TP 2 : Utilisation d'un simulateur de système d'exploitation

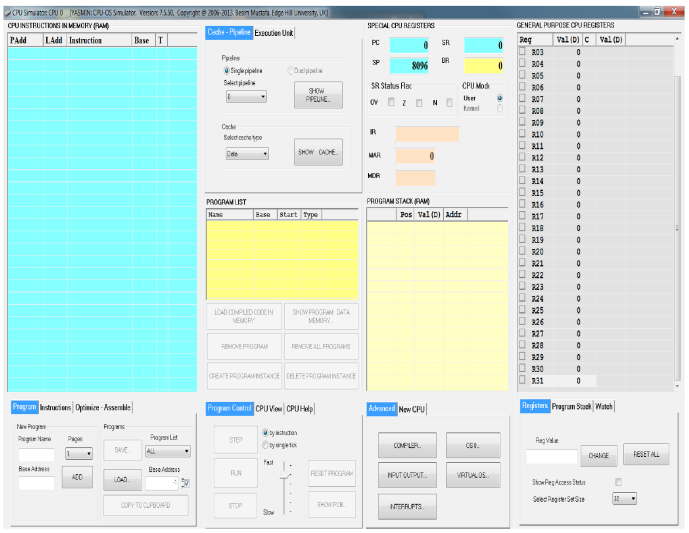
NFA003 : Principes et fonctionnement des systèmes d'exploitation

2014/2015

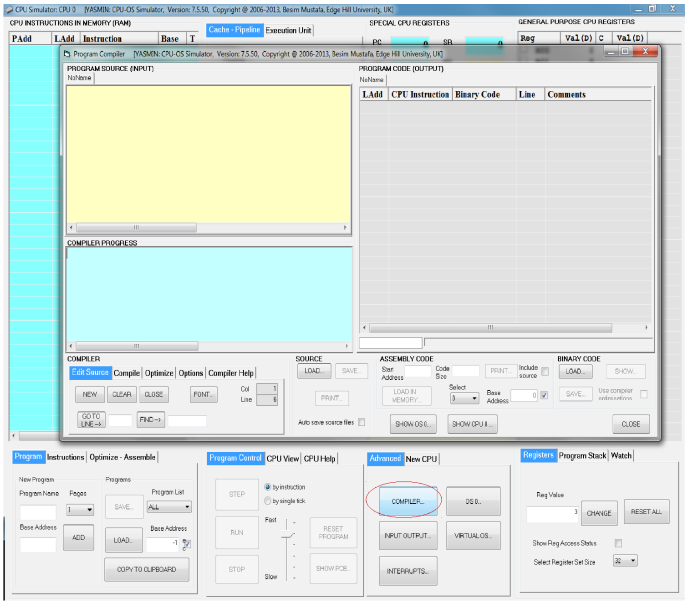
Supports :
Tous les supports pour ce tp sont disponibles sur à l'adresse :
`http://deptinfo.cnam.fr/new/spip.php?rubrique138`

Le but de ce tp est d'illustrer les concepts théoriques vus en cours à l'aide d'un simulateur de processeur et de système d'exploitation. Ce simulateur permet de visualiser et d'animer les mécanismes mis en oeuvre dans le fonctionnement du système d'exploitation. Il permet d'observer les taches cachées du système d'exploitation.

