UE Programmation Unix

TP2 Gestion des Processus

Les appels système : gestion de processus (suite)

Un appel système permet à un processus utilisateur d'accéder à une ou plusieurs fonctions internes au noyau du system d'exploitation et de les exécuter.

Ainsi, un programme peut invoquer un ou plusieurs services du système d'exploitation.

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction

```
main (int argc, char * argv [])
```

- **Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affiches avec `perror`.**
- **Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.**

La fonction main()

Arguments de la ligne de commandes

- Langage C offre des mécanismes qui permettent d'intégrer parfaitement un programme C dans l'environnement hôte
  - environnement orienté ligne de commande (Unix, Linux)
- Programme C peut recevoir de la part de l'interpréteur de commandes qui a lancé son exécution, une liste d'arguments

⇔ ligne de commande qui a servi à lancer l'exécution du programme

- Liste composée
  - du nom du fichier binaire contenant le code exécutable du programme
  - des paramètres de la commande

Entête à inclure

```
#include <stdlib.h>  // <cstdlib> en C++
```

Fonction *atoi*

```
int atoi( const char * theString );
```

Cette fonction permet de transformer une chaîne de caractères, représentant une valeur entière, en une valeur numérique de type `int`. Le terme d'`atoi` est un acronyme signifiant : ASCII to integer.

⚠ **ATTENTION :** la fonction `atoi` retourne la valeur 0 si la chaîne de caractères ne contient pas une représentation de valeur numérique. Du coup, il n'est pas possible de distinguer la chaîne `"0"` d'une chaîne ne contenant pas un nombre entier. Si vous avez cette difficulté, veuillez préférer l'utilisation de la fonction `strtol` qui permet bien de distinguer les deux cas.

La fonction `main ()`

Un processus débute par l'exécution de la fonction `main()` du programme correspondant

Definition

```
int main (int argc, char *argv[]);
ou
int main (int argc, char **argv);
```

- `argc`: nombre d'arguments de la ligne de commande y compris le nom du programme
- `argv[]`: tableau de pointeurs vers les arguments (paramètres) passés au programme lors de son lancement

A NOTER

- `argv[0]` pointe vers le nom du programme
- `argv[argc]` vaut `NULL`

- argc (argument count)
  - nombre de mots qui compose la ligne de commande (y compris le nom de la commande qui a servi à lancer l'exécution du programme)
- argv (argument vector)
  - tableau de chaînes de caractères contenant chacune un mot de la ligne de commande
  - argv[0] est le nom du programme exécutable

Exemple de cours

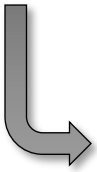
```c
#include <stdio.h>

int main (int argc, char *argv[]){
int i;
for (i=0;i<argc;i++){
    printf ("argv[%d]: %s\n",i, argv[i]);
}
return (0);
}// main
```

```
>./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
```



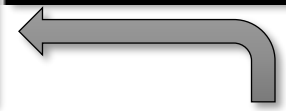
```
[ij04115@saphyr:~/unix_tpl]:ven. sept. 11$ nano affich_param.c
```



saphyr.ens.math-info.univ-paris5.fr - PuTTY

```
 GNU nano 2.5.3            Fichier : affich param.c
#include <stdio.h>

int main(int argc, char *argv[]){
        int i;
        for (i = 0 ; i < argc ; i++){
                printf("argv[%d]: %s\n", i, argv[i]);
        }
        return (0);
}//main
```



Spécifier le nom de l'exécutable avec l'option `-o`

```
ProgC > gcc -o toto premierProg.c
ProgC > ls
toto    premierProg.c
ProgC > ./toto
```

```
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ gcc -o affich_param affich_param.c
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ ls
affich_param  affich_param.c
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$
```

```
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ ./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$
```

Le fichier standard des erreurs
stderr

Structure *FILE ** et variables *stdin*, *stdout* et *stderr*

Entête à inclure

```
#include <stdio.h>
```

Structure *FILE ** et variables *stdin*, *stdout* et *stderr*

```
FILE * stdin;
FILE * stdout;
FILE * stderr;
```

La structure FILE permet de stocker les informations relatives à la gestion d'un flux de données. Néanmoins, il est très rare que vous ayez besoin d'accéder directement à ses attributs.

Effectivement, il existe un grand nombre de fonctions qui acceptent un paramètre basé sur cette structure pour déterminer ou contrôler divers aspects.

