

NOM Prénom :

Consignes

- L'examen dure 1h30. Prenez le temps de lire le sujet en entier (12 questions sur 6 pages)
- Sauf avis contraire, chaque question est indépendante. Le barème est purement indicatif.
- Répondez directement sur le sujet. Commencez par écrire votre nom ci-dessus.
- Écrivez lisiblement et surtout sans ratures. Utilisez un brouillon (vraiment).
- Documents et appareils interdits, sauf une feuille A4 recto-verso manuscrite.
- Pour les calculs en binaire, aidez-vous du tableau page 6.
- Dans les questions vrai/faux, chaque réponse correcte rapporte des points, chaque réponse incorrecte «annule» une réponse correcte (de la même question).

Noyau et Processus

Question 1 (1 pt) On s'intéresse au fragment de programme ci-dessous. Pour chaque proposition, entourez V si l'équation décrit un comportement possible du programme, ou au contraire entourez F pour un comportement impossible. Les lettres A et B désignent les deux nombres affichés par le processus parent, et de même les lettres C et D représentent les affichages du processus enfant.

- | | | |
|--------------------------|--------------------------|------------|
| <input type="checkbox"/> | <input type="checkbox"/> | A = C + 10 |
| <input type="checkbox"/> | <input type="checkbox"/> | C = A + 10 |
| <input type="checkbox"/> | <input type="checkbox"/> | D = B |
| <input type="checkbox"/> | <input type="checkbox"/> | D ≠ B |

```
if (fork() == 0)
{
    x = x + 5;
    printf("%d,%d\n", x, &x);
}
else
{
    x = x - 5;
    printf("%d,%d\n", x, &x);
}
```

Question 2 (2 pts) Complétez la fonction ci-dessous de telle sorte que l'exécution de `chaine(N)` crée N nouveaux processus, avec une structure de parenté en forme de chaîne. Autrement dit, on veut que chaque processus ait, au maximum, un seul enfant.

```
void chaine(unsigned int N)
{
    // ...
}
}
```

Multitâche et ordonnancement

Question 3 (1 pt) Lorsque le noyau fait un changement de contexte entre deux processus, il doit sauvegarder/restaurer l'état interne de certains composants matériels. Pour chaque élément ci-dessous, indiquez si le noyau doit en faire la sauvegarde/restauration (entourez V) ou bien s'il peut en «écraser» sereinement le contenu (entourez F).

- V F Cache CPU : L1, L2...
- V F Registres CPU généraux : R0, R1...
- V F Registres CPU spécialisés : PC, SP...
- V F Translation Lookaside Buffer

Question 4 (1 pt) Pour chaque stratégie d'ordonnancement ci-dessous, indiquez s'il s'agit d'ordonnancement préemptif (entourez V) ou d'ordonnancement coopératif (entourez F).

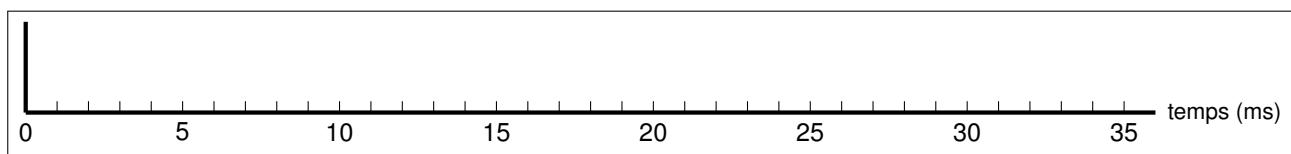
- V F First Come First Serve
- V F Shortest Job First
- V F Shortest Remaining Time First
- V F Round Robin

Question 5 (3 pts) On s'intéresse dans cette question et dans la suivante aux trois processus décrits par le tableau ci-dessous. L'activité de chaque processus consiste en une phase de calcul (CPU burst), suivie d'une phase d'entrées-sorties (IO burst) puis d'une seconde phase de calcul. À l'instant initial, le processus A est prêt à s'exécuter. Les autres instants d'arrivée sont indiqués dans le tableau. Tous les temps sont en millisecondes.

Processus	A	B	C
Instant d'arrivée	0	3	5
1 ^{ière} CPU-burst,	4	6	3
puis IO-burst	4	4	7
puis 2 ^{nde} CPU-burst	6	5	7

On suppose que la machine ne dispose que d'un unique processeur, ordonnancé selon la stratégie Round Robin avec un quantum de 5 millisecondes. On suppose également que le système ne peut traiter qu'une seule opération d'entrées-sorties à la fois ; les requêtes IO sont ordonnancées selon la stratégie Shortest Job First.

Sur une feuille de brouillon, dessinez un chronogramme indiquant la succession des tâches sur le processeur. Recopiez ensuite votre réponse au propre dans le cadre ci-dessous.



Question 6 (1 pt) Dans la question précédente, quel est le temps de séjour (*turnaround time*) de chacun des processus ? Répondez en millisecondes.