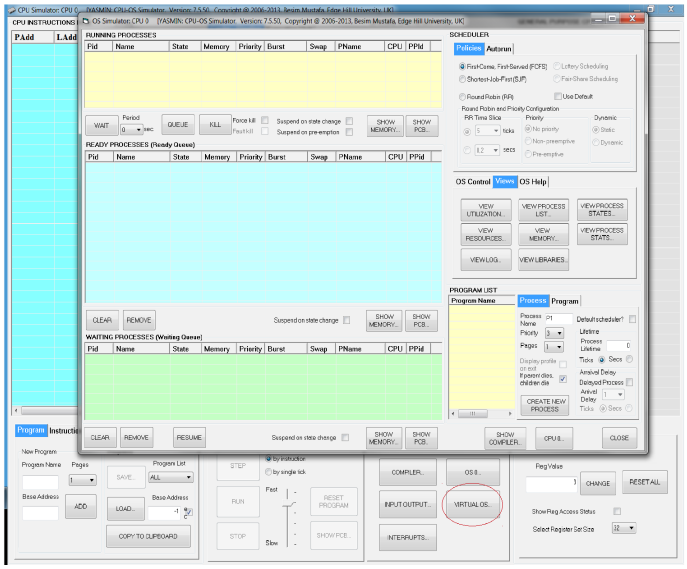
Lancer le simulateur, la fenêtre qui représente le `CPU` s'ouvre :



Vous pouvez accéder au compilateur via le bouton `COMPILER` :



Vous pouvez accéder au système d'exploitation virtuel via le bouton `VIRTUAL OS` :



Vous pouvez remarquer que sur les fenêtres `COMPILER` et `VIRTUAL OS`, il y a des boutons `SHOW OS`, `SHOW COMPILER` et `SHOW CPU` qui permettent la navigation d'une fenêtre à l'autre.

Exercice 1 — ***La compilation*** Le but de cet exercice est de vous illustrer les étapes de la compilation d'un programme. Pour rappel, un compilateur transforme un programme source en code objet (binary code) en trois étapes : l'analyse lexicale, l'analyse syntaxique et l'analyse sémantique. Au fil de ces trois étapes, le compilateur construit une table de symbole. Enfin, il génère le code objet. Dans cet exercice, nous ne considérerons pas d'édition de lien.

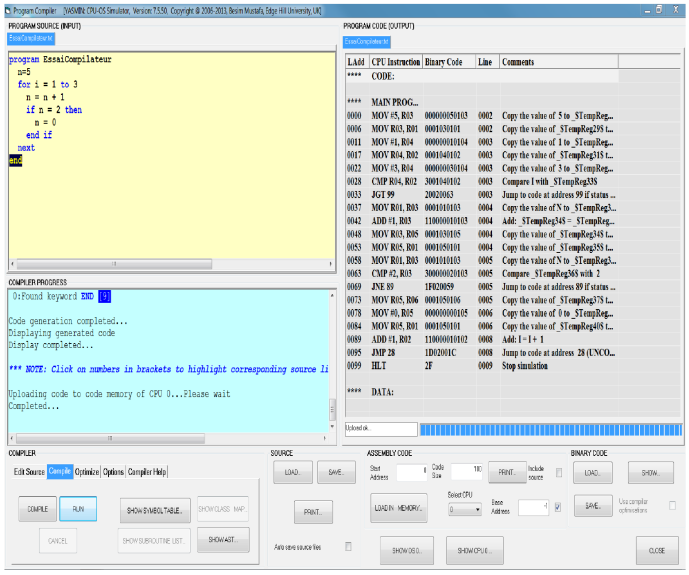
1) Dans la fenêtre `CPU`, cliquez sur le bouton `COMPILER` pour ouvrir la fenêtre `COMPILER`
2) Tapez dans le cadre `PROGRAM SOURCE (INPUT)` le programme suivant :

```
program EssaiCompilateur
  n=5
  for i = 1 to 3
    n = n + 1
    if n = 2 then
      n = 0
```

```
    end if
  next
end
```

Vous pouvez sauvegarder votre programme en cliquant sur le bouton `SAVE` du cadre `SOURCE`.

3) Compilez ce programme en cliquant sur le bouton `COMPILE` dans le cadre `COMPILER`, onglet `Compile`. Vous pouvez reconnaître le code machine, généré par la compilation, qui est affiché dans le cadre `PROGRAM CODE (OUTPUT)`. Plus précisément, vous pouvez voir la colonne `LAdr` qui est l'adresse logique de l'instruction, l'instruction dans la colonne `CPU Instruction`, le code binaire associé à l'instruction dans la colonne `Binary Code`, la ligne du code machine dans la colonne `line`, et dans la colonne `comments`, ce que fait l'instruction.



