Semaine 1 - Introduction

N. Sabouret

1 Exercice 1 – Questions de cours

1. Quel est le rôle d'un système d'exploitation?
2. Quelles sont les 5 principales fonctions d'un système d'exploitation?
3. Qu'est-ce que la multiprogrammation dans un système d'exploitation?
4. Quelle est la différence entre une exception (ou déroutement) et une interruption dans un système?

2 Exercice 2 – Structure et fonction d'un OS

1. Donnez deux exemples d'opérations qui ne peuvent pas être faites en dehors du mode « superviseur ».
2. Décrivez à l'aide d'un schéma le fonctionnement général d'un système d'exploitation

3 Exercice 3 – Conception d'un OS

Le but de cet exercice est de réfléchir concrètement à la conception d'un système d'exploitation. On considère un ordinateur très simplifié muni d'une UC 8 bits, d'une mémoire vive de 256o, d'un clavier à 5 touches (0, 1, effacer, espace, entrée) et d'un terminal capable d'afficher un octet par ligne.

On considérera que l'UC dispose de 8 registres d'entrée pour faire des calculs, d'un registre d'adressage (adresse de l'instruction) et d'un registre de sortie. On ne s'intéresse pas ici au jeu d'instructions de l'UC, au codage binaire des opérations ou à l'exécution des programmes, mais uniquement au fonctionnement de l'OS. Dans cet exercice, nous nous contenterons d'un système d'exploitation synchrone et mono-processus.

Notez que les programmes sont nécessairement saisis au clavier puisqu'il n'y a pas d'autre périphérique d'entrée. La saisie d'un programme se fait en saisissant des mots binaires, séparés par espace, et le programme se termine par entrée. Le clavier dispose d'un buffer de 8 octets (qui est le mot le plus long autorisé dans notre système informatique).

Nous souhaitons que l'UC puisse exécuter un programme pendant que l'utilisateur saisit un autre programme.

1. Proposez un jeu d'interruptions et un fonctionnement du système d'exploitation pour l'affichage
2. Proposez un jeu d'interruptions et un fonctionnement du système d'exploitation pour la gestion des saisies au clavier

3. Proposez une première structuration de la mémoire pour prendre en compte vos propositions

4. Décrivez le fonctionnement détaillé du système lorsque l'utilisateur appuie sur la touche espace pendant qu'un processus est en cours de traitement par l'UC.

4 Exercice 4 – Commandes shell sous Linux

Cet exercice se fera sous Linux, en utilisant le terminal.

Pour le compte-rendu (les TP sont relevés), vous mettrez dans un fichier les copies des instructions et des résultats affichés, précédés des numéros de questions, ainsi que vos explications ou réponses aux questions.

1. La commande `ps` permet de lister les processus qui s'exécutent actuellement sur l'ordinateur (gérés par le système d'exploitation). En utilisant sa documentation (pages man), déterminer la commande à utiliser pour lister ces processus.

2. Le répertoire `/dev` contient l'ensemble des fichiers d'accès aux périphériques. Quels sont les périphériques que vous pouvez utiliser?

3. La commande `od` permet de lire un fichier sous forme d'octets. Utilisez cette commande avec les périphériques suivants et décrivez ce qu'il se passe:

   - `/dev/random` et `/dev/urandom`
   - `/dev/zero`
   - `/dev/null`

4. Le répertoire `/dev/pts/` contient les flux d'entrée/sortie des terminaux. Nous allons voir comment ceci peut être exploité dans un système d'exploitation.

   (a) Ouvrez deux terminaux et tapez dans chacun d'eux la commande `ps -o pid,tty,cmd`. Que fait cette commande?

   (b) La commande `cat` permet d'affiche dans la console le contenu d'un fichier. Elle peut aussi être utilisée, avec la commande de redirection de flux `>`, pour écrire dans un fichier. En utilisant le bon `pts`, écrivez dans le terminal distant.

   (c) Utilisez la commande `cat` sans argument dans le terminal distant pour démarrer la saisie. Utilisez ensuite la commande `cat` avec le bon PTS pour "espionner" ce qui est tapé dans l'autre terminal. Que constatez vous ?

   (d) Pensez-vous que ce fonctionnement soit une faille de sécurité? Pourquoi?

5 Exercice 5 – Appels systèmes en C

Le langage C est un langage « proche de la machine » (on dit qu'une instruction C ne doit pas se traduire par plus de 3 instructions machine). La bibliothèque `stdlib` propose un ensemble de fonctions qui permettent de communiquer avec le système d'exploitation. Nous allons les étudier ici.

On rappelle la structure d'un programme C:

```
#include <stdlib.h>
#include <stdio.h>

int main() {
  ... declaration des variables ...
  ... instructions ...
  return 0;
}
```

et on rappelle que l'affichage se fait à l'aide de la fonction `printf`.

Cet exercice se fera sous Linux, en utilisant le terminal et le compilateur `gcc` qui s'utilise de la manière suivante:

```
$ gcc -O -Wall -o prog source.c
$ ./prog
```

Pour le compte-rendu (les TP sont relevés), vous mettrez dans un fichier les copies des instructions et des résultats affichés, précédés des numéros de questions, ainsi que vos explications ou réponses aux questions.

1. Utilisez la fonction `system` pour afficher le contenu du répertoire courant.

2. Le code de retour d'un programme peut s'afficher en utilisation l'instruction shell `echo $?`. Quel est le code de retour du programme précédent?

3. Utilisez ensuite la fonction `exit` dans votre programme avec un code de retour. Que se passe-t-il ?

4. Quel est le code de retour de la fonction `abort`? À quoi cela correspond-il?

5. Utilisez les fonctions `getenv` et `setenv` pour modifier et afficher la variable d'environnement `$TOTO`. On rappelle qu'une variable d'environnement se modifie dans le shell en utilisant la commande `export`:

   ```
   $ export TOTO=valeur
   ```

   et s'affiche à l'aide de la commande `echo`:

   ```
   $ echo $TOTO
   valeur
   ```

6. Expliquez pourquoi la variable n'est pas modifiée après l'exécution du programme.

Semaine 2 - Processus et threads

N. Sabouret

1 Exercice 1 – Questions de cours

1. Donnez la définition d'un processus
2. De quoi se compose un processus? Expliquez.
3. Peut-on avoir plusieurs processus prêts dans le système? Expliquez.
4. Le système d'exploitation est-il un processus? Justifiez.
5. Quels sont les avantages d'un processus multi-threadé?

