찾을 수 없을 때, 즉 마지막 명령어를 수행할 때, nopipe가 1이라면(pipeline개수가 0이라면) 두 번째 pipe를 생성한다. 마지막 명령어 처리는 <그림 11> 부분을 제외하고는 while 문 내의 알고리즘과 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 12>

Pipe\_idx가 0이라면 두 번째 pipe에서 입력을 받는다. Pipe\_idx가 1이라면 첫 번째 pipe에서 입력을 받는다. 출력은 shell에서 출력해야하므로 redirection 하지 않는다.

* Phase3

1. Job control 구조체

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 13>

job number, process id, command, state를 필드로 가지고 있는 BGProcess 구조체를 선언하여 하나의 job에 대한 상태를 가지고 있도록 한다. jobs list를 만든다. Jobs\_cnt는 현재 jobs 리스트에 저장되어 있는 job 개수이고, jobs max는 현재 jobs 리스트에 있는 job num 의 최대 값이다.

1. eval(char\* cmdline)

main 함수에서 command를 입력받은 후 호출하는 함수로, cmdline에 ‘&’ 문자가 끝에 있는지 확인한다. ‘&’ 문자가 있다면 bg를 1로 설정하고 없다면 bg를 0으로 설정한다. 이후 eval\_with\_pipe함수를 호출한다.

1. eval\_with\_pipe

phase 2에서 다음 부분을 제외하고는 동일한 flow로 진행된다.

<그림 5>(phase1 그림이나 phase2에서도 동일한 reaping 처리를 한다)에서 다음과 같이 변경되었다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 14>

bg가 0이라면 기존과 동일하게 Sigsuspend 함수를 호출하여 child process를 reaping할 때까지 기다린다. Bg가 1이라면 add\_jobs함수를 호출하여 job을 job 리스트에 append하고 blocked 상태 없이 shell 입력 문구를 출력한다.

1. process\_done

process\_done 함수는 child process가 종료된 이후에 호출되는 함수로 pid를 입력받아 job 리스트에서 찾은 후 동적할당한 메모리를 해제해준다.

1. add\_jobs

add\_jobs에서는 malloc함수를 통해 새로운 job을 동적할당하여, 각 필드에 정보를 저장한다.

1. find\_job

find\_job 함수는 pid를 입력받아 job 리스트에서 찾고 job number를 반환한다.

1. print\_job

print\_job함수는 job 리스트에 있는 job들을 출력한다.

1. exit\_all\_process

exit\_all\_process함수는 남아있는 모든 child process들에게 SIGKILL signal을 보낸다.

1. builin\_command

builtin\_command 함수에서 입력으로 exit이 입력되었을 때 처리가 수정되었고, jobs, bg, fg, kill이 입력되었을 때 처리가 추가되었다.

먼저 exit이 입력되었을 때 exit\_all\_process함수를 호출하여 모든 남아있는 child process에게 SIGKILL signal을 전송하고 현재 parent process에서는 reaping한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 15>

jobs가 입력되었을 때는 print\_jobs함수를 호출하여 job리스트를 출력한다.

Kill이 입력되었을 때는 job number에 해당하는 pid를 찾아 SIGKILL signal을 전송한다.

bg가 입력되었을 때는 입력받은 job number에 해당하는 프로세스의 state를 BACKGROUND로 변경한 후, SIGCONT signal을 전송하여 background process가 되도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 16>

fg가 입력되었을 때는 입력받은 job number에 해당하는 child process가 stop상태라면 SIGCONT signal을 전송하고, 프로세스의 state를 BACKGROUND로 변경한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 17>

이후 현재 terminal의 속성을 저장한 후, 현재 terminal에 대해 속성을 변경한다. TCSADRAIN을 두 번째 인자로 전달하여 모든 출력 이후에 변경하도록 한다. 이후 tcsetpgrp을 호출하여 terminal controlling process를 해당 child process로 변경하고, 부모 process는 child process가 끝날 때까지 wait한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 18>

1. sigchld\_handler

sigchld\_handler에서 변경된 부분은 두 가지이다. 먼저 child process가 정상적으로 종료되었든, 비정상적으로 종료되었든 종료된 child process가 background인지 확인한다. 이때 foreground에서 child process가 실행 중일 가능성이 있으므로 continue 문을 사용하여 다시 waitpid를 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<그림 19>

두 번째로는 SIGCHLD signal이 child process가 stop 상태가 되어 전송되었을 때이다. 만약 child process가 기존에 job 리스트에 있었다면 프로세스의 stat를 변경한다. 만약 job 리스트에 없다면 job 리스트에 추가한다.