A :

B :

C :

Gestion mémoire

Question 7 (2 pts) On s'intéresse dans cette question à un système avec mémoire paginée. Les adresses virtuelles et physiques sont codées sur 12 bits, et la taille des pages est de 512 octets. Le contenu de la table des pages est indiqué ci-dessous (le signe \emptyset indique un PTE invalide).

VPN	0	1	2	3	4	5	6	7
PPN	5	4	7	0	6	1	\emptyset	2

Pour chacune des adresses virtuelles ci-dessous (données en hexadécimal), indiquez l'adresse physique correspondante (également en hexa) ou bien répondez \emptyset si la traduction est impossible. Pour les calculs en binaire, vous pouvez vous aider du tableau page 6.

VA	789	4A5	5A5	31F
PA				

Question 8 (2 pts) Complétez la fonction ci-dessous de telle sorte que l'exécution de `count(N)` renvoie le nombre de bits non-nuls dans représentation binaire de N. Par exemple, `count(6) = 2`, et `count(42) = 3`.

```

unsigned int count(unsigned int N)
{

}

```

Stockage et systèmes de fichiers

Question 9 (1 pt) Pour chaque acronyme ci-dessous, donnez sa signification en toutes lettres :

INSA	Institut National des Sciences Appliquées
FAT	
FD	
HDD	
SSD	

Question 10 (1 pt) On s'intéresse à un disque dur magnétique d'une capacité de 4 Tio. Combien faut-il de bits, au minimum, pour encoder un numéro de secteur ? On suppose que la taille des secteurs est de 512 octets. Pour les calculs en binaire, vous pouvez vous aider du tableau page 6.

Synchronisation de threads

Question 11 (2 pts) On s'intéresse à trois threads concurrents A, B et C qui se partagent quatre ressources critiques X, Y, Z, T, elles-mêmes «protégées» par des sémaphores binaires SX, SY, SZ, et ST. Avant d'entrer en section critique, chaque thread verrouille correctement les trois ressources qu'il va utiliser. Cependant, l'interaction entre plusieurs threads risque de poser des problèmes à l'exécution. Pour chaque programme ci-dessous, indiquez s'il est correct (entourez V) ou bien s'il risque de causer un interblocage (entourez F).

A : while(true){ P(SX); P(SY); P(SZ); utiliser X+Y+Z; V(SX); V(SY); V(SZ); }
 B : while(true){ P(SY); P(SZ); P(ST); utiliser Y+Z+T; V(SY); V(SZ); V(ST); }
 C : while(true){ P(SZ); P(ST); P(SX); utiliser Z+T+X; V(SZ); V(ST); V(SX); }

A : while(true){ P(SY); P(SX); P(SZ); utiliser X+Y+Z; V(SY); V(SX); V(SZ); }
 B : while(true){ P(SY); P(SZ); P(ST); utiliser Y+Z+T; V(SY); V(SZ); V(ST); }
 C : while(true){ P(SX); P(SZ); P(ST); utiliser Z+T+X; V(SX); V(SZ); V(ST); }

A : while(true){ P(SY); P(SX); P(SZ); utiliser X+Y+Z; V(SY); V(SX); V(SZ); }
 B : while(true){ P(SZ); P(SY); P(ST); utiliser Y+Z+T; V(SZ); V(SY); V(ST); }
 C : while(true){ P(SX); P(SZ); P(ST); utiliser Z+T+X; V(SX); V(SZ); V(ST); }

A : while(true){ P(SX); P(SY); P(SZ); utiliser X+Y+Z; V(SX); V(SY); V(SZ); }
 B : while(true){ P(SZ); P(SY); P(ST); utiliser Y+Z+T; V(SZ); V(SY); V(ST); }
 C : while(true){ P(SZ); P(ST); P(SX); utiliser Z+T+X; V(SZ); V(ST); V(SX); }

Question 12 (6 pts) On s'intéresse au programme illustré ci-dessous, dans lequel un nombre arbitraire de threads concurrents accèdent à une même liste (simplement) chaînée. Il y a trois sortes de threads, avec les contraintes suivantes :

- Les *lecteurs* examinent les données en lecture seule. Plusieurs lecteurs peuvent donc travailler simultanément sans causer de problème.
- Les *ajouteurs* insèrent des éléments en fin de la liste. Pour préserver la cohérence du chaînage, il est interdit à deux ajouteurs de toucher en même temps à la liste. Mais on suppose que l'insertion ne gêne pas les lecteurs.
- Les *enleveurs* suppriment des éléments n'importe où dans la liste. Pendant une telle suppression, aucun autre thread n'est autorisé à accéder aux données : ni lecteur, ni ajouteur, ni un autre enleveur.

lecteurs

```
while(true)
{
    ...
    avant_lecture()
    LECTURE()
    apres_lecture()
    ...
}
```

ajouteurs

```
while(true)
{
    ...
    avant_insertion()
    INSERTION()
    apres_insertion()
    ...
}
```

