System Programming

Programming Assignment #2 Shlab

2013-11826 임주경

- eval

명령어를 받아서, 제공된 함수 parseline의 return 값을 변수 isBg로 지정하여 수행 명령어가 BG 인지 FG인지 체크한다. 다음 명령어가 빈 input이면 return으로 함수를 종료하고, builtin_cmd 함수를 호출해 return 값으로 built in 명령어임을 확인할 경우, built in명령어를 수행하고 return하도록 한다. Built in 명령어가 아닐 경우, 이때, signal 집합으로 사용할 sigset_t 변수 mask를 선언하여, sigemptyset(). Sigaddset() 함수 호출로 Signal 집합을 초기화한다. 다음, sigprocmask() 함수를 호출하여, Signal Block 상태로 만든다. 이때, 위 system call들의 return 값이 -1 즉, 에러가 발생할 경우 제공된 함수 unix_error()로 에러메시지를 출력한다. 다음, pid_t 타입 변수 pid를 fork()를 호출해 초기화하며, pid가 -1일 경우 system call error로 간주한다.

위의 과정까지 error 발생 없이 진행되었을 경우, Child 프로세스와 Parent 프로세스의 경우를 나누어 구현한다. Child 프로세스는 sigprocmask() 호출로 Signal UnBlock 상태를 만든 뒤, setpgid(0, 0) 결과로 새로운 process group을 설정한다. 마찬가지로, setpgid의 return 값으로 error도 check 한다. 다음, execvp() 함수를 호출하며, 이때 인자는 명령어의 첫번째 단어인 파일명으로 지정한다. Error가 없을 경우, 여기서 return되며, error가 발생할 경우, Command not found 문구를 출력하며, exit()으로 프로세스를 종료한다. Parent 프로세스는 addjob() 함수 호출로 job list에 job을 추가하며, Signal을 UnBlock 시킨다.

마지막으로, 변수 isBg 값으로 해당 변수가 BG인지 FG인지 체크하고, BG의 경우 해당 명령어 수행 문구를 출력하며, FG의 경우 waitfg() 함수 호출로, foreground 종료까지 기다리도록 한다.

- builtin_cmd

명령어의 첫번째 단어가 "quit"인 경우 exit() 호출, "jobs"인 경우 제공된 listjobs() 함수 호출 뒤, built in command임을 check 하는 1을 return, "bg" 또는 "fg"인 경우 do_bgfg() 함수 호출 뒤, 1을 return 한다. 이외의 경우에는 built in command가 아님을 check 하는 0을 return한다.

- do_bgfg

사용하는 변수로는 int 변수 pid, jid와 job 구조체 job_t 타입의 포인터 *job을 선언한다. 우선, 함수 인자 argv의 두번째 string이 NULL일 경우, pid 또는 jid인자가 필요하다는 문구를 출력하며,

return 한다.

Strstr() 함수를 호출하여, ID 인자가 %를 포함하지 않는 PID라면, isdigit() 함수로 Valid한 pid인지 체크하고, valid 하다면, 해당 pid의 job을 getjobpid() 함수 호출로 찾아낸다. 만약 pid에 해당하는 job이 없다면, No such process 문구를 출력한다.

ID 인자가 JID라면, PID에서와 마찬가지 과정을 수행한다.

해당하는 job이 job list에 존재한다면, BG의 경우 해당 명령어 수행 문구를 출력하고 job state를 BG로 바꾼다. 다음, kill() 함수를 호출해 SIGCONT signal로 해당 process group을 재실행하도록 한다. FG의 경우 job state를 FG로 바꾸고, 마찬가지 kill() 함수 호출로 SIGCONT signal을 전달하며, waitfg() 호출로 해당 job의 종료까지 기다린다.

- waitfg

While 문으로, 인자로 받은 pid 값이 fgpid() 호출로 check 한 현재 foreground process pid 값과 같은 동안, sleep() 호출로 유지한다. Foreground process가 종료되면, 함수가 return 되며, wait가 끝나게 된다.

- sigchld_handler

While 문으로 waitpid() 함수를 호출해 세번째 인자로 WNOHANG | WUNTRACED 를 입력한다. 이는, 기다리는 자식 process들의 종료 상태를 회수할 수 없거나, 중단된 자식의 process 들의 상태를 받기 위함이다. 이때, WIFEXITED()에 해당하는 경우 deletejob() 호출로 해당 job을 제거한다. WIFSTOPPED()에 해당하는 경우 signal 20에 의해 process가 멈췄음을 문구로 출력하고, 해당 job state를 ST로 변경한다. WIFSIGNALED()에 해당하는 경우 signal 2에 의해 process가 종료되었음을 문구로 출력하고, 해당 job을 deletejob() 호출로 제거한다. 이외의 경우 unix_error() 호출로 SigChld Error문구를 출력한다. 만일 while 문에서 탈출했을 때, errno가 ECHILD에 해당할 경우 SigChld Error로 간주하여 return 하며, pid가 -1일 경우 waitpid Error 문구를 출력한다.

- sigint_handler / sigtstp_handler

Fgpid() 호출로 현재, FG의 process id를 변수로 초기화한다. 만약, 해당 process가 없다면, 아무일도 수행하지 않고 return하며, 해당 process가 존재한다면, kill(-pid, Signal)을 호출한다. 이때, -pid는 같은 group의 process들에게 signal을 전달하기 위함이다. Sigint의 경우 SIGINT를 sigtstp의 경우 SIGTSTP를 kill 함수의 두번째 인자로 입력한다.

- 마치며

각 함수의 구현에 있어서 어려움을 겪은 이유 중 하나는, System level에서의 함수들과 Signal 변수, Signal Set 등이 추상적으로 다가왔기 때문이다. 특히, Signal emptyset, Signal addset, SigprocMask와 같은 함수들에 대한 정보는 과제를 수행하며, 공부하게 되었다. 본 과제를 수행하며, Shell Programming은 user level에서 수행되지만, 보이지 않는 Signal과 System call들에 대해서 기본 원리를 잘 파악해야 한다는 중요성을 느낄 수 있었으며, Signal을 control하는 handler들이 각 process 간에 어떠한 관계를 갖고 전달해주는지 수행 결과를 어떤 System call을 통해 전달받는지, Signal set을 이용해 Signal Block / UnBlock 상태를 안정적으로 유지하는 것이 중요하다는 점들에 대해서 배울 수 있었다.