Programmation Système

Labo-Unix - http://www.labo-unix.net

2001-2002

Table des matières

Droits de ce document

1 Rappels

1.1 Les fonctions printf() et scanf()

```
int printf (const char *format, ...);
int scanf (const char *format, ...);
```

Ces instructions font des écritures et des lectures **formattées** sur les flots de sortie et d'entrée standard. Les spécifications de format sont décrites dans la page de manuel printf(3).

1.2 Les fonctions sprintf(), sscanf(), snprintf()

```
int sprintf (char *str, const char *format, ...);
int sscanf (const char *str, const char *format, ...);
```

Similaires aux précédentes, mais les opérations lisent ou écrivent dans le **tampon str**.

Remarque importante : la fonction *sprintf()* ne connait pas la taille du **tampon str**, il y a donc un risque de débordement[1]. Il faut prévoir des tampons assez larges, ou (**bien mieux**), utiliser la fonction :

```
#include <stdio.h>
int snprintf(char *str, size_t size, const char *format, ...);
```

de la bibliothèque GNU, qui permet d'indiquer une taille à ne pas dépasser.

1.3 La fonction system()

```
#include <stdlib.h>
int system (const char * string);
```

permet de lancer une ligne de commande depuis un programme. L'entier retourné par *system()* est le *code de retour* fourni en paramètre à *exit()* par la commande.

1.3.1 Exemple

Faire un programme qui va envoyer un fichier à une personne, en utilisant la fonction *system()* (qui appellera les commandes uuencode (man uuencode) et mail (man mail)), et la fonction *snprintf()*. Vous devrez utiliser uuencode pour encoder le fichier, et l'envoyer au destinataire voulu via la commande mail.
Voir le feuillet d'exercices, à l'exercice 1 pour la correction.

[1] Ce type de fonction a fait couler beaucoup d'encre. En effet, un concept d'attaque se base sur ces debordements (overflow), ce qui permet de deborder d'un buffer, et cela permet ainsi à des personnes malententionnées de faire executer des commandes arbitraires sur le système. C'est donc un risque pour la sécurité du système. Pour de plus amples renseignements, rechercher des informations sur les buffers overflows et les formats bugs.

2 Communication avec l'environnement

2.1 Paramètres de main(...)

Le lancement d'un programme C provoque l'appel de sa fonction principale *main(...)* qui a 3 paramètres optionnels :

- *argc* : nombre de paramètres sur la ligne de commande (y compris le nom de l'exécutable lui-même) ;
- *argv* : tableau de chaînes contenant les paramètres de la ligne de commande ;
- *envp* : tableau de chaînes contenant les variables d'environnement au moment de l'appel, sous la forme variable=valeur.

Remarques :

1. les dénominations *argc, argv, envp* sont purement conventionnelles.
2. *envp* est considérée comme obsolète. Voir plus loin la fonction *getenv()*.

2.1.1 Exemple

Faire un programme qui affichera le nombre d'arguments passés en paramètre (avec **argc**), qui listera les arguments en paramètres (avec **argv[]**), et listera les variables d'environnement (avec **envp[]**. Ce programme utilisera la fonction *exit()*.
Voir la correction sur le feuillet d'exercices, au niveau 'Exercice 2'.

2.2 La fonction getopt() : analyse de paramètres de la ligne de commande

L'analyse des options d'une ligne de commande est facilitée par l'emploi de *getopt()*.

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;
```

getopt() analyse le tableau de paramètres **argv** à **argc** éléments, en se basant sur la *chaîne de spécification d'options* **optstring**. **optstring** va donc être une chaine de caractères contenant toutes les options que l'on veut pouvoir spécifier à notre programme. Exemple : Soit un programme où l'on veut pouvoir specifier '-h' pour obtenir de l'aide, '-a' pour ajouter une ligne et '-s' pour en enlever une. Alors, l'appel à *getopt()* sera du type :