2 Exercice 2 – Un peu de shell

Pour le compte-rendu (les TP sont relevés), vous mettrez dans un fichier les copies des instructions et des résultats affichés, précédés des numéros de questions, ainsi que vos explications ou réponses aux questions. Vous donnerez le code C dans le compte-rendu.

Finissez tout d'abord le TP de la semaine passée: les exercices de cette semaine s'appuient sur les notions vues au TP précédent. Attention: n'accumulez pas de retard d'une séance à l'autre!

1. Nous avons vu au premier TP la commande `ps` qui permet de voir les processus s'exécutant dans le système. Utilisez la commande `pstree` pour voir les dépendences père/fils entre les processus. Quelle est la parenté de votre commande pstree? Expliquez à quoi correspondent ces différents processus. Quel lien y a-t-il entre votre commande et la session graphique?
2. Que fait le programme C suivant? Écrivez et compilez ce programme.

```
#include <stdio.h>
#define N 100000

int main() {
  int i,j;
  for(i=3;i<=N;i++) {
    j=2;
    while ((j*j<=i)&&(i%j!=0))
      j++;
    if (i%j!=0)
      printf("%d ",i);
  }
  printf("\n");
```

```
  return 0;
}
```

3. Utilisez la commande shell time pour déterminer le temps pris par l'exécution d'un processus généré à partir de ce programme. Qu'en concluez-vous?

4. Utilisez la fonction `gettimeofday` de la bibliothèque `sys/time.h` pour récupérer et afficher le temps dans votre programme C. Comparez le résultat avec celui fournit par la commande shell.

5. Utilisez la fonction `clock` de la bibliothèque `time.h` pour récupérer et afficher le temps dans votre programme C. La valeur renvoyée par clock peut être manipulée comme un `double` et vous pouvez utiliser la constante `CLOCKS_PER_SEC` pour obtenir une valeur en secondes. Comparez le résultat avec celui fournit par la commande shell.

   Remarque: cette solution pour calculer le temps n'est pas standard et ne fonctionne pas correctement si vous avez un processeur à vitesse variable, comme c'est souvent le cas sur les portables, ou si vous utilisez un autre système d'exploitation que Linux.

3 Exercice 3 – Création de processus en C

1. Écrivez en C un premier programme `toto` qui attend 4 secondes puis écrit « au revoir » sur la sortie standard.

2. Écrivez un autre programme qui fait un fork tel que, en parallèle: le père attend 2 seconds, écrit « bye » et termine; le fils attend 4 secondes puis écrit « fini » et termine. Que se passe-t-il ?

3. Rajoutez l'instruction `abort` dans le père. Que se passe-t-il ? Même question avec `exit`.

4. Utilisez l'instruction `kill` de la bibliothèque `signal.h` pour envoyer un signal de terminaison au fils depuis le père. Expliquez ce qu'il se passe

5. Modifiez le code du fils pour qu'au lieu d'attendre et d'écrire "fini", il appelle le programme toto. Que se passe-t-il?

6. Utilisez la commande shell `pstree` pour observer le lien entre le programme `toto` et le programme qui l'a lancé.

4 Exercice 4 – Threads POSIX en C

La norme POSIX spécifie les fonctions que doit offrir une bibliothèque de threads. Elle est implémentée en standard sous Unix et il existe des implémentations libres sous Windows (Win32 propose sa propre API de threads, suivant une autre norme).

1. En utilisant les fonctions `pthread_create` et `pthread_join` de la bibliothèque `pthread.h`, écrivez un programme C qui crée une thread affichant « bonjour » toutes les secondes pendant 10 secondes. Vous devrez utiliser un pointeur vers une fonction (généralement appelée `run`) qui effectue la boucle et se termine par `pthread_exit`. Toutes ces fonctions sont documentées dans les pages man.

   Attention: vous devrez utiliser l'option `-pthread` dans `gcc` pour l'édition de liens.

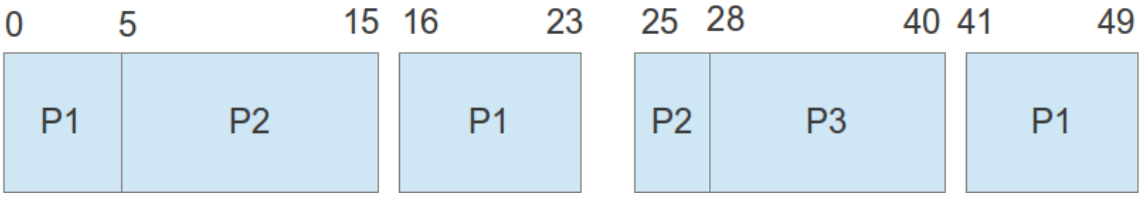
2. Que se passe-t-il si vous n'utilisez pas l'instruction `pthread_join` ? Expliquez.

Semaine 3 - Ordonnancement et threads en Java

N. Sabouret

1 Exercice 1 – Questions de cours

1. Quels sont les critères d'évaluation d'un ordonnanceur? Indiquez pour chaque critère comment il doit être utilisé.
2. On considère le diagramme de Gantt suivant. Indiquez les valeurs pour chaque critère (on suppose que tous les procesus étaient présents avant le début de l'ordonnancement).



3. Citez deux inconvénients de l'algorithme d'ordonnancement FIFO
4. Citez deux inconvénients de l'algorithme d'ordonnancement « plus court d'abord »
5. Citez deux inconvénients des algorithmes d'ordonnancement à base de priorité
6. Citez deux inconvénients de l'algorithme d'ordonnancement « round robin »

2 Exercice 2 – Ordonnancement

On considère les processus suivants, définis par leur durée (réelle ou estimée), leur date d'arrivée et leur priorité:

P1 durée: 10, date 0, priorité 3

P2 durée: 1, date 0, priorité 1

P3 durée: 2, date 2, priorité 3

P4 durée: 5, date 4, priorité 2

P5 durée: 1, date 6, priorité 4

1. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement FIFO et indiquez le temps d'attente moyen.
2. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement par priorité et indiquez le temps d'attente moyen.

3. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement « plus court d'abord » non préemptif et indiquez le temps d'attente moyen.

4. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement « plus court d'abord » préemptif et indiquez le temps d'attente moyen.

5. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement « round robin » avec un quantum de temps fixé à 2 et indiquez le temps d'attente moyen.

