**System Programming Project 1**

담당 교수 :

이름 :byeon sung jin

학번 :20180546

1. **개발 목표**

* **동일한 데이터를 가지는 자식 프로세스를 추가적으로 형성하는 fork, 프로세스들 간에 연결을 제공하는 signal, 파일 디스크립터 간에 연결해주는 pipe라는 시스템 콜을 통해 커널과 운영체제 위에서 유저에게 서비스를 제공하는 shell을 구현했다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

phase1 에서는 파이프가 없는 단일 명령어를 구현하였다. ls,cat,clear,mkdir,less,grep 과 같은 명령어는 우분투 20 기준 /bin 내부에 실행파일이 있어 경로를 파싱하여 execve해주면 기존의 bash shell 과 같이 동일한 명령어가 출력되었다

다만 cd, exit, jobs,bg,fg은 직접 함수를 만들어 구현하여야만 했고, 최대한 bash shell의 동작과 일치하도록 구현하였다. 현재 jobs에는 정상작동하지만, kill 명령어 직후 실행할 시 깨진 문자가 출력되는 버그가 있다. kill은  pid를 argument로 받는 /bin/kill 다르게 job number를 인자로 받는 형태였다. kill 은 별도의 출력은 없고 job pgid 내부의 모든 프로세스들을 terminate한다

1. Phase 2

phase 2는 명령어의 output이 다른 명령어의 Input으로 사용되는 파이프를 구현하였다. 한 프로세스마다 최대 2개의 파이프를 사용하며, 이전의 명령어가 존재하면 이전 명령어의 파이프로부터 입력을 받고, 다음 명령어가 존재하면 다음 명령어의 파이프에 출력을 써서 마지막 명령어에서 해당 결과를 출력한다

1. Phase 3

백그라운드 작업이 끝나면 reap\_dead\_jobs과 print\_done\_jobs를 통해 다음 명령어 출력시 [1] Done sleep 30과 같은 명령어를 출력한다`

jobs : 현재 Suspended, Running 상태인 Background 명령어를 출력한다 [1] Suspended sleep 30

bg : suspended background job을 다시 background에서 돌게하는 명령어다.

