```
#include <sys/types.h> /* pid_t */
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <assert.h>
#include <errno.h>


#define Close(FD) do {                                      \
    int Close_fd = (FD);                                    \
    if (close(Close_fd) == -1) {                            \
      perror("close");                                      \
      fprintf(stderr, "%s:%d: close(" #FD ") %d\n",         \
              __FILE__, __LINE__, Close_fd);                \
    }                                                       \
  }while(0)



  //const char *sort[] = { "sort", "-k9", NULL };
char ** command[101];   //   = { ls, sort, less };


void error_and_exit(char *s)
{
     printf("%s.\n", s);
     _exit(0);  // EXIT_FAILURE);
} // end of error_and_exit
```

```
/* move oldfd to newfd */
static void redirect(int oldfd, int newfd) {
  if (oldfd != newfd) {
    if (dup2(oldfd, newfd) != -1)
      Close(oldfd); /* successfully redirected */
    else
      error_and_exit("dup2");
  }
}



static void run(char* const argv[], int in, int out) {
  redirect(in, STDIN_FILENO);   /* <&in  : child reads from in */
  redirect(out, STDOUT_FILENO); /* >&out : child writes to out */

  execvp(argv[0], argv);
  // return(0); // error_and_exit("execvp");
}



int shellexec(int numcmds) {
  // const char *ls[] = { "ls", "-l", NULL };
  // const char *sort[] = { "sort", "-k9", NULL };
  // const char *less[] = { "less", NULL };
  // const char** command[] = { ls, sort, less };
  // int n = sizeof(command) / sizeof(*command);

  /* run all commands but the last */
  int i = 0, in = STDIN_FILENO; /* the first command reads from stdin */
```

```
  for ( ; i < (numcmds-1); ++i) {
    int fd[2]; /* in/out pipe ends */
    pid_t pid; /* child's pid */

    if (pipe(fd) == -1)
      error_and_exit("pipe");
    else if ((pid = fork()) == -1)
      error_and_exit("fork");
    else if (pid == 0) { /* run command[i] in the child process */
      Close(fd[0]); /* close unused read end of the pipe */
      run((char * const*)command[i], in, fd[1]); /* $ command < in > fd[1] */
      return(0);
    }
    else { /* parent */
      assert (pid > 0);
      Close(fd[1]); /* close unused write end of the pipe */
      Close(in);    /* close unused read end of the previous pipe */
      in = fd[0]; /* the next command reads from here */
    }
  }
  /* run the last command */
  run((char * const*)command[i], in, STDOUT_FILENO); /* $ command < in */
}
```

```
void skipblank()
{
     char c = ' ';
     while ((c = getchar()) == ' ' || c  == '\t') ;
     ungetc(c, stdin);
```

```
}



int main()
{

      while(1)   // while-1
      {
            int numcmds = 0;
            int no_wait = 0;
            // char E_name[101];

            // while(1)   // while-2  read a line
            {
                  //char E_name[101];
                  // char arg[101][101];
                  char **arg;
                  arg = (char **)malloc(sizeof(char *) * 101);
                  command[numcmds] = (char **) arg;

                  printf(">");

                  // scanf("%s", E_name);
                  // arg[0] = (char*)malloc(sizeof(E_name));
                  // strcpy(arg[0], E_name); //first argument should be process name

                  int i=0, j = 0;
                  char c;
                  arg[j] = (char*)malloc(sizeof(char)*101);
```

```
                skipblank();
                //    for(i = 0; i < 100 && j < 100 && (c = getchar()) != '\n'; /* no i ++ here. */  )
                while (1)   // while-3
                {
                      c = getchar();
printf("111111=%c.\n", c);
                      if(c == ' ' || c == '\t')
                      {
                            arg[j][i] = '\0';
                            skipblank();
                            i = 0;
                            j++;
                            arg[j] = (char*) malloc(sizeof(char) * 101);
                      }
                      else if (c == '|')
                      {
                            //a new command
                            char *tmp;
                            if (i > 0)
                            {
                                  arg[j][i] = '\0';
                                  arg[j+1] = NULL;
                                  tmp = NULL;
                            }
                            else // else i == 0
                            {
                                  tmp = arg[j];
                                  arg[j] = NULL;
                            }
                            arg = (char **)malloc(sizeof(char *) * 101);
                            numcmds ++;
                            command[numcmds] = arg;
                            skipblank();
```

```
        i = 0;
        j = 0;
        if (tmp == NULL)
        {
              arg[0] = (char *)malloc(sizeof(char) * 101);
        }
        else
        {
              arg[0] = tmp;
        }
}
else if (c == '>')
{
        arg[j][i] = '\0';
        // the next word should be a file name.
