TP 5 – Mini-projet : Tubes et Fichiers

⟶ *Vademecum* : je compile, je vérifie que les exe n'ont pas changé de nom dans les `execl`, je donne un usage, je mets des messages d'erreurs explicites, je fais un *readme* pour indiquer quoi lancer s'il y a plusieurs fichiers ⟵

Mini-projet :

- ☞ Cet exercice sera à déposer sur ARCHE pour le **jeudi 04 mai 2023 avant 23h55**.
- ☞ Outre le programme lui-même, la qualité de la programmation, des commentaires et l'affichage lors de l'exécution du programme est important
- ☞ Vous travaillez pour vous (et notamment pour le TP6 noté, et le semestre qui suivra)

L'objectif final du programme est de réussir à décoder et à trouver le fichier qui vous concerne ! En effet 90 fichiers placés dans un dossier contient un message et vous devez retrouver celui qui contient votre nom. Mais les fichiers ont été cryptés avec la technique classique de décalage de lettre (par exemple A vaut C, B vaut D, C vaut E...). Les fichiers seront lus et tous les décalages de lettres devront être testés et au final le résultat sera affiché à l'écran. La transmission des données se fera par des tubes anonymes. Suivez progressivement les étapes indiquées : il y a 3 objectifs.

Premier objectif : décrypter un fichier

Les fichiers cryptés ont un format particulier avec un en-tête.

Exemple de fichier :
`CRint1int2XYZXYZXYZXYZXYZXYZXYZMESSAGEMESSAGEMESSAGEXYZXYZXYZXYZ`
int1 et int2 sont des entiers en binaire, XYZ des caractères de remplissage et MESSAGE le message réel.

Vous trouverez des fichiers de test dans le fichier zip fourni. Ils se nomment *Testxxx.cod*. Utilisez ces fichiers pour tester votre programme. Le répertoire *MessagesCodes* contient les fichiers pour l'objectif final.

- Réalisez un programme qui teste l'existence d'un fichier (donné en argument), l'ouvre et vérifie qu'il s'agit bien d'un fichier crypté : c'est-à-dire que les 2 premiers octets sont 'C' et 'R'. Utilisez les fichiers Testxxx.cod pour vérifier votre programme. Le fichier Test_sansCR.cod doit conduire à l'affichage d'un message d'erreur.

> **aide C :**
> ☞ N'oubliez pas de fermer le fichier.

- Le fichier a une structure particulière car l'entête contient aussi (après C et R) des informations : un entier (en binaire) qui donne la taille du message à décrypter et un entier (en binaire) qui donne le nombre d'octets avant le message hors en-tête (en effet des caractères ont été insérés avant le message réel). Retrouvez ces informations en les affichant. Vous procéderez en lisant int par int avec la fonction `read` (et non plus octet par octet). Des messages d'erreurs devront indiquer si ces informations sont manquantes (c'est le cas pour le fichier Test_sansdebut.cod) : *on ne peut pas savoir si ces entiers sont réellement manquants puisque la fonction* `read` *lira quelque chose et récupérera de toute façon un entier. On peut alors tester la cohérence de la valeur : par exemple tester si elle est positive ou si elle ne dépasse pas la taille du fichier par exemple ou encore tester si la taille du message plus le nombre d'octets avant le message dépassent la taille du fichier.*

- Maintenant, avec la fonction `lseek` avancez jusqu'à début du message réel et lisez/affichez le message lettre par lettre.

> **aide C :**
> ☞ La fonction `lseek` sert à se déplacer dans le fichier, à partir d'un point donné (début SEEK_SET, courant SEEK_CUR, fin SEEK_END) et renvoie la valeur de la nouvelle position (si besoin).

- Créez maintenant un nouveau processus et un tube anonyme de communication entre les deux programmes (utilisez `execl`) et envoyez les lettres dans ce tube (au lieu de les afficher dans le processus parent). De l'autre côté le programme recevant ces lettres les décalera de 2 lettres (prenez le fichier de test TestAvautC.cod qui contient un message décalé de 2 lettres A=C). Le programme recevant les lettres affichera le message. *Pour résumer : c'est le processus parent qui ouvre, vérifie le fichier et avance jusqu'au message, et le processus enfant qui ne fait que décoder le message envoyé dans le tube par le processus parent.*

> **aide au décryptage**
> ☞ L'alphabet utilisé est constitué de 26 lettres en majuscule (de A à Z) et de l'espace. Donc voici le tableau de correspondance qu'il faut obtenir pour décrypter les lettres (ici pour un décalage de 2 lettres : A initial a été codé en C, on fait donc l'inverse

lettres codées : `<ABCDEFGHIJKLMNOPQRSTUVWXYZ >`

pour décoder) : ⇓

lettres décodées : `<Z ABCDEFGHIJKLMNOPQRSTUVWXY>`

☞ Une possibilité simple pour gérer le décalage : Soustraire le décalage à la lettre (par exemple 'B'-2) puis vérifier si elle est entre 'A' et 'Z', si ce n'est pas le cas (elle est donc avant 'A') ajustez le résultat pour revenir entre 'A' et 'Z' sans oublier ' '. [si la lettre de départ n'est pas entre A ou Z ni un espace ne pas faire de traitement mais gardez le symbole. Il s'agit peut-être d'une virgule ou d'un point]

- Intégrez maintenant le fait que le programme qui reçoit les lettres, vérifie si le message contient un mot particulier. Pour le fichier TestAvautC.cod, cherchez le mot TEST. Un façon de procéder consiste à sauvegarder au fur-et-à-mesure les lettres dans un tableau puis lorsque le caractère décodé est un espace, ajoutez '\0' dans le tableau et comparez le tableau (qui est alors une chaine de caractères) avec le mot à trouver, puis repositionnez le tableau pour le mot suivant.