6. Quel est le meilleur algorithme suivant le critère du temps d'attente min-max?

3 Exercice 3 – Premières threads en Java

Votre compte-rendu sera une archive de vos ***fichiers source*** *Java (et uniquement les fichiers sources), munis de suffisemment de commentaires et d'explications sur le travail effectué, ainsi que de jeux de tests (par exemple en copier-coller dans un commentaire à la fin du fichier). Inutile d'envoyer les fichiers compilés. Chaque question de chaque exercice doit être fait dans un fichier séparé. Toutes vos classes seront dans le package par défaut.*

Les threads en Java sont gérées à l'aide de la classe `Thread`: une thread Java est un objet instance de la classe `Thread` (ou d'une sous-classe). Les opérations effectuées par la thread sont définies dans la méthode `run`. La documentation de l'API Java pour la classe `Thread` explique comment implémenter une thread en Java:

```
class MaClasse extends Thread {
  ... /* attributs, constructeurs et autres
         méthodes spécifiques */
  @Override
  public void run() {
    ... /* le code de ma thread */
  }
```

Le lancement de la thread se fait en appelant la méthode `start`:

```
MaClasse t = new MaClasse(...);
t.start();
```

1. Implémentez en Java un programme similaire à celui de l'exercice 4 de la semaine précédente. Le programme écrit « début », lance une Thread affichant « bonjour » toutes les secondes pendant 10 secondes, attend 2 secondes et termine en écrivant « fin ». La thread fille et la fonction main vont s'exécuter en parallèle.

   Note: la thread courante peut être mise en sommeil en utilisant la fonction `Thread.sleep`.

2. Quelle différence observez-vous avec les threads POSIX ?

3. Supposons que la thread créatrice (*i.e.* la fonction `main`) ait besoin d'attendre la fin de la thread créée pour continuer son exécution. Quelle instruction faudrait-il utiliser? Montrez le résultat obtenu avec votre programme précédent.

4. Supposons que la thread créatrice veuille arrêter la thread fille après 2 secondes d'exécution. Quelles modifications faut-il faire? Attention: comme vous l'indique la javadoc de la classe Thread, **vous ne devez pas** utiliser la méthode `stop`. Donnez le code obtenu.

4 Exercice 4 – Produit de matrices multi-threadé

1. Implémentez un programme Java qui calcule le produit de deux matrices, en mono-thread dans la fonction main. Testez-le sur des matrices de grande taille et utilisez la commande shell time pour mesurer le temps d'exécution.

2. Implémentez une version multi-threadé du programme précédent: le calcul de chaque valeur de la nouvelle matrice est une thread séparée. Comparez les temps d'exécution.

3. Modifiez le programme précédent pour qu'une thread correspond maintenant au calcul d'une ligne entière de la matrice résultat, au lieu d'une seule valeur.

Semaine 4 - Synchronisation de processus

N. Sabouret

1 Exercice 1 – Questions de cours

1. Qu'est-ce qu'une section critique?
2. Quelles sont les propriétés attendues d'une section critique?
3. Quels sont les inconvénients du Mutex?
4. Quels sont les inconvénients des sémaphores?
5. Quelles sont les 4 conditions nécessaires de l'interblocage?

2 Exercice 2 (TD) – Synchronisation

Le problème des lecteurs-écrivains est un problème classique de synchronisation. On considère une base de données et un ensemble de processus lecteurs et écrivains dans cette base. Chaque processus attend un temps aléatoire puis demande à acquérir la base pour lire ou pour écrire. Le travail sur la base dure un certain temps puis le processus libère la base.

Plusieurs conditions sont à réunir :

C1. Plusieurs lecteurs peuvent accéder à la base en lecture en même temps.
C2. Un seul écrivain peut accéder à la base à la fois.
C3. Un écrivain ne peut pas accéder à la base en même temps que des lecteurs.

1. On propose ci-dessous le pseudo-code d'un processus (Thread Java) pouvant jouer alternativement le rôle de lecteur et d'écrivain. Satisfait-il les conditions C1, C2 et C3 ? Justifiez.

```
class Processus extends Thread
Base base;
Processus(Base b) { base = b; }
void run() {
  while(true) {
    sleep(random());
    choix aléatoire entre lire() et écrire();
  }
}
void lire() {
  sleep(random());
}
void ecrire() {
  sleep(random());
}
```

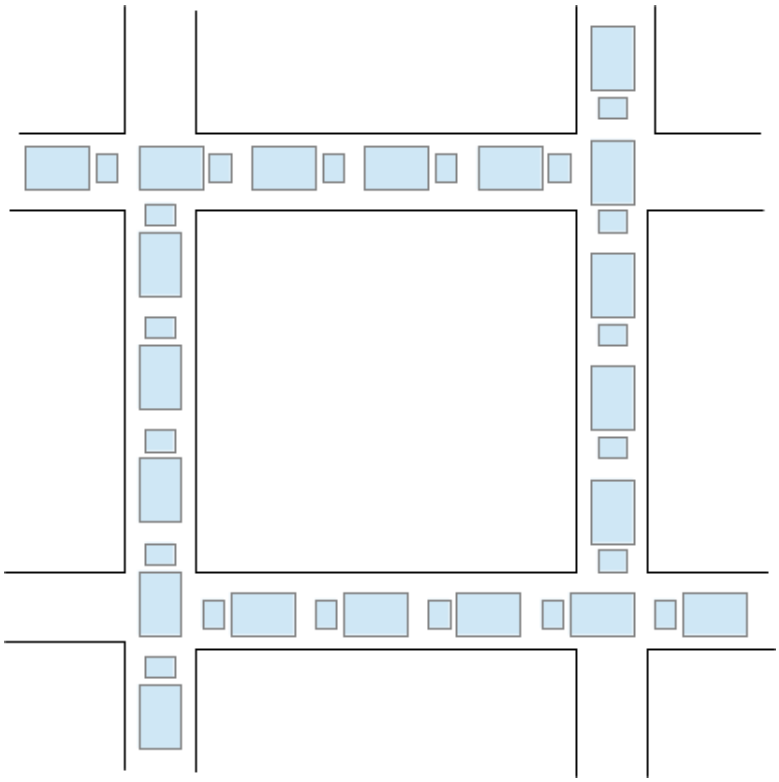
2. Proposez une correction au code de Processus, satisfaisant les conditions, dans le cas où on n'a uniquement que 2 processus. Vous pouvez utiliser un sémaphore (classe Semaphore de Java) pour contrôler l'accès en lecture et en écriture à la base. Vous devrez détailler le fonctionnement des méthodes d'acquisition et de libération de la base (pour la lecture et l'écriture).