fg : background job을 foreground에서 돌게하는데, 이때 parentpid는 자식이 종료할때까지 멈춘다

* 1. **개발 내용**
* **아래 항목의 내용만 서술**
* **(기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)**
* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow
* parent-child-son 3가지 계층의 프로세스가 일어난다. son은 명령어를 직접 실행한다(execve 나 함수형 명령어).
* parent는 아래 약술될 child를 fork 한뒤 foreground process 일때만 suspend상태로 대기한다.
* child는 son들을 모두 fork 한뒤 suspend 상태에서, 모든 명령어가 실행완료될때까지 loop를 도는 void sigchild\_handler\_child(int sig)에서 자식들을 정리한뒤 , child 본인에게 SIGCONT 를 보내 suspend상태에서 벗어난 후 종료한다.
* parent process는 sigchild 신호를 받게되면, phase 3에서 서술하겠지만 종료된 child pgid가 foreground 그룹일때만 parent 본인에게 SIGCONT를 보낸다(foreground child의 pgid는 pid\_t child\_pid라는 전역 변수에 저장되어있다). 종료된 process가 background면 job\_list에서 해당 job을 terminated상태로 바꿔준다.
* SIGTTOU,SIGTTIN처리
* 자식에서 setpgrp()함수를 통해 부모와 다른 프로세스 그룹을 생성함으로 more,less 명령어가 일어날때 SIGTTOU,SIGTTIN 시그널이 들어와 이를 처리하지않으면 정상적으로 실행이 되지않았다.
* 부모에서는 SIGTTOU,SIGTTIN 시그널이 들어올때 아무것도 하지 않게 하였고
* SIGINT 처리
* SIGINT의 처리는 부모에서는 SIGINT가 SIG\_IGN처리했고, child에서는 SIG\_DFL로 default행동을 복구해
* child에서는 SIGINT를 누르면 Terminated 되도록 처리했다.
* SIGTSTP 처리
* 부모 CHILD HANDLER 의 waitpid가 WUNTRACED 옵션으로 설정되어있어 Foreground 상태의 자식이 Stopped된다면 부모가 시그널을 받아 해당 자식의 pgid를 jobs\_list에 suspended상태로 추가한다
* foreground 자식이 아무것도 없을때에는 반응이 없도록 설정한다.
* SIGTTOU,SIGTTIN처리
* less ,more ,또 개인적인 욕심으로 추가한 vi 편집기 명령어 같은 경우는 자식 프로세스가 stdin과 stdout을 사용하고자한다. 항상 foreground 상태에서는 자식이 tcsetgroup(pid,STDIN\_FILENO)를 통해 input,output을 가져가도록 설정했다. 또한 fg 명령어에서도 foreground로 자식이 input/output을 사용하고자 할수 있기에 tcsetgroup함수를 사용했다.
* 그리고 항상 child가 stopped 되거나 terminated 될때는 부모의 child 시그널 핸들러에서 다시 input과 output을 가져오게끔 설정했다.
* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명
* interpreter.c에서 cmdline을 parsing하는데, command를 이중 연결 리스트(Redirecto to, Redirect From)로 형태로 만든다.
* execute.c 파일 execute\_commands(struct command\* cur\_cmd)의 child while loop의 매 fork에서 redirectfrom, 명령어 이전에 파이프가 있다면 dup2로 STDIN\_FILENO를 이전의 파이프(i)의 READ\_END로 리다이렉팅한다
* 마찬가지로 다음 명령어가 있다면 STDOUT\_FILENO를 다음 파이프(i+1)의 WRITE\_END로 리다이렉팅한다. 따라서 Pipeline의 개수와 상관없이 일관성있는 처리가 가능했다
* 다만, /bin의 디렉토리에 있는 명령어가 아닌 cd와 같은 명령어는 fork한 프로세스에 실행할 경우 처리가 되지 않음으로, 만약 파이프로 리다이렉팅할 다음 명령어가 없다면 fork로 들어가기전 부모프로세스에서 실행되도록 예외처리했다.
* 이외에 jobs | less, jobs | greps "abc" 등은 함수형 명령어라도 리다이렉팅이 정확하게 일어날수 있도록 처리했다.
* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명
* if(mode==FOREGROUND){
* child\_pid=pid;
* reap\_dead\_jobs();
* sigsuspend(&mask);
* }
* else if(mode==BACKGROUND){
* printf("[%d] : %s",jobs\_rear,buf);
* insert\_jobs(pid,buf,RUNNING);
* }
* parent에서 sigsuspend를 실행하여 sigchild\_handler를 통해 wakeup하는 foreground process와 다르게 background는 parent에서 suspend 하지않고 해당 cmdline을 job\_list[MAXJOBS] 배열에 추가하고 해당 명령어가 백그라운드에서 돌고 있음을 나타내는 [1] ls | grep "abc" & 형식을 출력한다.
* void sigchild\_handler(int sig){
* pid\_t pid;
* while( (pid=waitpid(-1,NULL,WNOHANG))>0){
* if(pid==child\_pid){
* *// Foreground Process*
* child\_pid=0;
* SEND\_CONTINUE(parent\_pid);
* }else{
* *// Background Process*
* int index=find\_jobs\_by\_pid(pid);
* jobs\_list[index]->state=TERMINATED;
* }
* }
* tcsetpgrp(STDIN\_FILENO,parent\_pid);
* tcsetpgrp(STDOUT\_FILENO,parent\_pid);
* }
* background child가 모두 son을 수거하고 종료되면, parent의 sigchld\_handler는는 child를 수거하며 job\_list[MAXJOBS]배열에서 해당 JOB을 찾아 구조체 내부의 state 플래그를 Terminated로 바꿔준다.
* 이 Terminated 된 JOB들을 어떻게 배열에서 제거할까 고민을 했었는데, thread를 통한 garbage collection을 만들까 고민을 했었지만 bash shell에선 단순히 입력이 들어왔을때 child 명령어를 실행하는동안 parent process에서 제거를 하는 형태라 마찬가지로 reap\_dead\_job 함수를 통해 Terminated 상태의 JOB을 jobs\_done[list]에 추가한다.
* 그리고 cmdline를 입력받은후 print\_done\_jobs함수를 통해 종료된 작업을 출력한다
* bg
* child는 son들의 pid list를 가지고 있다. bg 명령어를 실행하면 우선 child에 SIGCONT신호, 다시 child는 son들에게 sigcont신호를 보낸다.
* fg
* fg는 bg에서 waitpid(child\_pid,&st,WUNTRACED)로 자식을 기다린다는 점이 다르다
* 만약 WIFSTOPPED(st)면 자식을 suspended상태로 jobs\_list에 넣어준다.
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

구현할때 bash shell의 소스코드를 참조했는데, bash shell 에서 binary search를 통해 명령어를 찾고 있었다. myshell 프로젝트는 명령어의 갯수가 그렇게 많지않아 Linear하게 선형으로 찾는 것에 비해 큰 성능차이가 나지는 않을 것이나 해당 알고리즘을 프로젝트에 사용해보았다.

