Licence 2 - Informatique

Info4B - Principes des systèmes d'exploitation

Fascicule pour les Travaux Dirigés

Éric Leclercq

Révision : Janvier 2021

Résumé

Ce document contient l'ensemble des exercices de travaux dirigés du module de Principes des Systèmes d'Exploitation. Les exercices notés avec une étoile sont des extraits ou des compilations d'exercices d'examen ou de contrôle continu. Tous les exercices ne seront pas forcement traités durant les séances de travaux dirigés.

Table des matières

1 Notion de système d'exploitation

Exercice 1. Composants fondamentaux d'une machine

La figure 1 présente la carte mère d'un ordinateur de type PC (*Personnal Computer*). En vous appuyant sur les exemples d'architecture vus en cours, réalisez un schéma de l'architecture de la carte mère et identifier les éléments suivants et réaliser un schéma tel que celui présenté en cours pour une architecture de carte mère à base de *chipset* AMD :

— le logement du processeur (qui n'est pas présent) ;
— le(s) processeur(s) d'entrée-sortie ;
— les bus mémoire et périphériques (PCI), le bus E-IDE ;
— les logements pour la mémoire ;
— les logements pour les cartes d'entrée/sortie.

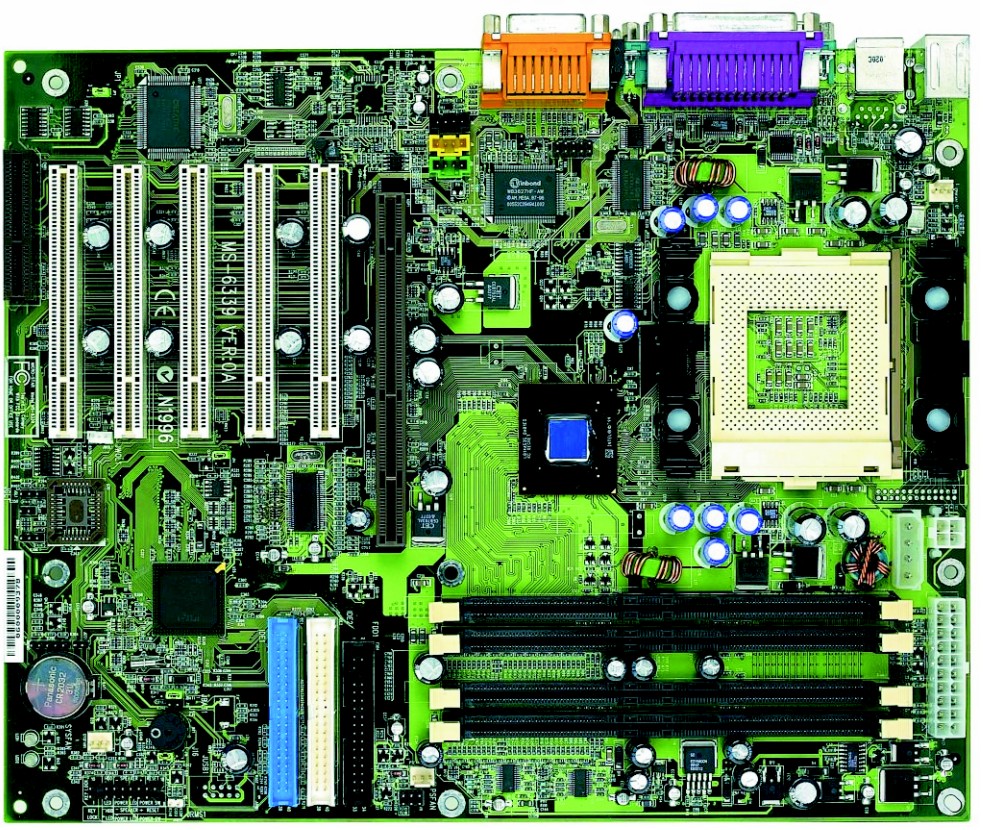


FIGURE 1 – Carte mère d'un ordinateur personnel (PC)

Exercice 2. Fonctionnalités d'un systèmes d'exploitation

Dans le domaine informatique, le système d'exploitation est un ensemble de programmes ajouté à la structure matérielle de la machine pour en tirer le meilleur parti.

Hors du cadre informatique, le système d'exploitation d'un objet (ou d'un système) peut être vu comme un ensemble d'éléments et/ou de procédures ajoutés aux éléments de base constituant l'objet (ou le système) afin d'assurer au mieux son exploitation (utilisation).

On considère les systèmes suivants :

— un photocopieuse utilisée par plusieurs personnes ;
— un téléphone portable ;
— une machine à laver ;
— les guichets d'une poste, ouverts à des clients ayant potentiellement plusieurs opérations de natures différentes à effectuer ;
— un carrefour routier dans un ville.

Questions :

1. Rappeler les cinq propriétés qui permettent de juger de la qualité d'un système d'exploitation. Définir chaque propriété en quelques mots.
2. Par analogie avec les composants d'un ordinateur et les structure utilisées dans un SE, identifier quelques points communs entre ces systèmes ?
3. Au travers des cinq propriétés précédentes et en spécifiant pour chaque cas ce qui représente la notion d'exploitation, illustrer ce pourraient représenter les systèmes d'exploitation des cinq systèmes décrits plus haut ;

Exercice 3.

On considère un ensemble de travaux $T = t_i, i = 1..n$ à effectuer par un ordinateur de façon séquentielle (les travaux sont exécutés les uns à la suite des autres). Pour simplifier, on suppose que tous les travaux sont constitués de trois phases :

1. lecture du programme et des données l_i (5 unités de temps)
2. exécution du programme e_i (30 ut)
3. affichage et enregistrement des résultats a_i (15 ut)

De plus on suppose que le temps mis pour passer d'un travail à un autre est négligeable. On considérant que $n = 30$, c'est-à-dire que 30 travaux sont à exécuter séquentiellement, calculer le temps de traitement total et le taux d'utilisation du processeur (ou son rendement) dans les deux cas suivants :

1. l'unité centrale de traitement (processeur) gère les périphériques d'entrée sortie ;
2. les périphériques sont gérés par un processeur d'entrée/sortie qui peut travailler en même temps que le processeur et ils possèdent tous les deux un accès direct à la mémoire.

Exercice 4. Gestion des tâches asynchrones

1. Écrire un programme Java qui compte indéfiniment mais qui se termine lorsque l'utilisateur appuie sur une touche. Dans les langages de programmation, les entrées-sorties sont, dans la majorité des cas, gérées de manière bloquantes. Dans cet exercice vous devez donc utiliser un thread séparé pour ne pas bloquer le thread principal.
2. Reprendre le programme précédent et le faire s'arrêter au bout d'un temps donné (passé en paramètre au lancement du programme par l'utilisateur).

Exercice 5. Capacité de traitement et architecture

En vous basant sur les architectures de type Von Neumann présentées en cours, proposer au moins trois stratégies permettant d'augmenter la capacité de traitement d'un ordinateur. Pour chacune réaliser un schéma de l'architecture proposée.

2 Processus, concurrence et synchronisation

Exercice 6. Création de processus

On considère le programme suivant (écrit en pseudo code) :

```
programme génération
  début
    entier i,pid
    pour i:=1 jusqu'à 5
      pid:=fork()
    fin pour
   fin
```

1. Faire un schéma de l'arborescence de processus créée par l'exécution de ce programme.
2. Identifier les problèmes engendrés par ce programme.
3. Proposer une modification en utilisant le code de retour de la fonction `fork` afin de créer uniquement 5 processus fils d'un même processus parent.
4. Écrire le même programme (et sa version corrigée) en Java avec des threads.

Exercice 7. Ressource partagée et incohérence

On considère deux threads qui modifient une variable partagée (`int compteur`) l'une en l'incrémentant, l'autre en la décrémentant avec les opérations `compteur++` et `compteur--`.

1. Écrire le programme en Java.
2. Si les deux exécutions sont successives T1 pui T2 (ou l'inverse) quelle est la valeur du résultat dans compteur ?
3. Même question si les exécution s'entremêlent en s'éxécutant de manière alternée et chacun son tour.
4. Le code en pseudo assembleur de l'incrémentation est le suivant :

```
LDA @compteur    // charger le registre A du processeur avec
                 // le contenu de l'adresse désignée par compteur
INC              // incrémenter le registre A
STA @compteur    // stocker le registre A dans la mémoire
                 // désignée par l'adresse compteur
```

   (a) écrire le pseudo code de l'autre opération

   (b) montrer que une exécution avec une commutation de contexte à un endroit particulier conduira à un état incohérent

   (c) dans le programme Java, identifier la section critique

   (d) modifier le programme Java pour assurer l'exclusion mutuelle

Exercice 8. Sémaphores

(d'après Philippe Durif http://www2.lifl.fr/~durif/)

La définition de sections critiques, au moyen de sémaphores, de verrous ou de moniteurs permet le résoudre le problème des accès concurrent à une ressource partagée par plusieurs

threads. Cependant, la notion de section critique ne permet pas à plusieurs threads de se **synchroniser** (coordonner) les uns par rapport aux autres. Par exemple, un thread doit pouvoir attendre qu'un autre soit arrivé à un certain point de son exécution pour terminer un calcul.

La classe `Object` de Java propose les méthodes `wait()` et `notify()` pour réaliser une forme de synchronisation. Ainsi, en plus d'un verrou interne utilisé par le moniteur (via la primitive `synchronized`), chaque objet Java possède un `wait-set` constitué de l'ensemble des threads qui ont exécuté la méthode `wait()` sur cet objet. Ces threads sont dit en attente. Un de ces threads sera débloqué lorsqu'un autre thread appellera la méthode `notify()` sur ce même objet.

Le `wait-set` d'un objet peut être manipulé par les méthodes suivantes de la classe `Object` :

- `wait()` permet au thread qui l'exécute de se mettre en attente dans le `wait-set` de l'objet récepteur. Ainsi, un thread peut décider de suspendre sa propre exécution ;
- `notify()` permet à un thread de réveiller un des threads qui était en attente dans le `wait-set` de l'objet récepteur de l'appel de la méthode. Attention, le choix du thread réveillé par `notify()` n'est pas spécifié par le langage, ainsi il se peut qu'on observe des phénomènes de famine quand plusieurs threads sont en attente sur le même objet ;
- `notifyAll()` est semblable à `notify()`, cette méthode réveille tous les threads qui étaient en attente.

Pour assurer la cohérence du `wait-set`, les méthodes doivent être exécutées dans une section critique verrouillée par l'objet dont on manipule le `wait-set`. Ceci n'est vérifié qu'à l'exécution. Pendant le `wait()` le verrou est relâché et le thread devra le reprendre lorsqu'il sera réveillé. En revanche, les autres verrous éventuellement obtenus par le thread en attente ne sont pas relâchés.

1. Pourquoi la classe Java présentée dans le listing 1 n'est pas correcte pour réaliser un sémaphore. D'ailleurs lors de l'exécution la machine virtuelle retourne une erreur : `Illegal Monitor State Exception`.
2. Corriger la définition de la classe sémaphore du listing 1.
3. Donner un programme exemple montrant comment l'utiliser.
4. La classe du listing 2 peut provoquer un interblocage, identifier le problème et proposez une correction.
5. Reprendre votre correction de la classe sémaphore du listing 1 en considérant le problème de l'interblocage vu précédemment. Comment expliquez vous ce phénomène ?

```
1  package Verrouillage;
   public class SemaphorePreemptif{
3   private int compteur=0;

5   public SemaphorePreemptif (int c){
     compteur=c;
7   }

9   public void P() throws InterruptedException{
      while (compteur==0) {this.wait(); }
11    compteur--;
    }
13
```

```
public void V() {
   compteur++;
   this.notify();
  }
 }
```

Listing 1 – Classe sémaphore

```
package Verrouillage;
public class SemaphoreSpinLock{
 private int compteur=0;

 public SemaphoreSpinLock (int c){
  compteur=c;
 }

 public synchronized void P() {
   if (compteur==0) System.out.println("Attente P");
   while (compteur==0) { ; }
   compteur--;
   System.out.println("Verrou obtenu");
  }
 }

 public synchronized void V() {
   compteur++;
   System.out.println("Verrou relache");
 }
}
```

Listing 2 – Classe sémaphore

Exercice 9. Verouillage

En TP vous avez réalisé une classe `Verrou` permettant à une méthode de définir une section critique et de verrouiller l'accès à une ressource. Vous ne devez pas utiliser le package `concurrent`.

1. Comparer cette méthode aux moniteurs proposés par Java, identifier au moins deux problèmes. Pensez à qui a le droit de déverrouiller une ressource.
2. Proposer au moins deux améliorations de votre classe verrou (préciser lequel des problème cité précédemment est résolu). Décrire comment vous allez implanter vos propositions et réaliser la modification de votre classe `Verrou`.

Exercice 10. *Sleeping Barber*

Le problème dit *Sleeping-Barber Problem* est un exemple classique, il illustre les problèmes d'accès concurrents aux ressources et la synchronisation des processus. On considère un salon de coiffure qui comporte une salle d'attente avec n chaises et une autre salle avec un fauteuil et un coiffeur. Si il n'y a pas de client, le coiffeur dort. Si un client entre et que toutes les chaises sont occupées, le client s'en va. Si le coiffeur est occupé et qu'au moins une chaise est libre le client s'assied et attend. Si le coiffeur est endormi l'arrivée d'un client le réveille.

