Système et Réseaux : projet Wikicomptines

avril 2024 — Pierre Rousselin

1 Introduction et consignes générales

Le but du projet Wikicomptines est de permettre à des utilisateurs du monde entier de partager des comptines sans sortir de leur terminal chéri.

Ce travail doit être fait seul.

Le point de départ doit être l'ensemble des fichiers fournis avec ce sujet sur `moodle`.

Vous rendrez votre travail sur le dépôt moodle avant le 17 mai 23h59.

— Vous devez fournir votre code complet avec un Makefile mis à jour de façon à ce que que les correcteurs puissent compiler avec la commande

```
$ make
```

et c'est tout.

— Chaque fichier `.c` ou `.h` commencera par

```
/* prénom nom numéro_étudiant
   Je déclare qu'il s'agit de mon propre travail.
   Ce travail a été réalisé intégralement par un être humain. */
```

— Il est interdit d'utiliser des IA génératives comme *chatGPT*, *copilot*, ... Vous devez apprendre à programmer un client et un serveur sans piller le travail collectif de l'humanité.

— Fournir aussi un rapport concis et clair au format texte ou pdf qui contient :

- La description détaillée et claire de vos ajouts au protocole WCP s'il y a lieu.
- La description des fonctionnalités mises en œuvre avec quelques éléments permettant de comprendre rapidement votre implémentation.
- Le cas échéant, ce qui ne fonctionne pas et vos tentatives pour résoudre les problèmes.
- Si vous avez utilisé une ressource externe au cours, le mentionner, donner la référence (l'URL si c'est sur le web).
- Indiquer si un autre étudiant vous a aidé.

— Derniers conseils :

- Le contenu du module permet déjà de faire énormément de choses, sans avoir à aller chercher des choses sur le web.
- Attention à la gestion de votre temps, équilibrez vos efforts avec les autres matières. Si vous êtes à jour, il est possible de mener ce projet à bien assez rapidement.
- Les 3 prochaines sections sont des exercices assez guidés à faire pas à pas, comme en TP. Il y a quelques petites difficultés mais elles ne sont pas insurmontables du tout. Si vous rendez des programmes clairs, propres et rigoureux pour ces 3 sections, vous aurez au moins 12 (et probablement une bonne note au Partiel 2 aussi...)
- La dernière section est plus à la carte, avec des suggestions très recommandées mais aussi des parties beaucoup plus libres. Vous êtes invités, si vous avez de bonnes idées, à faire évoluer WCP et à implémenter ces nouveautés.

2 Les fichiers de comptines

Un serveur Wikicomptines héberge dans un répertoire des fichiers de comptines.

Les fichiers de comptines sont au format « comptine » et ont pour extension `.cpt`. Le format comptine est assez rigide.

— Un fichier de format comptine est un fichier texte encodé en utf-8.

— Un tel fichier est une suite de *lignes*, chaque ligne est terminée par le caractère *Line Feed*, en C `'\n'`, en ASCII hexadécimal `0a`.

— Chaque ligne d'un tel fichier (avec son caractère de fin de ligne) fait au plus 256 octets.

— La première ligne du fichier est *le titre de la comptine*, elle est suivie d'une ligne vide.

— Le reste de la comptine est éventuellement séparé en *strophes*, chaque strophe étant séparée de la suivante par une ligne vide.

Voir les 6 exemples dans le répertoire `comptines`.

Exercice 1 : Utilitaires pour les comptines

1. Écrire dans `comptine_utils.c` la définition de la fonction

```c
/** Retourne 1 si nom_fich (chaîne terminée par un '\0') se termine par
 * .cpt et 0 sinon. */
int est_nom_fichier_comptine(char *nom_fich);
```

Tester dans un `main`.