```
while ((c = getopt(argc, argv, "has:")) != -1)
       switch(c){
                  .....
/*
 * Ici, on passera dans le while si des options que nous venons de
 * specifier au programme font partie de 'optstring'. Elles seront
 * ensuite interpretées
 */
```

À chaque étape, **optarg** retourne la valeur de l'option (si -a test, alors optarg=test) ou un point d'interrogation pour une option non reconnue).

A la fin *getopt* retourne -1, et le tableau **argv** a été réarrangé pour que les paramètres supplémentaires soient stockés à partir de l'indice **optind**.

2.2.1 Exemple

Faire un programme qui utilisera la fonction *getopt()* (voir man 3 getopt). Ce programme pourra prendre plusieurs paramètres sur la ligne de commande.

- On pourra lui spécifier un paramètre optionnel '-a', auquel on attribura une valeur.
- L'aide sera affichée via l'option '-h' en parametre.
- On pourra activer, ou non, l'option '-x'.
- Le reste des paramètres sera listé.

Pour ce faire, chercher dans la page de man de *getopt* les options **optopt**, **optind** et **optarg**.

Exemple :

```
[mathieu@battousai mathieu]$ ./essai-getopt -a un deux trois -x quatre
= option ‘-x’ activée
= paramètre ‘-a’ présent = un
3 paramètres supplémentaires
    ->  deux
    ->  trois
    ->  quatre

[mathieu@battousai mathieu]$ ./essai -x  -a 2
= option ‘-x’ activée
= paramètre ‘-a’ présent = 2
0 paramètres supplémentaires

[mathieu@battousai mathieu]$ ./essai -x 1 -a 2 3 4
= option ‘-x’ activée
= paramètre ‘-a’ présent = 2
3 paramètres supplémentaires
    ->  1
    ->  3
    ->  4
```

La correction est à la section 'Exercice 3' du feuillet d'exercices.

2.3 Variables d'environnement : *getenv()*

Il est plus commode de consulter les variables d'environnement avec la fonction *getenv()*.

```
#include <stdlib.h>
char *getenv(const char *name);
```

2.3.1 Exemple

```
   /* getterm.c */

#include <stdlib.h>
#include <stdio.h>


int main(void) {
  char *terminal = getenv("TERM");
  printf("Le terminal est ");

  if (terminal==NULL)
    printf("inconnu\n");
  else
    printf("un %s\n",terminal);

  exit(EXIT_SUCCESS); // ou return 0;
}
```

Exercice : Ecrire un programme env.c qui affiche les valeurs des variables d'environnement qui lui sont indiquées en paramètre.
Exemple d'exécution :

```
[mathieu@battousai mathieu]$ ./env TERM LOGNAME PWD
TERM=xterm
LOGNAME=mathieu
PWD=/home/mathieu/esi/LABO
```

Voir aussi les fonctions :

```
int putenv(const char *string);
int setenv(const char *name, const char *value, int overwrite);
void unsetenv(const char *name);
```

qui permettent de modifier les variables d'environnement du processus courant et de ses fils. Il n'y a pas de moyen de modifier les variables du processus père.

2.4 La fonction exit() : Fin de programme

Un programme se termine généralement par un appel à la fonction *exit()* :

```
#include <stdlib.h>
void exit(int status);
```

Le paramètre **status** est le *code de retour* du processus. On utilisera de préférence les deux constantes *EXIT_SUCCESS et EXIT_FAILURE* définies dans *stdlib.h*.

3 Erreurs

3.1 errno, et perror()

La plupart des fonctions du système peuvent échouer pour diverses raisons. On peut alors examiner la variable **errno** pour déterminer plus précisément la cause de l'échec, et agir en conséquence.

```
#include <errno.h>
extern int errno;
```

La fonction *perror()* imprime sur la sortie d'erreurs standard un message décrivant la dernière erreur qui s'est produite, précédé par la chaîne **s**.

```
#include <stdio.h>
void perror(const char *s);
```

Enfin, la fonction *strerror()* retourne le texte du message d'erreur correspondant à un numéro.