1. On souhaite programmer l'activité du coiffeur et le fonctionnement du système au moyen de threads. Identifier les classes qui constituent des ressources et les classes qui donneront naissance à des threads.
2. Y-a-t'il des problèmes de concurrence ? Si oui expliquez de façon précise la méthode que vous choisissez pour les éviter ?
3. Écrire les différentes classes du programme et la méthode `main()`.
4. On considère non plus 1 coiffeur mais 4 coiffeurs, quels sont les éléments à changer dans votre programme ?

Exercice 11. (*) Septembre 2009 (10pts)

Vous devez concevoir un programme pour simuler le fonctionnement d'un four utilisé dans une entreprise de mécanique qui fabrique des pièces en métal. Une zone de dépôt située avant le four contient 30 places pour accueillir des morceaux de métal brut qu'il faut chauffer pour pouvoir ensuite les travailler. Deux robots chargent le four qui comporte 6 compartiments pour accueillir simultanément 6 morceaux de métal. Les pièces sont déchargées automatiquement (sans l'intervention des robots) du four sur un convoyeur au bout d'un temps t (qui varie en fonction de la pièce et du métal).

1. Identifier les ressources, et les threads.
2. Des problèmes de concurrence peuvent-ils exister (illustrez votre réponse avec un exemple) ?
3. Écrire en Java les structures de données principales (classes) pour simuler le fonctionnement du four, des robots et de la zone de dépôt. Le corps des méthodes n'est pas à fournir.
4. Quelles structures doivent être protégées des accès concurrent ?
5. Proposer les constructions Java nécessaires à la résolution des problèmes de concurrence.
6. Écrire le programme principal qui lance les threads, simule l'arrivée aléatoire des pièces dans la zone de dépôt et simule le fonctionnement du four. Pour ce dernier on supposera que le temps de chauffage des pièces est un temps aléatoire d'un intervalle prédéfini.

Exercice 12. Simulation de parking

On souhaite réaliser une simulation de parking urbain. Le parking a un nombre de places fixé. Il dispose d'un ou plusieurs afficheurs indiquant le nombre de places restantes. L'entrée d'un véhicule n'est pas permise si le parking est plein. Le parking dispose de plusieurs entrées munies de barrières et de distributeurs de tickets. Les sorties sont également multiples (avec barrière et collecteur de ticket). On suppose que le paiement s'effectue sur une borne spécifique au milieu du parking.

1. On se concentre sur la simulation des entrées et des sorties, la validation du ticket n'est pas prise en compte. Identifier les ressources et les acteurs du système. Définir les différentes classes. Identifier les classes qui seront implantées comme des threads. Identifier les ressources qui peuvent être utilisées par plusieurs threads.
2. Y-a-t'il des problèmes de concurrence ? Si oui identifier les sections critiques et expliquez de façon précise la solution retenue pour éviter les problèmes de concurrence.

3. Écrire le programme.

Exercice 13. Les philosophes

Cinq philosophes sont assis en rond autour d'une table ronde. Un philosophe passe sa vie alternativement à penser et à manger. La table comporte 5 assiettes de spaghetti et 5 fourchettes. Pour manger un philosophe utilise deux fourchettes. Une fourchette est placée entre chaque philosophe. Pour manger, un philosophe utilise uniquement la fourchette placée à sa droite et celle placée à sa gauche. Un philosophe ne prend pas une fourchette qui est déjà détenue par un autre philosophe. Une fois qu'il a mangé, un philosophe repose les deux fourchettes qu'il a utilisé.

1. Identifier les éléments qui constituent les ressources et les threads
2. Expliquer le(s) problème(s) de concurrence
3. Proposer un programme pour simuler le fonctionnement des philosophes
4. La situation peut-elle faire apparaître un problème de verrou mortel ?
5. Comment résoudre ce problème : proposer au moins deux stratégies et expliquer en quoi elles sont différentes (ne se rapportent pas au même principe).

Exercice 14. Verrous mortels

1. Écrire un programme qui génère, un verrou mortel. Il faut utiliser au moins 2 threads, 2 ressources liées et des méthodes qui utilisent le liens entre les ressources pour afficher leur état.
2. Proposez deux stratégies pour résoudre le problème des verrous mortels.

Exercice 15. Interblocage prévention

1. Déterminer 4 conditions nécessaires pour qu'une application puisse générer une situation d'interblocage.
2. À partir de l'exercice des philosophes établir un graphe des ressources utilisées et des attentes.
3. Comment déterminer si il existe un interblocage (ou une situation conduisant à une interblocage) ?
4. Traduire le graphe sous le forme de listes de ressources en attente pour chaque processus, en déduire un algorithme de détection d'interblocage (verrous mortels).

Exercice 16. Barrière de synchronisation

On suppose qu'une application est découpées en activités constituées d'étapes. Une activité peut passer d'une étape i à une étape $i+1$ lorsque toutes les autres activités ont réalisé leur étape i. Ainsi, une activité doit attendre que toutes les autres activités aient terminé leur étape en cours pour passer à la suivante. Cette modélisation permet de simuler avec un système asynchrone des calculs synchrones.

1. Quelle est la relation entre barrière et sémaphore ?
2. Écrire la classe barrière au moyen d'une classe sémaphore.
3. Écrire la classe barrière en utilisant les méthodes `wait()`, `notifyAll()`.

4. Comment étendre la notion de barrière à des communication entre threads utilisant le réseau.

Exercice 17. (*) Threads vs Acteurs, concurrence (7pts)

1. Quels sont les problèmes engendrés par l'utilisation des méthodes de gestion de la concurrence en Java et en général dans les langages de programmation (`synchronized`, `lock`, `unlock`, `wait`, `notify`, etc.).
2. On souhaite implanter un système, inspiré de la notion d'acteur de Scala, utilisant des messages pour la communication entre threads. Chaque thread a un id unique et une seule méthode publique permettant de déposer un message dans sa boite aux lettres (stockée dans le thread et représentée par une file). Ainsi, un thread est libre de traiter comme il le veut les messages envoyés par les autres threads. Pour diffuser les messages entre les threads, un ou plusieurs threads routeurs sont utilisés.

   Pour envoyer un message, on invoque une méthode `envoyerMessage` en donnant l'id du thread destinataire, l'id du thread émetteur, et le contenu du message sous la forme d'un couple clé-valeur (par exemple Temperature,5).

   Spécifier [1] la classe `Routeur`. Écrire l'entête de la méthode `envoyerMessage`. Existe-t-il des problèmes de concurrence ?
3. Spécifier la classe `Acteur`, c'est-à-dire un thread pouvant envoyer et recevoir des messages (en connaissant un routeur). Existe-t-il des problèmes de concurrence ?
4. Donner un exemple de code d'envoi de message entre deux acteurs au moyen d'un routeur (instancier le routeur, les acteurs, invoquer une méthode d'envoi).
5. Que pensez-vous de l'utilisation de messages pour gérer les interactions entre processus ?

Exercice 18. (*) Lignes de tramway

Une ligne de tramway reliant deux stations (A et B) comporte un tronçon à voie unique (Tc - figure suivante). Les rames sont autonomes et se déplacent de A vers B ou de B vers A puis repartent dans l'autre direction en changeant de voie. Tous les trains engagés à un instant sur le tronçon à voie unique circulent tous dans le même sens.

1. Identifier les différentes classes et déterminer celles qui sont des threads et celles qui constituent des ressources. Donner la signature des classes et de leurs méthodes.
2. Chaque rame se déplaçant de A vers B commence par utiliser les tronçons T1, puis Tc, et enfin T2 puis arrivée en B, change de voie et repart dans l'autre sens en suivant les tronçons T3, Tc, T4 (elle effectue donc ainsi le trajet de B vers A). Le cycle ne s'interrompt pas. On souhaite gérer l'accès au tronçon à voie unique au moyen de sémaphore(s),. On suppose que le nombre de rames sur la voie unique est égal à 1. Écrire la ou les méthodes de déplacement.
3. Répondre à la question 2 avec l'hypothèse que le nombre de rames sur le tronçon à voie unique est illimité. Vous pouvez proposer des classes supplémentaires pour contrôler l'accès à Tc.
4. Répondre à la question 2 en supposant que la limite du nombre de rames sur la voie unique est fixé (N). Vous pouvez proposer des classes supplémentaires pour contrôler l'accès à Tc.

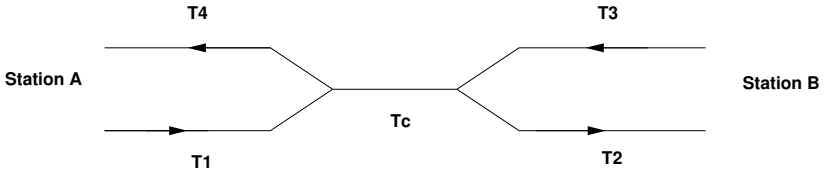


FIGURE 2 – Représentation schématique des voies

Exercice 19. (*) Robots (7pts)

On considère un système robotisé pour le déplacement de produits dans une entreprise. Les robots ne sont pas complètement autonomes, ils sont coordonnées par un programme central. On souhaite traiter le cas d'une intersection (figure 1), les robots arrivent de deux directions (lignes A et B) et doivent emprunter un chemin unique (ligne C). Pour éviter les collisions ils passent chacun leur tour au niveau de l'intersection (lignes A, puis ligne B) puis poursuivent leur chemin sur a ligne C.

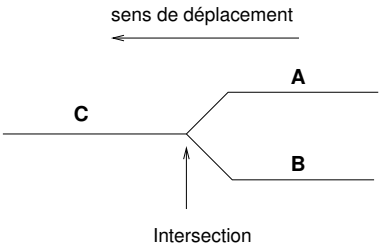


FIGURE 3 – Intersection

1. Identifiez les ressources et les threads. Décrire les classes que vous devez créer pour simuler de ce système.
2. Expliquer les problèmes de concurrence éventuels.
3. Si il n'y a pas de robots en attente sur une ligne les robots de l'autre ligne passent les uns à la suite des autres. Pour implanter ce fonctionnement on souhaite modifier la classe `Verrou` réalisée en TP en incluant un compteur pour savoir combien de demandes de verrouillage sont en attente et qui demande. Écrire la nouvelle version de la classe `Verrou`.
4. Écrire le programme principal (main) de la simulation des robots.
5. Écrire la classe qui implémente l'intersection.

Exercice 20. Prévention des verrous mortels

Afin d'éviter les verrous mortels on suppose que les processus doivent déclarer le nombre maximum de ressources qu'ils utiliseront. On appelle état sûre, un état du système qui permet d'allouer les ressources demandées par chaque processus dans un certain ordre en évitant les verrous mortels. Un système est dans un état sûre si il existe un séquence sûre.

1. Soit P_i une séquence de processus, proposer une définition de la notion de séquence sûre.

1. attributs et entête de méthode sans le corps des méthodes

2. Soit 3 processus et 1 type de ressource disponible en 12 exemplaires. Chaque processus a déclaré le nombre maximum de ressource dont il a besoin, soit respectivement 10, 4 et 9. Au temps t_k les processus ont respectivement 5, 2 et 2 ressources. Déterminer la séquence qui satisfait la condition d'état sûre.
3. Au temps t_l, on suppose que le processus P_3 demande et obtient une ressource de plus, montrez que le système n'est plus dans un état sûre et conduit à une situation d'interblocage.

Exercice 21. (*) Contrôle continu (8pts)

D'après un problème écrit par Anthony Joseph à l'Université de Californie à Berkeley.

À coté de Redmond dans l'état de Wahsington, il existe un ferry qui est utilisé à la fois par des hackers [2] Linux et des employées de Microsoft afin de traverser une rivière.

Le ferry peut transporter 4 personnes, il ne quittera pas la rive si il y a moins de 4 personnes ou si il y en a plus de 4. Afin de garantir la sécurité de tous, les configurations suivantes ne sont pas stables : 1 hacker, 3 employés Microsoft, ou 3 hackers et 1 employé Microsoft. Tout autre combinaison est considérée comme fiable (stable). Lorsque tout le monde est embarqué et que la configuration est stable, une des personnes invoque le capitaine pour lui dire de démarrer (peut importe qui le fait mais cela doit être fait). On ne se préoccupe pas du sens du voyage (aller ou retour).

1. Identifier les éléments qui constituent les ressources et les threads (il est conseillé de faire un schéma et de l'expliquer).
2. Expliquer le(s) problème(s) de concurrence et leurs solutions.
3. La situation peut-elle faire apparaître un problème de verrou mortel ?
4. Proposer un programme pour simuler le fonctionnement du ferry :
   (a) donner le squelette des classes (membres) ;
   (b) donner prototype (signature) des méthodes (ne pas donner le code constituant le corps de la méthode mais expliquer au moyen d'un commentaire ce que le méthode réalise) ;
   (c) écrire les méthodes `run` pour les thread ;
   (d) écrire la méthode `main`.