2. Écrire dans `comptine_utils.c` la définition de la fonction

```c
/** Lit des octets dans le fichier de descripteur fd et les met dans buf
 * jusqu'au prochain '\n' rencontré, y compris.
 * retourne : le nombre d'octets lus avant le premier '\n' rencontré
 * précondition : buf doit être assez grand pour contenir les octets lus
 * jusqu'au premier caractère '\n' y compris. */
int read_until_nl(int fd, char *buf);
```

Si, par exemple, la première ligne du fichier de descripteur `fd` est

```
Lol !\n
```

un seul appel à cette fonction devrait écrire ces 6 caractères à l'adresse `buf`, retourner 5 (et faire avancer la tête de lecture dans `fd` de 6 caractères mais c'est automatique).
Indication : le plus simple est de lire les octets 1 par 1.
Tester dans un `main`.

3. Écrire dans `comptine_utils.c` la définition des fonctions

```c
/** Alloue sur le tas et initialise une struct comptine avec le fichier
 * de format comptine (extension ".cpt") de nom de base base_name situé dans
 * le répertoire dir_name. Le titre et le nom de fichier de la comptine
 * retournée sont eux-mêmes alloués sur le tas pour contenir :
 * - la première ligne du fichier, avec son '\n', suivi de '\0'
 * - une copie de la chaine base_name avec son '\0'
 * Retourne : l'adresse de la struct comptine nouvellement créée ou bien
 * NULL en cas d'erreur */
struct comptine *init_cpt_depuis_fichier(const char *dir_name,
                const char *base_name);

/** Libère toute la mémoire associée au pointeur de comptine cpt */
void liberer_comptine(struct comptine *cpt);
```

Par exemple, si l'on se trouve dans le répertoire parent du répertoire `comptines` fourni, l'appel

```c
init_cpt_depuis_fichier("comptines", "escargot.cpt");
```

devrait retourner un pointeur vers une `struct comptine` sur le tas ayant deux champs pointeurs vers des chaînes aussi sur le tas contenant respectivement 16 octets (le titre : `"Petit escargot\n\0"`) et 13 octets (le nom de base : `"escargot.cpt\0"`).
Tester dans un `main`.

4. Écrire dans `comptine_utils.c` la définition des fonctions

```c
/** Alloue sur le tas un nouveau catalogue de comptines en lisant les fichiers
 * de format comptine (ceux dont le nom se termine par ".cpt") contenus dans le
```

```
 * répertoire de nom dir_name.
 * retourne : un pointeur vers une struct catalogue dont :
 * - nb est le nombre de fichiers comptine dans dir_name
 * - tab est un tableau de nb pointeurs de comptines, avec pour chacunes
 *   + nom_fichier égal au nom de base du fichier comptine correspondant
 *   + titre égal à la première ligne du fichier
 * retourne NULL en cas d'erreur. */
struct catalogue *creer_catalogue(const char *dir_name);

/** Libère toutes les ressources associées à l'adresse c et c lui-même */
void liberer_catalogue(struct catalogue *c);
```

Par exemple, si l'on se situe dans le répertoire parent du répertoire `comptines`, l'appel `creer_catalogue("comptines");` devrait retourner un pointeur vers un tableau de 6 pointeurs de comptines initialisées avec les comptines présentes dans ce répertoire.
Indications : il faut ouvrir le répertoire donné en argument. Voir l'exercice 3 du TP0 et la documentation de `opendir`, `closedir`, `readdir` (dans la section 3 du manuel), `rewinddir`.
Attention aux noms de fichiers pour `open` (vous aurez sans doute à concaténer des chaînes de caractères).