        // to be done later ......
}
else if (c == '&')
{
        if (i == 0)
        {
              // arg[j][0] = (char)NULL;
              free(arg[j]);
              arg[j] = (char*)NULL;
              no_wait = 1;
        }
        else  // else i > 0
        {
              arg[j][i] = '\0';
              arg[j+1] = (char*)NULL;
              no_wait = 1;
        }
        skipblank();
```

```
                        if ((c = getchar()) != '\n' && c != EOF) {
                              error_and_exit("input & is not at the end of line.\n");
                        } else {
printf("2222-%c.\n", c);
                              break;
                        }

                  }
                  else if (c == '\n' || c == EOF)
                  {
                        if (i == 0)
                        {
                              // arg[j][0] = (char)NULL;
                              free(arg[j]);
                              arg[j] = (char*)NULL;
                              // no_wait = 1;
                        }
                        else  // else i > 0
                        {
                              arg[j][i] = '\0';
                              arg[j+1] = (char*)NULL;
                              // no_wait = 1;
                        }
printf("3333-%c.\n", c);
                        break;
                  }
                  else
                  {
                        arg[j][i] = c;
                        i ++;
                  }
            } // end of while-3
      }  // end of while-2
```

```
            printf("numcmds = %d.\n", numcmds);
            for(int ii = 0; ii <= numcmds; ii ++)
            {
                  int jj = 0;
                  while (command[ii][jj] != NULL)
                  {
                        printf("%s.\n", command[ii][jj]);
                        jj ++;
                  }
                  printf("create a new command.\n");
            }
            shellexec(numcmds);
            printf("6666.\n");
            // fprintf(stderr, "5555.\n");
      } // end of while-1

      return(0);
}  // end of main
```

```
/*
          int no_wait;
          if(arg[j][0] == '&')
          {
               // arg[j][0] = (char)NULL;
               free(arg[j]);
               arg[j] = (char*)NULL;
               no_wait = 1;
          }
          else
          {
               if (i == 0)
               {
                    free(arg[j]);
                    arg[j] = (char *)NULL;
               }
               else
               {
                    arg[j][i] = '\0';
                    // arg[j+1][0] = (char)NULL;
                    arg[j+1] = (char *)NULL;
               }

               no_wait = 0;
          }
*/
```

```
/*

            //  int numcmds = 1; //  dividearg();  // number of commands linked by pipes
                                  // e.g., ls | sort | less means numcmds = 3
            int ii;
            for (ii = 0; ii < numcmds - 2; ii++ ) {

            }

            if (numcmds== 1) {

            pid_t pid = fork();

            if(pid == 0)     //child process
            {
                  if(no_wait == 1) //no wait, creating a grandchild process and kill this child process
                  {
                        pid_t G_pid = fork();
                        if(G_pid == 0)   //grandchild process
                        {
                              //execvp(E_name, (char* const *)arg); //execution of grandchild process
                              return 0;
                        }
                        else
                        {
                              return 0;  //kill child process
                        }
                  }
                  else  //wait, no need of grandchild process
                  {
                        // execvp(E_name, (char* const *)arg);//execution of child process
                        return 0;
                  }
```

```
            }
            else if(pid > 0) //parent process
            {
                  waitpid(pid, NULL, 0);
            }
            else
            {
                  //error
            }

            // for(int i = 0; arg[i] != NULL; i++)
            // {
                  //free(arg[i]);
            //}

            } else {  // numcmds > 1
                  // usingpipes(numcmds);
            }
      }     //end of while-1

      return 0;
}

*/
```