- **stdin** (Standard input) : ce flot correspond au flux standard d'entrée de l'application. Par défaut, ce flux est associé au clavier : vous pouvez donc acquérir facilement des données en provenance du clavier. Quelques fonctions utilisent implicitement ce flux (**scanf**, par exemple).
- **stdout** (Standard output): c'est le flux standard de sortie de votre application.
  Par défaut, ce flux est associé à la console d'où l'application à été lancée. Quelques fonctions utilisent implicitement ce flux (**printf**, par exemple).
- **stderr** (Standard error) : **ce dernier flux est associé à la sortie standard d'erreur de votre application.** Tout comme stdout, ce flux est normalement redirigé sur la console de l'application.

fprintf it is the same as printf,
except now you are also specifying the place to print to :

```
printf("%s", "Hello world\n"); // "Hello world" on stdout (using printf)
fprintf(stdout, "%s", "Hello world\n"); // "Hello world" on stdout (using fprintf)
fprintf(stderr, "%s", "Stack overflow!\n");//Error message on stderr (using fprintf)
```

Messages d'erreurs affiches avec perror

C Library - <stdio.h>

C library function - perror()

Description

The C library function **void perror(const char *str)** prints a descriptive error message to stderr. First the string **str** is printed, followed by a colon then a space.

Declaration

Following is the declaration for perror() function.

```
void perror(const char *str)
```

Parameters

- **str** – This is the C string containing a custom message to be printed before the error message itself.

Return Value

This function does not return any value.

```c
#include <stdio.h>

int main () {
   FILE *fp;

   /* first rename if there is any file */
   rename("file.txt", "newfile.txt");

   /* now let's try to open same file */
   fp = fopen("file.txt", "r");
   if( fp == NULL ) {
      perror("Error: ");
      return(-1);
   }
   fclose(fp);

   return(0);
}
```

Let us compile and run the above program that will produce the following result because we are trying to open a file which does not exist –

```
Error: : No such file or directory
```

2 Le shell

Ecrire le programme d'un shell simplifié capable d'exécuter, en premier plan ou en arrière-plan, n'importe quelle commande entrée par l'utilisateur avec un nombre arbitraire de paramètres.

Fonctions utiles :

fork	**fork - create a child process**
	`#include <sys/types.h>`
	`#include <unistd.h>`
	
	`pid_t fork(void);`

wait	**wait - wait for a child process to stop or terminate**
	
	`#include <sys/types.h>`
	`#include <sys/wait.h>`
	
	`pid_t wait(int *stat_loc);`

exit	**exit - cause normal process termination**
	`#include <stdlib.h>`
	
	`void exit(int status);`

execvp	**execvp - execute a file**
	`#include <unistd.h>`
	
	`int execvp(const char *file, char *const argv[]);`

fgets	**fgets - reads a line from the specified stream and stores it into the string pointed to by str.**
	It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached. If an error occurs, a null pointer is returned.
	
	`#include <stdio.h>`
	`char *fgets(char *str, int n, FILE *stream)`

strtok	**strtok - breaks string str into a series of tokens using the delimiter delim.**
	This function returns a pointer to the first token found in the string. A null pointer is returned if there are no tokens left to retrieve.
	
	`#include <string.h>`
	`char *strtok(char *str, const char *delim)`

strrchr	**strstr - function finds the first occurrence of the substring needle in the string haystack.**
	The terminating '\0' characters are not compared.
	
	This function returns a pointer to the first occurrence in haystack of any of the entire sequence of characters specified in needle, or a null pointer if the sequence is not present in haystack.
	
	`#include <string.h>`
	`char *strstr(const char *haystack, const char *needle)`

strlen	**strlen() - function takes a string as an argument and returns its length.**
	The returned value is of type size_t (the unsigned integer type).
	
	strlen() function doesn't count the null character \0 while calculating the length.
	
	`#include <string.h>`
	`size_t strlen(const char *str)`

strcpy	**strcpy() - copies the string pointed to, by src to dest.**
	This returns a pointer to the destination string dest.
	
	`#include <string.h>`
	`char *strcpy(char *dest, const char *src)`

malloc	**malloc() - allocates the requested memory and returns a pointer to it.**
	This function returns a pointer to the allocated memory, or NULL if the request fails.
	