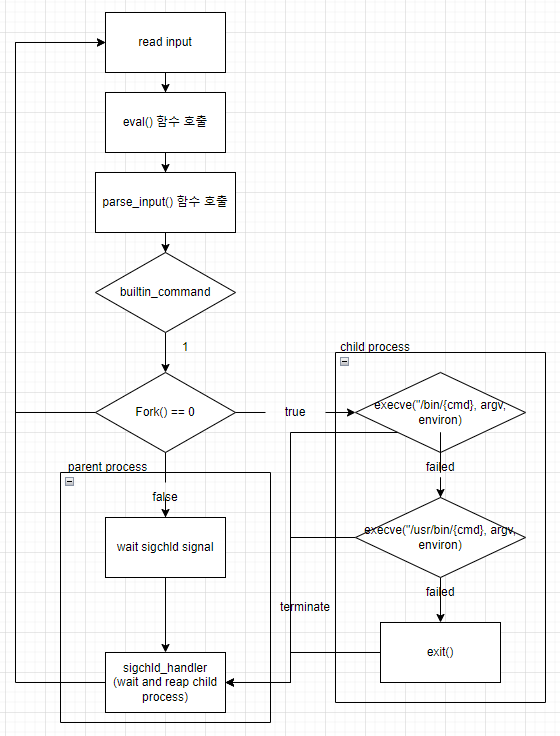
텍스트이(가) 표시된 사진

자동 생성된 설명<그림 20>

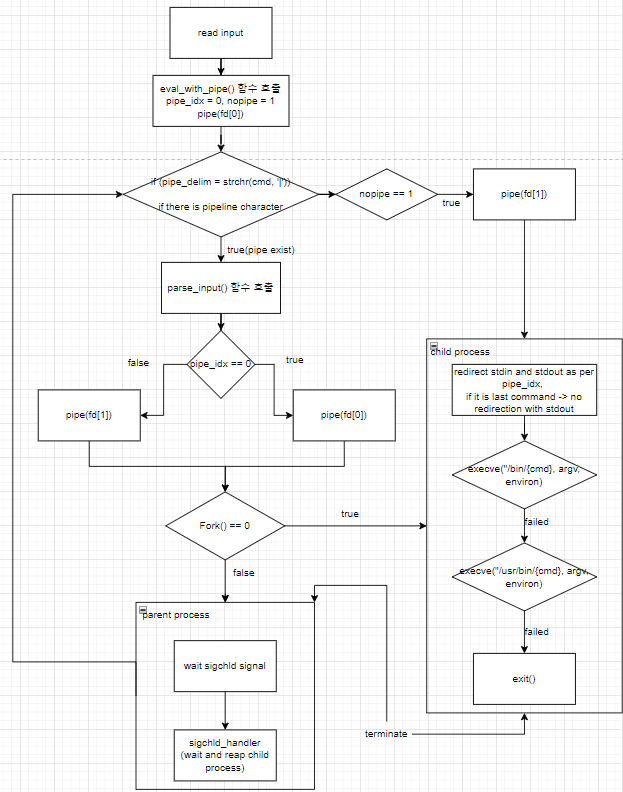
1. **구현 결과**
   1. **Flow Chart**

* **2.B.개발 내용에 대한 Flow Chart를 작성.**
* **(각각의 방법들에서 추가된 내용(fork, pipeline, background)만 특성이 잘 드러나게 그리면 됨.)**

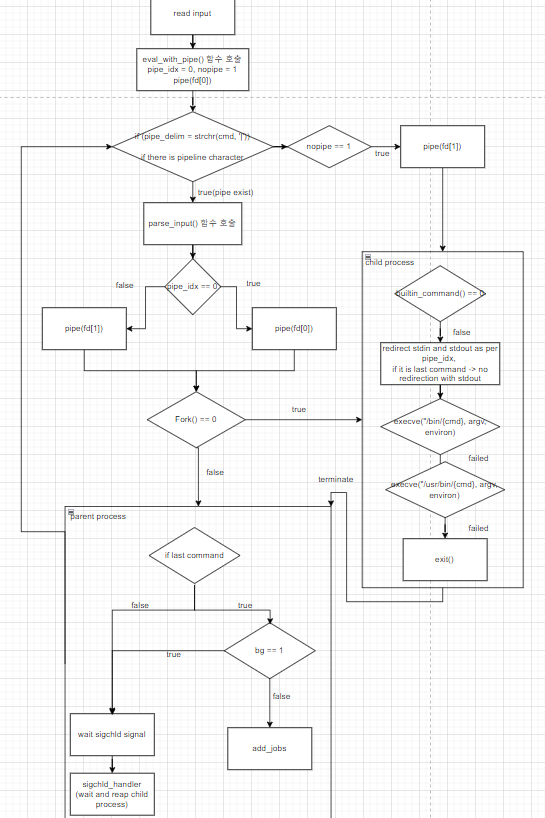
1. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

****