Operation system project1 – oh my pretty shell report

1. Outline how programs are launched and how arguments are passed

1)Exit 명령어 처리부분, 2)prompt 명령어처리부분, 3)cd 명령어 처리부분, 4)timeout 명령어 처리 부분, 5)for 명령어 처리부분 그리고 6)기타 나머지 명령어들 처리 부분 6 가지 policy 를 만들고, 6 가지를 구분하는 메커니즘은 if – else if – else 문을 활용하여 command 에서 첫번째 로 파싱 되어 들어오는 tokens[0]에 들어있는 문자열과 각 명령어를 비교시키는 함수 "strncmp()"를 사용하여 구현했습니다. 각각의 명령들이 어떤 메커니즘으로 작동하는지 기술하겠습니다.

첫 째로 cd 명령어입니다. 이것도 다른 명령들과 다를 바 없는 줄 알고, fork() – execvp() 를 사용하여 처리하려고 했습니다. 당연히 자식프로세스에서 cd 명령어를 수행하고 탈출해버리니 pwd 명령어를 사용하면 아무것도 안 변했습니다. 원인을 알기 위하여 /bin 디렉토리에 가보았더니 cd 명령어는 존재하지 않았고, 이는 기존의 명령어들과 달리 "execvp()"를 이용하는 메커니즘과 차별을 줘야된다는 사실을 깨닫게 되었습니다. 다시 말해, fork 시키지 않고 부모 프로세스에서 즉, 구현하는 쉘에서 "chdir()"와 "getenv()"를 사용해서 구현해야 한다는 사실을 알았습니다. 교수님께서 hint 로 주셨는데 그것을 빨리 발견했으면 덜 헤맸을 것 같습니다. 여기서도 cd ~ 와 cd ("..." , 디렉토리 명, 상대경로 등) 두 부분으로 나눠서 구현했습니다 cd ~는 "getnev()"의 인자로 "HOME"이라는 환경변수를 넘기면 홈으로 가는 절대경로를 참조한 다는 사실을 이용하였습니다. "getnev()"의 함수 결과로 나온 포인터를 "chdir()"의 인자로 넘겨 디렉토리를 이동시켰습니다. 또한, 실패 시 -1 을 반환하기 때문에 실패했을 때, "No such file or directory" 메시지를 띄우게 구현했습니다.
cd ("..." , 디렉토리 명, 상대경로 등) 은 "chdir()"의 인자로 tokens[1]을 넘겼습니다. 파싱 되어 들어오는 문자열이 모두 경로를 의미해주는 환경변수 이기 때문입니다.

두 번째로 기타 나머지 명령어들 처리 부분입니다. 자식프로세스는 fork 의 반환 값으로 0 을받고, 부모프로세스는 자식프로세스의 pid 값을 반환 받습니다. 이 사실을 활용하여, If((pid=fork())==0){...}else{...}의 메커니즘으로 부모 프로세스와 자식 프로세스를 구분 시켰고 부모 프로세스는(쉘)은 자식 프로세스(사용자가 입력하는 명령어)가 끝날 때까지 대기상태에 있도록 "waitpid(pid,&status,양수)"를 넣었습니다. 인자로는 status 변수의 주소 값을 넘겼는 데 이는, 자식프로세스가 끝나면 이 값을 반환하여 wait 함수가 이 값을 보고 resource 를 환원하게 됩니다. 자식프로세스에서는 "execvp()"를 사용하여 잘려 들어오는 tokens[0]을 첫 번째인자로 넘기고 두 번째인자로 tokens 배열 전체를 넘겼는데, array of

stirng 이라고 수업시간에 배웠기 때문입니다. 이 함수는 환경변수를 참조하여 그곳에 있는 명령이나 파일을 실행하는 기능을 수행합니다. 쉽게 말하면, 첫 번째 인자 tokens[0]을 참조하여 그곳에 있는 tokens 를 실행하는 것입니다. "execvp()"는 인자로 "파일/명령어(tokens[0])", 실행시킬 문자열("tokens")를 넘기면 실행되어 exec 계열의 다른 함수들 보다 편했습니다. 그리고 정상적으로 끝나면 탈출하도록 exit(0)을 넣었습니다.

이 함수가 실패하면 -1 이 반환되므로 실패 시 "No such file or directory"라는 메시지가 출력되도록 구현했습니다. execvp 함수가 실패하면 이 프로세스는 프로세스를 시그널을 보내서 죽이거나 비정상 종료를 시켜 줘야 합니다. 왜냐하면 execvp 함수는 실패 시 이 함수 호출지점 밑 줄부터 다시 코드를 실행해서 (부모와 자식은 exec 계열의 함수를 쓰지 않는 이상 코드영역을 공유하기 때문에)이 프로세스가 죽지 않고 살아있습니다. 처음에는 자동종료인 줄 알아서 종료 시키는 코드를 넣지 않았는데, make test-run 명령을 실행했을 때, not_exist binary 명령이 들어가면 실패하고 fprintf 함수를 실행하고 main 함수의 루틴으로 가는 것을 보았습니다. 그래서 실패 시 어떻게 되는지를 알게 되었고 오류를 수정했습니다.

