|  |
| --- |
| 시스템 프로그래밍 |
| Simple Shell |
| 간단하게 동작하는 쉘 만들기 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **담당교수님** | 박정규 | | **제출일자** | 2015. 05. 06 | | **소속학과** | 컴퓨터공학 | | **학번** | 201017093 | | **이름** | 이상윤 | |

**Simple Shell**

**1. 프로그램 개요**

**1-1. Simple Shell**

Simple Shell은 커맨드라인을 해석해주는 프로세스이다. 사용자로부터 커맨드라인을 입력 받아서 입력 받은 라인을 해석하여 해당하는 명령을 실행시킨다.

이 프로그램은 표준입력을 통해서 명령어를 읽어온 후, 명령을 실행시킬 자식 프로세스를 fork해 동작한다. 부모 프로세스(mysh)는 다른 명령을 읽어 들이기 전, 자식 프로세스가 종료할 때까지 기다린다.

본 과제에서는 쉘의 기본적인 동작에 추가적으로 백그라운드 프로세스(background process), 리다이렉션(redirection), 히스토리(history) 기능을 구현하였다.

**1-2. Simple Shell 동작환경**

Simple Shell(이하 mysh)은 gcc version 4.8.3으로 컴파일되었으며, Red Had 4.8.3.-9에서 테스팅되었다.

**2. 프로그램 실행화면**

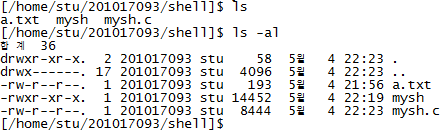
**2-1. 쉘 실행**



쉘이 실행되면 [작업디렉터리]$ 와 같은 모양의 프롬프트를 띄우게 된다.

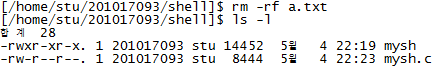
쉘이 정상적으로 실행되었다.

**2-2. Built-In 명령어 실행**



리눅스 기본 내장 명령어인, ls를 실행한 모습이다.

ls 명령어의 옵션을 주었을 때도 정상적으로 결과를 출력해준다.



또 다른 명령어인 파일 삭제 명령어를 rf(하위폴더 포함 삭제, 삭제 프롬프트 없이) 옵션을 주어서 실행시켰을 때도 정상적으로 파일이 지워지게 된다.



알 수 없는 명령어를 입력했을 때는 위와 같이 에러 메세지를 출력한다.

이 쉘 프로그램은 execvp를 이용하므로 간단하게, execvp의 리턴 값을 이용하여 에러가 발생하였을 시 perror 함수를 통해서 출력해주었기 때문에 위와 같은 에러 메세지를 출력하게 된다.



쉘은 exit 명령을 통해서 종료할 수 있다.

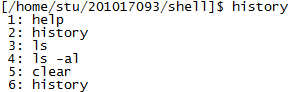
|  |  |
| --- | --- |
| [/home/stu/201017093/shell]$ | *Simple shell 프롬프트* |
| [201017093@cslab2 shell]$ | *실습서버 shell(bash) 프롬프트* |

와 같이 프롬프트가 변경되었음을 확인할 수 있다.



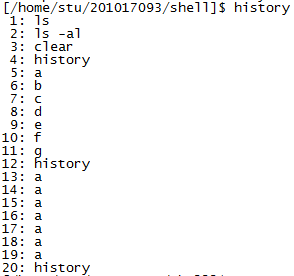
help 명령은 현재 simple shell에 대한 도움말을 출력해준다.

**2-3. History 기능**



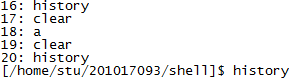
History 명령은 현재까지 입력했던 명령어의 히스토리를 확인할 수 있다.

히스토리 버퍼는 크기가 20이며, 20개를 넘어서면 처음부터 지워가면서 히스토리를 갱신한다.

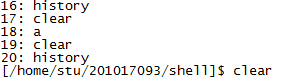


히스토리 명령을 확인 해 보면 위의 그림의 1, 2번이 사라지고 명령어가 계속 기록됨을 확인할 수 있다.

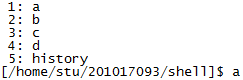
이러한 히스토리 기능은 Bash 쉘과 같이 키보드 방향키(위, 아래)를 이용해서 History를 자동으로 입력할 수 있다.



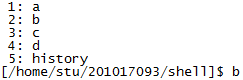
키보드 위 방향키를 누르게 되면 가장 마지막으로 입력한 명령이 자동적으로 불러와진다.



