ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours de Programmation Orientée Système, sections IN et SC
Chappelier J.-C.

NOM : Hanon Ymous
(000000)
Place : 0

#0000

PROGRAMMATION ORIENTÉE SYSTÈME

Examen

12 août 2020

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez de deux heures pour faire cet examen (16h15 – 18h15).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée ; ne joignez aucune feuille supplémentaire ; **seul ce document sera corrigé**.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte trois exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 100 points) ; tous les exercices comptent pour la note finale :
   — question 1 : 22 points ;
   — question 2 : 54 points ;
   — question 3 : 24 points.

Question 1 Questions brèves [22 points]

Question 1.1 Acquis C ? [8 points]

Expliquez ce qu'affiche le code suivant :

```
char s[] = "Shotaro:Kaneda:x:880716:821220";
for (char *p = s, *q = s; *p; ++p) {
    if (*p == ':') {
        *p = '\0';
        printf("\"%s\", ", q);
        q = p + 1;
    }
}
```

Donnez en particulier un schéma de la mémoire à un moment pertinent en *milieu* d'exécution de la boucle (c.-à-d. ni au début, ni à la fin). Indiquez sur ce schéma le contenu de `s`, ainsi que les pointeurs `p` et `q` à ce moment là.

Réponse :

Question 1.2 Qu'est-ce que C ? [3 points]

Ce programme compile-t-il et s'exécute-t-il correctement ? Si oui, qu'affiche-t-il ? Dans tous les cas, justifiez votre réponse.

```
#include <stdio.h>
void f(void* p) {
  if (p == NULL) puts("Cessez !");
  else           puts("C'est C !");
}
int main(void)
{
  int* p = NULL;
  f(&p);
  return 0;
}
```

Réponse :

Question 1.3 Ah, c'est du C ! [4 points]

Ce programme compile-t-il et s'exécute-t-il correctement ? Si oui, qu'affiche-t-il ? Dans tous les cas, justifiez votre réponse.
On supposera que `s` est stocké à l'adresse 8899, `"Angus"` à l'adresse 2211 et que le caractère `'s'` se représente par le nombre 115 en mémoire.

```
#include <stdio.h>
int main(void)
{
  const char* s = "Angus";
  printf("%d\n", (int) s[5]);
  return 0;
}
```

Réponse :

Question 1.4 Dans C maintenant. [3 points]

Ce programme compile-t-il et s'exécute-t-il correctement ? Si oui, qu'affiche-t-il ? Dans tous les cas, justifiez votre réponse.

```
#include <stdio.h>
int main(void)
{
  char s[6];
  s = "cigale";
  printf("%s\n", s);
  return 0;
}
```

Réponse :

Question 1.5 C'est à C. [4 points]

Ce programme compile-t-il et s'exécute-t-il correctement ? Si oui, qu'affiche-t-il ? Dans tous les cas, justifiez votre réponse.

```
#include <stdio.h>
int main(void)
{
  const char* s = "orque";
  const char* t = s;
  for (size_t i = 0; *t != 0; ++i) {
    t = s + i;
    printf("%s", t);
  }
  return 0;
}
```

Réponse :

Question 2 Gestion d'un disque dur [54 points]

On s'intéresse à la conception (partielle) d'un système de gestion de fichiers sur disque dur.

Un disque dur est décomposé en T secteurs, dont la taille est typiquement de 512 octets (voir fig. 1 ci-contre). Une lecture ou écriture sur un disque ne peut opérer que sur des secteurs entiers. Ainsi, pour écrire le 21212^{e} octet du disque, il faut d'abord lire le secteur 42 (car $21212/512 = 42$), modifier (en mémoire) l'octet position 220 du secteur lu (car $21212 \bmod 512 = 220$), puis réécrire ensuite le secteur ainsi modifié à sa place d'origine (42).

Le disque stocke non seulement le contenu des fichiers eux-mêmes, mais également toutes les « méta-données » permettant de gérer le système de fichiers (cf. fig. 1) : informations sur les fichiers, répertoires, mais aussi les secteurs libres. Pour information (on ne l'utilisera pas), le premier secteur du disque (appelé « superblock ») contient une structure qui mémorise l'emplacement et la taille sur le disque de toutes les méta-données.