	`#include <stdlib.h>`
	`void *malloc(size_t size)`

```c
#include <stdio.h> // printf(), fgets(), perror()
#include <string.h> // strcpy(), strlen(), strrchr(), strtok()
#include <stdlib.h> // malloc()
#include <sys/types.h> // wait(), fork()
#include <unistd.h> // fork(), exevp()
#include <sys/wait.h> // wait()

#define MAX_NB_ARGS 40
#define MAX_LONG_CMD 250
int main(int argc, char* argv[]) {

    char cmd[MAX_LONG_CMD]; // La commande
    char* argv_execvp[MAX_NB_ARGS]; // Les mot de la ligne de commande
    int bg, i, status, fork_result;
    char* ptrBg;
    char* token;

       while(1) { // boucle infinie
           printf("----->");
           fgets(cmd, MAX_LONG_CMD - 1 , stdin); // la commande + \n
           i = 0;
           while(cmd[i] != '\n') //on remplace \n par \0
               i++;
           cmd[i] = '\0';

         //on remplace & par \0 et on retient la présence sa dans variable bg
         ptrBg = strrchr(cmd, '&');
         bg = 0;
         if(ptrBg!= NULL){
            *ptrBg = '\0'; // on remplace & par \0
            bg = 1;
         }

         token = strtok(cmd, " "); //get the first token
         for( i = 0 ; token != NULL ; i++) {  // walk through other tokens

           argv_execvp[i] = (char*) malloc( strlen(token) + 1 );
           strcpy(argv_execvp[i], token);
           token = strtok(NULL, " ");

         }// for
         argv_execvp[i] = NULL; // Le tableau se termine par NULL

         if( i > 0) { // ligne de commande PAS vide
            fork_result = fork();
            if(fork_result == 0) { // CODE DU FILS
               printf("\n");
               execvp(argv_execvp[0], argv_execvp);
               perror("Erreur function execvp() : ");
            }
            else // CODE DU PERE
            {
               if(bg == 0) // si la commande est PAS executee en arrière plan
                 wait(&status); //On attend le processus
            }
         }
    } // while
} // main
```

```
[ij04115@saphyr:~/unix_tp2]:jeu. oct. 15$ nano question2.c
[ij04115@saphyr:~/unix_tp2]:jeu. oct. 15$ gcc question2.c -o question2
[ij04115@saphyr:~/unix_tp2]:jeu. oct. 15$ ./question2
----->ls

bonjour             question1_A1.c  question1_B2    question1_C2.c
bonjour.c           question1_A2    question1_B2.c  question2
environment_list    question1_A2.c  question1_C1    question2.c
environment_list.c  question1_B1    question1_C1.c  test_questionC
question1_A1        question1_B1.c  question1_C2    test_questionC.c
----->ls -l

total 120
-rwxr-xr-x 1 ij04115 licence3in 7352 sept. 26 14:33 bonjour
-rw-r--r-- 1 ij04115 licence3in   83 sept. 26 14:31 bonjour.c
-rwxr-xr-x 1 ij04115 licence3in 7360 sept. 26 20:12 environment_list
-rw-r--r-- 1 ij04115 licence3in  204 sept. 26 20:10 environment_list.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 14:53 question1_A1
-rw-r--r-- 1 ij04115 licence3in  346 sept. 26 14:52 question1_A1.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:06 question1_A2
-rw-r--r-- 1 ij04115 licence3in  368 sept. 26 15:06 question1_A2.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:52 question1_B1
-rw-r--r-- 1 ij04115 licence3in  347 sept. 26 15:51 question1_B1.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:58 question1_B2
-rw-r--r-- 1 ij04115 licence3in  372 sept. 26 15:57 question1_B2.c
-rwxr-xr-x 1 ij04115 licence3in 7524 sept. 26 20:26 question1_C1
-rw-r--r-- 1 ij04115 licence3in  526 sept. 26 20:36 question1_C1.c
-rwxr-xr-x 1 ij04115 licence3in 7524 sept. 26 20:39 question1_C2
-rw-r--r-- 1 ij04115 licence3in  614 sept. 26 20:39 question1_C2.c
-rwxr-xr-x 1 ij04115 licence3in 7804 oct.  15 02:39 question2
-rw-r--r-- 1 ij04115 licence3in 1899 oct.  15 02:39 question2.c
-rwxr-xr-x 1 ij04115 licence3in 7552 sept. 27 10:46 test_questionC
-rw-r--r-- 1 ij04115 licence3in  371 sept. 27 10:46 test_questionC.c
```