셋 째로는 prompt 기능 구현입니다. 이 기능은 단순히 __prompt 배열의 담겨있는 값을 tokens[1]로 오는 문자열로 바꿔주면 되기 때문에, "strcpy(__prompt,tokens[1])"을 이용하 여 구현했습니다.

2. How the time-out feature is implemented

timeout null, timeout 0, timeout number 3 가지 policy 를 제공하고 timeout null 인경우는 "Current timeout is 0 second"가 출력되고 그리고 기타명령어가 처리되는 메커니즘에서 if(timer_set){ alarm(__timeout); }라는 코드를 넣어주어 timeout 0 일 때는 타이머 세팅 변수 (bool timer_set)을 false 로 만들어서 alarm 함수가 무시되도록 했고, timeout number 일 때는, 타이머 세팅 변수를 ture 로 만들어서 number 의 시간으로 자식프로세스가 생성과 함께 타이머를 맞추도록 구현하였습니다.

타이머 시간>자식프로세스 수행시간, 타이머 시간<자식프로세스 수행시간 구현은 제일 헤맸던 부분인데, 처음에는 timeout 이 되었을 때 핸들러 함수에서 waitpid 함수의 리턴 값을 이용하여 자식종료 여부를 판단, 종료가 되었으면 무시하고 종료가 안되었을 시, kill(pid,SIGKILL)을 실행시켜 자식프로세스를 죽이려고 했는데 waitpid 함수의 결과는 자식프로세스가 끝장나야 나오는 것이므로 잘못된 메커니즘이었습니다. 그 사실을 재활용하여

waitpid(pid,&status,양수)함수 밑 줄에 alarm(0);을 넣었는데 alarm 함수는 제일 늦게 수행된 것에 맞춰지고 그것의 인자가 0 이라면 알람을 취소하는 사실을 활용하여, 만약 자식프로세스가 수행 중이라면 부모는 아직 waitpid 함수에 머물러 있을 것이고 alarm(0)을 수행하지 못해 timeout 이 되면 핸들러 함수로 이동하고, 자식 프로세스가 종료되었으면 waitpid 함수 밑의 코드를 실행 시킬 것이므로 alarm(0)을 수행하여 보냈던 알람을 취소시킬 것이라는 판단 이 서서 구현했더니 성공적으로 되었습니다.

다음은 핸들러 함수 부분입니다. sigaction 함수를 이용하여 구현하였는데 sigaction(SIGALRM, &act, 0); 이것의 의미는 SIGALRM 은 alarm 함수가 시그널을 보내면 그 신호를 받는 역할입니다. &act 는 시그널을 받았을 시 어떤 핸들링을 해야 하는지 지정하는 구조체 변수의 주소 값입니다. 이 구조체 내부에는 핸들러 함수를 지정할 수 있는 변수가 있어서 이 변수에 제가 구현한 timeout_handler 함수 포인터를 넣어 줌으로서 핸들링함수로 timeout_handler 함수를 지정했습니다. 그리고 기타 나머지 구조체 멤버 변수들은 사용 필요가 없었기 때문에 모두 0 으로 초기화 시켰습니다. 그리고 핸들러함수 timeout_handler 는 호출되면 KILL 함수를 사용하여 자식프로세스에게 죽으라는 신호를 보내게끔 만들었습니다. Kill(pid,SIGKILL) 그리고 자식프로세스가 수행중인 명령 이름을 기타 명령어처리 메커니즘에서 childname 포인터에 담았고, 이 명령 이름을 출력해 줌으로써 무엇이 죽었는지 출력시켰고, 이름을 담았던 배열을 초기화하는 코드를 넣었습니다.

3. How the for built-in command is implemented

Tokens[0]의 값이 for 와 같다면 for 명령이 실행되는 것을 의미하므로 메커니즘으로 진입 시켰고, tokens[index]가 null 이 아닐 때까지 반복하면서, 사용자가 입력한 명령에서 숫자인 것 만을 골라서(atoi 함수 사용) 누적 곱을 구한다. 초기값은 1 이고 loop 변수가 숫자가 들어올 때 마다 자기자신과 들어온 수를 곱한다. (중첩 루프이기 때문에) 그리고 뒤에 나오는 for 부터는 무시하고 "for"도 아니고 "숫자"도 아닌 문자열이 명령어 일 것이므로 그에 해당하는 문자열을 command 배열에 담았다. 그리고 while(loop--){ /*command 배열에 담긴 내용을 수행한다.*/ } 이렇게 하면 반복하는 효과가 나온다. 이 반복문 내부에는 cd 명령어와 기타 명령어 처리 메커니즘을 넣었다. 그리고 반복하면서 fork 를 하면 자식이 계속 생기므로 execvp 함수의 수행이 끝나면 exit 함수를 사용하여 바로바로 탈출 시켜 자식이 수행되고 바로 죽게 만들었습니다.

4. Lesson learned

수도 없이 강의노트와 구글검색을 하면서 완성했습니다. 어려운 걸 해내서 굉장히 뿌듯합니다.