Exercice 22. (*) Examen 2010 - 8 pts

Le père Noël dort dans son magasin au pôle nord et ne peut être réveillé que si : 1) ses 9 rennes sont de retour de la coupe du monde de football (ils reviennent lorsqu'ils le souhaitent) 2) certains lutins ont des difficultés à construire des jouets et demandent de l'aide. Afin de permettre au Père Noël de dormir, les lutins n'ont le droit de le réveiller que si ils sont au moins 3 à avoir des problèmes. Si le Père Noël est réveillé et constate qu'à sa porte il y a 3 lutins, il les aide, si il y a aussi les 9 rennes, il prépare son traîneau et laisse attendre les lutins. Les nombre de lutins n'est pas déterminé. Le départ du Père Noël se fait le 24 décembre au matin.

1. Identifier les ressources et les threads ?

2. Hacker est à l'origine un mot anglais signifiant bricoleur, bidouilleur, utilisé pour désigner en informatique les programmeurs astucieux et débrouillards. Plus généralement il désigne le possesseur d'une connaissance technique lui permettant de modifier un objet ou un mécanisme pour lui faire faire autre chose que ce qui était initialement prévu (d'après Wikipédia).

2. Peut-il y avoir des problèmes de concurrence et de synchronisation de processus ? Comment les résoudre ?
3. La situation peut-elle faire apparaître un interblocage ?
4. Décrire les classes `Renne`, `Lutin` et `PereNoel` (uniquement les noms de méthodes et leurs paramètres)
5. Écrire la méthode pour simuler le comportement du Père Noël.

Exercice 23. (*) Examen 2015 (8 pts)

On considère un pont sur une rivière qui est juste assez large pour laisser passer le trafic sur une seule voie. Par conséquent les véhicules ne peuvent utiliser le pont de manière concurrente que si ils se dirigent dans la même direction. Un problème de sécurité apparaît si deux véhicules se déplaçant dans des directions opposées et entrent sur le pont en même temps.

1. Identifier la ou les ressources et les threads.
2. Expliquer les problèmes de concurrence.
3. Quelles méthodes ou constructions Java allez vous utiliser pour résoudre les problèmes de concurrence.
4. Spécifier la classe pont (sans le corps des méthodes)
5. On veut éviter le problèmes de sécurité au moyen de feux de signalisation. Bien entendu des files d'attente de véhicules vont se créer :
   (a) Identifier les nouvelles ressources et nouveaux threads éventuels.
   (b) Décrire les autres classes nécessaire à la modélisation de la situation.
   (c) Identifier les nouveaux problèmes de concurrence et préciser leurs solutions.
   (d) Écrire la classe feu de signalisation (uniquement les signatures de méthodes et variables membres). Expliquer comment les deux feux de signalisation vont synchroniser leur état.

Exercice 24. (*) Examen (8pts)

1. Écrire un programme Java dans lequel deux threads incrémentent chacun leur tour un compteur. Les threads s'arrêtent lorsque le compteur a atteint une valeur maximum définie au lancement du programme.
2. Modifier votre programme pour le généraliser à n threads, qui incrémenteront chacun leur tour le compteur (le thread 1 commence, puis le 2 et ainsi de suite, lorsque le thread n a terminé le thread 1 recommence).

Exercice 25. (*) Synchronisation de processus, rendez-vous (7pts)

On considère deux threads A et B, le thread A exécute une méthode composées de deux portions de code a_1 et a_2, le thread B suit le même schéma, les portions de code sont b_1 et b_2.

On veut garantir que la portion de code a_1 sera exécutée avant la portion de code b_2 et que la portion de code b_1 sera exécutée avant la portion a_2.

1. Proposez un programme dans la syntaxe Java utilisant les primitives de synchronisation `wait` et `notify`.

2. Reprendre le même programme en utilisant deux sémaphores afin d'indiquer respectivement que A ou B est arrivé au point de rendez-vous.
3. Un ordre particulier des primitives de synchronisation peut-il provoquer un verrou mortel ?

Exercice 26. (*) Contrôle continu (7pts)

On considère un carrefour routier de 3 routes (a,b,c) sans feu ni rond-point. La règle de priorité à droite du code de la route s'applique. Un véhicule venant de la route a peut aller vers b ou c, respectivement b vers c ou a et, c vers b ou a.

1. Identifier ressources et threads.
2. Déterminer les conditions pour avoir accès à la ressource carrefour ?
3. Peut-il y avoir un inter-blocage ? Si oui expliquer pourquoi la situation n'est pas exactement la situation minimum type vue en cours.
4. Écrire la portion de code Java permettant de faire passer un véhicule d'une route à une autre. Si besoin donner les variables membre des différentes classes.

Exercice 27. (*) Thread compteurs revisités (6pts)

On considère deux threads A et B qui comptent jusqu'à 10 000, l'un les nombre pairs, l'autre les nombres impairs.

1. Proposez un programme dans la syntaxe Java utilisant un sémaphore ou un verrou
2. Modifier votre programme pour utiliser `wait` et `notify`.
3. On ajoute un troisième thread puis n quelle solution utiliser (celle de la question 1 ou celle de la question 2), justifiez. Modifier le programme en conséquence.
4. Ces programmes peuvent-il provoquer un verrou mortel ?

Exercice 28. (*) Pipeline (7pts)

Soit p processus de type producteur (p est petit de 2 à 5) qui génèrent des objets o_i d'une classe C. Chacun de ces objets doit subir un certain nombre de transformations T_j, $j = 1, \ldots, k$ dont chacune est réalisée par un thread spécialisé. Les transformations sont effectuées dans l'ordre et dès qu'un thread a terminé avec un objet, il le rend disponible pour le thread suivant (indice $j + 1$) et en prend un autre. Tant qu'il y a des objets à traiter les threads sont actifs.

1. Faire un schéma du système, identifier le ou les types d'interactions, mettre en évidence les structures nécessaires à la communication entre les threads et à leur coordination.
2. Lister les classes nécessaires pour la réalisation du programme et indiquer celles qui sont des ressources et celles qui sont des threads.
3. Identifier et décrire les problèmes de concurrence. Comment les résoudre ? La simulation du fonctionnement du système peut-elle provoquer un verrou mortel ?
4. Donner le code Java pour le programme principal (`main`) pour 2 producteurs et 3 traitements (T).
5. Écrire la classe T_1 et ses méthodes pour un type de thread de traitement.

3 Ordonnancement

Exercice 29. Allocation du processeur, ordonnancement

Les processus suivants demandent à être exécuté par le processeur. Ils arrivent en file d'attente pour accéder au processeur au top de temps spécifié dans le tableau donné ci-après.

Nom	p0	p1	p2	p3	p4	p5
Top d'arrivée	1	3	4	12	15	16
Durée	2	11	5	8	7	3

Pour chacun des quatre systèmes décrits dans les questions suivantes :
— on suivra l'évolution de l'exécution de tous les processus (utiliser les annexes) ;
— on calculera le taux de retard de tous les processus ;
— on calculera le taux de retard moyen.
À l'issue des simulations vous devrez conclure sur le comportement de chaque stratégie vis-à-vis de processus longs et des processus courts.

1. **Traitement jusqu'à terminaison :** le processeur est réservé à un processus pendant toute la durée de son exécution. Pour suivre l'exécution de ce système, utiliser les colonnes 1 et 2 de l'annexe 1.

   (a) **Ordonnancement FIFO :** chaque nouveau processus est placé à la fin de la file d'attente

   (b) **Ordonnancement plus court d'abord :** le nouvel arrivant est placé avant les processus de durée plus longue (inter-classement).

2. **Traitement avec préemption :** on considère désormais un mécanisme d'accès au processeur par tourniquet. On suppose que chaque file d'attente du processeur est gérée par la méthode du plus court d'abord.

   (a) **Tourniquet simple :** le tourniquet est composé d'une seule file d'attente, le quantum de temps est de 4 unités. Pour suivre l'évolution du système, on utilisera la dernière colonne de l'annexe 1.

   (b) **Tourniquet à 3 files d'attente :** on considère maintenant un système à 3 files d'attente dont les quantums de temps associés sont 2, 3 et 4. Les règles de fonctionnement concernant l'allocation du processeur sont les suivantes :
   — un processus qui passe dans le processeur en arrivant de la file d'attente i est remis (lorsqu'il est vidé du processeur) dans la file d'attente i+1 ;
   — les processus d'une file d'attente i n'ont accès au processeur que lorsque les files d'indice inférieur (1 à i-1) sont vides ;
   — l'arrivée d'un processus dans le première file, quand elle est vide, provoque le vidage (réquisition) du processus en cours d'exécution s'il est issu des files 2 ou 3.

   Utiliser l'annexe 2 pour suivre l'évolution du système. Critiquer les solution d'ordonnancement choisies.

3. Reprendre le tourniquet à une file avec un quantum de 4 unités et utiliser une prévision (cf cours) en supposant que les phases d'entrée/sortie sont négligeables par rapport au temps de calcul.

Exercice 30. Gestion des demandes et files d'attentes de processus

Dans le chapitre 3 nous avons étudié les algorithmes d'ordonnancement. Dans la famille d'algorithmes avec préemption, un processus fait plusieurs passages dans le processeur, afin de sauvegarder l'état d'un processus le SE utilise un PCB (*Process Control Bloc*).

1. Quels sont les informations nécessaire au PCB ?
2. Chaque processus a besoin de ressources (mémoire, E/S etc.). Proposer une méthode utilisant des files d'attentes afin de gérer les demandes de ressources en vous basant sur le PCB.

Exercice 31. (*) Extrait d'examen (noté sur 7 points)

On considère une mémoire de 2000 unités gérée par page de 200 unités. Les 1000 premières unités sont constituées par la mémoire centrale, les autres par la mémoire virtuelle. On suppose que les allocations de mémoire se font par page et ne sont pas nécessairement contiguës. Une allocation ne peut se faire qu'en mémoire centrale ce qui signifie que si il n'y a plus de page libre dans la mémoire centrale et qu'il reste encore des pages libres dans la mémoire virtuelle alors il y a déclenchement du mécanisme de commutation de pages (swap). La commutation de pages est gérée avec la stratégie LRU.

On dispose également d'un ordonnanceur préemptif sans réquisition à une seule file (FIFO) associée à quantum de temps de 2 unités.

Top Arrivée	Processus	Opérations
0	P1	All 450, Acc 21, Acc 22, Acc 30, Calc 2, Acc 200, Calc 1, Acc 22, Acc 5, All 50, Acc 22, Acc 439
2	P2	All 250, Acc 1, Acc 201, Acc 3, Acc 149, Acc 1, Calc 1, Lib 250
4	P3	All 350, Acc 10, Acc 205, Acc 301, Acc 14, Calc 2, All 500

TABLE 1 – Opérations sur la mémoire (All=allocation (en unité), Lib=libération (en unité), Acc=Accès à une adresse), phase de calculs (Calc n) données en unité de temps

1. Faire un schéma du système (mémoire et ordonnanceur) avant les opérations.
2. Suivre le comportement global du système :
   (a) Donnez l'état de la file d'attente au processeur et spécifier le processus en exécution.
   (b) Donnez l'état de la mémoire à chaque changement (commutation de page par exemple).
   (c) Précisez le nombre total de défauts de pages.

   Pour vous aidez vous pouvez, à partir du tableau 1, refaire un tableau permettant de retrouver les situation vues dans les exercices précédents.

   Indications :

   Les actions réalisées par chaque processus sont données dans le tableau 1. Les adresses mémoires sont exprimées au moyen d'offset à partir de l'adresse de base

déterminée lors de l'allocation. On suppose qu'un accès mémoire consiste à charger un registre du processeur et utilise 1 unité de temps. Les allocations et libérations de mémoire sont gérées par le noyau et utilisent 1 unité de temps. Les phases de calcul ne s'effectuent que sur les registres du processeur, c'est-à-dire après un accès mémoire.

Exercice 32. (*) Septembre 2005 (10pts)

On souhaite concevoir un ordonnanceur pour une machine bi-processeurs. Afin d'évaluer différentes stratégies vous utiliserez les jeux de tests des tableaux 2 et 3 (donnant pour chaque processus son top d'arrivée et sa durée).

Nom	p0	p1	p2	p3	p4	p5
Top d'arrivée	1	3	4	12	15	16
Durée	2	11	5	9	8	3

TABLE 2 – Jeu de test

1. rappelez le rôle de l'ordonnanceur dans les systèmes d'exploitation
2. décrivez le fonctionnent du système avec une stratégie de tourniquet à une file (sans réquisition) et quantum de 2 unités
3. simulez votre stratégie avec le jeu de processus du tableau 2, calculez le taux de retard moyen
4. modifiez votre stratégie en utilisant 2 files l'une avec un quantum de 2 l'autre avec un quantum de 6
5. simulez cette nouvelle stratégie, calculez le taux de retard moyen
6. on souhaite ajouter un processeur d'E/S au système. Dans l'hypothèse de la question 4, faire le schéma de ce nouveau système et expliquez son fonctionnement
7. en considérant maintenant que les processus du tableau 2 effectuent également des phases d'E/S suivant le schéma (phase exécution, phase E/S, phase exécution etc.) et en vous référant au tableau 3, simulez le comportement de ce nouveau système
8. chaque processeur fonctionne à 2,8Ghz et travaille sur 64 bits. En supposant qu'un processeur peut exécuter une instruction par cycle d'horloge calculez la bande passante du bus mémoire (en Mo ou Go) nécessaire dans le cas d'utilisation d'un processeur puis de deux.