  Exemple : "UN TEST POUR TESTER" (mot à trouver TEST)

```
U -> <U>
N -> <UN>
  -> <UN\0>  (comparaison avec strcmp --> faux)
T -> <T>
E -> <TE>
S -> <TES>
T -> <TEST>
  -> <TEST\0> (comparaison avec strcmp --> vrai)
P -> <P>
O -> <PO>
...etc...
```

  Il est possible aussi d'utiliser des fonctions de recherche de sous-chaines mais attention on veut un mot entier (ici TESTER ne doit donc pas être considérer comme contenant TEST)

- Maintenant, faites passer la valeur du décalage comme argument du deuxième programme (pour le rendre plus générique). La valeur sera donc envoyée par le `execl`.
- Dans le parent, pour savoir si l'enfant a trouvé le bon message, utilisez la valeur de retour du processus enfant pour indiquer la valeur du décalage (0=pas trouvé). Le parent affichera alors le nom du fichier et la valeur du décalage après la mort du processus enfant.

Rappel

☞ Le parent devra attendre la mort du processus enfant avec la fonction `wait` et WEXITSTATUS pour récupérer la valeur de retour. Voir le TP2.

- Avant de passer à la suite assurez-vous que :
  - le programme parent a pour argument un nom de fichier, un mot à rechercher et le décalage que l'on veut tester (ce dernier argument sera supprimé par la suite),
  - le programme parent affiche le nom du fichier et la valeur du décalage.
  - le programme enfant a pour argument le mot à rechercher, une valeur de décalage et le descripteur du tube pour la lecture,
  - le programme enfant affiche le message **s'il a trouvé le mot** (sinon rien),
  - le programme parent fonctionne avec les fichiers de test fournis (notamment pour les messages d'erreur).

Réalisez le deuxième objectif (si et seulement si ce qui précède fonctionne) :

- Le parent (le premier programme) doit maintenant générer autant de processus enfant que de décalage possible (soit 25, A ne vaut pas A ni l'espace) et aussi créer le même nombre de tube (1 par enfant). Le parent, lorsqu'il lira une lettre du message, l'enverra dans tous les tubes. Vous utiliserez alors un tableau de descripteur par exemple descr[NB][2] avec NB le nombre de processus/tubes créés. Exemple : descr[4][1] est le descripteur pour l'écriture du tube 4.

> **Aide**
> ☞ N'oubliez pas de fermer tous les bouts de tube qui ne vous servent pas.
> ☞ Le parent devra maintenant attendre la mort de tous les processus enfants avec une boucle (sur le nombre de processus) et la fonction `wait`. Le parent affiche toujours le nom du fichier et la valeur du décalage (lorsqu'elle est non nulle).

- Avant de passer à la suite assurez-vous que :
  - le programme parent a pour argument un nom de fichier et un mot à rechercher,(il n'y a plus besoin de la valeur de décalage)
  - le programme parent affiche le nom du fichier et la valeur du décalage.
  - le programme enfant a pour argument le mot à rechercher, une valeur de décalage et le descripteur du tube pour la lecture,
  - le programme enfant affiche le message s'il a trouvé le mot,
  - le programme parent fonctionne avec les fichiers de test fournis (notamment pour les messages d'erreur).

Réalisez l'objectif final (si et seulement si ce qui précède fonctionne) :

- Réalisez un troisième programme qui parcourt un répertoire et, pour chaque fichier, lance le programme parent précédent (en utilisant `fork` et `execl`). Il y aura donc autant de processus à lancer que de fichier (qui eux même lancent 25 processus !). Le programme prend en

arguments le nom du répertoire et le mot à retrouver. Le mot à retrouver est votre NOM en majuscule (pour faciliter le décodage tous les espaces, tirets ou apostrophes ont été supprimés : exemple le nom "De La Maison d'En Face-Face" est DELAMAISONDENFACEFACE).

- À la fin, le programme doit attendre la mort de ses processus enfants et indiquer si le message a été décrypté ou non (utilisez `wait` et la valeur de retour des parents : 0 ou 1 par exemple). Normalement un des enfants a affiché le numéro du fichier (et le décalage) et un processus petit-enfant a affiché le message décrypté. Si aucun enfants n'a rien trouvé, le programme doit mettre un message (ce qui permet de savoir si le programme est terminé ou non ;-)).
- Testez sur les 90 fichiers et trouvez le vôtre ! !