4) En observant l'exécution de la compilation dans le cadre `COMPILER PROGRESS` répondez aux questions suivantes :

   a) Quelles sont les 4 étapes que le compilateur a effectuées ?

   b) Combien d'unités lexicales (ou "token") y a-t-il dans le code de `EssaiCompilateur` ?

   c) Dans l'onglet `Compiler Help` du cadre `COMPILER`, cliquez sur `SHOW HELP INFORMATION`, vous pouvez regarder les règles syntaxiques du langage utilisé dans votre programme.

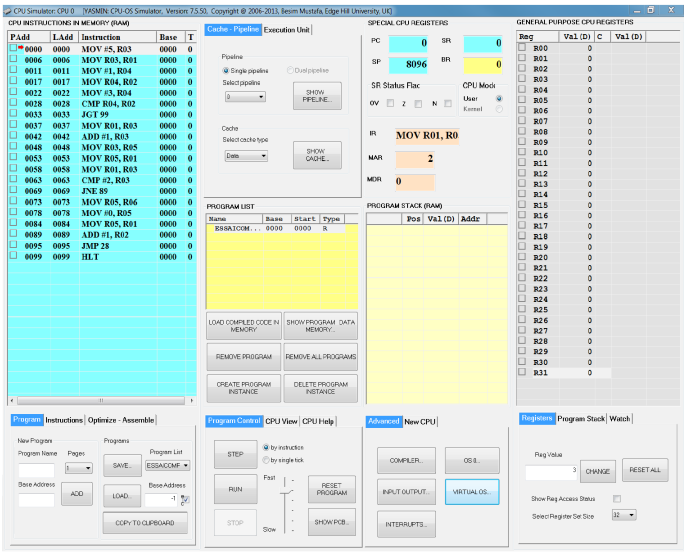
   d) A votre avis, quelle est la taille de la table des symboles ?

   e) Cliquez maintenant sur le bouton `SYMBOL TABLE` pour faire apparaître la table des symboles.

      i) A quoi est associée chaque ligne de la table des symboles ? Détaillez les informations contenues dans cette table.

      ii) Dans le cadre `PROGRAM CODE`, les nombres dans la colonne `LAdd` sont les adresses logiques des instructions. Que représentent-elles ?

5) Nous allons maintenant exécuter le programme. Chargez le code machine en mémoire Centrale en cliquant sur le bouton `LOAD IN MEMORY`. Allez maintenant sur la fenêtre `CPU` (bouton `SHOW CPU`). Faites maintenant tourner le programme en cliquant sur le bouton `RUN` :



Observez ce qui se passe au niveau du processeur dans la fenêtre `CPU` du simulateur :

a) Dans le cadre `CPU INSTRUCTION IN MEMORY`, vous pouvez voir l'état de la mémoire centrale (ou RAM) après chargement du programme. La première colonne `PAdr` est l'adresse physique de chaque ligne du programme, la deuxième `LAdr` leur adresse logique et la troisième `Instruction` est l'instruction en code machine.

b) Dans le cadre `SPECIAL CPU REGISTER` vous pouvez observer la valeur des registres particuliers du processeur :
   — `PC (Program Counter)` (ou compteur ordinal) : adresse de la prochaine instruction à exécuter,
   — `SP (Stack pointeur)` : adresse du pointeur de pile,
   — `SR (Status Register)` : le registre contenant des informations sur l'exécution de la dernière instruction,
   — `BR (Base Register)` : le registre de base,
   — `SR (Status Flag)` : le registre PSW,
   — `CPU mode` : mode utilisateur ou superviseur,
   — `IR (Instruction Register)` : le registre d'instruction,
   — `MAR (Memory Address Register)` : le registre d'adresse associé à la mémoire centrale,
   — `MDR (Memory Data Register)` : le registre de donnée associé à la mémoire centrale.

c) Dans le cadre `GENERAL PURPOSE CPU REGISTER` vous pouvez observer la valeur des registres généraux du processeur.

d) Dans le cadre `PROGRAM LIST` vous pouvez observer la liste des programmes chargés en mémoire centrale.

e) Dans l'onglet `Program control` vous pouvez choisir la vitesse du processeur (molette `slow-fast`), puis exécuter les programmes chargés en mémoire centrale avec les boutons `RUN` (exécution de la totalité du programme) ou `STEP` (exécution par ligne).