Le but de cet exercice est de fournir certaines parties d'un programme de gestion de disque dur dont on possède déjà certaines autres parties. En particulier, le type `hdd_t` est déjà fourni et permet de représenter un disque dur. Ce type possède (entre autres) un champ `sector_size` de type `unsigned short int` contenant la taille d'un secteur en octets (typiquement 512). Ce type possède également un champ `root` de type `size_t` contenant le numéro du secteur utilisé pour stocker le premier fichier (A sur la fig. 1) et un champ `nb_sectors` de type `size_t` contenant le nombres total de secteurs (T sur la fig. 1).

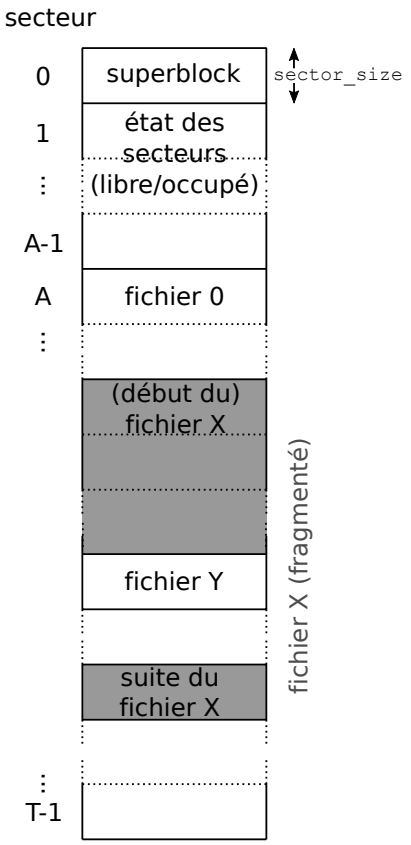

FIGURE 1 – Organisation physique d'un disque dur.

Question 2.1 Lecture de fichiers [14 points]

On suppose également avoir une fonction

```
int hdd_read_sector(char* destination, hdd_t* hdd, size_t sector_number);
```

qui permet de lire en mémoire un secteur donné : elle recopie en mémoire à l'adresse `destination` le contenu du secteur numéro `sector_number` (du disque `hdd`) et retourne 0 si tout s'est bien passé ou un code d'erreur non nul en cas d'erreur.
Note : `destination` est de type `char*` non pas parce que c'est une chaîne de caractères, mais simplement pour lire la mémoire octet par octet (en C, `char` est synonyme d'« octet » (« *byte* », plus exactement) : `sizeof(char) == 1`).

Définissez sur la page ci-contre une fonction `hdd_read_bytes()` qui lit dans une zone mémoire donnée, un nombre spécifié d'*octets* à partir d'un numéro de secteur donné (c'est à vous de décider son prototype). On supposera que la zone mémoire donnée contient la place nécessaire (en clair, ce n'est pas à cette fonction de faire l'allocation).

Indications : le nombre d'octets à lire peut être plus grand que plusieurs secteurs et faites attention au cas où il n'est pas multiple de la taille d'un secteur : la zone mémoire de destination n'a pas la place pour lire la fin du dernier secteur.

Réponse :

suite au dos ☞

Question 2.2 Recherche de place [25 points]

Les secteurs 1 et suivants (jusque $A-1$ sur la fig. 1) contiennent, bit par bit (dans l'ordre de numéro de bit croissant ; voir la fonction `get_bit()` ci-dessous), le statut libre (0) ou occupé (1) de chacun des secteurs. Ainsi, si le secteur 42 est occupé, le bit 2 ($42 \bmod 8 = 2$) de l'octet 5 ($42/8 = 5$) du secteur 1 ($42/4096 = 0$) [1] sera à 1 ; si le secteur 123456789 (disque d'au moins 60 Go) est libre, le bit 5 ($123456789 \bmod 8 = 5$) de l'octet 418 ($(123456789 \bmod 4096)/8 = 418$) du secteur 30141 ($123456789/4096 = 30140$) sera à 0. [2]

On suppose avoir à disposition une fonction

```
int get_bit(char* p, unsigned int index);
```

qui retourne la valeur (0 ou 1) du bit d'index « `index` » dans l'octet pointé par `p`.