--- * ---

3 Client et serveur, TCP au-dessus d'IPv4

Exercice 2 :

1. Écrire dans `wcp_clt.c` la définition de
   ```
   /** Retourne (en cas de succès) le descripteur de fichier d'une socket
    *  TCP/IPv4 connectée au processus écoutant sur port sur la machine d'adresse
    *  addr_ipv4 */
   int creer_connecter_sock(char *addr_ipv4, uint16_t port);
   ```
   Cette fonction met convenablement fin au programme avec un message d'erreur informatif en cas d'échec.
2. Écrire dans `wcp_srv.c` la définition de
   ```
   /** Retourne en cas de succès le descripteur de fichier d'une socket d'écoute
    *  attachée au port port et à toutes les adresses locales. */
   int creer_configurer_sock_ecoute(uint16_t port);
   ```
3. Dans `wc_srv.c`, écrire une fonction `main` appelant la fonction précédente, puis une boucle infinie dans laquelle le serveur accepte les connexions.
   Tester grossièrement en utilisant aussi le client `wc_clt.c`.

--- * ---

4 Le protocole WCP

Le protocole WCP (Wikicomptines Protocol) permet au client et au serveur Wikicomptines de dialoguer. Ce protocole précise que le port du serveur est 4321, le protocole transport utilisé est TCP.

Exercice 3 : La liste des comptines

Le protocole WCP précise que lorsqu'on souhaite interroger la liste des comptines d'un serveur, celui-ci doit envoyer, pour chaque comptine de son catalogue, une ligne (terminée par `'\n'`) contenant le titre de cette comptine, précédée d'un espace et de son numéro (indice dans son catalogue) sous forme

décimale sur 6 caractères (voir l'exemple à la fin de l'exercice). Enfin, une ligne vide signale la fin de la liste des comptines.

Le protocole WCP ajoute une restriction sur la taille du catalogue : le nombre de comptine doit pouvoir tenir sur 2 octets (en représentation non signée).

1. Écrire dans `wcp_srv.c` la fonction

```
/** Écrit dans le fichier de desripteur fd la liste des comptines présents dans
 *  le catalogue c comme spécifié par le protocole WCP, c'est-à-dire sous la
 *  forme de plusieurs lignes terminées par '\n' :
 *  chaque ligne commence par le numéro de la comptine (son indice dans le
 *  catalogue) commençant à 0, écrit en décimal, sur 6 caractères
 *  suivi d'un espace
 *  puis du titre de la comptine
 *  une ligne vide termine le message */
void envoyer_liste(int fd, struct catalogue *c);
```

Indication : la fonction `dprintf` peut vous aider.

2. Écrire dans `wcp_clt.c` la fonction

```
/** Lit la liste numérotée des comptines dans le descripteur fd et les affiche
 *  sur le terminal.
 *  retourne : le nombre de comptines disponibles */
uint16_t recevoir_liste_comptines(int fd);
```

3. Ajouter dans chaque `main`, les instructions faisant en sorte que juste après la connexion, le serveur envoie au client sa liste de comptine et le client l'affiche à l'écran. Essayer sur boucle locale et sur des machines distantes.

Avec le répertoire `comptines` donné, on devrait obtenir l'affichage suivant pour un client :

```
$ ./wcp_clt 127.0.0.1
     0 Les petits poissons dans l'eau
     1 Petit escargot
     2 La famille tortue
     3 Mon petit lapin
     4 Dans sa maison, un grand cerf
     5 G comme Gaston
```

--- * ---

Exercice 4 : Choix de la comptine et réponse du serveur

Le protocole WCP précise qu'ensuite, le client envoie un numéro de comptine en représentation non signée sur 2 octets en *network byte order*. Ensuite, s'il peut répondre à sa requête, le serveur lui envoie le contenu de la comptine correspondant sous la forme de lignes terminées par `'\n'`, deux lignes vides consécutives signalant la fin du message.