Exercice 2 — *Les états des processus* Le but de cet exercice est de décrire les trois principaux états des processus et d'en identifier les transitions valides.

1) Création d'un premier processus :

a) Cliquez sur le bouton `COMPILER` pour ouvrir la fenêtre `COMPILER`

b) Téléchargez le programme `BoucleInfini` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

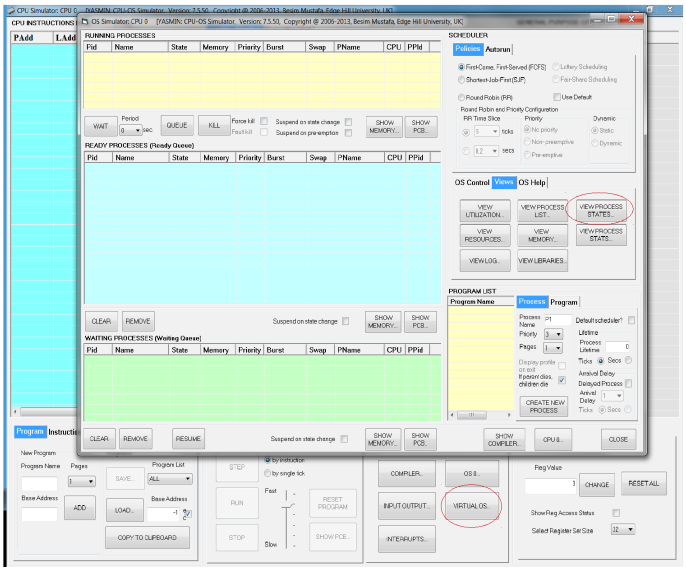
```
program BoucleInfini
  n=0
  while true
    n = n + 1
  wend
end
```

c) Compilez ce programme en cliquant sur le bouton `COMPILE` dans le cadre `COMPILER`, onglet `Compile`.

d) Enregistrez ce programme en cliquant sur le bouton `SAVE` du cadre `SOURCE` sous le nom `BoucleInfini`.

2) Les états des processus et les transitions

a) Dans la fenêtre `CPU` du simulateur, cliquez sur `VIRTUAL OS` une nouvelle fenêtre s'ouvre :



Sélectionnez l'onglet `VIEWS` et cliquez sur le bouton `VIEW PROCESS STATES`. La fenêtre qui apparaît est une représentation graphique des états des processus. On peut y voir la file des processus qui sont dans l'état prêt (`Ready state`), le CPU où il peut y avoir un seul processus dans l'état élu (`Running state`), et la file des processus qui sont dans l'état bloqué (`Waiting state`).

b) Dans cette fenêtre, cochez les checkbox `Stay on top` et `Animate`, afin de pouvoir modifier les états d'un processus que vous allez maintenant créer.

c) Chargez le programme `BoucleInfini` dans la mémoire centrale du simulateur. Pour cela retournez sur la fenêtre du compilateur et cliquez sur le bouton `LOAD IN MEMORY` du cadre `ASSEMBLY CODE`. Vous pouvez observer dans la fenêtre `CPU` que le programme est chargé dans la mémoire centrale.

d) Créez maintenant un processus à partir du programme chargé en mémoire. Pour cela retournez dans la fenêtre `VIRTUAL OS`, vous pouvez voir dans la liste `PROGRAM LIST` le programme `BoucleInfini`. Cliquez maintenant sur le bouton `CREATE NEW PROCESS` de l'onglet `Process`. Dans la fenêtre qui représente les états des processus, on peut voir ce processus dans la file des processus prêts avec son numéro (pid).