```
#include <string.h>
char *strerror(int errnum);
```

3.1.1 Exemple

Ecrire un programme qui va changer les droits d'accès à des fichiers grâce à la commande système *chmod* (voir man 2 chmod).

Ce programme devra changer les droits du fichier (par exemple 0600), en verifiant que l'utilisateur executant notre programme a les droits de le faire, que l'on peut acceder à notre fichier, qu'il n'y a pas des problemes avec les liens symboliques (ELOOP), que le nom n'est pas trop long, et que le fichier existe. S'il y a un probleme, l'erreur sera affichée. (Utilisez errno, et strerror()).

Pour cela, voir dans la page de manuel de chmod :
- S_IREAD, S_IWRITE, S_IEXEC
- ELOOP, et d'autres options que vous devez trouver.

Pour la correction, voir dans le feuillet d'exercices à la section 'Exercice 4'.

4 Allocation dynamique : malloc(), free(), realloc(), strdup()

```
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

malloc() demande au système d'exploitation l'allocation d'un espace mémoire de taille supérieure ou égale à **size** octets. La valeur retournée est un pointeur sur cet espace (**NULL** en cas d'échec).

free() restitue cet espace au système. *realloc()* permet d'agrandir la zone allouée. Il est très fréquent de devoir allouer une zone mémoire pour y loger une copie d'une chaîne de caractères. On utilise pour cela la fonction *strdup()*

```
#include <string.h>
char *strdup(const char *s);
```

L'instruction `nouveau = strdup(ancien)` est approximativement équivalente à :

```
nouveau = strcpy( (char *)malloc(1+strlen(ancien)),ancien);
```

4.1 Exemple

La fonction *lireligne()* ci-dessous lit une ligne de l'entrée standard et retourne cette ligne dans un tampon d'une taille suffisante. Elle renvoie le pointeur NULL si il n'y a plus de place en mémoire.

```
#include <stdlib.h>
#include <stdio.h>
#define TAILLE_INI 16

char * lireligne(void)
{
   char * chaine, *nchaine;
   int taille_allouee, taille_courante;
   int c;

   chaine = (char *) malloc(TAILLE_INI);

   if (chaine == NULL) return(NULL);

   taille_courante = 0;
   taille_allouee = TAILLE_INI;
   while(1) {
       c = getchar();
       if ((c=='\n') || (c==EOF)) break;
       chaine[taille_courante++] = c;
       if(taille_courante == taille_allouee)
       {
           taille_allouee *= 2;
           nchaine = (char *) realloc(chaine, taille_allouee);
           if (nchaine == NULL) {
                free(chaine);
                return(NULL);
           }
           chaine=nchaine;
       }
   }
   chaine[taille_courante] = '\0';
   return(chaine);
}

int main(void)
{
   char * chaine;
   printf("tapez une grande chaîne\n");
   chaine = lireligne();
   if (chaine == NULL)
   {
     fprintf(stderr,"Plus de place en mémoire\n");
     exit(EXIT_FAILURE);
   }
   printf("%s\n",chaine);
```

```
   free(chaine);
   exit(EXIT_SUCCESS); // ou return 0;
}
```

5 Manipulation des fichiers

5.1 Flots standards

Il y a trois flots prédéclarés, qui correspondent à l'entrée et la sortie standards, et à la sortie d'erreur :

```
#include <stdio.h>
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

Quelques fonctions agissent implicitement sur ces flots :

```
int printf(const char *format, ...);
int scanf(const char *format, ...);
int getchar(void);
char *gets(char *s);
```

printf() écrit sur **stdout** la valeur d'expressions selon un format donné. *scanf()* lit sur **stdin** la valeur de variables.

5.1.1 Exemple

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

int main(void)
{
  char article[10];
  float prix;
  int quantite;
  printf("article, prix unitaire, quantité ?\n");
  scanf("%s %f %d",article,&prix,&quantite);
  printf("%d %s a %10.2f =  %10.2f\n",
         quantite,article,prix,prix*quantite);

  exit(EXIT_SUCCESS); // ou return 0;
}
```

scanf() renvoie le nombre d'objets qui ont pu être effectivement lus sans erreur.

gets() lit des caractères jusqu'à une marque de fin de ligne ou de fin de fichier, et les place (marque non comprise) dans le tampon donné en paramtre, suivis par un caractère

NUL. *gets()* renvoie finalement l'adresse du tampon. Attention, prévoir un tampon assez grand, ou (bien mieux) utilisez *fgets()*.

getchar() lit un caractère sur l'entrée standard et retourne sa valeur sous forme d'entier positif, ou la constante EOF (-1) en fin de fichier.