1. Écrire dans `wcp_clt.c` les fonctions

```
/** Demande à l'utilisateur un nombre entre 0 (compris) et nc (non compris)
 *  et retourne la valeur saisie. */
uint16_t saisir_num_comptine(uint16_t nb_comptines);
/** Écrit l'entier ic dans le fichier de descripteur fd en network byte order */
void envoyer_num_comptine(int fd, uint16_t nc);
```

2. Écrire dans `wcp_srv.c` les fonctions

```
/** Lit dans fd un entier sur 2 octets écrit en network byte order
 *  retourne : cet entier en boutisme machine. */
uint16_t recevoir_num_comptine(int fd);

/** Écrit dans fd la comptine numéro ic du catalogue c dont le fichier est situé
```

```
 *  dans le répertoire dirname comme spécifié par le protocole WCP, c'est-à-dire :
 *  chaque ligne du fichier de comptine est écrite avec son '\n' final, y
 *  compris son titre, deux lignes vides terminent le message */
void envoyer_comptine(int fd, const char *dirname, struct catalogue *c, uint16_t ic);
```

3. Écrire dans chacun des `main` les instructions permettant effectivement :

- au client de demander à l'utilisateur un numéro de comptine, de l'envoyer au serveur ;
- au serveur de répondre avec la comptine demandée ;
- au client de lire la réponse du serveur et de l'afficher sur le terminal, sans les deux lignes vides finales.

Voici un exemple, côté client, de ce qui est attendu :

```
$ ./wcp_clt 127.0.0.1
     0 Les petits poissons dans l'eau
     1 Petit escargot
     2 La famille tortue
     3 Mon petit lapin
     4 Dans sa maison, un grand cerf
     5 G comme Gaston
Quelle comptine voulez-vous ? (Entrer un entier entre 1 et 6) : 3
Mon petit lapin

Mon petit lapin
A bien du chagrin
Il ne saute plus
Il ne danse plus

Saute, saute, saute
Mon petit lapin
Saute, saute, saute
Tu auras du thym
```

4. Pour finir cette partie minimale, faire en sorte que le ménage soit bien fait, c'est-à-dire :

- chaque fichier ouvert est bien fermé (ceci vaut bien sûr pour les *sockets* ;
- chaque `malloc` a son `free` ;
- les appels système les plus risqués sont bien testés avec messages d'erreur et comportements appropriés.
- Tester vos programmes, en boucle locale ou sur machines distantes. Le protocole étant pour l'instant fixé, vous devriez à ce stade pouvoir vous connecter avec n'importe quel client WCP à n'importe quel serveur WCP donc essayer entre vous.

--- * ---

5 Approfondissement

Important : chaque ajout ou changement dans le protocole doit être soigneusement documenté. Rien ne doit être ambigü. En particulier, les lectures dans les *sockets* TCP doivent être très rigoureuses (combien d'octets ? quand est-ce qu'on peut s'arrêter de lire ?).

Sur cette base, vous êtes fortement incités à ajouter des fonctionnalités. Voici quelques pistes (les quatre premières sont vraiment conseillées) :

- Le format `.cpt` évolue pour accepter les lignes de tailles arbitrairement grandes. Il faut alors allouer dynamiquement de la mémoire pour gérer les éventuelles très longues lignes.
- Le serveur est multithreadé de façon à pouvoir servir de nombreux clients en même temps.

— Le serveur maintient à jour un fichier de *log* contenant les connexions (adresses IP) et actions demandées. Attention : ce sont les *threads* qui remplissent ce fichier donc gare aux phénomènes de compétition. La fonction `inet_ntop` servira sans doute.

— Les clients peuvent demander plusieurs comptines à la suite, un numéro de comptine spécial (ou hors de l'intervalle) met fin à la requête.

— Les clients peuvent téléverser de nouvelles comptines dans le répertoire de comptines du serveur. Attention : c'est une *écriture* dans le répertoire. Il faut mettre à jour le catalogue de comptine et donc mettre en place des synchronisations convenables. Faire des choix pertinents lorsque les fichiers existent déjà et les expliquer.

— À ce stade, on peut distinguer au moins 3 requêtes possibles chez le client : lister les comptines, lire des comptines, ajouter des comptines. Ce serait bien d'avoir un tout petit langage de requête qui permette au client de dire ce qu'il veut faire au serveur.

— Autres ajouts que vous imaginez (mais documentez soigneusement), il y a largement de quoi faire.