IUT Belfort-Montbéliard
Département Informatique

Examen final

28 octobre 2021

Durée : 2 heures. Documents autorisés.

Indications

- répondre aux questions de manière **précise** et **concise** ; numéroter les **copies**.
- les réponses aux questions de programmation seront données en langage C ; la liste complète des lignes `#include <...>` n'est pas nécessaire.

1 Entrées / sorties

Le but de l'exercice est d'écrire une commande *xxd* qui permet d'afficher le contenu d'un fichier en hexadécimal (base 16) et en texte, comme dans l'exemple ci-dessous.

```
00000000: 2550 4446 2d31 2e35 0a25 d0d4 c5d8 0a34  %PDF-1.5.%.....4
00000010: 2030 206f 626a 0a3c 3c0a 2f4c 656e 6774   0 obj.<<./Lengt
00000020: 6820 3236 3236 2020 2020 2020 0a2f 4669  h 2626      ./Fi
00000030: 6c74 6572 202f 466c 6174 6544 6563 6f64  lter /FlateDecod
00000040: 650a 3e3e 0a73 7472 6561 6d0a 78da 9d59  e.>>.stream.x..Y
00000050: 596f dc46 127e d7af 1860 1fc4 59cc d0ec  Yo.F.~...‘..Y...
00000060: e6bd 592c                                ..Y,
```

Chaque ligne est composée de :

- L'adresse depuis le début du fichier, en base 16 sur 8 chiffres, suivie d'un double-point.
- Une espace, puis les valeurs de 16 octets en base 16 sur deux chiffres, avec une espace pour séparer tous les groupes de deux octets.
- Deux espaces, puis les caractères correspondant aux octets. Si le caractère n'est pas imprimable, il est remplacé par un point (.).
- Si la taille du fichier n'est pas un multiple de 16, la dernière ligne est tronquée, comme dans l'exemple.

1.1 Écriture d'une ligne

Écrire une fonction `afficher_ligne()` permettant d'afficher une ligne telle que décrite ci-dessus. Le paramètre *adresse* est la valeur de l'adresse à afficher en début de ligne ; *tampon* est un tableau contenant les valeurs des *taille* octets à afficher (16 au maximum).

```
void afficher_ligne(int adresse, unsigned char tampon[], int taille);
```

Indication Pour déterminer si un caractère est imprimable ou non, on utilisera la fonction `int isprint(int c);` qui retourne une valeur différente de zéro si le caractère *c* est imprimable, et zéro sinon.

1.2 Écriture de l'entrée standard

Écrire un programme complet lisant les données sur son entrée standard, puis les affichant comme présenté dans l'introduction. Vous utiliserez la fonction `afficher_ligne()` définie précédemment.

1.3 Avec un fichier quelconque

Donnez les modifications à apporter au programme précédent pour que, si un nom de fichier est donné sur la ligne de commande, c'est le contenu du fichier qui est affiché au lieu de l'entrée standard.

2 Des processus

(a) Dans un système d'exploitation, qu'est-ce qu'on appelle un processus ? Quelles sont les différentes manières pour un processus de terminer ?

(b) Donner les trois principaux états possibles au cours de la vie d'un processus. Faire un schéma décrivant les transitions entre les états, sans oublier d'y faire figurer la création et la terminaison du processus.

(c) Dans un système Unix, qu'est-ce qu'un processus « zombie » ? Compléter le schéma précédent pour y faire figurer l'état « zombie ».

3 Des signaux

Dans un programme, on souhaite que le message « Au revoir ! » soit affiché lorsque l'utilisateur utilise la combinaison de touches Ctrl-C. Pour cela, on décide d'intercepter le signal correspondant : SIGINT.

(a) Écrire la fonction de gestion du signal qui affiche le message « Au revoir ! » sur la sortie standard, puis termine le processus avec le code de retour 0.

(b) Donner l'extrait de code à utiliser pour installer la fonction en tant que gestionnaire du signal SIGINT, en utilisant la fonction `signal()`.

(c) Donner l'extrait de code à utiliser pour installer la fonction en tant que gestionnaire du signal SIGINT, en utilisant la fonction `sigaction()`.

(d) Dans le cas général, quelle fonction faut-il privilégier entre `signal()` et `sigaction()` ? Pourquoi ?

4 Des redirections

On souhaite exécuter deux commandes en faisant en sorte que la deuxième commande reçoive, sur son entrée standard, la sortie de la première commande. Pour cela, on écrit un programme `redir` avec la syntaxe suivante :

```
redir commande1 [arguments...] -- commande2 [arguments...]
```

Avec un fichier temporaire. Dans un premier temps, on décide d'utiliser un fichier temporaire, d'exécuter la première commande en redirigeant sa sortie standard dans le fichier puis, lorsque la première commande est terminée, d'exécuter la deuxième commande en redirigeant son entrée depuis le fichier.

Donner une implémentation possible pour le programme `redir`.

Indications importantes

- Le programme devra se terminer après exécution de la deuxième commande.
- Le fichier temporaire sera créé avec la fonction `FILE *tmpfile(void)` qui retourne un descripteur de fichier correctement ouvert en lecture et écriture, ou NULL en cas d'erreur.
- Les redirections seront mises en place en utilisant les fonctions de bas niveau. La fonction `int fileno(FILE *stream)` permet d'obtenir le descripteur de fichier bas niveau correspondant au descripteur haut niveau donné.
- Lorsque la première commande est terminée, et avant d'exécuter la deuxième commande, appeler la fonction `void rewind(FILE *stream)` pour remettre le curseur de lecture au début du fichier.
- Enfin, pour simplifier, l'analyse de la ligne de commande pourra être remplacée par la déclaration des variables nécessaires et un simple commentaire décrivant précisément leur état.

Question subsidiaire : avec un tube. Modifier le programme précédent en utilisant un tube entre les deux processus à la place du fichier temporaire.