Programmation système avancée
Devoir maison le 20 mai 2020, durée 24h, le début 9h00

Documents autorisés et consignes

Vous devez impérativement signer est joindre à vos copies la déclaration fournie sur moodle qui précise les modalités de devoir. Vous pouvez soit compléter la déclaration, imprimer, signer, soit faire une copie manuscrite. Après avoir signé, vous faites une photo que vous devez joindre à votre copie sur moodle (ou envoyer par mail).

Le sujet comporte 5 pages. Pour de raison de formatage ou l'insertion de dessin une longue zone blanche à la fin d'une page ne signifie pas la fin du sujet, regardez s'il n'y a pas de page suivante.
Documents autorisés : voir la charte de l'examen. L'utilisation de pages man est bien sûr autorisée.

Si vous faites les réponses sous forme manuscrite tâchez d'écrire de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin de blocs de code (fin de boucles, etc.).

Il est inutile d'écrire les `#include` mais si vous avez déjà des includes dans vos fichier `*.c` ne les supprimez pas.

1 Projection mémoire

Exercice 1 :

Il est facile de tronquer la longueur de fichier en supprimant le octets à la fin du fichier (les fonctions `truncate` et `ftruncate`). Supprimer une zone au milieu de fichier nécessite un peu de travail.
Écrire le programme (la fonction `main`) tel qu'en compilant on obtient la commande `tronquer` telle que

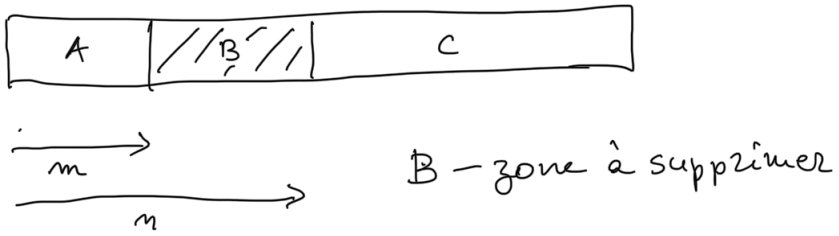
```
./tronquer m n file
```

prend en paramètre deux entiers non-négatifs `m` et `n` et un chemin `file` vers un fichier.
Le programme supprime dans le fichier `file` une zone de `(n - m)` octets à partir de la position `m`.
Notez que le programme ne crée pas un nouveau fichier mais supprime une partie de fichier existant.
Plus précisément,

- si `m >= n` alors le programme ne fait rien,
- si `m >= longueur de fichier` le programme ne fait rien,
- si `m < n < longueur de fichier` on supprime `n - m` octets à partir de la position `m`,

– si `m < longueur de fichier <= n` on supprime tous les octets à partir de la position `m` jusqu'à la fin du fichier.

Le dessin suivant présente la zone à supprimer quand `m < n < longueur de fichier`.



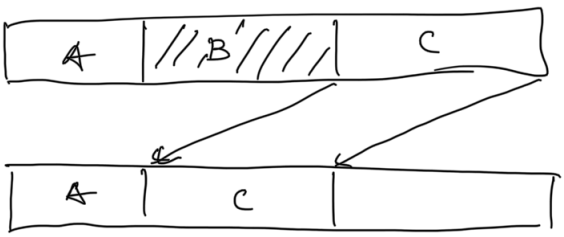
Exemples.
`./tronquer 0 10 toto.txt` supprimera le 10 premiers octets de `toto.txt` (où tous les octets si le fichier possède moins de 10 octets.)
`./tronquer 10 100 toto.txt` supprimera le 90 octets à partir de la position 10 (où tous les octets à partir de la position 10 si `toto.txt` contient moins de 100 octets).
Vous devez écrire un programme basé sur la projection de fichier en mémoire :

(1) faire la projection en mémoire,

(2) dans la projection, déplacer les octets de la zone qui se trouve après la zone à supprimer (de préférence sans boucle), voir le dessin ci-dessous,

(3) synchroniser la projection avec le fichier,

(4) tronquer la partie à la fin de fichier pour ajuster la longueur de fichier.