Nom	p0	p1	p2	p3	p4	p5
Top d'arrivée	1	3	4	12	15	16
Durée	(1,1)	(5,2,3,1)	(2,1,2)	(6,2,1)	(4,1,3)	(2,1)

TABLE 3 – Jeu de test avec phases d'E/S

Exercice 33. (*) Examen (6pts)

On considère un tourniquet à 2 files d'attente associées à des *quantums* de 2 (pour la file 1) et de 4 (pour la file 2) permettant l'accès à un processeur double cœur. Si un processus

se termine sans épuisement du quantum, on attend la fin du processus s'exécutant dans l'autre cœur pour lancer l'ordonnancement. La gestion des files se fait par priorité (priorité haute en tête de file). Il n'y a pas de réquisition. La priorité d'un nouveau processus est de 1, elle augmente d'une valeur à chaque passage dans le processeur (elle est plafonnée à la valeur 6). Les processus n'ont accès à la file 2 que si leur niveau de priorité est strictement supérieur à 3.

Nom	p0	p1	p2	p3	p4	p5	p6
Top d'arrivée	1	2	4	6	8	11	12
Durée	3	4	5	7	6	2	1

1. Simuler le comportement du système avec les données fournies dans le tableau ci-dessus.
2. Calculer le taux de retard pour chaque processus et le taux de retard moyen, conclure.

Exercice 34. (*) Contrôle continu (7pts)

On considère un tourniquet à une file d'attente associée à un *quantum* de 2 unités de temps (ut). L'ordonnanceur utilise une stratégie Earliest Deadline First (échéance la plus proche en premier) pour trier les processus et déterminer celui qui doit être admis dans le processeur. Lors son lancement chaque processus déclare son échéance.

Nom	p0	p1	p2	p3	p4	p5	p6
Top d'arrivée	1	2	4	6	8	11	12
Durée (ut)	3	4	3	6	2	1	1
Échéance (au top)	6	20	12	14	15	18	20

1. Quel type de système d'exploitation et quelles applications sont concernés par cette stratégie d'ordonnancement ?
2. Simuler le comportement du système avec les données fournies dans le tableau ci-dessus.
3. Calculer le taux de retard pour chaque processus et le taux de retard moyen. Quels sont les processus qui n'auraient pas satisfaits l'échéance demandée ?
4. Comment modifier le système pour améliorer son fonctionnement sans modifier la stratégie ?

Exercice 35. (*) Ordonnancement et E/S (7 pts)

On étudie une stratégie d'ordonnancement incluant une prise en compte des demandes d'entrée sortie (E/S). On se limitera à traiter les entrées sorties relatives aux périphériques de stockage (disque dur par exemple).

Pour l'accès au processeur (qui dispose d'un seul cœur), on considère un tourniquet à deux files d'attentes associées à des *quantum* respectifs de 2 et 4 unités de temps (ut), gérées en FIFO. Ainsi, un processus qui a déjà été exécuté au moins une fois est mis en file numéro 2. Les processus de la file numéro 2 n'ont accès au processeur que si la file d'attente numéro 1 est vide.

Pour l'accès aux périphériques de stockage, on dispose d'une file d'attente FIFO associée à un traitement jusqu'à terminaison par le processeur d'E/S. Quand un processus

demande une E/S il est placé dans cette file d'attente. Le processeur et le processeur d'E/S peuvent travailler en parallèle. Quand le processeur d'E/S a terminé une exécution, il en informe le processeur via une interruption qui provoque une réquisition en plaçant le processus concerné en début (tête) de la file d'attente numéro 1.

Nom	p1	p2	p3	p4
Top d'arrivée	1	2	4	6
Comportement (en ut)	(3,2,3,2)	(8,1)	(1,1)	(2,1,2,1)

1. Faire un schéma du système.
2. Simuler le comportement de la stratégie d'ordonnancement avec les données fournies dans le tableau ci-dessus. Chaque processus alterne des phases d'utilisation du processeur avec des phases d'E/S (écriture disque) et commence bien entendu par une phase utilisant le processeur. Par exemple, le processus p1 calcule pendant 3ut puis effectue des écritures sur le disque pendant 2ut et recommence un calcul durant 3ut puis des écritures disque durant 2ut avant de se terminer.
3. Calculer le taux de retard pour chaque processus et le taux de retard moyen. Que peut-on remarquer ?

Exercice 36. (*) Les compteurs de Robert (1977) (8pts)

On étudie un problème de type lecteurs et rédacteurs. On considère une ressource `r` (instance d'une classe `Ressource`) et différents threads qui peuvent lire (opération *lect*) ou écrire (opération *red*) dans `r`. Les propriétés suivantes doivent être garanties : 1) il ne peut pas y avoir deux écritures concurrentes sur la ressource en même temps ; 2) il ne doit pas y avoir de lecture de la ressource pendant une écriture ; 3) les lectures concurrentes sont autorisées.

1. Soit 6 threads, $T_i, i = 1 \dots 6$ effectuant les opérations décrites dans le tableau 1. On suppose que les différents threads sont dans une file en attente d'exécution dans l'ordre du tableau 1. Le système dispose de plusieurs processeurs (8 par exemple). Dans les questions suivantes vous devez donner l'ordre des opérations et vérifier que la méthode d'ordonnacement respecte les trois propriétés énoncées ci dessus.

position	1	2	3	4	5	6
thread	T_1	T_2	T_3	T_4	T_5	T_6
opération	$lect_1$	$lect_2$	red_3	$lect_4$	red_5	$lect_6$

TABLE 4 – Thread et opérations lecteur/rédacteur

   (a) Pour un mécanisme d'ordonnancement qui donne priorité aux lecteurs, donner l'enchainement des opérations et spécifier les opérations qui peuvent être exécutées en parallèle en utilisant une notation parenthésée. Par exemple $(op_1, op_3, op_3), op_4$ signifie que op_1, op_2 et op_3 sont exécutées en parallèle et que op_4 est exécutée lorsque les trois autres opérations sont terminées.

   (b) Même question avec priorité aux rédacteurs.

   (c) Même question avec priorités égales, c'est-à-dire avec une gestion FIFO de la file.

2. On souhaite programmer les interactions entres les lecteurs, les rédacteurs et la ressource, sans avoir recours à un ordonnanceur spécifique. On utilise différentes variables pour enregistrer le nombre de lecteurs ou de rédacteurs en cours ou en attente. La classe ressource contient une variable compteur (de type `int`) et les deux méthodes `lect()` et `red()`. La méthode `red()` modifie le compteur (par exemple en l'incrémentant) et la méthode `lect()` retourne sa valeur.
   (a) Pour chacune des méthodes `lect()` et `red()`, justifiez si elle doit être déclarée `synchronized` ou non.
   (b) Utiliser un ou plusieurs sémaphores pour assurer les trois propriétés et donner le code Java de la classe `Ressource`.

Exercice 37. (*) Ordonnancement simple (7pts)

On considère un tourniquet à deux files d'attentes associées à des *quantums* respectifs de 1 et 2 unités de temps (ut). Le processeur possède un seul cœur. Les files sont triées en FIFO, il n'y a pas de réquisition. Les processus de la file 2 n'ont accès au processeur que si la file 1 est vide.

Nom	p0	p1	p2	p3	p4	p5
Top d'arrivée	1	2	4	6	8	10
Durée (ut)	3	2	3	5	2	1

1. Un processus qui a déjà été exécuté au moins une fois est mis en file numéro 2. Simuler le comportement du système avec les données fournies dans le tableau ci-dessus. Calculer le taux de retard pour chaque processus et le taux de retard moyen.
2. On modifie la stratégie d'accès au processeur. Les processus font deux passages en file 1 avant d'accéder à la file 2. Si il y a ambiguïté dans la gestion des files c'est-à-dire lors de l'arrivée d'un processus et d'une remise en file, dans le cas d'une file vide, la priorité est donnée au nouveau processus. Refaire la simulation, calculer le taux de retard de chaque processus et le taux de retard moyen.
3. Comparer les deux simulations.

4 Entrées sorties et mémoire cache

Exercice 38. Entrées sorties et DMA

Comme vu au chapitre 1 du cours, une unité centrale est généralement composée d'un processeur et de plusieurs processeurs d'entrées sorties, connectés à travers un bus commun.

Pour démarrer une opération d'entrée sortie, le processeur charge les registres appropriés à l'intérieur du processeur d'entrées sorties. Celui-ci, à son tour, examine le contenu de ces registres pour déterminer l'action à mener. Par exemple, s'il trouve une requête de lecture, il démarre le transfert des données du périphérique vers une mémoire tampon locale. Dès que le transfert des données est terminé, le processeur d'entrées sorties informe le processeur principal qu'il a terminé son opération. Il effectue cette communication en provoquant une **interruption**.

Cette situation est en général le résultat d'une exécution d'un processus utilisateur demandant des entrées sorties. Dès que l'entrée sortie est démarrée, deux méthodes sont possibles :

— dans le cas le plus simple, l'entrée sortie est démarrée ; ensuite, lorsqu'elle est terminée, le contrôle est rendu au processus utilisateur. Ce cas est connu sous le nom **d'entrée sortie synchrone**.
— l'autre solution, appelée **entrée sortie asynchrone**, rend le contrôle au programme utilisateur sans attendre l'achèvement de l'entrée/sortie. Celle-ci pourra continuer à s'exécuter tandis que d'autres opérations système se produisent.

Question 1 : faire un schéma pour chaque méthode d'entrée sortie (synchrone et asynchrone). Le schéma devra faire apparaître 4 couches : le processus demandeur, le gestionnaire d'interruption, le pilote de périphérique, le matériel. On précisera quelles couches appartiennent au noyau et à l'utilisateur.

Question 2 : l'attente de l'achèvement de l'entrée/sortie peut s'exécuter selon différentes stratégies. Certains machines ont une instruction spéciale `wait` qui rend le processeur inactif jusqu'à la prochaine interruption. Les machines qui n'ont pas une telle instruction peuvent avoir une boucle d'attente : `Loop: jmp Loop`. Cette boucle très courte continue simplement jusqu'à ce qu'une interruption se produise, en transférant le contrôle à une autre partie du système d'exploitation.

Une autre stratégie vue en cours permet au processeur d'E/S d'être informé qu'un périphérique a terminé une opération d'E/S : décrire cette méthode et ses interactions avec le noyau du système d'exploitation. Quel est l'inconvénient de cette stratégie (par exemple pour la gestion des cartes réseau) ?

Exercice 39. Gestion de la mémoire cache

La mémoire cache est un élément important dans l'architecture d'une machine.

1. Rappeler son rôle.
2. Quelle particularité doit posséder un mémoire cache pour être efficace ?
3. Proposer un schéma de fonctionnement en identifiant les grandes unités fonctionnelles (utiliser les connaissances acquises au premier semestre en cours d'architecture).
4. Quel problème peut-il se poser lorsqu'un processeur d'E/S a un accès direct à la mémoire et que le processeur central dispose d'un cache ? Comment résoudre ce type de problème ?

5 Gestion de la mémoire et organisation des données

Exercice 40. Gestion de la mémoire principale (compaction)

Soit une situation d'occupation mémoire décrite à l'annexe 3.

1. Quelle zone mémoire sera sélectionnée lors d'une demande d'allocation de n unités (cf cas suivants a-c) avec une stratégie *first-fit* ? On supposera que les demandes sont indépendantes les unes des autres.
   a) 6 unités
   b) 4 unités
   c) 2 unités
2. Même question avec une stratégie *best-fit*.
3. Donnez l'état de la mémoire après : a) une compaction totale, b) une compaction partielle, c) une compaction optimisée. Les compactions partielle et optimisée seront faites dans l'hypothèse d'une demande d'allocation de 10 unités.

4. **Travail personnel :** déterminer l'algorithme pour une compaction optimisée.

Exercice 41. Pagination

Soit un système d'exploitation gérant la mémoire au moyen du mécanisme de pagination (gestion par pages de taille fixe). La machine dispose de 600 unités (ou blocs) de mémoire centrale (RAM). Le système d'exploitation utilise des pages de 200 unités, il est configuré de façon à pouvoir gérer 14 pages de mémoire virtuelle (simulée au moyen du disque dur).

Le système est soumis aux demandes d'accès décrites dans le tableau 4.