e) Dans la fenêtre `VIRTUAL OS` cochez maintenant les trois checkbox `Suspend and state change` dans le cadre `READY PROCESSES`. Cela permettra au simulateur d'arrêter son exécution et d'afficher un message à chaque changement d'état d'un processus.

f) Pour modifier les états d'un processus, il faut faire des "cliquer-glisser". Si un transition est impossible, le message `"**ERROR:Illegal state transition"` est affiché en bas de la fenêtre. Cliquez sur le bouton `RESUME`, effectuez les actions suivantes et remplissez les 3 colonnes de droite :

Actions	Succès	Échec	État résultant
Glissez le processus dans la file des processus bloqués (`waiting queue`)			
Glissez le processus dans la poubelle pour le terminer (`bin`)			
Glissez le processus dans le CPU. Dans la fenêtre `VIRTUAL OS`, onglet `OS Control`, cliquez sur la bouton `RESUME`			
Glissez le processus dans la file des processus bloqués (`waiting queue`). Dans la fenêtre `VIRTUAL OS`, onglet `OS Control`, cliquez sur la bouton `RESUME`			
Glissez le processus dans le CPU			
Glissez le processus dans la poubelle pour le terminer (`bin`)			
Glissez le processus dans la file des processus prêts (`ready queue`). Dans la fenêtre `VIRTUAL OS`, onglet `OS Control`, cliquez sur la bouton `RESUME`			
Glissez le processus dans la file des processus prêts (`ready queue`). Dans la fenêtre `VIRTUAL OS`, onglet `OS Control`, cliquez sur la bouton `RESUME`			
Glissez le processus dans la poubelle pour le terminer (`bin`)			

g) Résumez dans le tableau suivant les transitions d'états possible pour un processus, et dessinez le graphe des états des processus.

Depuis l'état	Vers l'état

3) Étude des états des processus

a) Téléchargez le programme `AttenteLecture` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

```
program AttenteLecture
  while true
    wait read(caractere)
    writeln(caractere)
  wend
end
```

Chargez le programme en mémoire à l'aide du bouton `LOAD IN MEMORY`.

b) Dans la fenêtre `CPU` du simulateur, puis dans l'onglet `ADVANCED` cliquez sur le bouton `INPUT OUTPUT`. Une console (ou invite de commande) apparaît, cochez la checkbox `Stay on top`, puis cliquez sur le bouton `SHOW KBD` pour ouvrir une fenêtre clavier.

c) Dans la fenêtre `VIRTUAL OS`, vérifiez qu'aucune des trois checkboxes `Suspend on state change` ne sont cochées, et que la vitesse du CPU est à la position la plus rapide. Créez un processus avec le bouton `CREATE NEW PROCESS`.

d) Cliquez sur le bouton `START`, et attendez que le processus soit dans l'état `Waiting` apparaisse dans la console. Dans quel état se trouve le processus ? (on peut le voir dans la fenêtre `CPU`)

e) En utilisant la fenêtre clavier, cliquez sur n'importe quel bouton. Que se passe-t-il dans le simulateur ?

f) A votre avis que fait ce programme ?

Exercice 3 — *L'ordonnancement des processus* 1) Création de processus :

a) Cliquez sur le bouton `COMPILER` pour ouvrir la fenêtre `COMPILER`

b) Téléchargez le programme `BoucleOrdo` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

```
program BoucleOrdo
  i = 0
  for n = 0 to 40
    i = i + 1
  next
end
```

Enregistrez ce programme sous le nom `BoucleOrdo`

c) Compilez en cliquant sur le bouton `COMPILE` dans le cadre `COMPILER`, onglet `Compile`.

d) Chargez le programme dans la mémoire centrale du simulateur. Pour cela, cliquez sur le bouton `LOAD IN MEMORY` du cadre `ASSEMBLY CODE`. Vous pouvez observer dans la fenêtre `CPU` que le programme est chargé dans la mémoire centrale.

e) Créez maintenant un processus à partir du programme chargé en mémoire. Pour cela cliquez sur le simulateur `VIRTUAL OS`. La fenêtre s'ouvre et vous pouvez voir dans la liste `PROGRAM LIST` le programme `BoucleOrdo`. Il est maintenant possible de créer autant de processus que souhaité à partir de de programme. Pour cela cliquez sur le bouton `CREATE NEW PROCESS`. Créez 3 processus. On peut observer les quatre processus dans la file des processus prêt qui est représentée par le vue `READY PROCESSES`.