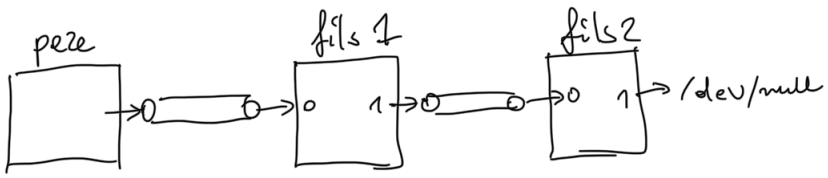
2 exec et redirections

Exercice 2 :

On suppose que après la compilation le programme suivant donne un exécutable `aoub`.

```c
/* aoub.c */
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
int main(int argc, char **argv){
  if( argc != 3 ){
    fprintf( stderr, "usage : %s mot_a mot_b\n", argv[1]);
    exit(1);
  }
  int terminal = open("/dev/tty", O_WRONLY);
  char c;
  while( read( 0, &c, sizeof(c) ) > 0 ){
    if ( c == 'a' )
      write( terminal , argv[1] , strlen(argv[1]) );
    else if( c == 'b' )
      write( terminal , argv[2] , strlen(argv[2]) );
    else
      continue;
    char n = '\n';
    write(terminal, &n , sizeof(n) );
    write( 1 , &c, sizeof( c ) ) ;
  }
}
```

Le code de ce programme est disponible sur moodle.
Comme vous pouvez constater le programme `aoub` écrit sur le terminal soit `argv[1]` soit `argv[2]` en fonction de la lettre lue sur le descripteur 0. Le programme écrit ensuite cette lettre sur le descripteur 1. Le terminal est ouvert explicitement avec `open(/dev/tty, O_RDWR )` parce que, à cause de redirection possible, le descripteur 1 ne correspond peut-être pas au terminal (ce sera le cas dans notre exercice).
Le but de l'exercice est écrire le programme, appelons-le `pingpong` qui créa deux processus fils, `fils1` et `fils2`.
Le trois processus, `pere`, `fils1`, `fils2` communiquent par des tubes anonymes comme montre ce dessin :



Donc

(1) le processus père écrit dans le premier tube et les octets envoyés sont récupérés par le `fils1`,

(2) `fils1` écrit dans le deuxième tube et les octets écrits seront récupérés par `fils2`.

Les deux processus fils doivent exécutés le programme `aoub` mais avec les paramètres de `main` différents :

- `fils1` exécutera [1] la commande `./aoub ping titi`,
- `fils1` exécutera la commande `./aoub pong toto`,

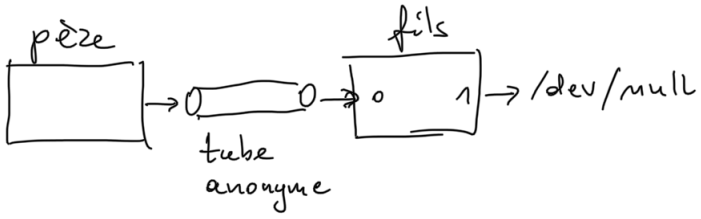
Avant de lancer l'exécution de `./aoub` par les processus fils il faut faire les redirections appropriés :

- quand `fils1` lit sur le desripteurs 0 il lira sur le tube qui le connecte à son père et quand il écrit sur le descipteur 1 il écrira dans le tube qui le connecte au `fils2`,
- quand `fils2` lit sur le descripteur 0 il lira sur le tube qui le connecte avec son frère et quand il écrit sur le descripteur 1 il écrira dans le fichier `/dev/null` [2]

Après avoir mis en ouvre les liens de communication décrites ci-dessus le processus père envoie une suite infinie aléatoire de lettres `a` et `b` vers son premier fils.

Indication. `random()&01` produit une suite aléatoire de 0 et 1.

Remarque. Si vous avez du mal à mettre tout en oeuvre avec deux fils essayer la configuration avec un seul fils qui exécutera la commande `./aoub ping pong` :



Cette solution rapporte moins de points que la solution avec deux fils.

3 Les verrous de fichiers

Dans un fichier on stocke les structures

```
typedef struct{
  unsigned numero_compte;
  long     solde;
} compte;

```

Chaque structure est composée d'un numéro de compte et d'un solde.

Exercice 3 :

Écrire la fonction

```
int ajouter_compte(const char *comptes, unsigned numero, long sld)

```

1. grâce à exec de votre choix
2. Vous ouvrirez `/dev/null` en écriture comme n'importe quel autre fichier. `/dev/null` est un puits, tout ce qui y est écrit disparaît sans trace.

qui ajoute un nouveau compte `numero` avec les solde `sld` à la fin du fichier `comptes`. La fonction renvoie 0 en cas de succès et -1 si l'opération échoue pour une raison quelconque.

Exercice 4 :

Écrire la fonction

```
1 long   debiter( const char *comptes, unsigned numero, long val)
2
```

qui ajoute dans le compte `numero` la somme `val`. On suppose que la fonction peut être utilisée sur le même fichier `comptes` par plusieurs processus en même temps. En posant les verrous appropriés il faut s'assurer que l'opération se déroule de façon correcte.
On préfère les verrous les moins contraignants possibles pour assurer le maximum de parallélisme. La fonction renvoie le nouveau solde.