1. Suivez le fonctionnement du système en supposant que les pages de la mémoire centrale sont vidées dans l'ordre où elles ont été chargées (stratégie FIFO). Complétez le tableau 4 sachant que dans un système de gestion de mémoire par page, une adresse mémoire est convertie en une adresse de page (*base*) et un déplacement (*offset*) à l'intérieur de la page. Simulez le fonctionnement en utilisant la première colonne de la fiche. Notez a) le nombre de défauts de pages (chargement d'une page de la mémoire virtuelle vers la mémoire centrale, c'est-à-dire nom présence en mémoire centrale de la page demandée) jusqu'à la douzième demande (comprise) et b) le nombre total de défauts de pages.
2. Suivez le fonctionnement du système avec une stratégie qui décharge prioritairement la page utilisée il y a le plus longtemps. Utilisez la deuxième colonne de la fiche. Notez le nombre de défauts de pages (mêmes conditions que la question 1). Comparez les résultats obtenus. Que s'est-il passé à partir de la douzième demande ?
3. La dernière simulation utilise une stratégie de déchargement qui sélectionne en priorité les pages ayant le moins servi. En cas d'égalité, on utilise la stratégie de la question 2. Utiliser la colonne 3 de la fiche pour effectuer la simulation . Afin de maintenir un compteur d'utilisation des pages, le système maintient une liste (tableau horizontal colonne 3). Comptabilisez à nouveau les défauts de pages, comparez-les avec les résultats précédents. Qu'en déduire ? Cette stratégie est rarement utilisée, pourquoi ?
4. On fait l'hypothèse suivante : les demandes sont mises en attente. De plus on suppose que l'ordre de traitement n'est pas fondamental. Quelle serait la stratégie optimale ?
5. Refaire la question 2 en considérant un taille de page de 300 unités. Expliquez les résultats. Utilisez la fiche suivante pour simuler le système.
6. Que pensez vous d'une taille de page de 10 unités (justifier) ?
7. **Travail personnel :** écrire les formules de passage permettant d'associer à une adresse mémoire son couple (numéro de page, offset) et inversement du couple (numéro de page, offset) à adresse mémoire réelle. Dans ce dernier cas, vous aurez besoin de connaître l'adresse réelle de la page chargée. En effet, suite à plusieurs chargements déchargements, une page i peut se retrouver en position j. Par exemple, la page numéro 3 peut se retrouver en page 0. Écrire l'algorithme qui permet au système d'exploitation de gérer la mémoire virtuelle.

Exercice 42. Examen - 6 pts

On considère une mémoire de 200 pages dont 80 sont des pages physiques, les autres étant des pages de la mémoire virtuelle. Chaque page contient 16 blocs de 512 octets.

Demande	Conversions adresse =(page,offset)			Mémoire centrale	
	Adresse	Page	Offset	Page	Adresse
1	657	3	57	0	57
2	523	2	123	1	323
3	170	0	170	2	570
4	725				
5	1133				
6	145				
7	190				
8	658				
9	573				
10	56				
11	598				
12	888				
13	1134				
14	170				
15	472				

TABLE 5 – Demandes d'accès mémoire

1. Faire un schéma de la situation, calculer la taille de la mémoire physique et de la mémoire virtuelle en *ko*.
2. Calculer les numéros de page et l'offset dans la page pour les 5 demandes d'accès suivantes (exprimée en octets) : 131 200, 820 000, 138 300, 131 205, 780 000.
3. Simuler le fonctionnement du mécanisme de gestion de la mémoire virtuelle et donner le nombre de défauts de pages, en supposant que :
   — dans la situation initiale chaque page est à sa place, par exemple, la page 0 occupe la page physique 0 de la mémoire, etc. ;
   — la date du dernier accès (en top horloge) à une page physique correspond à son numéro ;
   — on utilise une stratégie de déchargement de la page la moins récemment utilisée.

Exercice 43. Les B-arbres

On considère une famille particulière de B-arbre : les B-arbres binaires. Dans un cadre général, un arbre binaire est une structure dans laquelle chaque nœud comporte une valeur et deux pointeurs ou références vers deux fils : le fils gauche et le fils droit. Un nœud qui n'a pas de fils est une feuille (les références qui désignent ses fils sont vides). L'objectif de cet exercice est d'identifier les propriétés des B-arbres, sur l'exemple des arbres binaires, qui favorisent la rapidité des recherches dans le contenu d'un fichier.

Par convention, dans le cas d'utilisation d'arbres binaires, pour un nœud donné, on range tous les éléments plus petits que la valeur du nœud à gauche et tous éléments les plus grands à droite. Ce principe est appliqué récursivement à tous les nœuds de l'arbre.

Pour ranger une nouvelle valeur dans l'arbre, on applique le principe suivant (exprimé sous la forme d'un algorithme récursif) :

```
Pour ranger un élément e dans un arbre B
  Si B est vide
```

```
   On crée une nouvelle feuille f dont le corps est e
   et B devient cette feuille
Sinon
   Si Valeur(e) < Valeur(Racine(B))
      Ranger e dans le sous-arbre gauche de B
   Sinon
      Ranger e dans le sous-arbre droit de B
fin
```

1. Construire l'arbre binaire permettant de ranger les éléments suivants (insérés dans l'ordre de la liste) : chêne, baobab, mélèze, arbousier, aulne, châtaignier, cèdre, eucalyptus, cyprès, figuier, pin, platane, peuplier.
2. Construire l'arbre binaire, selon la même technique que celle adoptée précédemment, permettant de ranger les éléments suivants (insérés dans l'ordre de la liste) : arbousier, platane, aulne, pin, baobab, peuplier, cèdre, mélèze, châtaignier, figuier, chêne, eucalyptus, cyprès.
3. Écrivez en pseudo-code l'algorithme qui permet de rechercher un élément dans l'arbre à partir de la valeur de sa clé. Prendre exemple sur l'algorithme donné plus haut.
4. Comment obtenir une liste triée des valeurs contenues dans l'arbre ?
5. Déterminer un critère permettant de mesurer la qualité d'une recherche par l'algorithme précédent en utilisant le rapport entre le nombre d'éléments stockées réellement dans le B-arbre et sa capacité de stockage maximum. Comparer la qualité des recherches dans le cas le plus défavorable pour les arbres que vous avez construits dans les questions 1 et 2.
6. Combien d'éléments peut-on stocker dans B-arbre binaire complet ?
7. Proposer une méthode pour construire un arbre équilibré avec l'hypothèse que toutes les données sont connues au départ. Comment faire si cette hypothèse est abandonnée ?
8. On souhaite, en plus des noms d'arbres, enregistrer le nom latin, une description textuelle ainsi qu'une photographie. Sachant que les recherches de font essentiellement sur le nom commun ou le nom latin. Proposer, dans ce cas, une structure de données permettant le stockage et l'indexation en vue d'optimiser des recherches.
9. On modifier les nœuds du B-arbre afin d'y stocker plusieurs éléments (5 éléments par nœud). Quelles modifications apporter à l'algorithme ?

Exercice 44. Structure de hachage (*hash-coding*)

On dispose d'une mémoire (tableau) de 14 unités. Chaque unité peut contenir un chaîne de caractères de longueur 80 (par conséquent un nom d ?arbre). On veut modifier l'organisation de cette mémoire pour en faire une mémoire à accès aléatoire en la découpant en zones de tailles fixes.

On suppose que les débordements, lorsqu'une zone est pleine, s'effectuent toujours dans la première zone vide suivant la zone saturée.

Dans les questions suivantes, vous devez ranger successivement, dans la mémoire, chacun des noms d'arbre de l'exercice précédent pris dans l'ordre alphabétique. Pour calculer la place de chaque élément, remplir le tableau 5.

1. On considère tout d'abord des zones d'une unité numérotées de 0 à $NbZ = 13$ ainsi que la fonction de hachage suivante :

$$F_1(valeur) = Ord(v[1])Mod(NbZ + 1)$$

   $Ord(c)$ est la fonction qui retourne la valeur ASCII du caractère passé en paramètre. Mod est la fonction modulo. La fonction F_1 fait correspondre à chaque valeur de la clé le reste de la division entière de la valeur ASCII de la première lettre de la clé avec le nombre de zones.
2. Les zones ont deux unités (de 0 à $NbZ = 6$). Utilisez la même fonction de hachage que celle de la question précédente.
3. Avec des zones d'une unité, on considère la fonction de hachage suivante :

$$F_1(valeur) = (Ord(v[1]) + 26 \times (longueur(v) - 1))Mod(NbZ + 1)$$

4. On utilise maintenant F_2 avec des zones de 2 unités.
5. Dans chaque cas, calculer le nombre de débordement. Qu'en pensez vous ? Pourquoi le fonction F_2 fait-elle intervenir 26 ?

	$Ord(v[1])$	mod 14	mod 7	$Ord(v[1]) + 26 * (lg - 1)$	mod 14	mod 7
arbousier	97					
aulne						
baobab						
cèdre						
châtaignier						
chêne						
cyprès						
eucalyptus						
figuier						
mélèze						
peuplier						
pin						
platane						

TABLE 6 – Enregistrements à traiter

Exercice 45. (*) Examen (6pts)

On considère une gestion de mémoire par blocs de 1024 octets et par pages de 16 blocs. Le système dispose de 10 pages en mémoire physique et 20 pages en mémoire virtuelle.

65→A 66→B 67→C 68→D 69→E 70→F 71→G 72→H 73→I 74→J 75→K
76→L 77→M 78→N 79→O 80→P 81→Q 82→R 83→S 84→T 85→U 86→V
87→W 88→X 89→Y 90→Z 91→[92→“ 93→] 94→ˆ 95→˙ 96→‘ 97→a 98→b
99→c 100→d 101→e 102→f 103→g 104→h 105→i 106→j 107→k 108→l 109→m
110→n 111→o 112→p 113→q 114→r 115→s 116→t 117→u 118→v 119→w
120→x 121→y 122→z

TABLE 7 – Table ASCII des caractères

1. Évaluer la taille de la mémoire principale et de la mémoire virtuelle, faire un schéma.
2. Sachant qu'un processus demande une quantité de mémoire exprimée en octets et qu'un bloc n'appartient qu'à un seul processus, déterminer l'état de la mémoire (faire un schéma) après les demandes suivantes effectuées dans l'ordre de la gauche vers la droite du tableau :

Nom du processus	p0	p1	p2	p1	p2	p3
Demandes (octets)	12280	5000	24000	5234	60200	62000

3. Évaluer la fragmentation interne (aux blocs) et externe suite à ces demandes.
4. Décrire le problème de la fragmentation externe et proposer une stratégie pour réduire la fragmentation externe lorsque le système est inoccupé.

Exercice 46. (*) Examen (6pts)

On considère un système avec 4Go de mémoire RAM et 4Go de mémoire virtuelle. La taille des pages, fixée par le système d'exploitation, est de 2048Ko.

1. Combien de pages sont nécessaires pour couvrir les 8Go de mémoire adressable ?
2. Combien de bits sont utilisés pour coder les numéros de page ?
3. Un processus demande l'accès a l'adresse 30 012 032, cette adresse est elle en mémoire principale ou virtuelle ? Donnez le couple numéro de page, offset pour cette adresse.
4. Un processus demande un accès à la page 2060 offset 1245, donnez l'adresse.
5. On suppose que le système d'exploitation en se chargeant a occupé les 100 premières pages, ensuite deux processus de calcul (P_1 et P_2) ont été lancés, le premier à premier a occupé 1 600 pages et le second tout le reste de la mémoire physique. P_1 demande une allocation de 200 pages puis lit les adresses allant de 310 000 000 à 350 000 000 pour réécrire de nouvelles valeurs calculées dans les nouvelles pages allouées (en commençant au début de la première nouvelle page). On suppose que la stratégie de gestion de pages est FIFO. Simulez le comportement du système pour quelques opérations, généralisez. Que pouvez vous en conclure ?
6. Quelle est la taille mémoire nécessaire pour stocker les données utiles à la stratégie FIFO ? Combien de pages devront être utilisées ? Quelle sera leur particularité (pensez aux pages qui stockent le noyau du système d'exploitation) ?

Exercice 47. (*) Gestion mémoire - 5 pts

On considère une mémoire de 25 blocs (numérotés de 0 à 24), chaque bloc a une taille de 1 024 octets. On considère les demandes suivantes dans l'ordre chronologique, P_1 demande 10 400 octets, P_2 demande 3 800 octets, P_3 demande 7 410 octets.

1. Dans le cas d'une allocation contigüe, donner l'état de la mémoire après les trois demandes d'allocation. Évaluer la fragmentation interne globale en nombre d'octets et la fragmentation externe en nombre de blocs.
2. P_2 se termine et libère ses blocs. P_4 demande une allocation de 5 020 octets. Choisir une stratégie et donner l'état de la mémoire après l'allocation de P_4, évaluer la fragmentation externe.
3. Reprendre la question 1 puis la question 2 dans le cas d'une gestion de la mémoire autorisant des allocations non-contigües.

Exercice 48. (*) Ordonnancement, mémoire virtuelle - 7 pts

On considère un tourniquet à trois files d'attentes associées à différentes classes de processus : temps réel pour la file 1, interactifs pour la file 2 et batch ou longs pour la file 3. Leurs quantums respectifs sont 2, 2, 4 unités de temps. Lors de leur démarrage les processus sont placés dans la file qui correspond à leur type. Le processeur possède deux cœurs. Les files sont gérées en FIFO, il n'y a pas de réquisition. Les processus des files 2 et 3 n'ont accès aux cœurs du processeur que si la file 1 est vide. Les files sont prises en compte dans l'ordre de leur numéro.

1. Faire un schéma du système.
2. Suivre l'exécution des processus décrits aux tableau 1.
3. Calculer le taux de retard pour chaque processus et le taux de retard moyen, discutez du fonctionnement de l'ordonnanceur.
4. Peut-on améliorer le fonctionnement du système avec des *quantums* différents ?

Nom et type	p0 (L)	p1 (I)	p2 (RT)	p3 (I)	p4 (I)	p5 (RT)
Top d'arrivée	1	2	3	5	9	10
Durée (ut)	8	4	2	6	2	2

TABLE 8 – Processus, durée, arrivée et type (I= interactif, RT= temps réel, L=long)

Exercice 49. (*) Gestion de la mémoire 2019 - 6 pts

On considère une mémoire de 16 pages (numérotées de 0 à 15) comportant 10 pages de mémoire virtuelle simulée par un disque dur. Chaque page a une taille de 512 octets.