3. Proposez une solution au problème des lecteurs-écrivains avec un nombre arbitraire de processus. Vous pouvez utiliser plusieurs sémaphores.

4. Votre solution prévient-elle du risque de famine ? Proposez si besoin une solution à ce problème.

3 Exercice 3 (TD) – Interblocage

On considère le carrefour représenté par la figure suivante:



1. Montrez que les 4 conditions sont réunies pour un interblocage

2. Proposez une solution simple pour éviter cette situation.

4 Exercice 4 – Synchronisation en Java

1. Implémentez en Java le problème des lecteurs-écrivains vu en TD (exercice 2) en utilisant les sémaphores.

2. Implémentez en Java le même problème mais en utilisant cette fois un **moniteur**. Vous pouvez prendre en compte le fait que votre base est un objet Java et peut jouer le rôle de moniteur. Vous aurez besoin d'un booléen pour savoir si un écrivain tient la base et d'un compteurs de lecteurs, comme dans la solution par sémaphores. Vous utiliserez les instructions Java `wait` et textttnotify de la classe Object pour la mise en attente sur le moniteur, et vous devrez synchronized vos méthodes (sections critiques).

5 Exercice 4 (TP) – Producteur-consommateur

L'objectif de cet exercice de TP est de mettre en place une infrastructure de communication entre des threads.

Nous allons créer un système de type « producteurs - consommateurs » où un ensemble de threads produit des messages que consomme un autre ensemble. Chaque thread doit s'identifier par son numéro dans tous ses affichages (pour qu'on puisse suivre ce qu'il se passe).

1. Commencez par une version simple avec un seul producteur et un seul consommateur. Dans votre fichier, vous écrirez 4 classes:

   - la classe principale (publique) contenant la fonction static main;
   - une classe Producteur pour la thread productrice;
   - une classe Consommateur pour la thread consommatrice;
   - une classe BAL représentant la boite-aux-lettres, structure de communication entre le producteur et le consommateur. La boite aux lettre est contient une file de message représentée par une LinkedList (si vous n'avez pas l'habitude de cette classe, vous pouvez utiliser ArrayList qui est moins complète)

   Le producteur affiche un message chaque fois qu'il poste dans le BAL et le consommateur affiche le message lu dans la BAL.

2. Modifiez le programme précédent de manière à pouvoir traiter le cas de plusieurs producteurs et plusieurs consommateurs échangeant à travers une seule boîtes à lettres.

3. Complétez ce code pour mettre en place une politique de terminaison propre : au bout d'un certain nombre de messages (aléatoire), un producteur se termine et poste un message de fin dans la boite. Toutes les threads doivent alors arrêter de produire et se terminer proprement.

4. On rajoute un 3e type d'agent dans notre système: le perturbateur, qui inverse tous les messages de la boite aux lettres. Retirer les instructions `sleep` et ajoutez une thread perturbatrice. Que se passe-t-il? Expliquez.

Semaine 5 - Mémoire

N. Sabouret

1 Exercice 1 – Questions de cours

1. Expliquez ce qu'est l'édition de liens (en quelques mots).
2. Donnez trois principales méthodes d'allocation mémoire.
3. En quoi consiste la pagination de la mémoire ?
4. Quel est le principal problème de la pagination ?
5. Donnez les deux grandes classes d'algorithmes de remplacement de pages.

2 Exercice 2 – Allocation mémoire contigüe

1. Soit un système à partitions fixes de mémoire avec allocation contiguë. A un instant donné les partitions libres sont (par ordre croissant des adresses):

   10K 5K 20K 18K 7K 9K 4K 12K 15K

   On considère une liste d'arrivée des processus qui demandent 12K 10K 9K

   Donner le comportement des algorithmes selon les stratégies de la première zone libre (First Fit), du meilleur ajustement (Best Fit) et du plus grand résidu (Worse Fit).

2. Calculer le taux de fragmentation pour chaque cas. Quel est le meilleur algorithme ?

3. On se place maintenant dans un système disposant de 1000 Ko de mémoire haute non partitionnée et on applique un algorithme de swapping pour allouer la mémoire des processus. Les processus se présentent à la mémoire dans l'ordre suivant:

   - P1: 212Ko de mémoire, durée 2ms
   - P2: 417Ko de mémoire, durée 4ms
   - P3: 112Ko de mémoire, durée 6ms
   - P4: 426Ko de mémoire, durée 3ms

   Déroulez l'algorithme sur la même séquence de processus (212Ko, 417Ko, 112Ko et 426Ko dans cet ordre) et donnez la valeur de la fragmentation à chaque étape.

3 Exercice 3 – Pagination

On considère un système utilisant la technique de pagination et ayant les caractéristiques suivantes :

- Une table de page ayant 2^{16} entrées
- Chaque entrée de la table de pages est codée sur 16 bits.
- Une entrée contient un numéro de cadre de page et un bit de de présence/absence
- Le déplacement (offset) est codé sur 16 bits
- Une adresse virtuelle indexe 1 octet

1. Quelle est la taille d'une page?
2. Quelle est la taille de la mémoire physique?
3. Quelle est la taille de la mémoire virtuelle?
4. En considérant les huit premières entrés de la table de page présentée par la figure suivante, donner les adresses logiques correspondantes aux adresses physiques 33792 et 66048?

N° cadre de page	N° de page	Bit de présence/absence
7	0	0
6	0	0
5	0	1
4	1	1
3	0	0
2	0	0
1	2	1
0	3	1

4 Exercice 4 – Algorithmes de remplacement

Durant son exécution, un programme accède successivement à la liste suivante de pages virtuelles de son espace d'adressage:

```
0, 1, 4, 2, 0, 1, 3, 0, 1, 4, 2, 3
```

On suppose que le système d'exploitation a alloué 3 cadres de pages au processus. Donner la suite des pages présentes en mémoire ainsi que le nombre de défauts de page pour chacun des cas suivant :

1. Si on utilise l'algorithme de remplacement FIFO;
2. Si on utilise l'algorithme de remplacement « clock-based »;
3. Si on utilise l'algorithme de remplacement LRU.

Enfin, quel serait le remplacement optimal (si on pouvait deviner à l'avance les appels que va faire le programme) et donc le taux de défaut de page minimum?