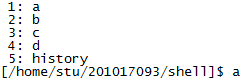
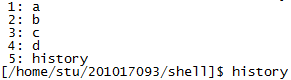
이 상태로 한번 더 키보드 위 방향키를 누르게 되면 그 전 명령이 불러와짐을 확인할 수 있다.



위와 같은 히스토리 상태에서, 키보드 아래 방향키를 누르게 되면 가장 처음으로 입력한 명령이 불러와 진다.



한 번 더 키보드 아래 방향키를 누르게 되면 두 번째로 입력한 명령이 불러와짐을 확인할 수 있다.



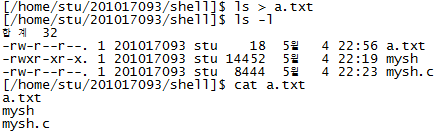
맨 마지막 명령어를 불러온 후, 한번 더 키보드 아래키를 누르면 가장 먼저 입력한 a가 입력됨을 확인할 수 있다.

**2-4. 리다이렉션**

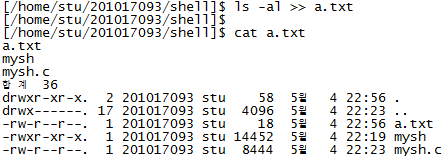
이 쉘에서 구현된 리다이렉션은 >(Output to Input)과 >>(Append)이다.

먼저 >(output to input) 리다이렉션이 정상적으로 동작하는지 확인해보자.

ls의 결과를 a.txt로 리다이렉션 하였다. 이 결과를 cat 명령을 통해서 확인한 스크릴 샷이다.



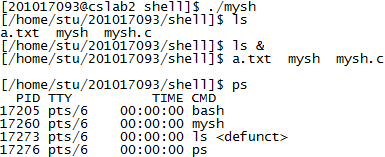
이 상태에서 >>(Append) 리다이렉션이 동작함을 확인해보자.



ls –al의 결과를 a.txt에 추가시키도록 명령을 실행하였다

정상적으로 추가되었음을 확인할 수 있다.

**2-5. Background 실행**

.

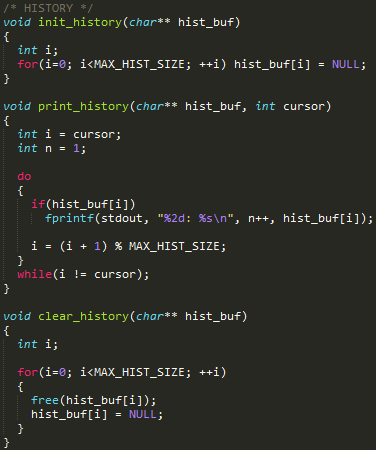
어떤 작업을 Background에서 실행시켰을 때와 그렇지 않을 때의 차이다.

명령어에 &을 추가하면 작업을 백그라운드에서 실행할 수 있다.

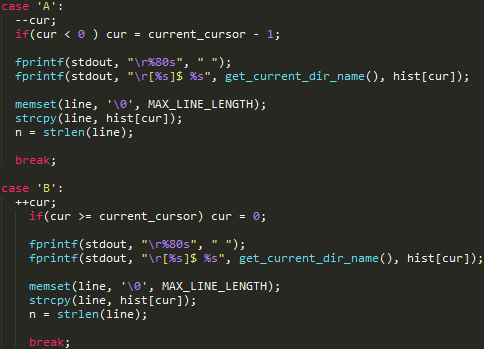
ps를 통해서 확인해보면 ls가 백그라운드에서 실행됨을 확인할 수 있다.

**3. 소스코드 부가설명**

**3-1. 히스토리**

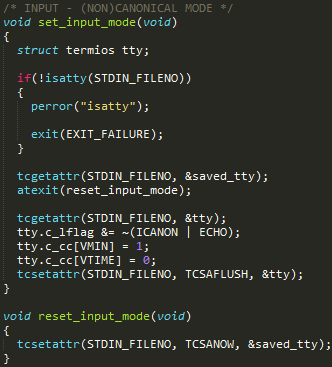


히스토리 기능은 Circular Buffer를 이용해서 간단하게 구현하였다. 리스트를 구현해도 되지만 간단하게 구현하기 위해서 배열을 이용하였다.



키보드 방향키에 따라서 (A-위, B-아래) 자동완성 기능에 응용되었다.

