Lezione 16

Modificare il programma `smallsh` della quindicesima lezione, aggiungendo le seguenti funzionalità (dopo aver implementato le funzioni `gettok` e `runcommand` lasciate per esercizio):

- aggiungere il comando `exit` per terminare l'esecuzione della shell;
- implementare il comando `cd` per cambiare la directory corrente;
- aggiungere la possibilità di riconoscere ed ignorare i commenti (i.e., tutto ciò che segue il carattere speciale `#`);
- fare in modo che, usando il metacarattere di escape (\), sia possibile includere i caratteri speciali (e.g. spazi, tabulazioni, `&`, `;`, `#`) negli argomenti dei comandi.

I cambiamenti da apportare a `smallsh.h` e `smallsh.c`, per implementare le funzionalità richieste, sono i seguenti:

- aggiungere la costante simbolica `COMMENT`:

  ```
  # define COMMENT 5
  ```

  per gestire il caso in cui il token corrente sia identificato come commento:

- Aggiungere al vettore di caratteri `special` il carattere che introduce un commento (`#`):

  ```
  static char special[]={' ','\t','&',';','\n','#','\0'};
  ```

- Aggiungere il caso relativo ai commenti nel comando `switch` della funzione `gettok`:

  ```
  case '#':
    type=COMMENT;

    while(*ptr!='\n')
      ptr++;

    break;
  ```

  I comandi precedenti non fanno altro che impostare la variabile `type` con il valore `COMMENT` e saltare (incrementando il puntatore `ptr`) tutti i caratteri del commento fino ad incontrare il newline.

- Modificare il caso di default del comando `switch` della funzione `gettok` nel modo seguente:

  ```
  default:
    type=ARG;

    while(inarg(*ptr))
  ```

```
{

   if(*ptr=='\\')
     ptr++;

    *tok++=*ptr++;
}
```

Così facendo, se viene rilevato il carattere di escape (\) si fa avanzare il puntatore memorizzando nel token buffer il metacarattere successivo.

- Il comando `exit` viene implementato per mezzo dell'istruzione `if` seguente, inserita all'inizio della funzione `runcommand`:

```
if(strcmp(*cline,"exit")==0)
  exit(0);
```

- Il comando `cd` viene simulato per mezzo del codice seguente (inserito dopo quello per il comando `exit`):

```
if(strcmp(*cline,"cd")==0)
  {
    if(*(cline+1)==NULL)
      status=chdir((char *)getenv("HOME"));
    else
      status=chdir(*(cline+1));

    if(status==-1)
      {
        perror("Impossibile accedere alla directory specificata");
        return -1;
      }
    else
      return 0;

  }
```

Si noti l'utilizzo della funzione `getenv` per recuperare il percorso dell'home directory dell'utente quando quest'ultimo digita `cd` senza argomento. Si ricorda infatti che in tal caso la shell imposta la directory corrente a quella home dell'utente che ha invocato il comando.

Per completezza si riporta di seguito il codice sorgente completo dei file `smallsh.h` e `smallsh.c`:
• File `smallsh.h`:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

# define EOL 1 /* end of line */
```

```
# define ARG 2 /* normal arguments */
# define AMPERSAND 3
# define SEMICOLON 4
# define COMMENT 5

# define MAXARG 512 /* max. no. command arguments */
# define MAXBUF 512 /* max. length input line */

# define FOREGROUND 0
# define BACKGROUND 1
```

• File `smallsh.c`:

```
#include "smallsh2.h"

static char inpbuf[MAXBUF], tokbuf[2*MAXBUF],
*ptr = inpbuf, *tok = tokbuf; /* buffer e puntatori globali */
static char special[]={' ','\t','&',';','\n','#','\0'};

int userin(char *p)
{
  int c, count;
  ptr=inpbuf;
  tok=tokbuf;
  printf("%s",p); /* stampa il prompt */
  count=0;

  while(1)
    {

      if((c=getchar())==EOF)
        return(EOF);

      if(count<MAXBUF)
        inpbuf[count++]=c;

      if(c=='\n' && count<MAXBUF)
        {
          inpbuf[count]='\0';
          return count;
        }

      if(c=='\n') /* se linea troppo lunga, ricomincia */
        {
          printf("smallsh: input line too long\n");
          count=0;
          printf("%s",p);
        }

    }
```

```
}

int inarg(char c)
{
  char *p=special;

  while(*p!='\0')
    {

      if(c==*p++)
        return 0;

    }

  return 1;
}

int gettok(char **outptr)
{
  int type;
  *outptr=tok;

  while(*ptr==' ' || *ptr=='\t')
    ptr++;

  switch(*ptr) {
  case '\n':
    type=EOL;
    ptr++;
    break;
  case '&':
    type=AMPERSAND;
    ptr++;
    break;
  case ';':
    type=SEMICOLON;
    ptr++;
    break;
  case '#':
    type=COMMENT;

    while(*ptr!='\n')
      ptr++;

    break;
  default:
    type=ARG;

    while(inarg(*ptr))
      {
```

```
        if(*ptr=='\\')
            ptr++;

        *tok++=*ptr++;
      }

  }

  *tok++='\0';
  return type;
}

int runcommand(char **cline, int where)
{
  pid_t pid;
  int status;

  /* Implementazione del comando 'exit' */
  if(strcmp(*cline,"exit")==0)
    exit(0);

  /* Implementazione del comando 'cd' */
  if(strcmp(*cline,"cd")==0)
    {
      if(*(cline+1)==NULL)
        status=chdir((char *)getenv("HOME"));
      else
        status=chdir(*(cline+1));

      if(status==-1)
        {
          perror("Impossibile accedere alla directory specificata");
          return -1;
        }
      else
        return 0;

    }

  switch(pid=fork()) {
  case -1:
    perror("Errore nella chiamata alla fork");
    return -1;
  case 0:
    execvp(*cline,cline);
    perror(*cline);
    exit(1);
  default:
```

```
    if(where==BACKGROUND)
      {
        printf("Process id: %d\n",pid);
        return 0;
      }
    else
      {

        if(waitpid(pid,&status,0)==-1)
          return -1;
        else
          return status;

      }

  }

}

int procline(void)
{
  char *arg[MAXARG+1]; /* array di puntatori per runcommand */
  int toktype; /* typo del token */
  int narg; /* numero di argomenti correnti */
  int type; /* FOREGROUND o BACKGROUND */
  narg = 0;

  for(;;) /* ciclo infinito */
    {
      switch(toktype = gettok(&arg[narg])){
      case ARG: if(narg < MAXARG)
        narg++;
        break;
      case EOL:
      case SEMICOLON:
      case AMPERSAND:
      case COMMENT:

        if(toktype == AMPERSAND)
          type = BACKGROUND;
        else
          type = FOREGROUND;

        if(narg != 0)
          {
            arg[narg] = NULL;
            runcommand(arg, type);
          }

        if(toktype == EOL)
```

```
          return;

        narg = 0;
        break;
      }

    }

}

char *prompt = "Command> "; /* prompt */

main()
{
  while(userin(prompt) != EOF)
    procline();
}
```