Vous pouvez donner l'allocation sous la forme d'un tableau où figurent:

- sur la première ligne les pages appelées
- chaque ligne suivante représente un cadre de page. On indique la valeur de la nouvelle page dans le ligne correspondant au cadre de page utilisé.

Cette représentation permet de compter facilement les défauts de page.

5 Exercice 5 – Gestion de la mémoire en C

L'allocation dynamique de la mémoire consiste à étendre, pendant l'exécution d'un programme, la mémoire qui lui est attribuée.
Les principales fonctions d'allocation dynamiques sont :

- **malloc** pour allouer un bloc de mémoire,
- **calloc** pour allouer un bloc de mémoire et l'initialiser à zéro,
- **realloc** pour agrandir la taille d'un bloc de mémoire,
- **free** pour libérer un bloc de mémoire.

Ces fonctions se trouvent dans la bibliothéque standard <**stdlib.h**> Les prototypes de ces quatres fonctions sont décrits dans leurs pages `man`. Le type size_t qui est utilisé est equivalent au type unsigned long int sous Linux.

1. On souhaite créer un tableau de n entiers dans une fonction. Sachant que les tableaux sont gérés comme des pointeurs en C, expliquez pourquoi le code suivant n'est pas correct:

   ```
   int * creer_tableau(int n) {
     int tab[n];
     return tab;
   }

   int main() {
     int * t = creer_tableau(100);
     ...
     return 0;
   }
   ```

2. Écrire la fonction **int* allouer_tableau(int dimension, int val)** qui alloue la mémoire d'un tableau de taille **dimension**, puis qui l'initialise en mettant chacune de ses cases à la valeur **val**.

   Testez vos fonctions au fur et à mesure dans le programme (fonction `main`).

3. Écrire une fonction **int* recuperer_n_entiers(int n)** qui récupère n entiers entrés au clavier et les stocke dans un tableau de taille n. Vous pouvez utiliser la fonction `allouer_tableau` définie précédemment.

   Testez vos fonctions au fur et à mesure.

4. Écrire la fonction **void_liberer_tableau(int* vecteur)**, qui libère le tableau **vecteur** et une fonction **void afficher_tableau(int* vecteur, int dimension)** qui affiche ce vecteur d'entiers (de taille **dimension**).

   Testez vos fonctions.

5. Écrire une fonction **int* recuperer_entiers(int n, int taille_max)** qui après avoir alloué un tableau **tab** de taille **n**, récupère des entiers entrés au clavier. Lorsque plus de n entiers sont récupérés, cette fonction augmente la taille du tableau tab de n jusqu'à ce que la taille de tab soit supérieure ou égale à taille_max

   Pour tester la fonction, on pourra prendre, par exemple, $n = 7$ et $taille\_max = 15$.

6. Écrire la fonction **int** allouer_matrice(int lignes, int colonnes, int val)** qui alloue la mémoire d'une matrice de taille **lignes** $\times$ **colonnes** puis qui initialise tous ses éléments à la valeur **val**.

7. Écrire la fonction **void liberer_matrice(int** matrice, int n)**, qui libère la matrice **matrice**. Ensuite, écrire une fonction **void afficher_matrice(int** matrice, int lignes, int colonnes)** qui affiche une matrice d'entiers de taille **lignes** $\times$ **colonnes**. Vous pouvez utiliser les fonctions définies sur les tableaux.

   Tester vos fonctions.

Semaine 6 - Mémoire (suite)

N. Sabouret

1 Exercice 1 – Questions de cours

1. Donnez deux avantages de la segmentation.
2. Comment est déterminée une erreur de segmentation?
3. Comment peut-on combiner segmentation et pagination? Expliquez pourquoi on souhaite le faire.
4. Donnez deux avantages de la mémoire virtuelle.
5. Qu'est-ce que l'allocation proportionnelle de pages?

2 Exercice 2 – Segmentation

1. On suppose la table de segments suivante:

Segment	base	limite
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

   Quelles sont les adresse physiques des adresses logiques suivantes (données en base 10, le premier chiffre correspondant au segment):

   - 0430
   - 1010
   - 2500
   - 3400
   - 4112

2. On considère une mémoire segmentée-paginée pour laquelle les cadres de page en RAM sont de 4 Ko. La mémoire compte au total 16 cadres de pages numérotés de 0 à 15. Dans ce contexte, on considère deux processus A et B.

   - Le Processus A a un espace d'adressage composé de trois segments S1A, S2A et S3A qui sont respectivement de 6 Ko, 11 Ko et 1 Ko.

     Pour le processus A, seules les pages 1 et 2 du segment S1A, la page 2 du segment S2A et la page 1 du segment S3A sont chargées en mémoire, respectivement dans les cadres de page 4, 5, 10 et 6.

- Le processus B a un espace d'adressage composé de deux segments S1B et S2B qui sont respectivement de 13 Ko et 7 Ko.

  Pour le processus B, seules les pages 2 et 3 du segment S1B et la page 1 du segment S2B sont chargées en mémoire, respectivement dans les cadres 11, 2 et 15.

Représentez sur un dessin les structures allouées (table des segments, tables des pages) et la mémoire centrale correspondant à l'allocation décrite.

3. Le processus A souhaite accéder aux données et aux instructions situées sur le segment S2A aux adresses 8180, 8184, 8188 et 8192. Expliquez ce qu'il se passe en précisant les adresses physiques de ces instructions et comment elles sont obtenues.

3 Exercice 3 – Mémoire virtuelle

On considère un système de gestion de mémoire paginée a deux niveaux tel que :

- les adresses virtuelles et physiques sont toutes deux codées sur 32 bits ;
- les 10 premiers bits de l'adresse virtuelle forment le premier index, les 10 bits suivants forment le deuxième index, et les 12 bits restants le déplacement ;
- on suppose que chacune des entrées de ces tables est sur 32 bits.

1. Quelle est la taille maximale de la table des pages ? Quelle est la taille maximale de l'espace d'adressage utile ?
2. Avec cette structure mémoire, quelle fraction de l'espace mémoire d'un processus doit-on utiliser pour gérer sa table des pages ? Discuter la valeur du ratio entre l'espace mémoire du processus et celui de sa table de pages en fonction de la taille du processus et d'autres considérations pertinentes.

4 Exercice 4 – Gestion de la mémoire en C

La fonction **void *memcpy(void *dest, const void *src, size_t n)** de la bibliothèque <**string.h**> permet de copier **n** octets depuis la zone mémoire allouée **src** vers la zone mémoire **dest**. Attention : Les deux zones ne doivent pas se chevaucher.

