Lab report #8

CSED 20170302 Kim Dae Hui

이번 lab은 shell lab이다. 7가지 기능을 할 수 있는 shell을 직접 코딩해서 만들어내는 작업을 하는 lab이다. 대부분의 pid와 명령입력부를 잘 만들어 놓은 상태로 시작했기 때문에 몇몇의 skeleton code만 작성하게 되면 된다.

• eval: Main routine that parses and interprets the command line. [70 lines]

• builtin cmd: Recognizes and interprets the built-in commands: quit, fg, bg, and jobs. [25 lines]

• do bgfg: Implements the bg and fg built-in commands. [50 lines]

• waitfg: Waits for a foreground job to complete. [20 lines]

• sigchld handler: Catches SIGCHILD signals. 80 lines]

• sigint handler: Catches SIGINT (ctrl-c) signals. [15 lines]

• sigtstp handler: Catches SIGTSTP (ctrl-z) signals. [15 lines]

위 7가지 부분에 대한 코드만 해결하게 되면 되고 기타 쓰일 배열이나 변수들은 선언이 되어있고 쓰일 header들도 다 포함되어 있다. 따라서 본 보고서에서는 작성한 코드 위주로만 리뷰를 진행하고자 한다.

|  |
| --- |
| void eval(char \*cmdline)  {  char \*argv[MAXARGS];  int checkbg;  pid\_t pid;  sigset\_t mask;  checkbg = parseline(cmdline, argv); //bg를 parse한다.    //builtin\_cmd검사하면서 signal호출  if (!builtin\_cmd(argv)) {  // 우선 sig호출과 동시에 blocking  sigemptyset(&mask);  sigaddset(&mask, SIGCHLD);  sigprocmask(SIG\_BLOCK, &mask, NULL);  // pork시에 child를 포크하도록함  if ((pid = fork()) < 0) {  unix\_error("forking error");  }  // child가 성공적으로 포크 된 경우  else if (pid == 0) {  sigprocmask(SIG\_UNBLOCK, &mask, NULL);  setpgid(0, 0);  //커맨드가 없을경우, 음수여부로 검사하여 커맨드 유무를 분리  if (execvp(argv[0], argv) < 0) {  printf("%s: Command not found\n", argv[0]);  exit(1);  }  }  // parent process를 먼저 차지  else {  if (!checkbg) {  addjob(jobs, pid, FG, cmdline);  }  else {  addjob(jobs, pid, BG, cmdline);  }  sigprocmask(SIG\_UNBLOCK, &mask, NULL);  //bg의 경우와 fg의 경우를 분리한다.  if (!checkbg) {  //wait for fg  waitfg(pid);  }  else {  //bg의 출력  printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);  }  }  }  } |

eval부분은 cmd에서 읽어들인 명령어와 포크여부의 valid여부를 검사하는 부분으로 parsing해주며 검사에 들어간다.

|  |
| --- |
| int builtin\_cmd(char \*\*argv)  {  //이 함수에서 기본적인 command들에 대한 설계를 한다.  if (!strcmp(argv[0], "quit")) {  exit(0);  }  else if (!strcmp("&", argv[0])) {  return 1;  }  else if (!strcmp("jobs", argv[0])) {  listjobs(jobs);  return 1;  }  else if (!strcmp("bg", argv[0]) || !(strcmp("fg", argv[0]))) {  do\_bgfg(argv);  return 1;  }  return 0;  } |

이 함수에서는 전달받은 인자의 유효함을 받아들이고 quit의 명령어를 인지하게 해주고 명령어에 따라서 각기 맞는 함수를 실행하도록 나눠주는 부분이다.