Définissez une fonction `hdd_get_free_sectors()` qui retourne l'emplacement du premier secteur libre, ainsi que le nombre de secteurs libres contigus (qui le suivent, lui compris). Expliquez les paramètres ainsi que l'éventuelle valeur de retour.

Indication : pensez à modulariser : déléguez à une sous-tâche (qu'il vous faudra aussi écrire) déterminant si un secteur est libre ou non. On ne cherchera pas ici à minimiser le nombre d'accès au disque [3].

Réponse :

1. $4096 = 8 \times 512$.
2. En réalité, on peut faire un peu mieux et ne pas coder le fait que les A premiers secteurs sont de toutes façons occupés ; mais ceci serait un peu trop compliqué pour cet examen.
3. Ceci pourrait être fait par la suite (cache) et n'est pas du tout considéré dans cet examen.

(suite de la réponse à Question 2.2)

suite au dos ☞

Question 2.3 Fichiers fragmentés [15 points]

Il se peut que le disque ne contienne pas suffisamment de place contiguë pour stocker un fichier en un seul morceau sur des secteurs successifs (voir le fichier X sur la fig. 1). Ceci arrive en particulier lorsque les fichiers sont créés et supprimés dans le désordre. L'espace libre devient alors « fragmenté ». Dans ce cas, il est nécessaire de mémoriser le fichier en le divisant en plusieurs morceaux (fragments) ayant chacun un emplacement (numéro de secteur) et une taille (en octets).

Le système de fichiers doit donc pouvoir gérer une liste de fragments pour chaque fichier. Définissez (au moins) une structure `file_piece_t` permettant de le faire. Vous pouvez définir d'autres types intermédiaires si vous le souhaitez.

Définissez ensuite une fonction qui ajoute à une telle liste un nouveau fragment (débutant au prochain secteur libre ; aucune écriture n'est faite sur le disque ici).

Indications : cette fonction peut utiliser la fonction de la question précédente ; et pensez à gérer les cas d'erreur.

Réponse :

(suite de la réponse à Question 2.3)

suite au dos ☞

Question 3 Découpage de chaînes de caractères [24 points]

Le but de cet exercice est d'écrire une fonction `split()` qui retourne tous les constituants (« *tokens* ») séparés par un séparateur donné dans une chaîne de caractères donnée.

Par exemple :

- `split('/', "un/exemple/simple")` retournera un tableau contenant les trois chaînes `"un"`, `"exemple"` et `"simple"` ;
- `split('/', "/autre/exemple")` retournera un tableau contenant les trois chaînes `""` (chaîne vide), `"autre"` et `"exemple"` ;
- `split('/', "/autre/exemple/")` retournera un tableau contenant les *quatre* chaînes `""` (chaîne vide), `"autre"`, `"exemple"` et `""` (chaîne vide) ;
- `split('/', "/autre//exemple")` retournera un tableau contenant les *quatre* chaînes `""` (chaîne vide), `"autre"`, `""` (chaîne vide) et `"exemple"` ;
- enfin, `split('/', "autre exemple")` retournera un tableau contenant la seule chaîne `"autre exemple"`.

Question 3.1 Type de retour [4 points]

Définissez ici (et expliquez) le type de retour que vous proposez pour la fonction `split()`.

Réponse :

Question 3.2 Affichage [6 points]

Définissez ici (et sur la page d'en face) une fonction `affiche()` qui affiche une valeur du type que vous venez de définir (Question 3.1). Par exemple, l'appel à `affiche()` sur le résultat de `split('/', "un/exemple/simple")`, affichera :

```
"un", "exemple", "simple",
```

Vous êtes libre de choisir ses paramètres, mais, comme `printf()`, cette fonction devra retourner le nombre total de caractères affichés.

Réponse :

Question 3.3 `split()` [14 points]

Définissez ici (et au dos si nécessaire) la fonction `split()`.

Réponse :

(suite de la réponse à Question 3.3)