2) Politiques d'ordonnancement

a) Sélectionnez la politique d'ordonnancement `FCFS` dans `SCHEDULER/Policies`. Pour activer l'OS, réglez la vitesse au maximum et cliquez sur le bouton `START`. Qu'observez vous quand les processus s'exécutent ? (Regardez spécifiquement `RUNNING PROCESSES` and `READY PROCESSES`). Attendez la fin de l'exécution ou utilisez le bouton `KILL` pour terminer les processus.

b) Sélectionnez la politique d'ordonnancement `Round-Robin (RR)` dans `Scheduler/Policies`, et l'option `Non-preemptive` dans `Scheduler/Policies/Priority Type`. Créez 2 processus avec les priorités suivantes : 3 et 2. Pour cela utilisez la liste `Priority` dans `PROGRAM LIST/Processes`. réglez la vitesse à à la moitié et cliquez sur le bouton `START`. Pendant l'exécution créez un troisième processus de priorité 1. Qu'observez vous ? Attendez la fin de l'exécution ou utilisez le bouton `KILL` pour terminer les processus.

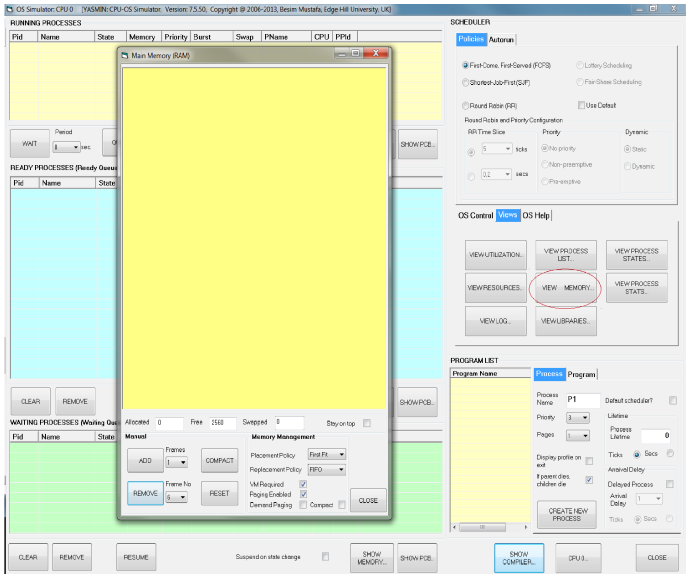
c) Sélectionnez maintenant l'option `Pre-emptive` dans `Scheduler/Policies/Priority Type`. Créez 2 processus avec les priorités 3 et 2. Choisissez `40 ticks` (40 battements d'horloge) dans `RR Time slice`. Réglez la vitesse à à la moitié et cliquez sur le bouton `START`. Pendant que le premier processus s'exécute créez un troisième processus de priorité 1. Qu'observez vous, quelles sont les différences avec la question b) ? Attendez la fin de l'exécution ou utilisez le bouton `KILL` pour terminer les processus.

Exercice 4 — *Allocation linéaire de la mémoire virtuelle* Après la création d'un processus, le système doit trouver un emplacement dans la mémoire centrale (ou RAM) pour charger le code et les données associés au processus.

Trois algorithmes sont utilisées pour déterminer un emplacement libre de la mémoire centrale : `First fit`, `Best fit` et `Worst fit`.

1) Il faut tout d'abord supprimer tous les programmes présents en mémoire centrale. Pour cela, dans la fenêtre `CPU`, retirez tous les processus qui sont dans les différentes files en utilisant dans l'onglet `Program List` le bouton `REMOVE PROGRAM` après avoir sélectionné les programmes.