La fonction **void *memset(void *dst, int c, size_t n)** remplit les **n** premiers octets de la zone mémoire allouée vers laquelle pointe **dst** avec le caractère dont le code ASCII est l'entier **c**.

La fonction **void *memmove(void *dest, const void *src, size_t n)** copie **n** octets depuis la zone mémoire **src** vers la zone mémoire **dest**. Les deux zones peuvent se chevaucher.

1. On souhaite utiliser la fonction **memcpy()** pour remplir un tableau de taille n avec n valeurs identiques (la valeur v sera passée en paramètre). Pour être plus efficace en mémoire, on souhaite utiliser une méthode de remplissage appelée "recopie de mémoire". Le principe est le suivant: au départ, seule la première case du tableau est initialisée (avec la valeur v). On recopie son contenu dans la case suivante puis on double la taille de l'espace considéré et on recommence (on recopie les deux premières cases dans les 2 suivantes, puis les 4 premières cases dans les 4 suivantes, etc). L'avantage de cette

méthode (par rapport à un remplissage classique de tableau) est qu'elle va mobiliser l'unité de gestion de la mémoire sans repasser par le processeur à chaque nouvelle case à remplir (chaque bloc sera recopié en une seule instruction pour le processeur).

Complétez le code suivant:

```
int* initialiser_vecteur_v1(int n, int valeur){
  int len = 1;
  int *vecteur = malloc(...);
  ....
  while(2*len<=n){
    .....
    .....
  }
  .....
  return vecteur;
}
```

2. En utilisant la fonction **memset**, écrivez la fonction **char* initialiser_tableau_char(int dimension, char c)** qui initialise les cases d'un tableau de caractère de taille **dimension**, en mettant dans chacune de ses cases le caractère **c**.

   Pour information, la fonction **memset** est justement l'implémentation de la méthode d'allocation que vous avez complétée ci-dessus.

3. En utilisation la fonction **memmove**, écrivez la fonction **char* copier_tableau_char(char* src, int dimension)** qui copie les cases d'un tableau de caractère **src** de taille **dimension** et retourne le pointeur vers la zone copiée.

   En regardant dans la documentation, expliquez la différence avec **memcpy**.

Semaine 7 - Système de fichiers

N. Sabouret

1 Exercice 1 – Questions de cours

1. Quels sont les éléments constitutifs de la structure d'un fichier? Expliquez leur rôle.
2. Quelles sont les opérations de bas niveau offertes par l'OS (sous forme d'appels systèmes) pour manipuler les fichiers?
3. Pourquoi faut-il ouvrir les fichiers?
4. Quels sont les différents modes d'accès possibles à un fichier (selon l'OS)?
5. À quoi servent les liens?
6. Donnez trois modes de contrôle d'accès différents.

2 Exercice 2 – Système de fichiers

On considère un système de fichiers tel que l'information concernant les blocs de données de chaque fichier est donc accessible à partir du i-noeud de celui-ci (comme dans UNIX). On supposera que :

- Le système de fichiers utilise des blocs de données de taille fixe 1K (1024 octets) ;
- L'i-noeud de chaque fichier (ou répertoire) contient 12 pointeurs directs sur des blocs de données, 1 pointeur indirect simple, 1 pointeur indirect double et 1 pointeur indirect triple.
- Chaque pointeur (numéro de bloc) est représenté sur 4 octets.

1. Quelle est la plus grande taille de fichier que ce système de fichiers peut supporter ?
2. On considère un fichier contenant 100,000 octets. Combien de blocs de données sont-ils nécessaires (au total) pour représenter ce fichier sur disque ?

3 Exercice 3 – Manipulations de fichiers

1. Créez un fichier texte appelé *fichier1* dans un nouveau répertoire appelé *ex2*. Sous Linux, il existe deux sortes de liens, les liens symboliques et les liens durs.
2. Modifiez les permissions de *fichier1* créé de telle façon que vous pouvez le lire, le modifier et l'exécuter alors que votre collègue puisse le lire mais ne puisse ni le modifier ni l'exécuter.
3. Créez un lien dur *liendur* sur ce *fichier* à travers la commande **ln**.

4. Créez ensuite un lien symbolique *liensymbolique* sur ce même fichier.

5. Avec la commande ls, affichez les numéros des inodes pour les trois fichiers. Que constatez-vous?

6. Supprimez le fichier original, essayer d'afficher le contenu des deux liens, que remarquez-vous? Expliquez?

4 Exercice 4 – Manipulation des fichiers en C

Le but de l'exercice est de gérer les comptes bancaires des clients en bancque. On représente un compte d'un client (ou fiche de client) par la struture suivante:

```
struct fiche {
  int numero;
  char nom[25];
  char prenom[25];
  float solde;
}
```

1. Créez *void creer(char* nom_fichier)* une fonction qui prend en paramétre le *nom_fichier* et qui crée le fichier *nom_fichier*.

2. Écrire une fonction *void ajout(char* nom_fichier)* qui permet de lire une fiche d'un compte à partir de l'entrée standard et de l'ajouter dans un fichier *nom_fichier*.

3. Ecrire une fonction *void afficher(char* nom_fichier)* qui permet d'afficher les fiches bancaires de tous les clients.

4. Ecrire une fonction qui permet d'afficher la fiche d'un client connaissant son numéro (Utilisez la fonction **fseek**.

Semaine 8 - Système de fichiers (suite)

N. Sabouret

1 Exercice 1 – Questions de cours

1. Quelles est la taille standard d'un bloc de fichier? Peut-on la modifier?
2. Qu'est-ce que le Master File Table?
3. Qu'est-ce qu'un File Control Bloc ou *inode*?
4. Quels sont les avantages et inconvénients des allocations contiguë, chaînée et indexée pour l'utilisation de blocs libres?
5. Qu'est-ce qui définit l'emplacement d'un bloc sur un disque?
6. Qu'est-ce qu'un code correcteur? Expliquez le principe général.

2 Exercice 2 – Ordonnancement d'accès aux blocs

On considère un disque contenant 256 cylindres (numérotés de 0 à 255) et les demandes d'accès suivantes aux cylindres, avec leur date d'arrivée (les requêtes arrivées à la même date sont donnés dans l'ordre d'arrivée: on suppose que le contrôleur ne met à jour sa liste de requêtes que tous les 100 unités de temps). On suppose que la tête est en position 24 au début.

temps 0: 54, 13, 22, 188, 245, 98, 24, 167

temps 100: 67, 93, 12, 25, 250, 220, 200

temps 200: 94, 230, 97, 30, 20

On suppose qu'il faut une unité de temps pour parcourir un cylindre et, le cas échéant, lire les données.