5.2 Opérations sur les flots : fopen(), fprintf(), fscanf()...

```
#include <stdio.h>
FILE *fopen(char *path, char *mode);
int fclose(FILE *stream);

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
```

fopen() tente d'ouvrir le fichier désigné par la chaîne **path** selon le mode indiqué, qui peut être :

- "r" (lecture seulement),
- "r+" (lecture et écriture),
- "w" (écriture seulement),
- "w+" (lecture et écriture, effacement si le fichier existe déjà),
- "a" (écriture à partir de la fin du fichier si il existe déjà),
- "a+" (lecture et écriture, positionnement à la fin du fichier si il existe déjà).

Si l'ouverture échoue, *fopen()* retourne le pointeur NULL.

```
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
```

Les trois premières fonctions ne diffèrent de *printf()*, *scanf()* et *getchar()* que par le premier paramètre, qui précise sur quel flot porte l'opération.

Comme *gets()*, *fgets()* lit une ligne de caractères pour la placer dans un tampon, mais :

- la longueur de ce tampon est précisée (ceci empêche les overflows)
- le caractère de fin de ligne (newline) est effectivement stocké dans le tampon.

5.2.1 Exemple

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

void compter (char nom[],FILE *f)
{
```

```
   int c,cars=0, lignes=0;
   while ((c=fgetc(f)) != EOF)
   {
      cars++;
      if (c=='\n') lignes++;
   };
   printf("%s: %d cars, %d lignes\n",nom,cars,lignes);
}

int main (int argc, char *argv[])
{
  char *nomfichier, *prog=argv[0];
  FILE *f;
  switch(argc) {
    case 1:
      compter("entrée standard",stdin);
      break;

    case 2:
      nomfichier = argv[1];
      f = fopen(nomfichier,"r");
      if (f == NULL) {
         fprintf(stderr,"%s: fichier %s absent ou illisible\n",
                 prog,nomfichier);
         exit(EXIT_FAILURE);
      }
      compter(nomfichier,f);
      fclose(f);
      break;

    default:
      fprintf(stderr,"Usage: %s [fichier]\n",prog);
      exit(EXIT_SUCCESS); // ou return 0;
   }
   exit(EXIT_SUCCESS); // ou return 0;
}
```

5.3 Positionnement : fseek(), feof(), ...

```
#include <stdio.h>
int feof(FILE *stream);
long ftell(FILE *stream);
int fseek(FILE *stream, long offset, int whence);
```

feof() indique si la fin de fichier est atteinte.
ftell() indique la position courante dans le fichier (0 = début).

fseek() déplace la position courante : si **whence** est SEEK_SET, la position est donnée par rapport au début du fichier, si c'est SEEK_CUR : déplacement par rapport à la position courante, si c'est SEEK_END : déplacement par rapport à la fin.

6 Manipulation des fichiers, opérations de bas niveau

La grande question : quelle est la différence entre les fonctions *fopen(), fread()*, ... et ces nouvelles fonctions (*open(), read()*, ... Sachez que ces fonctions font en fait, en gros, la même chose, sauf que par exemple, *fopen()* est la fonction ANSI alors qu'*open()* est la fonction du système.

Ainsi, *fopen()* sera bien plus portable que *open()*. L'inconvénient étant que *fopen()* est moins pratique d'utilisation que *open()*.

6.1 Ouverture, fermeture, lecture, écriture

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(char *ref, int mode, int perm);
```

Ouverture du fichier de référence (absolue ou relative à ".") *ref*.
Le *mode* d'ouverture est une conjonction des masques suivants :

```
O_RDONLY  /* ouvert en lecture */
O_WRONLY  /* ouvert en écriture */
O_RDWR    /* ouvert en lecture et écriture */
O_NDELAY  /* open non bloquant */
O_APPEND  /* mode append */
O_CREAT   /* open avec création de fichier */
O_EXCL    /* erreur à la création de fichier s'il existe deja */
```