2) Allez ensuite dans la fenêtre `VIRTUAL OS` dans l'onglet `Views` et cliquez sur le bouton `VIEW MEMORY`, une fenêtre représentant la mémoire centrale s'ouvre. Assurez vous que la liste `Frame` a la valeur `1` :



3) Nous allons maintenant remplir la mémoire centrale, pour cela cliquez sur le bouton `ADD` jusqu'à ce qu'elle soit pleine (c'est à dire 10 fois)

4) Nous allons maintenant libérer des emplacements mémoire, pour cela suivez les instructions suivantes :
   — Mettez la liste `Frame No` à la valeur `2` et cliquez sur `REMOVE`
   — Mettez la liste `Frame No` à la valeur `3` et cliquez sur `REMOVE`
   — Mettez la liste `Frame No` à la valeur `5` et cliquez sur `REMOVE`
   — Mettez la liste `Frame No` à la valeur `7` et cliquez sur `REMOVE`
   — Mettez la liste `Frame No` à la valeur `8` et cliquez sur `REMOVE`
   — Mettez la liste `Frame No` à la valeur `9` et cliquez sur `REMOVE`
   Il n'y a maintenant plus que les cadres ("frames") 0, 1, 4, et 6 qui sont alloués, avec des espaces libres de la mémoire centrale de différentes tailles entres ces cadres.

5) Choisissez l'algorithme de sélection d'emplacement dans la liste `Placement Policy` comme étant `First Fit`. Chargez ensuite le programme `BoucleInfini` dans la fenêtre `COMPILER`, compilez le (bouton `Compile`) et chargez le en mémoire (bouton `LOAD IN MEMORY`). Dans la fenêtre `VIRTUAL OS`, créez le processus associé (bouton `CREATE NEW PROCESS`). Dans quel emplacement mémoire le processus a-t-il été chargé ?
   Effacer le processus (mais pas le programme) en cliquant sur le bouton `REMOVE` de la zone `READY PROCESSES` (fenêtre `VIRTUAL OS`).

6) Choisissez l'algorithme de sélection d'emplacement dans la liste `Placement Policy` comme étant `Best Fit`, créez un nouveau processus. Dans quel emplacement mémoire le processus a-t-il été chargé ?
   Effacez à nouveau le processus (bouton `REMOVE`).

7) Choisissez l'algorithme de sélection d'emplacement dans la liste `Placement Policy` comme étant `Worst Fit`, créez un nouveau processus. Dans quel emplacement mémoire le processus a-t-il été chargé ?
   Effacez à nouveau le processus (bouton `REMOVE`), et cliquez sur le bouton `RESET` de la fenêtre `Main Memory` pour vider la mémoire.

8) Expliquez ce que fait chaque algorithme dans le tableau suivant :

Algorithme `First fit`	
Algorithme `Best fit`	
Algorithme `Worst fit`	

Exercice 5 — *La synchronisation* Dans cet exercice, nous illustrons comment le partage de ressources peut entraîner des situations de blocages lorsque ces ressources ne sont pas protégées. Pour cela, nous introduisons la notion de processus léger (ou "Thread"), qui est similaire à un processus, car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les "threads" d'un même processus se partagent sa mémoire virtuelle.

1) Cliquez sur le bouton `COMPILER` pour ouvrir la fenêtre `COMPILER`.
2) Téléchargez le programme `SectionCritique1` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

```
program SectionCritique1
  var g integer

  sub thread1 as thread
    writeln("In thread1")
    g = 0
    for n = 1 to 20
      g = g + 1
    next
    writeln("thread1 g = ", g)
    writeln("Sorti de thread1")
  end sub

  sub thread2 as thread
    writeln("In thread2")
    g = 0
    for n = 1 to 12
      g = g + 1
    next
    writeln("thread2 g = ", g)
    writeln("Sorti de thread2")
  end sub

  writeln("Nous sommes dans le main")
  call thread1
  call thread2

  wait
  writeln("Sorti du main")
end
```

