[Outline how programs are launched and how arguments are passed]

전체적인 개요를 설명하자면, 우선, 전역변수로 pid_t cpid,와 char*name을 선언하였다. 이들은 signal_handler를 overriding하는 경우, kill할 때 필요한 child의 pid와 "name is timed out"이라는 것을 출력해주기 위하여 선언하였다. 그 다음 signal_handler를 overriding하였는데, 이 부분은 강의노트를 보았다. 이는 timeout구현 방법에 대해 설명할 때 자세히 말하고자 한다.

우리가 구현해야 하는 run_command에서는 build-in command인 exit, prompt, timeout, for, cd는 각각 tokens[0]에 들어온 경우, 그에 해당하는 일을 처리하기 위해 if문으로 나누었다. 아래에서 설명할 time-out 과 for문 처리 방법 외의 것들에 대해 설명하고자 한다.

prompt는 기존의 prompt에 새로 들어온 tokens[1]값으로 덮어쓰는 것이므로 strcpy함수를 사용하였다.

cd는 change directory를 하는 것이므로, 우선 변경하고자 하는 값을 받기 위해 char*dir = tokens[1]을 선언하였다. 만약 cd뒤에 ~가 오는 경우, 우리는 HOME으로 가게하기 위해 change directory함수인 chdir안에 getenv("HOME")을 넣어주었다. 이는 "HOME"에 해당하는 "HOME"환경변수값을 반환하여 HOME으로 가게 해준다. 그 외의 directory로 가는 경우 chdir(dir)를 통해 해당 directory로 이동하도록 하였다.

externel command는 build-in command 외의 나머지 것이므로 else문으로 처리하였다. man page에서 fork()를 보면, fork 성공시, 반환되는 값이 부모님에게 child process pid가 전송되고, child에게 0이 전송되는 것을 알 수 있다. 따라서, cpid==0인 child인 경우와 cpid>0인 부모의 경우로 나누었다. 전역변수로 선언한 name에는 timeout 될 ./toy와 같은 것들이 들어가므로 tokens[0]값을 넣어주었다. chlid인 경우, execvp()를 통해 exec()를 처리하였다. execvp()는 path에 등록된 디렉토리에 있는 프로그램을 실행한다. man page를 통해 확인하면, execvp(const char*file, char*const argv[])이다. path에 있는 명령어인지 이는 확인이 되므로 실행될 file이름(tokens[0])을 첫번째 인자로 전달하고, 프로그램에 작동될 문자열인 tokens를 두번째 인자로 하였다. execvp는 본래는 return 값이 없지만, exec에 실패한 경우 -1을 반환하므로 반환 값이 0보다 작은 경우, file이 디렉토리에 없다는 문구를 출력하였다. 이 때는 이를 종료하기 위해 exit(0)만 하였었는데, no such file or directory이 나온 후, 새로운 명령어를 실행하였을 때, input들이 feedback되는 현상이 발생하였다. 따라서 이를 처리하기 위해, stdin을 닫아주는 close(0)함수를 넣어주었다. 부모(cpid>0) 인 경우는 자식을 데려오기 위해 waitpid(pid_t pid, int*wstatus, int options)를 사용하였다. 자식이 좀비로 남지 않게 하기 위해서이다.

[how the timed-out feature is implemented]

우선 built-in command인 timeout은 timeout만 들어온 경우, 현재 timeout시간을 보여주고, timeout 숫자로 들어온 경우에는 timeout을 설정하면 set_timeout에서 몇초로 설정되었는지, 0초가 들어온다면 설정이 불가능하다는 메시지가 출력이 된다. 따라서 나는 timeout만 들어온 경우와 timeout 숫자가 들어온 경우를 구분하기 위해 nr_tokens수를 사용하였다. nr_tokens수가 1인 경우, 'timeout'만 들어온 것이므로 현재 timeout시간을 출력해주고, 그렇지 않은 경우에는 timeout의 시간을 설정해주는 set_timeout함수를 호출해주었다.

다음으로 timeout이 될 때 까지 실행되지 않은 것들을 실행 종료를 하기 위해 , signal_handler를overriding하고 struct sigaction을 선언하였다. signal_handler에서는 SIGALRM 신호를 받게 된

경우, 시간이 초과되었으므로 child를 죽이고kill(cpid,SIGKILL), 시간이 다되었다는 메시지를 출력하기 위한 것을 선언하였다. 또한, alarm 설정 시간이 지난 후, 이 함수가 작동되므로, fprintf(stderr, "%s is timed out₩n",name)을 통해 시간이 다되었다는 것을 알려주는 메시지를 출력하였다. 그리고 signal_handler에서 이는 강의노트에 있는 것과 같이, sigaction(SIGALRM, &새로구현된sigaction, &기존의 sigaction)으로 구현하였다. 알람은 파라미터에 들어간 인자시간이 지난 후, signal을 보내는 명령어이다. 나는 위에서 해당시간이 지나면 Kill하는 것으로 overriding했으므로, 자식이 실행되는데 실행이 해당시간 안에 끝나지 않으면 sigaction으로 SIGALRM을 overriding한 것을 이용하여 kill하도록 한 것이다.

따라서 나는 부모님이 가진 child의 Pid인 cpid를 통해 kill한다고 생각하였으므로 부모님코드인(cpid>0)에서 sigaction(SIGALRM, %sa,&old_sa)로 SIGALRM에 kill을 구현하고, alarm(__timeout)으로 시간 재는 코드를 넣었다. 부모님은 자식을 기다리므로 wait_id = waitpid(cpid,&wstatus,0)을 하였다. 이 코드를 통해 부모는 자식이 죽으면 waitpid로 죽은 자식을 가져가도록 하였다.