Le paramètre **perm** n'a de sens qu'à la création du fichier, il permet de positionner les valeurs du champ **mode** de l'inode. Les droits effectivement positionnés dépendent de la valeur de **umask**. La valeur par défaut de umask est 0666 (valeur octale). La valeur de retour de *open()* est le numéro dans la table de descripteurs du processus qui a été utilisé

L'entier retourné par *open()* est un descripteur de fichier (-1 en cas d'erreur), qui sert à référencer le fichier par la suite.

```
#include <unistd.h>
#include <sys/types.h>
int close(int fd);
int read(int fd, char *buf, size_t count);
size_t write(int fd, const char *buf, size_t count);
```

close() ferme le fichier indiqué par le descripteur **fd**. Retourne 0 en cas de succès, -1 en cas d'échec.
read() demande à lire au plus **count** octets sur **fd**, à placer dans le tampon **buf**. Retourne le nombre d'octets qui ont été effectivement lus, qui peut être inférieur à la limite donnée pour cause de non-disponibilité (-1 en cas d'erreur, 0 en fin de fichier).
write() écrit sur le fichier les **count** premiers octets du tampon **buf**. Retourne le nombre d'octets qui ont été effectivement écrits, -1 en cas d'erreur.

6.1.1 Exemple

Créer un programme qui va pouvoir copier un fichier vers la sortie standard, et qui pourra copier un fichier vers un autre fichier, de telle sorte :

```
Usages:
      1: copie        (entrée-standard -> sortie standard)
      2: copie fichier      (fichier -> sortie standard)
      3: copie source dest  (source -> dest)
```

Il y aura la fonction erreur_fatale(), qui prendra la main en cas d'erreur, la fonction transfert(int entree, int sortie) qui copiera notre fichier vers la sortie standard ou un autre fichier. De plus, vous lui passerez aussi en paramètre la taille du tampon, afin de voir que celle ci influe fortement sur les performances.

Le principe :
- ouvrir le fichier
- le parcourir avec read, tout en copiant vers notre sortie
- fermer notre fichier.

Vous trouverez la correction sur le feuillet d'exercice, à la section 'Exercice 5'.

6.2 Suppression : remove()

```
#include <stdio.h>
int remove(const char *pathname);
```

Cette fonction supprime le fichier **pathname**, et retourne 0 en cas de succès (-1 sinon).

Exercice non corrigé : écrire un substitut pour la commande *rm*, avec gestions des erreurs.

6.3 Positionnement : lseek()

```
#include <fcntl.h>
off_t  lseek(int d, off_t offset, int direction)
```

lseek() permet de déplacer le curseur de fichier dans la **table des fichiers ouverts** du système.
- **offset** est un déplacement en octets,
- **d** le descripteur,
- **direction** une des trois macros L_SET, L_INCR, L_XTND.

1. **L_SET** : la nouvelle position est *offset* sauf si *offset* est supérieur à la taille du fichier, auquel cas la position est égale à la taille du fichier. Si l'offset est négatif, alors la position est zéro.
2. **L_INCR** : la position courante est incrémentée de *offset* place (même contrainte sur la position maximum et la position minimum).
3. **L_XTND** : Déplacement par rapport à la fin du fichier, cette option permet d'augmenter la taille du fichier (ne pas créer de fichiers virtuellement gros avec ce mécanisme, ils posent des problèmes de sauvegarde).

La valeur de retour de lseek est la nouvelle position du curseur dans le fichier ou -1 si l'appel a échoué.

6.4 Informations sur les fichiers, répertoires : stat(), fstat()...

```
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
```

Ces fonctions permettent de connaître diverses informations sur un fichier désigné par un chemin d'accès (*stat()*) ou par un descripteur (*fstat()*).

6.4.1 Exemple

Ecrire un programme qui renvoie la taille d'un fichier en Ko, en utilisant la fonction *stat()*. (voir man 2 stat).
Pour la correction, voir à la section 'Exercice 6'.