1. Faire un schéma de l'organisation de la mémoire, préciser la limite de la mémoire physique, numéroter les pages et donner les adresses des limites de page.
2. On suppose que le mécanisme de *swap* utilise une stratégie LRU (*Least Recently Used*), c'est-à-dire moins récemment utilisé. Donner les différents états de la mémoire physique en fonction des 14 demandes d'accès mémoire du tableau 1. On suppose que dans l'état initial la mémoire physique contient, dans l'ordre, les pages 5, 8, 2, 4, 1, 3. Ceci correspond également à leur ordre d'utilisation.
3. Pour chaque accès, donner le couple numéro de page, offset et le couple correspondant à la page placée en mémoire physique.
4. Quel est le nombre de défauts de page ?
5. Quels composants gèrent les défauts de page et les translations d'adresses dus au swap ?

demande	1	2	3	4	5	6	7
accès adresse	330	2 500	3 010	4 500	6 210	1 600	430
demande	8	9	10	11	12	13	14
accès adresse	6 100	6 400	7 200	3 002	2 060	180	2 400

TABLE 9 – Demandes d'accès à la mémoire

6 Fichiers

Exercice 50. Copie de fichier et thread

Écrire un programme qui permet d'afficher l'avancement d'une copie de fichier. L'affichage de l'avancement de la copie ne doit pas être inclus dans la classe qui effectue la copie : il s'agit d'une fonctionnalité supplémentaire (non intrinsèque) qui est ajoutée à la classe de copie de fichier via une autre classe. Cette autre classe se comportera comme un système de surveillance (*monitoring*) ou chien de garde (*watch dog*).

Exercice 51. (*) Extrait d'examen (8pts)

Réaliser un programme Java pour copier les fichiers contenus dans un répertoire source dans un répertoire destination. Vous devez utiliser au minimum 3 threads (c'est-à-dire au moins 2 threads qui copient des fichiers simultanément). Pour rappel la méthode `list()` de la classe `File` retourne un tableau de chaînes de caractères contenant la liste des éléments d'un répertoire donné (cf. extrait de code suivant).

```
File dir = new File("NomDuRepertoire");
String[] fichiers = dir.list();
```

Exercice 52. (*) Système de fichier Linux (6pts)

1. Donner la définition de la notion de i-nœud (*i-node*) dans les systèmes de fichiers.
2. On suppose que les blocs de données ont une taille de 4096 octets, quelle est la taille maximale d'un fichier dans le système ext2 de Linux.
3. Un fichier de 50Mo doit être stocké, combien de niveaux et de i-nœuds seront utilisés (faire un schéma).
4. On souhaiter garder les différentes versions d'un fichier, non pas au niveau du fichier lui même, mais au niveau des blocs de données modifiés. Proposer une solution en ajoutant des éléments complémentaires aux i-nœuds.

Exercice 53. Nombre de lignes d'un fichier et arguments

Écrire un programme Java qui pend en argument, sur la ligne de commande, un nom de fichier existant dans le répertoire courant et retourne le nombre de lignes du fichier ainsi que la longueur moyenne d'une ligne.

7 Principes des réseaux et communications TCP/IP

Exercice 54. Réseaux et débit

On considère un camion transportant 25 000 CD sur une distance de 1000km en roulant à 80Km/h en moyenne. Calculer le débit en Mb/s, comparer avec les débits de réseaux actuels.

Exercice 55. Transfert de fichiers

Quels sont les éléments nécessaire pour transférer un fichier entre deux ordinateurs connectés sur un réseau ? Afin que le transfert soit valide, il est nécessaire de s'assurer que le fichier est correctement arrivé sur la machine distante. Proposer deux stratégies

pour effectuer cette vérification, donner les deux algorithmes, discuter des avantages et inconvénients de vos propositions.

Exercice 56. Réseau Ethernet

1. Rappeler la méthode d'accès au médium physique utilisé dans les réseaux Ethernet.
2. Quels sont les défauts de cette méthode ? Conclure en montrant que le temps de transmission d'un message n'est pas a priori connu et expliquer pourquoi ce type de réseau est critiqué pour les applications de calcul scientifique.
3. Reprendre la même démarche et étudier le cas d'un anneau à jeton.

Exercice 57. Comparaison de topologies de réseau

On veut relier sur un réseau 65 machines. Calculer le nombre de liens (lignes de communication) distincts et le nombre de sauts en moyenne pour les topologies suivantes :

1. anneau
2. étoile
3. arbre binaire
4. réseau complet

Exercice 58. Adresse IP et mobilité

Pour identifier une machine sur le réseau Internet, on utilise une adresse unique (adresse IP). Par analogie avec la téléphonie, l'adresse IP correspond au numéro de téléphone. Que se passe t'il pour les téléphones mobiles ? Comment retrouver un correspondant ? Quel principe peut être est utilisé ? Peut-on l'appliquer au cas des ordinateurs portables ?

Exercice 59.

Soit une adresse données sous la forme textuelle FQDN (*fully qualified domain name*). Écrire un programme Java pour :

1. donner l'adresse IP de la machine ;
2. tester si la machine est joignable ;
3. tester si un service identifié par son port est actif sur le machine.

Exercice 60. Serveur Web

HTTP (*HyperText Transfer Protocol*), est un protocole utilisé pour le transfert de données (fichiers) depuis les serveurs Web vers des clients (navigateurs tels que Firefox ou Chromium). Il s'agit d'un langage au travers duquel dialoguent les navigateurs Web et les machines serveurs (ou plus précisément des processus serveur respectant le protocole HTTP) sur lesquels sont hébergés les sites Web (ensemble de fichiers contenant des descriptions de pages avec le langage de balises HTML). On utilise aussi le terme d'application Web pour désigner des sites Web qui réagissent en fonction des demandes du client pour lui construire dynamiquement un contenu spécifique (PHP permet par exemple de réaliser des applications Web).

HTTP est un protocole évolué de la couche 7 du modèle OSI [3], c'est-à-dire de la couche application, il repose sur TCP/IP. Il est décrit dans la RFC 2616 [4].

3. L'article de Wikipedia sur le sujet est bien détaillé
4. https://tools.ietf.org/html/rfc2616

Lorsqu'un client demande une page Web, il s'adresse au serveur HTTP (un processus), en envoyant une commande du protocole (ligne de texte contenant la commande et les arguments), comme par exemple : `GET /index.html HTTP/1.1`. Dans cet exemple, `GET` permet de déterminer si c'est une demande, le paramètre est le nom de la page demandée, la dernière partie spécifie la version du protocole utilisé ici c'est HTTP dans la version 1.1.

En réponse, le serveur Web va, avant d'envoyer la page demandée, envoyer des informations générales qui seront interprétées par le client :

```
HTTP/1.1 200 OK
Date: Mon,  24 May 2010 14:00:00 GMT
Server: Apache/1.3
Last-Modified: Wed, 26 May 2010 11:27:00 GMT
Content-Length: 1234
Content-Type: text/html
```

Ensuite va suivre le contenu du fichier sous la forme d'un flux.

Le navigateur reçoit le fichier et l'interprète. Si il s'agit d'un fichier HTML il l'affiche en effectuant le rendu à partir de l'analyse des balises, sinon il recherche un module ou une application externe qui pourra décoder le fichier, en dernière recours, il propose à l'utilisateur de choisir une application ou un enregistrement sur le disque.

Dans la suite de l'exercice nous allons spécifier et construire un mini serveur Web qui sera utilisable à partir d'un navigateur. L'exercice est long et peut être arrêté à la question 5.

1. Identifier les différents objets Java à utiliser pour réaliser un programme serveur web. Quel type de socket utiliser ?
2. Écrire un serveur Web initial renvoie, quelque soit la demande du client un fichier (toujours le même). Vous pourrez le tester avec un navigateur Web ou au moyen de la commande `telnet`.
3. Discuter de l'utilisation des Threads pour répondre à plusieurs demandes à la fois.
4. Modifier votre programme pour qu'il interprète les URL envoyées par le client comme un chemin local vers le système de fichiers et affiche la liste des fichiers d'un répertoire passé en paramètre et qui sera considéré comme la racine du site Web. Les répertoires seront affichés comme des liens hypertexte afin de permettre la navigation dans le système de fichiers (à partir de la racine du site seulement). Il faudra décoder le chemin local, renvoyer l'en-tête et former le fichier HTML qui liste les éléments du répertoire. Une ligne blanche sépare l'en-tête du contenu.
5. Modifier votre code pour que le serveur puisses envoyer aussi les fichiers correspondant aux URL.
6. Si les fichiers sont volumineux et donc longs à télécharger, si le serveur n'utilise pas de threads, les clients auront des problèmes de connexion (*time out*). Mais si il y a beaucoup de client les thread vont consommer beaucoup de ressources. Modifier le serveur pour lui donner la capacité à accepter plusieurs clients mais sans trop multiplier le nombre de threads.
7. À partir du serveur Web initial (question 2), développer un redirecteur, c'est-à-dire un serveur qui renvoie une demande vers au autre serveur.
8. À partir du serveur Web plus complet (question 5), vous pouvez développer un serveur dont le contenu des pages Web va être généré par de programmes Java. Cette

technique, similaire à PHP, est connue sous le nom de *servlet*[5] dans le monde Java. Ainsi, votre programme serveur doit donc en fonction de certaines URL spécifiques et pré-enregistrées, lancer une méthode d'un petit objet instancié à partir d'une classe sous-classe de la classe `Servlet` (que vous aurez aussi spécifiée pour rassemble les méthodes communes aux *servlets*). Cette méthode produit elle-même le code HTML renvoyé directement au client. Cependant, cette méthode présente l'inconvénient de pré-charger tout ce dont on peut avoir besoin. L'idéal serait de pouvoir charger à la demande des classes contenant les traitements. Ce dernier point est du niveau du cours de Master 1.

Exercice 61. (*) Serveur de temps (8pts)

On souhaite réaliser un serveur de temps pour des applications Java. Le serveur est synchronisé sur un horloge de référence et diffuse aux clients qui le demandent la date et l'heure. Les clients du serveur sont sur un réseau local.

1. Quel protocole TCP/IP utiliser (UDP ou TCP) ?
2. Le serveur doit-il utiliser des threads pour traiter les demandes des clients ?
3. Comment s'opère le dialogue entre un client et le serveur, définir la structure des messages échangés.
4. Écrire le programme serveur en supposant que le temps d'acheminement de l'information entre le client et le serveur n'est pas pris en compte.
5. Écrire un programme exemple d'utilisation du service offert par votre serveur.
6. On souhaite corriger l'heure en éliminant les erreurs dues à l'acheminement de l'information entre le serveur et un client. Pour cela le serveur mesure la durée d'acheminement en effectuant 10 échanges avec le client (aller-retour). Il calcule la moyenne et divise par 2 la durée puis communique cette information au client qui corrigera l'heure reçue.
   (a) Que devez vous modifier dans votre programme serveur pour prendre en compte cette nouvelle fonctionnalité ?
   (b) Redéfinir le dialogue entre le client et le serveur.
   (c) Définir la structure des messages échangés.
   (d) Écrire la portion de code nécessaire pour évaluer la durée d'acheminement, calculer la correction à apporter sur la date et la communiquer au client.

Indications : on suppose qu'il existe une classe `Date` munie d'une méthode `getDate()` qui retoune la date et l'heure dans le format `AAAA-MM-JJ hh:mm:ss:milli`. L'heure est donnée en millisecondes, par exemple une date complète retournée sera `2012-05-21 09:01:23:932`. Il existe aussi dans la classe `Date`, la méthode `getHeure()` qui retourne l'heure sous la forme d'une chaîne incluant les millisecondes et la méthode `getHeurems()` qui retourne l'heure en secondes sous la forme d'un float incluant les millisecondes.

Exercice 62. (*) Réseau de capteurs (8pts)

On souhaite réaliser une application qui collecte des données issues de capteurs. Les capteurs sont reliés à un réseau filaire ou à un réseau WiFi et possèdent tous une adresse IP. Lorsqu'un capteur reçoit une chaîne de caractère `GET VALUES`, il retourne un triplet sous la forme d'une chaîne de caractères, contenant la température, la pression atmosphérique

5. https://fr.wikipedia.org/wiki/Servlet

et le taux d'humidité. La température est mesurée en degrés Celsius au dixième de degrés, la pression est mesurée en hectopascal (hPa). Une réponse peut être par exemple `23,5 1025 67`. Tous les capteurs répondent sur le port 451.

1. Un fichier texte (`sensor.txt`) contient la liste des adresses IP des capteurs. Écrire la portion de code pour lire le fichier et mettre sont contenu dans un tableau de `String`.
2. Créer une classe `SensorHT` en utilisant une hash-table qui va recevoir pour chaque capteur (selon son adresse IP) la liste des mesures avec leur date (incluant l'heure à la seconde près). Une mesure est un objet contenant un champ date et trois champs (température, pression, humidité). On suppose que la date avec l'heure (y compris les secondes) est retournée par la méthode `getDate()` de la classe `Date`. Définir également dans votre classe `SensorHT` une méthode pour ajouter un triplet de mesure.
3. Créer un thread qui, toute les minutes, interroge, en boucle, tous les capteurs pour récupérer les valeurs et les stocker dans la table de hachage.
4. Votre programme doit pouvoir être interrogé par le réseau, il utilisera le port 4510. Un client doit récupérer des objets de type mesure lorsqu'il envoie une chaîne `GET` *ip date* ou les paramètres *ip* et *date* sont l'adresse IP du capteur et la date du jour. Que devez vous ajouter à votre programme ? Des problèmes de concurrence sont-ils à prévoir ?
5. Conclure sur la pertinence de l'utilisation d'une table de hachage pour cette application.

