+1/1/60+

LIFASR5 ECA
ECA du mardi 23/05/23 - 1h

No. étu. : .....

Utilisez un stylo bleu ou noir (ni crayon de bois, ni de crayon de papier, ni crayon à papier). Répondez uniquement dans les cadres prévus à cet effet.

Aucun document n'est autorisé. Les téléphones, ordinateurs, et toutes communication avec les autres étudiants sont interdits. Seule l'antisèche fournie est autorisée.

Numéro d'étudiant :

☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0
☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1
☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2
☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3
☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4
☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5
☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6
☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7
☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8
☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9

Dans tout le sujet, les programmes sont donnés sans les `includes` et les `using namespace` : on suppose que l'on saurait les ajouter pour compiler, mais on ne s'en préoccupe pas ici.

1 Autour des pipes

Question 1 On considère le programme ci-dessous.

```
1  int main(void) {
2    int p[2];
3    pipe(p); // p[0] pour la lecture, p[1] pour l'écriture
4    if(fork() > 0) { // processus père
5      close(p[0]);
6      for(char c = '0'; c < '9'; c++)
7        write(p[1], &c, 1);
8      wait(NULL);
9    }
10   else { // processus fils
11     close(p[1]);
12     char c;
13     while(read(p[0], &c, 1) == 1)
14       cout << "(fils) je lis " << c << endl << flush;
15   }
16   return 0;
17 }
```

A l'exécution, ce programme fonctionne normalement en affichant les chiffres de 0 à 9, mais il ne rend jamais la main dans le shell dans lequel on l'a lancé. Donnez précisément la raison pour laquelle le programme reste bloqué. On rappelle la règle "tout descripteur de fichier ouvert doit être fermé dès que possible" : votre explication doit donc être plus précise qu'un simple rappel de cette règle, et il ne suffit pas de dire ce qu'il faudrait faire pour que le blocage ne se produise pas.

☐0 ☐1 ☐2 ☐3

Le père ne ferme jamais p[1], donc il reste toujours un écrivain sur le pipe. Le fils (même après avoir lu toutes les données envoyées par le père) reste donc bloqué en attente de lecture à la ligne 13. Le père quant à lui attend la terminaison de son fils à la ligne 8, mais cela n'arrivera jamais !

2 Communication à l'aide de sockets

On veut écrire un client et un serveur qui communiquent en TCP/IP à l'aide d'une socket. Le serveur et le client n'échangeront que des messages dont la taille est fixée par la constante entière `MSG_LEN` (déjà définie par un `#define` dans le code source). Dans cette partie,

- vous utiliserez pour manipuler les sockets les primitives POSIX `recv()` et `send()`, ainsi que les fonctions de la `socklib` : l'utilisation de tout cela vous est rappelée dans l'antisèche.
- pour le code du serveur (Question 4), vous disposez d'une fonction `void get_temp(char msg[MSG_LEN])` qui, lorsqu'elle est appelée, place un message donnant la température dans la salle du serveur dans le paramètre `msg`.
- toujours pour le serveur (Question 4), vous devrez comparer deux chaînes de caractères stockées dans des tableaux de `char` (selon la méthode utilisée en C) : utilisez la fonction `int strcmp(char *s1, char *s2)`, qui retourne 0 si les deux chaînes stockées en mémoire à partir des adresses `s1` et `s2` sont identiques, et une valeur différente sinon.

Question 2 Écrivez une fonction d'en-tête `bool recv_msg(char msg[MSG_LEN], int sd)` pour recevoir l'intégralité d'un message de `MSG_LEN` octets dans `msg`, en lisant sur la socket de dialogue `sd`. On rappelle que la primitive `recv()` ne garantit pas de lire en un seul appel tous les octets qu'on lui demande de lire : veillez à lire exactement `MSG_LEN` sur la socket. En cas d'erreur, un appel à `recv_msg` retournera `false`, et il retournera `true` en cas de succès.

☐0 ☐1 ☐2 ☐3

```
bool recv_msg(char msg[MSG_LEN], int sd){
    int t, nbrd=0;
    do{
        t = recv(sd, msg+nbrd, MSG_LEN-nbrd, 0);
        if(t==-1) return false;
        nbrd += t;
    while(nbrd != MSG_LEN);
    return true;
}
```

Question 3 Écrivez maintenant une fonction d'en-tête `bool send_msg(int sd, char msg[MSG_LEN])` pour envoyer l'intégralité du message `msg` de `MSG_LEN` octets vers la socket de dialogue `sd`. On rappelle que la primitive `send()` ne garantit pas d'envoyer en un seul appel tous les octets qu'on lui demande d'envoyer : veillez à envoyer exactement `MSG_LEN` sur la socket. En cas d'erreur, un appel à `send_msg` retourne `false`, et `true` en cas de succès. ☐0 ☐1 ☐2 ☐3

```
bool send_msg(int sd, char msg[MSG_LEN]){
    int t, nbst=0;
    do{
        t = send(sd, msg+nbst, MSG_LEN-nbst, 0);
        if(t==-1) return false;
        nbst += t;
    while(nbst != MSG_LEN);
    return true;
}
```

Question 4 Vous devez maintenant donner la fonction principale d'un serveur qui se mettra à l'écoute sur le port `"9999"` et recevra les clients successivement. Dès qu'un client est accepté, le serveur se met en attente de réception d'un message avec `recv_msg()`. Quand un message `msg` est reçu :

- si ce message est `"TEMP"` alors le serveur envoie un message contenant la température au client, ferme la socket de dialogue, et se met en attente du client suivant.
- si ce message est `"STOP"` alors le serveur ferme la connexion et se termine