6.5 Parcours de répertoires

Le parcours d'un répertoire, pour obtenir la liste des fichiers et répertoires qu'il contient, se fait grâce aux fonctions :

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
int closedir(DIR *dir);
void rewinddir(DIR *dir);
void seekdir(DIR *dir, off_t offset);
off_t telldir(DIR *dir);
```

Exercice non corrigé : écrire une version très simplifiée de la commande ls, qui liste les fichiers.

7 Pipes et FIFOs

Les pipes et les FIFOs permettent la communication entre processus

7.1 Pipes

```
#include <unistd.h>
int pipe(int filedes[2]);
```

Ouvre un *pipe* (tuyau de communication). On lit dans **filedes[0]** ce qu'on a écrit dans **filedes[1]**.

7.2 Pipes depuis/vers une commande : popen(), pclose()

```
#include <stdio.h>
FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

popen() lance la commande décrite par la chaîne *command* et retourne un flot.
Si type est "r" le flot retourné est celui de la sortie standard de la commande (on peut y lire). Si type est "w" c'est son entrée standard. (man popen). La fonction *pclose()* referme ce flot.

```
//Exemple

#include <stdio.h>
#include <unistd.h>

int main(void)
{
      FILE *cmd;
      char buffer[1024];

      if((cmd = popen("/bin/ps aux", "r")) == NULL)
      perror("popen");

      while(!feof(cmd)) {
      fgets(buffer, 1023, cmd);
      puts(buffer);
      }
      return 0;
}
```

7.2.1 Exemple

Ecrire un programme qui va envoyer le résultat d'un "ls -l" par mail à un correspondant, via le programme 'mail'.

Il faut utiliser *snprintf()*, et *popen()*. La correction est dans le feuillet d'exercice à la section 'Exercice 7'.

7.3 FIFOs : mkfifo()

Les *FIFOs* sont également appelés ”tuyaux nommés”. Ce sont des objets visibles dans l’arborescence des fichiers et répertoires.À la différence des *pipes*, les *FIFOs* sont partageables par des processus appartenant à des utilisateurs différents.

```
#include <stdio.h>
int mkfifo (const char *path, mode_t mode);
```

La fonction *mkfifo()* crée un *FIFO* ayant le chemin indiqué par **path** et les droits d’accès donnés par **mode**. Si la création réussit, la fonction renvoie 0, sinon -1. Exemple :

```
if(mkfifo("/tmp/fifo.bla",0744) == -1)
    perror("mkfifo");
```

Les *FIFOS* sont ensuite utilisables par *open(), read(), write(), close()*.

8 Processus

8.1 Les fonctions fork() et wait()

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

pid_t fork(void);
pid_t wait(int *status)
```

La fonction *fork()* crée un nouveau processus (**le fils**) semblable au processus courant (**père**). La valeur renvoyée n’est pas la même pour le fils (0) et pour le père (numéro de processus du fils). -1 indique un échec.

La fonction *wait()* attend qu’un des processus fils soit terminé. Elle renvoie le numéro du fils, et son status (voir *exit()*) en paramètre passé par adresse.

Attention : Le processus fils **hérite** des descripteurs ouverts de son père. Il convient que chacun des processus ferme les descripteurs qui ne le concernent pas.

8.1.1 Exemple