이 상황에서 코드를 돌리면, externel command를 처리할 때, 다음자식에 대한 command가 이미 처리된 이후에도, alarm이 꺼지지가 않아서, alarm시간 이후 프로그램이 갑자기 죽는 현상같은 일이 나타났다. 우연히도 채점서버에서는 다 맞게 나와서 그냥 넘어가려다가, 뭔가 이상해서 우분투에서 혼자 계속 실행해본 결과, 이러한 문제가 있다는 것을 알 수 있었다. 이는 child의 명령어 처리가 다 끝났는데도 timeout에 대한 alarm이 계속 실행되어서 자꾸 프로그램이 종료되었다고 생각하였다. 이 부분을 해결하는데 굉장히 오래걸렸다. 오랜시간 뒤에 알게 되었던 것은, process가 끝나면 alarm(0)을 하여 alarm을 초기화하는 과정이 필요하다는 것이었다. 따라서, timeout되어 SIGALRM이 작동한다면, 작동 후 초기화 될 것이므로, 이와 반대인 timeout 되기 전에 child가 command처리를 끝낸 경우(if (wait_id != -1)), alarm을 취소하기 위해 alarm(0)을 추가하였다. 이렇게 하게 되면, 이미 실행된 externel command뒤에서 timeout시간에 맞춰 그냥 종료되는 현상이 고쳐졌다.

timeout이 완전하게 돌아가는 순간, 정말 행복했던 것 같다.

[How the 'for' built-in command is implemented]

nested for이라는 단어를 듣고, 재귀가 생각났다. 재귀를 사용하면 for 뒤에 숫자만큼 그 다음 명령어를 실행한다. 반복을 구현하기 위한 내가 아는 방법은 iteration과 recursion인데, for 3 for 4 for 5이런식으로 계속 반복되는 것은 iteration으로 구현하는 것이 적합하지 않으므로 recursion을 생각하게 되었다. for이 입력된 경우 반복횟수는 그 뒤에 숫자이므로 atoi(tokens[1])을 통해 그 뒤의 문자형 숫자를 정수형을 바꾸어주었다. 그리고 run_command를 다시 호출하였다. run_command(nr_tokens-2, tokens+2)를 통해 for 과 숫자 뒤에 명령어를 가지고 다시 그 함수를 실행하도록 한 것이다. 그 뒤의 명령어부터는 이번에 실행된 for과 숫자를 제외하므로, nr_tokens의 숫자가 2개가 줄어들 것이고, tokens 는 2개 뒤에서 시작할 것이다. 다음 command부터 run_command를 수행한다면 그 명령어(ex cd, pwd 등)에 따른 알맞은 동작을 하게 될 것이다.

[Lessons learned if you have any]

이번에는 과제에 접근하기까지 굉장히 많은 시간이 걸렸다. 어디서부터 어떻게 손을 대야할지 감이 안와서 굉장히 많은 시간동안 고민하였던 것 같다. built in command에서 prompt 와 cd와

for문,timeout 시간 설정하는 부분은 개념공부를 다시하고 보면 그래도 방법이 잘 떠올랐는데, externel command와 그 안에서의 timeout을 구현하는 것은 굉장히 많은 생각을 요구했던 것 같다. externel 명령어를 실행하기 위해 fork를 하여 child를 만들어서, execvp로 새로 시작한다는 개념은 익히고 있었지만, 구현하기까지의 과정이 길었던 것 같다. 수업시간에 exec()를 쓰면 pid는 유지되고 address spaces는 날린다고 되어 있었는데, 그 과정이 무엇인지에 대해서 알 수 있었다. 또한, 강의노트의 예제들이 굉장히 도움이 된다는 것을 알 수 있었다. 강의노트에 이번 fork같은 것은 이번처럼 예제가 많았으면 좋겠다는 생각을 하였다. 이번 과제는 코드자체가 길지는 않지만 정말 많은 생각을 할 수 있었던 과제였다고 생각한다. 이번 경험을 통해 항상 그냥 사용하던 터미널이 어떤 식으로 동작하는지에 대해 알 수 있었던 좋은 경험이 된 것 같다.

과제를 통해 Git과 우분투에 대하여도 굉장히 많은 부분 알 수 있었다. xcode로 코딩한 후, 그 코드를 복사하여 기존 우분투의 c파일의 코드를 지우고 붙여넣기를하면 Prompt에 control character가 들어가게 되어 ./mysh에 기묘한 프롬포트 모양이 들어간다. 그동안 이 잘못된 파일에 대해서 계속 내 git서버에 commit하면서 파일을 만들었었는데, 이는 다 지우고 다시 clone해야 ./mysh의 프롬포트 모양이 완전해진다는 Q&A답변을 듣고, 다시 하나하나 쳐서 옮겼는데 정상적으로 작동하는 것을 확인할 수 있었다. commit을 옮기기 위해 rebose명령어와 cherry pick 명령어를 처음으로 사용해보았는데 git에 익숙하지 않아서 정말 많은 시간이 걸렸다. 하지만 이 또한, 큰 경험이 되었다고 생각한다. 텍스트 파일을 복사 붙여넣기하면 파일이 완전하지 못할 수 있고, 오랜시간 걸려 commit을 다른 쪽 repository로 옮기는 방법까지 알 수 있게 되었기 때문에 과제와 더불어서 많은 것을 알 수 있었던 것 같다.