**3-2. 키보드 방향키 처리**



키보드 방향키의 처리를 위해서 터미널 모드를 정규모드(canonical mode)로 변경해야 한다.

정규모드는 기본적으로 라인단위로 입출력을 처리하게 된다. 여기서 말하는 라인은 EOF 혹은 개행문자(\n)를 만날 때까지의 문자열을 말한다.

비정규모드(Noncanonical mode)는 한 바이트씩 처리하게 되며, 정규모드에서 가능했던 특수문자들은 사용할 수 없다.

정규모드로 변경해주어야, 방향키나 백스페이스 등을 처리할 수 있게 된다.

이 부분은, <http://www.gnu.org/software/libc/manual/html_node/Noncanon-Example.html>를 참고해 작성했다.

**4. 소스코드**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <unistd.h>  #include <termio.h>  #include <fcntl.h>  #include <pwd.h>  #include <sys/types.h>  #include <sys/wait.h>  #define MAX\_LINE\_LENGTH 1 << 10  #define MAX\_ARG\_LENGTH 1 << 10  #define MAX\_ARG\_SIZE 100  #define MAX\_HIST\_SIZE 20  #define INPUT\_REDIRECTION ">"  #define OUTPUT\_REDIRECTION "<"  #define APPEND\_REDIRECTION ">>"  #define PIPELINE "|"  #define BACKGROUND "&"  #define ENTER 10  #define QUIT 4  #define BACKSPACE 8  #define ESC 27  #define LSB 91  #define STATE\_NONCANONICAL 0  #define STATE\_CANONICAL 1  #define DELIMS " \t\r\n"  void set\_input\_mode(void);  void reset\_input\_mode(void);  int prompt(char\*);  void init\_history(char\*\*);  void get\_history(char\*\*, int);  void print\_history(char\*\*, int);  void clear\_history(char\*\*);  bool parse(char\*, char\*\*, size\_t\*);  int lookupRedirection(char\*\*, size\_t, int\*);  bool lookupBackground(char\*\*, size\_t);  bool execute(char\*\*, size\_t);  struct termios saved\_tty;  char\* hist[MAX\_HIST\_SIZE];  int current\_cursor = 0;  int main(int argc, char\*\* argv)  {  char line[MAX\_LINE\_LENGTH];  init\_history(hist);  set\_input\_mode();  while(true)  {  char\* arguments[MAX\_ARG\_LENGTH];  size\_t argument\_count = 0;  if(prompt(line) == 0) continue;  hist[current\_cursor] = strdup(line);  current\_cursor = (current\_cursor + 1) % MAX\_HIST\_SIZE;  parse(line, arguments, &argument\_count);  if(argument\_count == 0) continue;  if(strcmp(arguments[0], "exit") == 0)  exit(EXIT\_SUCCESS);  else if((strcmp(arguments[0], "history") == 0))  print\_history(hist, current\_cursor);  else if(strcmp(arguments[0], "help") == 0)  {  fprintf(stdout, "간단한 쉘 만들기\n");  fprintf(stdout, "학번: 201017093\n");  fprintf(stdout, "이름: 이상윤\n");  }  else if(strcmp(arguments[0], "clear") == 0)  system(arguments[0]);  else  execute(arguments, argument\_count);  }  reset\_input\_mode();  clear\_history(hist);  return EXIT\_SUCCESS;  }  /\* HISTORY \*/  void init\_history(char\*\* hist\_buf)  {  int i;  for(i=0; i<MAX\_HIST\_SIZE; ++i) hist\_buf[i] = NULL;  }  void print\_history(char\*\* hist\_buf, int cursor)  {  int i = cursor;  int n = 1;  do  {  if(hist\_buf[i])  fprintf(stdout, "%2d: %s\n", n++, hist\_buf[i]);  i = (i + 1) % MAX\_HIST\_SIZE;  }  while(i != cursor);  }  void clear\_history(char\*\* hist\_buf)  {  int i;  for(i=0; i<MAX\_HIST\_SIZE; ++i)  {  free(hist\_buf[i]);  hist\_buf[i] = NULL;  }  }  /\* INPUT - (NON)CANONICAL MODE \*/  void set\_input\_mode(void)  {  struct termios tty;  if(!isatty(STDIN\_FILENO))  {  perror("isatty");  exit(EXIT\_FAILURE);  }  tcgetattr(STDIN\_FILENO, &saved\_tty);  atexit(reset\_input\_mode);  tcgetattr(STDIN\_FILENO, &tty);  tty.c\_lflag &= ~(ICANON | ECHO);  tty.c\_cc[VMIN] = 1;  