```
/*
   Exemple à deux processus reliés par un tuyau
   - l’un envoie abcdef...z 10 fois dans
      le tuyau
   - l’autre écrit ce qui lui arrive du tuyau
       sur la sortie standard, en le formattant
      (20 caractères par ligne).
*/
```

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

void genere(int sortie)
{
  char alphabet[26];
  int k;

  printf("dans genere\n");

  for (k=0; k<26; k++)
      alphabet[k] = 'a'+k;

  for (k=0; k<10; k++)
      if (write(sortie,alphabet,26) != 26)
      {
         perror("write");
         exit(EXIT_FAILURE);
      };
  close(sortie);
}


int lire(int fd, char *buf, size_t count)
{
  /* lecture, en insistant pour remplir le buffer */
  int dejalus=0, n;
  printf("dans lire\n");
  while (dejalus < count) {
      n = read(fd,buf+dejalus,count-dejalus);
      if (n == -1) return(-1);

      if (n ==  0) break; /* plus rien à lire */
      dejalus += n;
  };

  return(dejalus);
}


void affiche(int entree)
```

```
{
  char ligne[21];
  int nb,numligne=1;

printf("dans affiche\n");
  while( (nb=lire(entree, ligne,20)) > 0)
  {
      ligne[nb] = '\0';
      printf("%3d %s\n", numligne++, ligne);
  };

  if(nb < 0) { perror("read"); exit(EXIT_FAILURE); };

  close(entree);
}

int main(void)
{
  int fd[2], status;
  pid_t pid;

  if (pipe(fd) != 0)
  { perror("pipe"); exit(EXIT_FAILURE); };

  if ((pid=fork()) < 0)
     {perror("fork"); exit(EXIT_FAILURE); };

  if(pid == 0)
  {
      printf("pid == 0 !\n");
      close(fd[0]);
      close(1);
      genere(fd[1]);
      exit(EXIT_SUCCESS); // ou return 0;
  }
  else
  {
  printf("pid != 0\n");
      close(0);
      close(fd[1]);
      affiche(fd[0]);
  };

  wait(&status);
  printf("status fils = %d\n", status);
  exit(EXIT_SUCCESS); // ou return 0;
```

```
}
```

8.2 La fonction waitpid()

La fonction *waitpid()* permet d'attendre l'arrêt d'un des processus fils désigné par son **pid** (n'importe lequel si **pid** = -1), et de récupérer éventuellement son code de retour. Elle retourne le numéro du processus fils.
L'option **WNOHANG** rend *waitpid()* non bloquant (qui retourne alors -1 si le processus attendu n'est pas terminé).

```
#include <sys/types.h>
#include <sys/wait.h> pid_t

waitpid (pid_t pid, int *status, int options);
```

8.2.1 Exemple

```
int pid_fils;
int status;

if( (pid_fils = fork()) != 0) {
      code_processus_fils();
      exit(EXIT_SUCCESS); // ou return 0;
};
 ...

if (waitpid(pid_fils,NULL,WMNOHANG) == -1)
    printf("Le processus fils n'est pas encore terminé\n");
...
```

8.3 La fonction exec()

```
#include <unistd.h>
int execv (const char *FILENAME, char *const ARGV[])
int execl (const char *FILENAME, const char *ARG0,...)
int execve(const char *FILENAME, char *const ARGV[], char *const ENV[])
int execle(const char *FILENAME, const char *ARG0,...char *const ENV[])
int execvp(const char *FILENAME, char *const ARGV[])
int execlp(const char *FILENAME, const char *ARG0, ...)
```

Ces fonctions font toutes quasiment la même chose : lancer l'exécution d'un fichier *à la place du processus courant*. Elles diffèrent par la manière d'indiquer les paramètres.

- *execv()* : les paramètres de la commande sont transmis sous forme d'un tableau de pointeurs sur des chaînes de caractères (le dernier étant NULL). Exercice : Faire un programme qui va nous demander un nom de fichier source en C, et va appeller lui

même gcc pour compiler. Vous trouverez le corrigé dans le feuillet d'exercices à la section 'Exercice 8'.
- *execl()* reçoit un nombre variable de paramètres, le dernier est NULL. Faire le même programme que ci-dessus, sauf que vous devez utilisez *execl()*. (corrigé à la section 'Exercice 9') (voir man execl).
- *execve()* et *execle()* ont un paramètre supplémentaire pour préciser l'environnement.
- *execvp()* et *execlp()* utilisent la variable d'environnement **PATH** pour localiser l'executable à lancer. on pourrait donc écrire simplement :
  `execlp("gcc","gcc",fichier,"-o",prefixe,NULL);`

8.4 Numéros de processus : getpid() et getppid()

```
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

getpid() permet à un processus de connaître son propre numéro, et *getppid()* celui de son père.

8.5 Programmation d'un daemon

Les *daemons* (démons) sont des processus qui tournent normalement en arrière-plan pour assurer un service. Pour programmer correctement un daemon, il ne suffit pas de faire un *fork()*, il faut aussi s'assurer que le processus restant ne bloque plus de ressources. Par exemple il doit libérer le terminal de contrôle du processus, revenir à la racine (sinon il empèchera le démontage éventuel du système de fichiers à partir duquel il a été lancé etc.)

8.5.1 Exemple

Voici la fonction *devenir_daemon*. Cette fonction va utiliser les fonctions *setsid()* (man setsid), *chdir()* (man chdir).