|  |
| --- |
| void do\_bgfg(char \*\*argv)  {  //bgfg부분의 구현을 맡는 함수  struct job\_t \*jobtmp;  char \*tmpcom;  int jid;  pid\_t pid;  tmpcom = argv[1];  //명령어 조달이 안된 경우  if (tmpcom == NULL) {  printf("%s command requires PID or %%jobid argument\n", argv[0]);  return;  }  //command가 jid와 같을경우  if (tmpcom[0] == '%') {  jid = atoi(&tmpcom[1]);  //process를 갖게됨.  jobtmp = getjobjid(jobs, jid);  if (jobtmp == NULL) {  //예외처리  printf("%s: No such job\n", tmpcom);  return;  }  else {  //job의 pid를 pid에 넣어주어 나중에 kill하도록 해준다.  pid = jobtmp->pid;  }  }  // pid와 같을경우에  else if (isdigit(tmpcom[0])) {  //get pid하고  pid = atoi(tmpcom);  //get job한 후 예외처리  jobtmp = getjobpid(jobs, pid);  if (jobtmp == NULL) {  printf("(%d): No such process\n", pid);  return;  }  }  else {  printf("%s: argument must be a PID or %%jobid\n", argv[0]);  return;  }  //앞선 pid들을 kill하도록 한다.  kill(-pid, SIGCONT);  //bg fg 처리부분  if (!strcmp("fg", argv[0])) {  jobtmp->state = FG;  waitfg(jobtmp->pid);  }  else {  printf("[%d] (%d) %s", jobtmp->jid, jobtmp->pid, jobtmp->cmdline);  jobtmp->state = BG;  }  } |

이 함수는 bg와 fg를 구분해주는 부분으로 들어온 명령어에 따라서 bg/fg검사를 실행해주는 부분이다. 후에 프로세스 종료까지도 맡아주고 있다.

|  |
| --- |
| void waitfg(pid\_t pid)  {  struct job\_t\* jobtmp;  jobtmp = getjobpid(jobs, pid);  //pid의 예외처리  if (pid == 0) {  return;  }  if (jobtmp != NULL) {  //fg wiat할때 sleep의 역할  while (pid == fgpid(jobs)) {  }  }  return;  } |

이 함수에서는 fg의 경우 sleep이 일어나도록 해준다.

|  |
| --- |
| void sigchld\_handler(int sig)  {  int temp;  pid\_t pid;  //sig handler의 설계  while ((pid = waitpid(fgpid(jobs), &temp, WNOHANG | WUNTRACED)) > 0) {  if (WIFSTOPPED(temp)) {  //만약 temp가 바뀌게 되면은 stop signal이 간다.  getjobpid(jobs, pid)->state = ST;  int jid = pid2jid(pid);  printf("Job [%d] (%d) Stopped by signal %d\n", jid, pid, WSTOPSIG(temp));  }  else if (WIFSIGNALED(temp)) {  //temp가 삭제될 경우 signal이간다.  int jid = pid2jid(pid);  printf("Job [%d] (%d) terminated by signal %d\n", jid, pid, WTERMSIG(temp));  deletejob(jobs, pid);  }  else if (WIFEXITED(temp)) {  //exit한 경우  deletejob(jobs, pid);  }  }  return;  } |

이 함수는 냫쵱의 경우의 처리를 담고있고 삭제와 exit등 의 동작에 관한 handling을 크게 다루고 있다.

|  |
| --- |
| void sigint\_handler(int sig)  {  pid\_t pid = fgpid(jobs);  //pid가 제대로 입력되었는지를 체크하여 kill하게된다.  if (pid != 0) {  kill(-pid, sig);  }  return;  } |

이 함수에서는 sigint의 경우 호출되는 handler로 pid가 제대로 나오는지 보고 kill하는 과정을 진행한다.

|  |
| --- |
| void sigtstp\_handler(int sig)  {  pid\_t pid = fgpid(jobs);  //stp도마찬가지로 기능하도록하면 ehlsek.  if (pid != 0) {  kill(-pid, sig);  }  return;  } |

이 함수는 형태는 위와 같지만 sigstp일 경우 호출된다.

위와 같이 설정한 후에 make test01~ make test16과 make rtest01 ~ make rtest16을 비교하고 tsh>가 제대로 작동하는지 알 수 있고 잘못된 process와 제대로 된 process,그리고 백그라운드 작업등을 알 수 있다. 이는 xshell에서의 동작이고 trace파일에 대한 검사이기 때문에 결과와 설명은 생략한다.