Exercice 63. (*) Réseaux et stockage (8pts)

On souhaite réaliser une application qui collecte des images à partir de caméras de surveillance connectées à des petits boîtiers (de type Raspberry Pi par exemple) avec un système d'exploitation GNU/Linux. Les boîtiers sont connectés sur un réseau IP et envoient à un serveur, lorsque la caméra détecte un changement, un objet Java contenant la date (date + heure, minute, seconde), l'adresse IP de la caméra, sa localisation (une chaîne de caractères) et une image (tableau d'octets). On suppose que le serveur doit gérer 50 boîtiers au maximum. Le serveur stocke les objets reçus dans une structure qui doit permettre de les retrouver rapidement en fonction de la localisation.

1. Quel protocole TCP/IP utiliser (UDP ou TCP) ?
2. Le serveur doit-il utiliser des threads pour traiter les demandes des clients (les boîtiers) ?
3. Quelle structure de données utilisez-vous pour stocker les objets reçus ? Comment traiter les problèmes de concurrence éventuels ?
4. Écrire l'extrait de code Java qui permet d'établir les sockets de connexion avec les clients et de recevoir un objet pour l'ajouter dans la structure de donnée retenue.
5. On souhaite enregistrer, toutes les heures, les objets reçus dans un fichier dont la nom comporte la date et l'heure (exemple `messages-2013-05-16-10H.dat`). On suppose que la méthode `DateHourToString` de la classe `Date` retourne la date du jour avec l'heure. Écrire la portion de code réalisant l'enregistrement.
6. On souhaite étendre le principe pour d'autres types de capteurs (température, CO_2, etc.) Ces capteurs renvoient des messages simples, contenant la date (complète), l'adresse IP du boîtier, la localisation, le type et la valeur de la mesure dans une

chaîne de caractères, toutes les 5 minutes. On veut pouvoir gérer plusieurs centaines de capteurs. Que devez vous modifier ou ajouter au niveau des mécanismes de communication de votre application ? Que devez vous ajouter comme structure(s) de stockage ?

Exercice 64. Sémaphore réseau (6pts)

Un développeur crée une application qui utilise intensivement le réseau entre plusieurs postes clients et serveurs. Afin d'optimiser les échanges, sans saturer le réseau, il souhaite développer un mécanisme de sémaphore pour l'accès au réseau (l'ouverture de socket client ou serveur).

Le sémaphore est un objet qui contient des jetons (eux mêmes des objets). Chaque jeton comporte une durée d'utilisation. Un programme qui a obtenu un jeton peut utiliser le réseau pendant la durée déterminée par le jeton (il peut créer des sockets serveur ou client pendant cette durée). Les jetons sont récupérés par le mécanisme de sérialisation. Le programme qui héberge le sémaphore (serveur de jetons) dialogue sur le port 8900.

1. Définir les classes nécessaires, sans donner le code, identifier celles qui sont des threads et celles qui sont des ressources.
2. Quel protocole allez vous utiliser pour implanter le serveur de jetons ?
3. Déterminer les problèmes de concurrence éventuels. Comment les résoudre ?
4. Écrire la portion de le code pour l'obtention d'un jeton.

Exercice 65. (*) Messagerie instantanée (7pts)

On souhaite développer une application (par exemple pour un système de messagerie instantanée) qui utilise TCP/IP pour permettre la communication entre plusieurs milliers de machines et donc plusieurs milliers d'utilisateurs.

Un socket serveur écoute sur le port 8990 en TCP. Au lieu de créer un thread pour chaque connexion, on utilise un nombre prédéterminé de threads dit *workers* (TW_i) pour prendre en charge des groupes de connexions. Au démarrage du programme n threads TW_i sont crées en plus du thread principal.

Chaque connexion possède un identifiant (id de type entier) et est associée à un nom de login unique. Le thread principal, suite à une acceptation sur le socket serveur, crée deux flux et transmet leur références à un des threads *worker* TW_k (celui qui a le moins de connexion en cours) qui les stocke dans une structure adaptée et effectuera les traitements (réception, émission).

Chaque thread *worker* TW_i considère, les un à la suite des autres, les flux en entrée et pour récupérer les données et les traiter. Pour les envois, chaque thread *worker* TW_i propose un méthode. Les différents threads worker doivent pourvoir communiquer entre eux et savoir qui gère une connexion donnée (selon son id ou login).

1. Faire un schéma du système en faisant figurer les différents threads et les clients. Déterminer les problèmes de concurrence éventuels. Comment les résoudre ?
2. Écrire l'extrait code de la classe principale pour : créer le socket serveur, n threads *worker* et leur passer les flux suite à l'acceptation d'une connexion. n est une constante du programme.
3. Définir la classe `Worker` : spécifier les variables de classe, d'instance ainsi que l'entête de chaque méthodes.

4. Quelles sont les limites de cette approche ? Peut-on gérer 200 000 utilisateurs ?
5. Quels sont les avantages et inconvénients de l'utilisation d'UDP au lieux de socket TCP ?
6. Donnez deux pistes de solutions pour pouvoir gérer 1 000 000 d'utilisateurs (faire un schéma pour chacune).

Exercice 66. (*) Calculs distribués (6pts)

On développe une application réseau pour effectuer des calculs sur des données hébergées sur différentes machines. Chaque machine envoie à la suivante le résultat de son calcul quelle a combiné avec ses données locales. Une machine maître collecte les résultats. Pour implanter ce système on utilise le port 8990 en TCP. Si une machine a plusieurs cœurs de processeurs, elle peut accepter plusieurs calculs. Dans les messages échangés entre les machines, figurent, en plus des données (une matrice de réels), la liste des machines qui doivent, successivement exécuter leur calcul et en fin de liste la machine maître.

1. Faire un schéma détaillé du système (on prendra le cas de 3 machines et un maître) en faisant figurer les parties serveur et client. Faire apparaître les adresses IP, les ports (vous pouvez utiliser des ports supplémentaires), les ressources. Déterminer les problèmes de concurrence éventuels. Comment les résoudre ?
2. Quels sont les mécanismes de Java que vous allez utiliser pour implanter ce système ?
3. Écrire l'extrait code de la classe principale de chaque machine pour : créer le socket serveur et créer un thread lors d'une connexion.
4. Définir les attributs de la classe Java relative aux messages (elle peut être composée).
5. Écrire l'extrait de code pour envoyer les données d'une machine à une autre (en général et non pas dans le cas de trois machines).

Exercice 67. (*) Application réseau, 2019 (8pts+q10+q11)

On développe une application réseau pour permettre à des utilisateurs de poster des photographies. À partir d'un programme client, l'utilisateur envoie une photographie à un serveur sous la forme d'un objet de la classe `Photo` contenant son nom d'utilisateur, un tableau d'octets (l'image), le nom de l'image incluant son format (jpg, png, etc.) et une liste d'annotations (tags ou hashtags). Le serveur range ces objets dans une structure de données permettant de retrouver toutes les photographies postées par un utilisateur et celles correspondant à une annotation. Le serveur doit supporter plusieurs clients simultanés.

1. Faire un schéma et y faire figurer le serveur, les clients, les threads, les structures de données partagées, les fichiers, etc. Décrire et justifier les constructions Java que vous allez utiliser pour implanter votre système.
2. Quel type de socket utilisez-vous pour les connexions entre les clients et le serveur ?
3. Écrire le squelette de la classe `Photo` (attributs et nom des méthodes, paramètres, sans le corps des méthodes).
4. Écrire le code de la méthode de la classe `Photo` qui permet de lire un fichier contenant une image.
5. Écrire l'extrait de code qui permet au serveur de prendre en charge plusieurs clients et de réceptionner des objets de la classe `Photo`.

6. Définir la classe `BasePhoto` du programme serveur qui permet de gérer et d'organiser les objets de la classe `Photo` (ajouter, rechercher par nom d'utilisateur, par annotation). Vous ne donnerez que les attributs, les noms des méthodes, leurs paramètres et leurs modifieurs (le code des méthodes n'est pas demandé). Indications : vous devez utiliser des tables de hachage, la classe `BasePhoto` ne contient qu'un objet.
7. Expliquer les problèmes de concurrence éventuels côté serveur et comment les résoudre ?
8. Écrire l'extrait de code qui permet d'ajouter une photo dans la classe `BasePhoto`.
9. Comment stocker l'objet de la classe `BasePhoto` dans un fichier ?
10. Comme se fera le stockage des données du serveur ? Écrire la portion de code permettant de stocker les différentes structures dans un fichier.
11. Est-il possible de lancer plusieurs serveurs ? Comment faire évoluer votre système pour répondre à cette demande ?

Exercice 68. (*) Gestion de messages, hastable 2019 (8pts)

Une entreprises familiale de vente de confitures et de produits régionaux veut informatiser son service d'expédition de colis.

1. Vous devez écrire un programme Java qui :
   (a) permet aux utilisateurs, depuis différents PC, de saisir au clavier une référence client et une référence de commande pour indiquer que la commande est expédiée. On suppose que la référence du client est une chaîne de caractères composée de son nom suivi d'un nombre à 6 chiffres. La référence commande est composée de l'année, du mois et d'un numéro d'ordre de commande dans le mois. On ne vérifie pas le format de l'entrée clavier.
   (b) range les éléments (couples client/commande) donnés par les utilisateurs dans une table de hachage ayant pour clé le client.
   (c) stocke toutes les 10 minutes la table de hachage sur le disque dur dans un fichier.
   (d) envoie toutes les 25 minutes la table de hachage à un serveur ayant pour adresse IP `192.168.10.11`.
2. Écrire le programme serveur qui réceptionne les tables de hachage provenant des différents postes et les fusionnent dans une grande table commune.

8 Divers

Exercice 69. (*) Extrait Examen de Juin 2010 (8pts)

1. Décrire la notion de socket des systèmes d'exploitation.
2. Quelles sont les relations entre flux Java et socket.
3. Un socket peut-il être partagé par plusieurs thread ?
4. Combien faut-il de niveaux pour stocker 2 321 éléments dans un B-arbre comportant 2 fils par nœud ?
5. Même question avec 4 fils par nœud.
6. Un algorithme de type Best Fit est-il utile dans le cas d'une allocation non contigüe ?
7. Quelle est la relation entre la notion de port TCP/IP et la notion de processus ?
8. Écrire un programme Java pour lire un fichier texte et recopier une ligne sur 2 dans un fichier destination spécifié en paramètre au lancement du programme.

Exercice 70. (*) Questions de cours (6pts)

1. Expliquer les notions de préemption et de réquisition.
2. Un constructeur de classe peut-il être déclaré `synchronized` ? Si oui expliquer au travers d'un exemple simple l'intérêt du modificateur `synchronized` sur le constructeur.
3. Expliquer la différence entre les termes *swap* et mémoire virtuelle.
4. Pourquoi l'allocation contiguë est généralement associée à un mécanisme de compactage ?

Exercice 71. (*) Questions de cours (6pts)

1. Pourquoi le mécanisme de réquisition est difficile à implanter dans le noyau d'un système d'exploitation ?
2. Expliquez les différences entre les méthodes `wait()` et `sleep()` de Java.
3. Expliquez la notion d'interblocage.
4. Un système d'exploitation installé sur une machine M fournit 2^{32} octets de mémoire adressable. La machine possède 2^{31} octets de mémoire physique. La gestion de la mémoire repose sur une gestion par pages. Chaque page a une taille de 8192 octets. Un processus utilisateur demande l'adresse 10023456. Qu'elle est la page de cette adresse ? Cette adresse est-elle en mémoire principale ou virtuelle ? Si l'adresse est en mémoire virtuelle expliquez comment le système d'exploitation y accède.

Exercice 72. (*) Examen (6 pts)

1. Définir et expliquer la notion de sérialisation dans Java.
2. Sur un système d'exploitation orienté serveur, gérant plusieurs centaines de processus et plusieurs milliers de threads, on souhaite améliorer l'ordonnanceur en remplaçant la liste des processus (processus et thread) en attente par une structure de hachage ou un B-arbre. Discuter des avantages et inconvénients de ces deux structures pour l'ordonnanceur.

3. Décrire trois méthodes de communication entre threads.
4. Définir la notion de classe dans l'adressage de TCP/IP.

Exercice 73. (*) Examen (6pts)

1. Expliquer le problème de concurrence induit par l'utilisation de flux vers des fichiers. Comment le résoudre ?
2. En considérant qu'une table de hachage et qu'un B-arbre sont des ressources utilisées par plusieurs threads, aussi bien en lecture qu'en écriture, comparer les avantages inconvénients des deux structures.
3. On considère une liste de 3 zones mémoire libres (non consécutives) de 50, 20, 40 blocs, chaque bloc a une taille de 1024 octets, un processus demande une allocation de 20 021 octets. Avec l'algorithme *best-fit* quelle zone sera choisie et quelles sont les tailles des fragments internes et externes produits.