```
int devenir_daemon(void)
{
  int fd;

  /* Le processus se dédouble, et le père se termine */
  if (fork() != 0)
  exit(EXIT_SUCCESS); // ou return 0;

  /*
   *le processus fils devient le leader d'un nouveau
   * groupe de processus
   */
```

```
  setsid();

  /*
   * le processus fils crée le processus démon, et
   * se termine
   */
  if (fork() !=0) exit(EXIT_SUCCESS); // ou return 0;

  /* le démon déménage vers la racine */
  chdir("/");

  /* l'entrée standard est redirigée vers /dev/null */
  fd = open("/dev/null",O_RDWR);
  dup2(fd,0);
  close(fd);

  /* et les sorties vers /dev/console */
  fd = open("/dev/console",O_WRONLY);
  dup2(fd,1);
  dup2(fd,2);
  close(fd);
}
```

9 Signaux

Dans cette section sont présentés les signaux Unix classiques, qui ont un inconvénient majeur : ils ne fonctionnent pas de la même façon d'un Unix à l'autre. On préfèrera les signaux Posix.

9.1 Les fonctions signal(), kill()

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

void (*signal(int signum, void (*handler)(int)))(int);
```

La fonction *signal()* demande au système de lancer la fonction **handler** lorsque le signal **signum** est reçu par le processus courant. La fonction *signal()* renvoie la fonction qui était précédemment associée au même signal.

Il y a une trentaine de signaux différents, parmi lesquels :
- SIGINT (program interrupt, émis par Ctrl-C),
- SIGTST (terminal stop, Ctrl-Z)
- SIGTERM (fin, obtenu par la commande kill
- SIGFPE (erreur arithmétique),
- SIGALRM (fin de delai, voir fonction *alarm()*),

– etc.

La fonction *handler()* prend en paramètre le numéro du signal reçu, et ne renvoie rien.

9.1.1 Exemple

Faire un programme qui va pouvoir recevoir et utiliser les signaux **SIGTSTP** (quand on fait un controle-z lors du programme), **SIGINT** (controle-c), et **SIGTERM** (si on kill le processus).
Pour ce faire, vous allez utiliser la fonction *signal()*, que vous appellerez plusieurs fois[2] au debut de votre main, créant ainsi un nouveau 'signal handler' pour votre programme. Vous devez créer la fonctiontraitement(), qui saura quoi faire suivant les signaux. Utilisez les pages de manuel de *getpid()*, *signal()*.

Vous trouverez la correction dans le feuillet d'exercices, à la section 'Exercice 10'.

9.2 La fonction kill()

```
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>

int kill(pid_t pid, int sig);
```

La fonction *kill()* envoie un signal à un processus. (man 2 kill).

9.3 La fonction alarm()

```
#include <unistd.h>

long alarm(long delai);
```

La fonction *alarm()* demande au système d'envoyer un signal **SIGALRM** au processus dans un délai fixé (en secondes). Si une alarme était déjà positionnée, elle est remplacée. Un délai nul supprime l'alarme existante. (man alarm)

10 Remerciements

Ce cours s'appuie sur les polycopiés de Dominique Revuz. Si par malheur vous trouviez des fautes, seul le Labo-Unix sera en faute.

[2] N fois le nombre de signaux que vous voulez gerer, donc ici 3 fois, avec les parametres que l'on vous a demandé (man signal).

11 GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom : to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation : a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals ; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

11.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

11.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

11.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts : Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material

on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

11.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version :

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.

- Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another ; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

11.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you in-

clude in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History” ; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

11.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

11.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

11.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the

original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

11.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http ://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM : How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page :

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation ; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST” ; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.