   Ce programme crée un programme principal, appelé `SectionCritique1`, dans lequel il crée deux processus légers : `Thread1` et `Thread2`. Chacun de ces processus légers incrémente la variable globale `g` dans une boucle.
   Quelle est à votre avis la valeur attendue pour `g` à la fin de l'exécution de `SectionCritique1` ?
3) Compilez ce programme en cliquant sur le bouton `COMPILE` dans le cadre `COMPILER`, et chargez le programme dans le CPU en cliquant sur le bouton `LOAD IN MEMORY`. Affichez ensuite la console en cliquant sur le bouton `INPUT/OUTPUT` de la fenêtre `CPU`, dans laquelle vous cocherez la checkbox `Stay on top`.

4) Créez maintenant un processus à partir du programme chargé en mémoire. Pour cela cliquez sur le bouton `VIRTUAL OS` de la fenêtre `CPU`. La fenêtre `VIRTUAL OS` s'ouvre et vous pouvez voir dans la liste `PROGRAM LIST` le programme `SectionCritique1`. Créez un instance de ce programme en cliquant sur le bouton `CREATE NEW PROCESS`.

5) Sélectionnez la politique d'ordonnancement `Round-Robin (RR)` dans `Scheduler/Policies`, et la valeur `10 ticks` dans `RR Time Slices`. Réglez la vitesse à la plus rapide et cliquez sur le bouton `START`. Attendez que le programme termine son exécution. Regardez les valeurs de `g` à la fin de l'exécution du programme. Sont-elles celles que vous attendiez ?

6) Changez `RR Time Slices` à `5 ticks` et relancez le programme. Dans la console, regardez les valeurs de la variable `g`, est-elle différente ? Si oui, pourquoi ?

7) Téléchargez le programme `SectionCritique2` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

```
program SectionCritique2
  var g integer

  sub thread1 as thread synchronise
    writeln("In thread1")
    g = 0
    for n = 1 to 20
      g = g + 1
    next
    writeln("thread1 g = ", g)
    writeln("Sorti de thread1")
  end sub

  sub thread2 as thread synchronise
    writeln("In thread2")
    g = 0
    for n = 1 to 12
      g = g + 1
    next
    writeln("thread2 g = ", g)
    writeln("Sorti de thread2")
  end sub

  writeln("Nous sommes dans le main")
  call thread1
  call thread2

  wait
  writeln("Sorti du main")
end
```

Où la commande `synchronise` s'assure que les deux processus légers s'exécutent en exclusion mutuelle.
En suivant les mêmes instructions qu'aux questions précédentes, lancez le programme et regardez les valeurs de `g`. Les résultats sont-ils différents des questions 5) et 6) ? Si oui, pourquoi ?

8) Téléchargez le programme `SectionCritique3` à l'adresse `http://deptinfo.cnam.fr/new/spip.php?rubrique138` et ouvrez le dans l'éditeur de texte du compilateur :

```
program SectionCritique3
  var g integer

  sub thread1 as thread
    writeln("In thread1")
```

```
    enter
    g = 0
    for n = 1 to 20
      g = g + 1
    next
    writeln("thread1 g = ", g)
    leave
    writeln("Sorti de thread1")
  end sub

  sub thread2 as thread
    writeln("In thread2")
    enter
    g = 0
    for n = 1 to 12
      g = g + 1
    next
    writeln("thread2 g = ", g)
    leave
    writeln("Sorti de thread2")
  end sub

  writeln("Nous sommes dans le main")
  call thread1
  call thread2

  wait
  writeln("Sorti du main")
end
```

Où le code compris entre les commandes `enter` et `leave` est exécuté sans partager le CPU. Comment s'appellent les zones de code entre ces deux commandes ?
En suivant les mêmes instructions qu'aux questions précédentes, lancez le programme et regardez les valeurs de `g`. Les résultats sont-ils différents des questions 5), 6) et 7) ? Si oui, pourquoi ?