1. Appliquez l'algorithme FCFS et calculez le temps de traitement total
2. Appliquez l'algorithme SSTF et calculez le temps de traitement total
3. Appliquez l'algorithme C-SCAN (en supposant qu'on est ascendant au début) et calculez le temps de traitement total
4. Quel est le meilleur algorithme sur cet exemple?

3 Exercice 3 – Droits d'accès

1. Essayer de supprimer ou de modifier le fichier */etc/passwd*. Que se passe-t-il? Expliquer à l'aide de la commande *ls -l*.

2. A l'aide de la commande *id*, vérifier votre identité et le(s) groupe(s) au(x)quel(s) vous appartenez.

3. Créer un fichier texte nommé *Lecture* (de contenu quelconque), qui soit lisible par tout le monde, mais non modifiable (même pas par vous).

4. Créer un répertoire nommé *Secret*, dont le contenu est visible uniquement par vous même. Les fichiers placés dans ce répertoire sont-ils lisibles par d'autres membres de votre groupe?

5. Créer un répertoire nommé *Connaisseurs* tel que les autres utilisateurs ne puissent pas lister son contenu mais puissent lire les fichiers qui y sont placés.

6. Un administrateur désire s'assurer chaque matin que tous les fichiers placés sous $REPERTOIRE sont lisibles par tout le monde, mais non modifiables excepté par leur propriétaire. Quel doit être le mode de ces fichiers et répertoires?

7. La commande `umask` permet de définir les droits que les fichiers et les répertoires ont à leur création (voir la page `man`).

   Déterminer un masque permettant d'attribuer les droits désirés aux nouveaux répertoires :

	Répertoires		
Droits maximaux	rwx	rwx	rwx
Masque	???	???	???
Droits désirés	rwx	rw-	r–

4 Exercice 4 – Gestion des fichiers sous Shell

Créer un script permettant d'afficher la liste des fichiers du répertoire */etc* accessibles en lecture.

5 Exercice 5 - Manipulation des fichiers en Java

La gestion de fichiers proprement dite se fait par l'intermédiaire de la classe *File* du package *java.io.File*. Cette classe possède des méthodes qui permettent d'interroger ou d'agir sur le système de gestion de fichiers du système d'exploitation. Un objet de la classe `File` peut représenter un fichier ou un répertoire.

1. Écrire la méthode `public static void racine()` qui affiche toutes les racines du système. On utilise la méthode statique `listRoots` de la classe `File`. Cette méthode renvoie un tableau des racines du système.

2. Écrire la méthode `public static boolean test_fichier(File file)` qui affiche le type de `file` (fichier ou répertoire) et retourne `true` si `file` est un fichier ou `false` si c'est un répertoire. Utiliser les méthodes `isFile` et/ou `isDirectory`.

3. Écrire la méthode `public static void creer_fichier(File file)` qui teste si le fichier `file` existe, s'il n'existe pas, il est créé en utilisant la méthode `createNewFile`.

4. Écrire une méthode `public static void lister(File file)` qui affiche tous les fichiers et répertoires contenu dans un répertoire `file` passé en paramètre. Vous pouvez utiliser les méthodes `listFiles` et `test_fichier` .

5. Écrire une méthode *public static void lister_ rec(File file)* qui affiche récursivement le contenu des répertoires, c'est-à-dire qui affiche tous les fichiers et répertoires contenu dans un répertoire, ainsi que le contenu des répertoires, etc.

6. Écrire une méthode *public static void lire(String filename)* qui ouvre le fichier et affiche son contenu sur la sortie standard.

7. Vous devrez rendre un programme complet dans lequel toutes vos fonctions sont testées. Vous mettrez en commentaire un copier-coller de votre sortie standard (console Eclipse) qui montre les résulats de l'exécution de votre programme.

Semaine 9 - Système de fichiers (suite et fin)

N. Sabouret

1 Exercice 1 – Questions de cours

1. Quels sont les avantages et inconvénients du transfert de fichier anonyme et du transfert avec identification? Illustrez votre réponse par des exemples.
2. Quels sont les principaux services de nommage?
3. Quel est l'élément de base du fonctionnement de NFS?
4. Comment se fait l'identification en NFS?
5. À quoi correspondent les niveaux RAID0, RAID1 et RAID5? Quels sont les avantages de chacun?

2 Exercice 2 – RAID

Soit les données hexadécimales suivantes à écrire sur un agrégat par bandes avec parité constitué de 6 disques:

```
FF 16 27 39 A2 - 17 47 12 BD E3 - 89 C4 38 62 8C
```

On suppose que les bandes sont constituées d'un seul octet par disque, que la première bande utilise le sixième disque pour stocker la parité. La deuxième bande verra sa parité sur le cinquième, la troisième sur le quatrième, etc.

1. Quelle est la capacité totale de ce sous-système RAID5 sachant que chaque disque comporte 60Go d'espace disponible?
2. Déterminer sur quel disque sera écrit le premier bloc de chaque bande?
3. Calculer la parité associée à chaque bande et compléter le tableau suivant :

	Disque 0	Disque 1	Disque 2	Disque 3	Disque 4	Disque 5
Bande 0						
Bande 1						
Bande 2						

4. Supposons que le disque 1 est tombé en panne. Comment faire pour recalculer les trois parties de bandes effacées de ce disque.

3 Exercice 3 – Code de Hamming (7,4)

Le code de Hamming (7,4) est un code correcteur d'erreur. Il repose sur l'utilisation de 3 bits de parités (C_0, C_1, C_2) pour 4 bits de données (D_0, D_1, D_2, D_3) (d'où son nom). Les bits sont placés de la manière suivante dans le mot de 7 bits enregistré sur le disque:

$$D_3, D_2, D_1, C_2, D_0, C_1, C_0$$

Les valeurs de C_0, C_1 et C_2 sont calculées de la manière suivante:

- C_0 code la parité de $D_0 + D_1 + D_3$ (xor)
- C_1 code la parité de $D_0 + D_2 + D_3$
- C_2 code la parité de $D_1 + D_2 + D_3$

On vérifie un mot enregistré en calculant C'_0, C'_1 et C'_2 de manière similaire:

- C'_0 est la parité de $C_0 + D_0 + D_1 + D_3$
- C'_1 est la parité de $C_1 + D_0 + D_2 + D_3$
- C'_2 est la parité de $C_2 + D_1 + D_2 + D_3$

S'il n'y a pas d'erreur, ces trois bits doivent prendre la valeur 0. Sinon, leur valeur donne la position du bit faux.