Exercice 74. (*) Capteurs et architecture de collecte de données (8 pts)

On développe une application réseau pour collecter des données de capteurs avec une architecture à 3 niveaux : 1) les capteurs, 2) les machines de collecte de données, 3) un serveur qui centralise des mesures moyennes. Pour simplifier, on suppose que les données sont stockées dans un tableau d'entiers sur les machines de collecte. À chaque capteur est attribué un élément du tableau. Le capteur met à jour les données à intervalle régulier et écrase la précédente mesure. Le tableau possède 2000 éléments. On ne se préoccupe pas de la communication entre les capteurs et les machines de collecte.

1. Sur une machine de collecte, deux threads (S1 et S2) issus de la même classe, effectuent la somme des éléments du tableau en prenant en charge chacun une moitié du tableau. Un troisième thread M est informé dès que les deux threads (S1 et S2) ont terminé. M calcule alors la moyenne, attends 2 minutes et informe S1 et S1 qu'ils peuvent recommencer.
   (a) On suppose que le tableau de mesures est passé en paramètre au constructeur de thread. Existe-t-il des problèmes de concurrence par rapport à cette ressource ? Si oui comment les résoudre ?
   (b) Écrire le code Java du constructeur pour les threads S1 et S2.
   (c) Écrire le code Java de la méthode `run()` pour les threads S1 et S2.
   (d) Comment les threads S1 et S2 informent le thread M qu'ils ont terminé leur calcul ? Donnez les lignes à ajouter à la méthode `run()`.
   (e) Comment le thread M récupère les résultats de S1 et S2 ?
   (f) Écrire la méthode `run()` du thread M.
2. Après avoir calculé la moyenne, le thread M crée ensuite un objet mesure pour l'envoyer au serveur qui centralise les données provenant de plusieurs machines.
   (a) Écrire la classe mesure sachant qu'une mesure contient une date, un numéro d'identification du serveur (id) et une valeur.
   (b) Quel type de socket doit utiliser le thread M pour se connecter au serveur central sachant que ce dernier doit pouvoir traiter plusieurs envois simultanés de plusieurs clients ?

(c) Écrire le code Java pour envoyer une mesure au serveur central.

3. Sachant que le serveur central doit garder pour chaque serveur de collecte (une centaine environ) l'historique des mesures et que les recherches se font en fonction du numéro d'identification du serveur de collecte, quel type de structure de données utiliser ? Existe-t-il des problèmes de concurrence lorsque le serveur central reçoit plusieurs mesures et doit les insérer dans la structure ? Comment les résoudre ? Illustrez avec des extraits de code Java.

Exercice 75. (*) Examen (6pts)

1. Expliquer la signification du terme famine appliqué aux processus.
2. Montrer que les primitives `wait()`, `notify()` et `notifyAll()` peuvent provoquer une famine.
3. On considère n processus chaque processus utilise trois ressources (variables entières) R_1, R_2 et R_3. Chaque processus lit R_1 et R_2 puis R_3 et pour produire un résultat qui, en fonction de la valeur, est remis dans R_1 ou R_2 ou R_3. Ce comportement peut-elle provoquer un situation de verrou mortel ?

Exercice 76. (*) Examen seconde session (8pts)

Les réponses aux questions de cet exercice sont limitées à 1/2 page de texte par question (hors schémas et/ou extraits de code).

1. Expliquer la notion de *buffer* (mémoire tampon) et son rôle dans les systèmes d'exploitation.
2. Comparer les notions de mémoire cache et *buffer*.
3. Définir les modes d'interactions producteur(s)/consommateurs(s) et lecteur(s)/rédacteur(s), donner des exemples d'illustration. Comment la notion de *buffer* intervient dans ces deux modes d'interactions ?
4. Comparer les notions de moniteur et de sémaphore. Donner leur traduction en Java. Montrez en quoi les sémaphores sont utiles pour répondre à des problèmes de synchronisation de processus.

Exercice 77. (*) Contrôle continu (6pts)

1. Pourquoi utiliser des pages plutôt que directement les blocs pour mettre en place un mécanisme de mémoire virtuelle ?
2. Expliquez pourquoi l'extrait de programme suivant provoque le message `Illegal Monitor State Exception` lors de son exécution. Que doit-on corriger (ajouter) ?

```
1  ...
    public void P() throws InterruptedException{
3     while (compteur==0) {this.wait(); }
      compteur--;
5    }

7  public void V() {
      compteur++;
9     this.notify();
    }
11 ...
```

Listing 3 – Classe sémaphore

3. Quel est la signification du terme attente active ? Quel sont les inconvénients engendré par ce type d'attente ? Comment y remédier ?
4. Définir la notion de section critique, donnez un exemple de code.

Exercice 78. (*) Ordonnancement et gestion de la mémoire (7pts)

1. On considère un tourniquet à deux files d'attentes associées à des *quantums* respectifs de 2 et 3 unités de temps (ut). Le processeur possède deux cœurs par conséquent deux processus peuvent s'exécuter en même temps avec des *quantums* différents. Un processus qui a déjà été exécuté au moins une fois est mis en file numéro 2. Les processus de la file 2 n'ont accès au processeur que si les processus de la file d'attente 1 sont peu nombreux (2 au maximum). Les files sont triées en FIFO, il n'y a pas de réquisition.

Nom	p0	p1	p2	p3	p4	p5	p6
Top d'arrivée	1	2	4	6	7	10	12
Durée (ut)	3	2	3	5	2	1	1

   Simuler le comportement du système avec les données fournies dans le tableau ci-dessus. Calculer le taux de retard pour chaque processus et le taux de retard moyen. Que constatez vous ?
2. Le système dispose de 1200 unités de mémoire dont 400 de RAM. La mémoire est organisée en pages de 100 unités. Les demandes d'accès mémoire sont dans l'ordre d'arrivée : 60, 180, 70, 205, 80, 320, 630, 62, 920, 210, 89, 340, 805, 751, 219. Donner les différents états de la mémoire et le nombre de défauts de page total.

Exercice 79. (*) Ordonnancement, mémoire virtuelle (7pts)

1. On considère un tourniquet à trois files d'attentes associées un *quantum* de 2 unités de temps (ut). Le processeur possède un seul cœur. Les files sont gérées en FIFO, il n'y a pas de réquisition.

Nom	p0	p1	p2	p3	p4	p5
Top d'arrivée	1	2	3	5	9	10
Durée (ut)	3	5	2	7	1	3

   (a) Un processus qui a déjà été exécuté au moins une fois est mis en file numéro 2 et ainsi de suite. Simuler le comportement du système avec les données fournies dans le tableau ci-dessus. Calculer le taux de retard pour chaque processus et le taux de retard moyen.

   (b) Peut-on améliorer le fonctionnement du système avec des *quantums* différents associés à chaque file. Si oui, donner un exemple précis d'une séquence d'exécution de processus.

2. On considère une mémoire de 10 000 éléments (blocs) dont 4 000 de mémoire physique (RAM), le reste en mémoire virtuelle. La mémoire est gérée par pages de 1 000 éléments.

   (a) Faire un schéma de l'organisation de la mémoire globale, numéroter les pages.

   (b) Suivre l'exécution de l'état de la mémoire sachant que la stratégie de déchargement des pages est en FIFO. Donnez le nombre de défaut de page et précisez les correspondances d'adresses (page, offset) pour les demandes décrites dans le tableau 1. Au départ la mémoire est vide.

Demande	Conversions adresse =(page,offset)			Mémoire centrale	
	Adresse	Page	Offset	Page	Adresse
1	1657	1	657	0	657
2	523				
3	2170				
4	3725				
5	1133				
6	6145				
7	5190				
8	1658				
9	4573				
10	8156				
11	1598				
12	8888				
13	1134				
14	2170				
15	1477				
16	6472				
17	3479				
18	8172				
19	1472				
20	5000				

TABLE 10 – Demandes d'accès mémoire

(c) Qu'observez-vous ? Que pouvez-vous en déduire comme amélioration de la stratégie ?

Exercice 80. (*) Questions de cours, 2019 (6pts)

1. Expliquez pourquoi les noyaux des systèmes d'exploitation gèrent les différents types de mémoires par blocs ou par pages.
2. Les moniteurs sont disponibles dans le langage Java avec le mot-clé `synchronized`, montrez pourquoi cette construction syntaxique est bien adaptée à la programmation orientée objet. Vous pouvez comparer les moniteurs et les verrous sur ce point.
3. Expliquez ce qu'est la mémoire virtuelle et comment fonctionne le mécanisme de *swap*.

Exercice 81. Sémaphores, révision

Ce problème est adapté de l'ouvrage de A. Tanenbaum (Operating Systems : Design and Implementation). Dans le parc national Kruger en Afrique du Sud, il y a un canyon profond. Une seule corde (liane) enjambe le canyon. Les babouins peuvent traverser le canyon utilisant la liane, mais si deux babouins allant dans des directions opposées se rencontrent au milieu, ils se battront et tomberont à l'eau (l'issue sera fatale aux deux). En outre, la corde n'est pas suffisamment solide pour contenir plus de 5 babouins. S'il y a plus de 5 babouins sur la corde en même temps, elle se cassera (l'issue sera fatale pour tous). En supposant que nous pouvons apprendre aux babouins à utiliser des sémaphores, vous devez concevoir un schéma de synchronisation respectant les propriétés suivantes :

— une fois qu'un babouin a commencé à traverser, il est garanti qu'il pourra se rendre de l'autre côté sans tomber sur un babouin arrivant dans l ?autre sens ;
— il n'y a jamais plus de 5 babouins sur la corde.

1. Identifier les classes qui constituent des ressources et des threads.
2. Comment coordonner l'activité des babouins (synchroniser) ? Faire un schéma.
3. Quels sont les problèmes de concurrence ?
4. Implanter le programme.
5. eut-il y avoir un problème de famine ? Si oui proposer une modification de votre programme pour le résoudre.

Annexe 1

FIFO

Top	File d'attente						processeur
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							

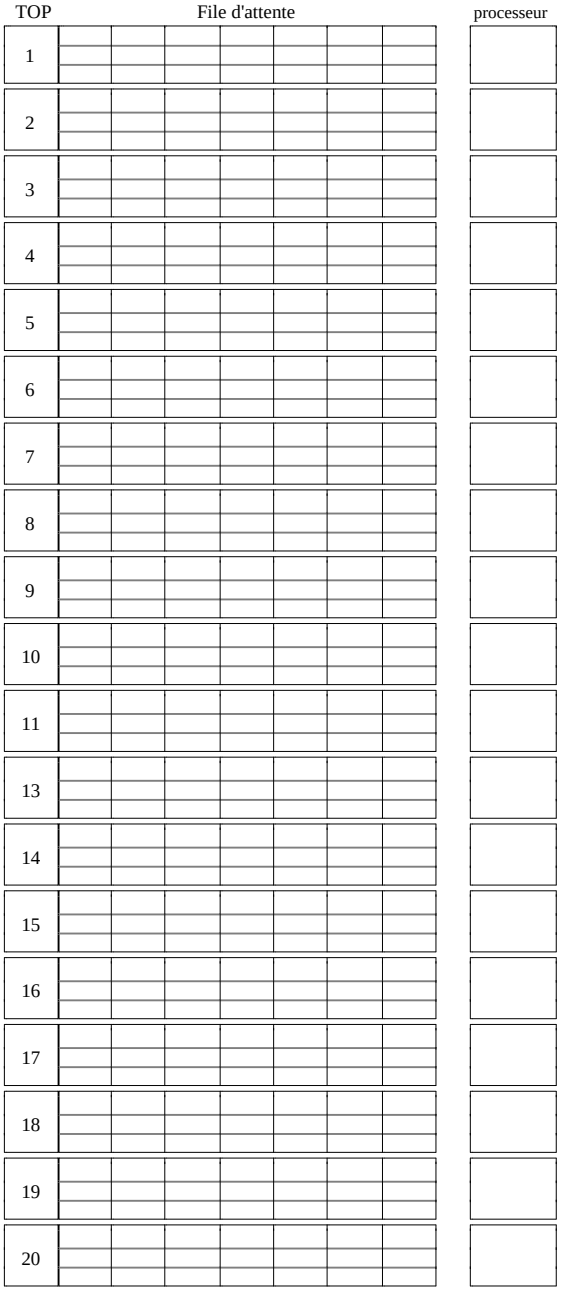
Plus court d'abord

Top	File d'attente						processeur
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							

Tourniquet simple

Top	File d'attente						processeur
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							

Annexe 2


TOP
File d'attente
processeur
1
2
3
4
5
6
7
8
9
10
11
13
14
15
16
17
18
19
20

		Totale	Partielle	Optimisée
1	P1			
2	P1			
3				
4				
5				
6				
7	P2			
8	P2			
9	P2			
10				
11	P3			
12				
13				
14				
15				
16				
17				
18				
19				
20	P4			
21	P4			
22	P4			
23	P4			
24	P4			
25				
26				
27				
28				
29	P5			
30	P5			
31	P5			
32	P5			
33				
34				
35				
36				
37				
38				
39	P6			
40				
41				
42	P7			
43	P7			
44	P7			
45	P7			

TABLE 11 – Annexe 3