tty.c\_cc[VTIME] = 0;  tcsetattr(STDIN\_FILENO, TCSAFLUSH, &tty);  }  void reset\_input\_mode(void)  {  tcsetattr(STDIN\_FILENO, TCSANOW, &saved\_tty);  }  /\* SHELL PROMPT \*/  int prompt(char\* line)  {  int n = 0;  int len = 0;  char c;  int cur = current\_cursor;  fflush(NULL);  fprintf(stdout, "[%s]$ ", get\_current\_dir\_name());  while(c = fgetc(stdin))  {  if(c == QUIT)  {  fputc('\n', stdout);  return 0;  }  if(c == ENTER)  {  fputc('\n', stdout);  break;  }  if((c != BACKSPACE) && (c <= 26)) break;  switch(c)  {  case BACKSPACE:  if(n == 0) break;  fputc('\b', stdout);  fputc(' ', stdout);  fputc('\b', stdout);  line[--n] = (char) 0;  break;  case ESC:  if((c = fgetc(stdin)) != LSB) break;  switch(fgetc(stdin))  {  case 'A':  --cur;  if(cur < 0 ) cur = current\_cursor - 1;  fprintf(stdout, "\r%80s", " ");  fprintf(stdout, "\r[%s]$ %s", get\_current\_dir\_name(), hist[cur]);  memset(line, '\0', MAX\_LINE\_LENGTH);  strcpy(line, hist[cur]);  n = strlen(line);  break;  case 'B':  ++cur;  if(cur >= current\_cursor) cur = 0;  fprintf(stdout, "\r%80s", " ");  fprintf(stdout, "\r[%s]$ %s", get\_current\_dir\_name(), hist[cur]);  memset(line, '\0', MAX\_LINE\_LENGTH);  strcpy(line, hist[cur]);  n = strlen(line);  break;  }  break;  default:  fputc(c, stdout);  line[n++] = (char) c;  break;  }  }  line[n] = '\0';  len = strlen(line);  if(len == 0) return 0;  for(n=0; n<len; ++n)  if(line[n] != ' ' && line[n] != '\t') return 1;  return 0;  }  int lookupRedirection(char\*\* argv, size\_t argc, int\* flag)  {  int i;  for(i=0; i<(int)argc; ++i)  {  if(strcmp(argv[i], INPUT\_REDIRECTION) == 0)  {  \*flag = O\_WRONLY | O\_CREAT | O\_TRUNC;  break;  }    if(strcmp(argv[i], APPEND\_REDIRECTION) == 0)  {  \*flag = O\_WRONLY | O\_CREAT | O\_APPEND;  break;  }  }  return i;  }  bool lookupBackground(char\*\* argv, size\_t argc)  {  return (strcmp(argv[argc - 1], BACKGROUND) == 0);  }  bool parse(char\* line, char\*\* argv, size\_t\* argc)  {  size\_t n = 0;  char\* temp = strtok(line, DELIMS);  if(temp == NULL) return false;  while(temp != NULL)  {  argv[n++] = temp;  temp = strtok(NULL, DELIMS);  }  argv[n] = NULL;  \*argc = n;  return true;  }  bool execute(char\*\* argv, size\_t argc)  {  char\* params[MAX\_ARG\_LENGTH];  int fd = -1;  int flag = 0;  int idx = lookupRedirection(argv, argc, &flag);  bool bg = lookupBackground(argv, argc);  pid\_t pid;  int status;  int i;  if((pid = fork()) == -1)  {  perror("fork");  return false;  }  else if(pid == 0)  {  for(i=0; i<idx; ++i)  {  if(strcmp(argv[i], BACKGROUND) == 0) break;  params[i] = argv[i];  }  if(flag > 0)  {  if((fd = open(argv[i+1], flag, 0644)) == -1)  {  perror("open");  return false;  }  if(close(STDOUT\_FILENO) == -1)  {  perror("close");    return false;  }  if(dup2(fd, STDOUT\_FILENO) == -1)  {  perror("dup2");  return false;  }  if(close(fd) == -1)  {  perror("close");  return false;  }  }  if(execvp(argv[0], params) == -1)  {  perror("execvp");  return false;  }  exit(EXIT\_SUCCESS);  }  else  {  if(bg == true)  {  return true;  }  else  {  if((pid = waitpid(pid, &status, 0)) == -1)  {  perror("waitpid");    return false;  }  return true;  }  }  } |