1. On souhaite stocker le mot `1010` (D_0 est le bit de fin). Donnez le mot de Hamming créé.
2. Vérifier le code de Hamming sur le mot suivant: `1010110`. S'il y a erreur, déterminez le bit erronné.
3. Vérifier le code de Hamming sur le mot suivant: 1010000. S'il y a erreur, déterminez le bit erronné.

4 Exercice 4 – Partage NFS

NFS est un service qui permet à un ordinateur d'accéder à des fichiers à distance: la machine sur laquelle les fichiers sont localisés est dite serveur, et les machines pouvant formuler des demandes d'accès à ces fichiers sont des clients.

Les clients vont avoir le droit de "monter" (commande `mount`) des (parties de) partitions du serveur.

Le fichier `/etc/exports` donne la liste des partitions autorisées à l'exportation, ainsi que d'éventuelles restrictions quant à l'exportation (seulement certaines machines sont autorisées, leurs droits d'accès sont restreints,?).

1. Ce TP se déroule sur une machine virtuelle sur laquelle vous pouvez être administrateur (`root`).
   - Redémarrez votre ordinateur sous CentOS et connectez-vous avec votre compte utilisateur habituel.
   - La machine virtuelle est la même que votre machine réelle, précédée d'un `x`. Par exemple, si vous êtes sur la machine `toto.dep-informatique.u-psud.fr`, votre machine virtuelle est `xtoto`.

     Pour accéder à la machine virtuelle, vous pouvez:

  - Utiliser ssh: `ssh root@xtoto`
  - Utiliser virt-manager: `sudo virt-manager` (ou depuis le menu)

  Le mot de passe de `root` est `fiifo640` sur la machine virtuelle.

- Désactivez le firwall de votre machine virtuelle (sous virt-manager, une fois connecté, vous pouvez accéder au firewall via le menu système).
- Vous pouvez vérifier l'adresse IP de votre machine à l'aide de la commande `ifconfig`.

Le TP est à faire en binôme: un étudiant en `root` sur une machine virtuelle qui sera le serveur, et un autre étudiant sur une machine client (machine virtuelle ou compte habituel). Vous pouvez aussi faire l'exercice seul, sur la même machine (la machine virtuelle étant le serveur, l'OS réel étant le client: chacune a une adresse IP routable).

2. En utilisant la commande `man`, indiquez la structure du fichier `/etc/exports` et expliquez chaque champ.

3. Sur le serveur, créez un dossier (contenant des fichiers) et exportez le avec les droits d'écriture et de lecture. Vous devrez alors (re)démarrer votre serveur NFS:

   ```
   # service nfs restart
   ```

   Vous pouvez vérifier les répertoires exportés NFS à l'aide de la commande `exportfs`.

4. Sur le client, montez le dossier exporté avec la commande `mount`. Donnez la syntaxe de la commande à utiliser pour monter le répertoire distant.

5. Testez les accès (en lecture et en écriture) à partir du client (lister le contenu d'un répertoire et le contenu d'un fichier)

Semaine 10 - Entrées-Sorties

N. Sabouret

1 Exercice 1 – Questions de cours

1. Comment fonctionne un port ou un bus d'E/S?
2. Quelles sont les différences entre le bus PCI, le bus d'extension et le bus SCSI?
3. À quoi sert le contrôleur d'interruptions?
4. Qu'est-ce qu'un accès direct à un périphérique (dans l'OS)?

2 Exercice 2 – Redirection des entrées/sorties Shell

Les entrées/sorties standard sont représentées par trois fichiers standards qui sont associés à chaque commande du shell Linux :

Flux	Nom	Sescripteur	Matériel
Entrée standard	stdin	0	Associée au clavier
Sortie standard	stdout	1	Associée à l'écran pour afficher les résultats d'une commande
Sortie erreur standard	stderr	2	Associée à l'écran pour afficher les messages d'erreur d'une commande

1. Exécuter `ls -l > fichierls`. Ensuite, `ls -l >> fichierls`. Conclure sur le rôle de `>` et `>>`.
2. Utiliser la commande `grep` pour afficher les lignes contenant le mot `home` de `fichierls` sans tenir compte des majuscules et minuscules.
3. Utiliser la commande `wc` pour compter le nombre de lignes dans `fichierls`. Ensuite le nombre de caractéres.
4. Utiliser la commande `cut` sur le fichier `/etc/passwd` pour afficher pour chaque nom du compte son répertoire d'accueil.

3 Exercice 3 - E/S en C

1. A l'aide de la fonction *fgetc* écrire :
   - La fonction *int compter_lettres (FILE *f)* qui rend le nombre de caractères d'un fichier.
   - La fonction *int compter_lignes(FILE *f)* qui rend le nombre de lignes d'un fichier.
   - La fonction *int compter_mots (FILE *f)* qui rend le nombre de mots d'un fichier.

   Vous devrez écrire une fonction *main* pour tester vos fonctions au fur et à mesur.

2. A l'aide des fonctions *fwrite* et *fread*, écrire les fonctions *int mon_putc(char c, FILE *f)* et *char mon_getc(FILE *f)* qui agissent comme les fonctions *putc* et *getc* de la bibliothèque système.

3. Écrire la fonction *void copier_fichier_v1 (FILE *fs,FILE *fd)* qui copie le contenu d'un fichier source (*fs*) dans un fichier destination *(fd)* en utilisant les fonctions *fgetc* et *fputc*.

4. Écrire la fonction *void copier_fichier_v2 (FILE *fs,FILE *fd)* identique à la précédente mais qui utilise les fonctions *fscanf* et *fprintf*.

4 Exercice 4 - E/S en Java

1. Écrire la fonction *public static void lire_fichier(File fichier)* qui lit et affiche le contenu de *fichier*. Utiliser les classes *FileReader* et *BufferedReader*

2. Écrire la fonction *public static int counter_lignes(File fichier)* qui rend le nombre de lignes contenus dans *fichier*.

3. Écrire la fonction *public static void ecrire_v2(File fichier, String[] data)* qui permet d'écrire dans *fichier* les éléments de *data*, à raison de 1 élément par ligne.