☐0 ☐1 ☐2 ☐3 ☐4 ☐5

```
int main(void){
    char msg[MSG_LEN];
    int s, sd;
    s = create_server_socket("9999");
    while (true) {
        sd = accept_connection(s);
        recv_msg(msg, sd);
        if (strcmp(msg, "STOP") == 0) {
            close(sd);
            break;
        }
        if(strcmp(msg, "TEMP") == 0){
            get_temp(msg);
            send_msg(sd, msg);
        }
        close(sd);
    }
    close(s);
    return 0;
}
```

3 Systèmes de fichiers

On considère un système de fichier qui utilise une table d'allocation : il s'agit d'un tableau d'entiers sur 16 bits, avec des indices codés sur 16 bits. Les blocs du système de fichier sont alloués par chaînage dans la table d'allocation : chaque entrée de la table d'allocation contient une valeur qui est : soit l'adresse du bloc suivant d'un fichier, soit une valeur spéciale, par exemple une valeur marquant la fin du fichier. Voici la signification des valeurs possibles (elles sont données en hexadécimal) :

valeur	signification
0x0000	le bloc est libre et peut être utilisé pour l'allocation de blocs
0x0001 à 0xFFEF	adresse du bloc suivant
0xFFF0 à 0xFFF6	le bloc est réservé par l'algorithme d'allocation de blocs
0xFFF7	le bloc est défectueux et ne peut plus être utilisé pour l'allocation de blocs
0xFFF8 à 0xFFFF	dernier bloc d'un fichier

Question 5 Donnez le nombre de valeurs possibles pour les entrées de la table d'allocation, sous la forme d'une puissance de 2, puis en décimal. ☐0 ☐1 ☐2 ☐3

Les valeurs sont codées sur 16 bits, donc il y en a :
$2^{16} = 65536$

Question 6 Combien de blocs peut comporter au maximum un fichier dans ce système de fichier ? Précisez votre calcul, et donnez votre réponse en décimal. ☐0 ☐1 ☐2 ☐3

Au maximum, un fichier occupe tous les blocs, mais il faut de compter ceux avec une valeur spéciale : 65536-17=65519.

Question 7 La taille en octets des blocs dans ce système de fichier est codée sur 2 octets. Quelle est la taille maximale d'un fichier ? Donnez le calcul à faire pour avoir le résultat exact et une estimation en Gio. ☐0 ☐1 ☐2 ☐3

Tous les blocs sont de taille maximale, et sont tous occupés : $(2^{16}-1) \times (2^{16}-17) \simeq 2^{32}$ o = 4 Gio

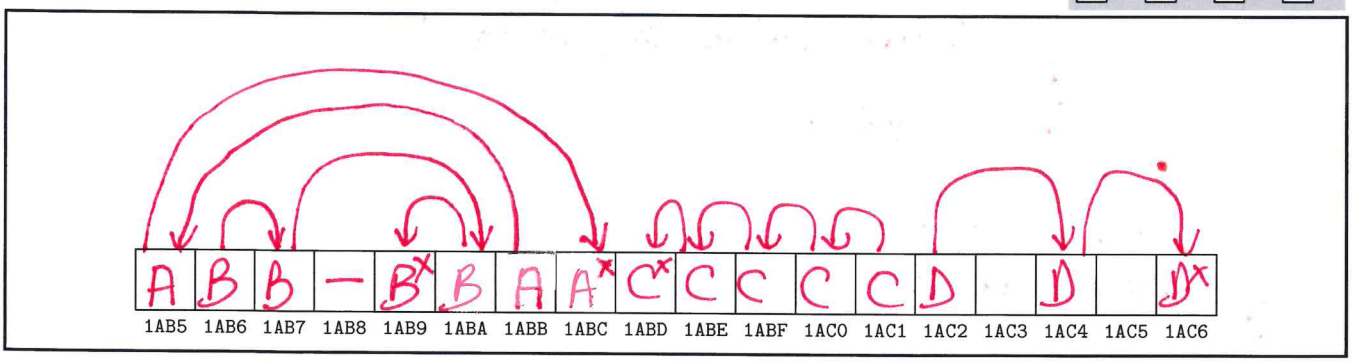
Voici un extrait de la table d'allocation d'un disque, avec les adresses et les valeurs en hexadécimal :

adresse	1AB5	1AB6	1AB7	1AB8	1AB9	1ABA	1ABB	1ABC	1ABD	1ABE	1ABF	1AC0	1AC1	1AC2	1AC3	1AC4	1AC5	1AC6
valeur	1ABC	1AB7	1ABA	FFF7	FFF8	1AB9	1AB5	FFF9	FFF8	1ABD	1ABE	1ABF	1AC0	1AC4	0000	1AC6	0000	FFF8

Question 8 Complétez le schéma ci-dessous en :

- en précisant bien les chaînages entre les blocs à l'aide de fléches,
- en attribuant une lettre (`A`, `B`, `C`...) à chaque fichier présent,
- en marquant les blocs qui terminent un fichier par une croix (×) et les blocs défectueux par un tiret (−),
- en laissant vide les cases correspondant aux blocs libres.

☐0 ☐1 ☐2 ☐3



Question 9 On doit ajouter 2 blocs à la fin du fichier commençant avec le bloc d'adresse `0x1AC1` : expliquez quels sont les chaînages à modifier dans la table d'allocation décrite pour allouer ces blocs (plusieurs réponses sont possibles).

☐0 ☐1 ☐2 ☐3

Le dernier bloc actuel du fichier est 0x1ABD.
- pour suivant de 0x1ABD on met 0x1AC3
- 0x1AC3 ——— 0x1AC5
- 0x1AC5 ——— 0xFFF8 pour marquer la fin du fichier.