원래 프로젝트에서 주어지는 기본 소스코드는 eval()함수를 통해 명령어를 실행하는 방식이었는데, interpreter로 이름을 바꾸고 매 루프마다 interpreter에서 parser 함수를 실행한다. parser 함수는 명령어를 파싱한후 status를 반환하는데, status에는 SYNTAX ERROR, NOCOMMAND ERROR등 에러값이 반환된다. switch case문에서 OK값이 아니라면 에러메세지를 출력하며 즉시 종료한다.

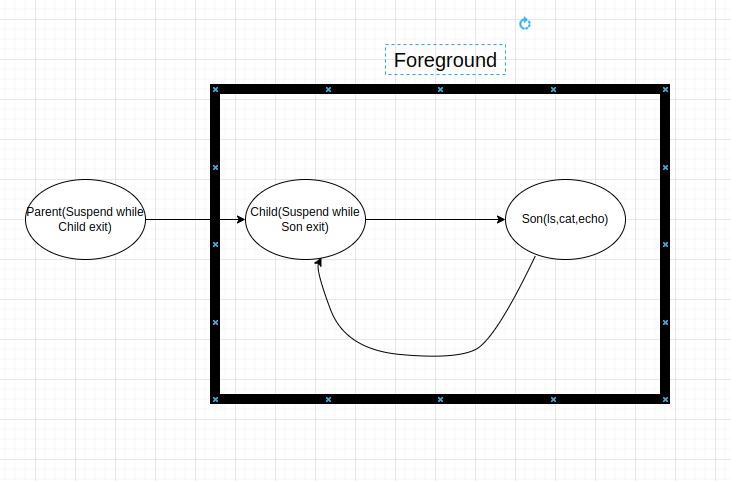
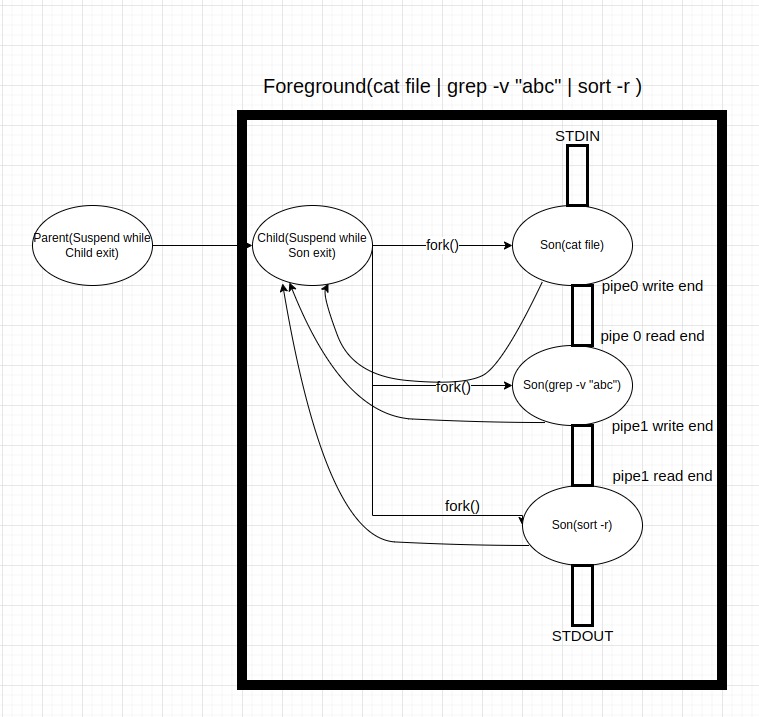
* struct command{
* unsigned short argc;
* char\*\* arguments;
* cmd\_flag f;
* struct command\* redirectto;
* struct command\* redirectfrom;
* union
* {
* builtin\* builtin;
* variable\* variable;
* };
* };

또한 COMMAND 구조체에는 arguments, 함수 커맨드인지, RELATIVE PATH 커맨드인지, ABSOLUTE PATH 커맨드인지를 나타내는 enum형 flag가 있다. switch\_case\_cmd함수에서 flag를 활용해 명령어를 올바르게 실행한다. union 구조체 내의 builtin과 함께 있는 variable 구조체 포인터는 시간이 된다면 환경변수 구조체도 만들어보려고 했으나, 시간 상 하지 못한 레거시이다. 과제의 채점 포인트는 아니니 추후 구현해볼 생각이다.

* typedef struct \_builtin{
* char\* name;
* void (\*fp)(char\*\*);
* }builtin;

만약 switch\_case\_cmd함수에서 flag==function이라면 함수의 포인터를 통해 함수를 실행한다.

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)**
* 
* **2.     Phase 2 (pipeline)**
* 
* **3.     Phase 3 (background)**
* 