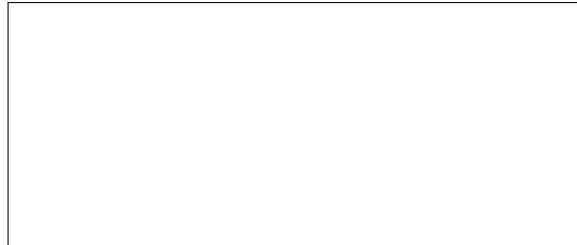
enleveurs

```
while(true)
{
    ...
    avant_suppression()
    SUPPRESSION()
    apres_suppression()
    ...
}
```

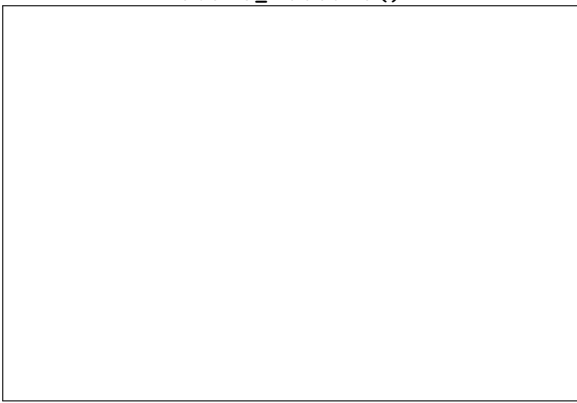
Pour les besoins de cet exercice, on ne s'intéresse pas aux détails d'implémentation de la liste chaînée (i.e. les fonctions en majuscules ci-dessus) ni à l'activité des threads en dehors des accès à la ressource critique (i.e. les points de suspension).

Votre travail consiste à implémenter, dans les cadres ci-dessous, les six fonctions `avant_qqch()` et `apres_qqch()` de façon à respecter les contraintes de synchronisation. Pour chaque sémaphore ou variable partagée que vous utiliserez, pensez à préciser sa valeur initiale.

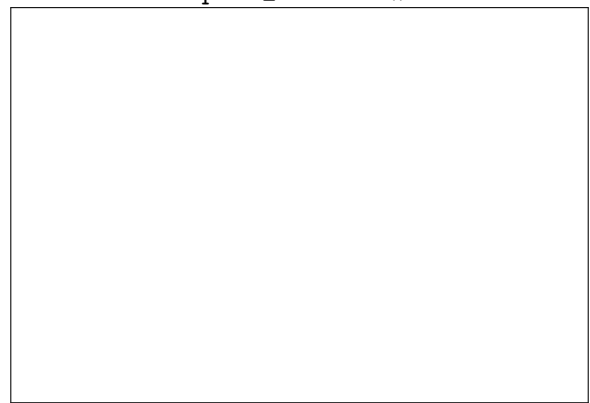
Conditions initiales



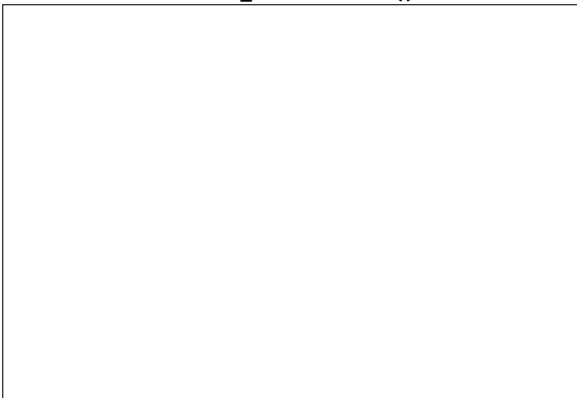
`avant_lecture()`



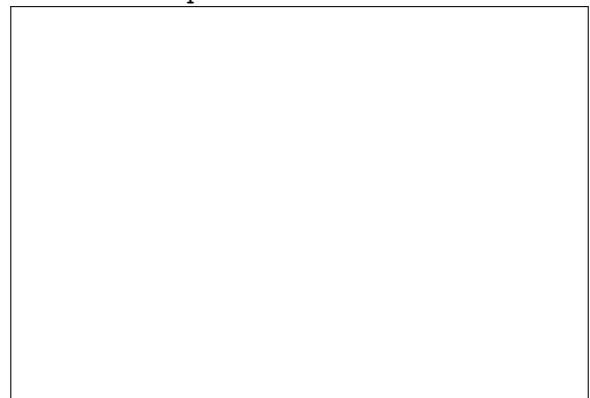
`apres_lecture()`



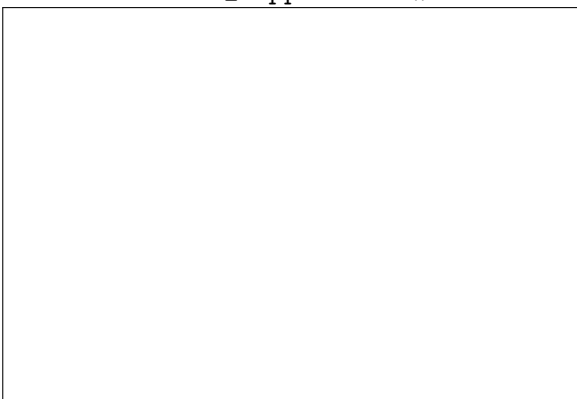
`avant_insertion()`



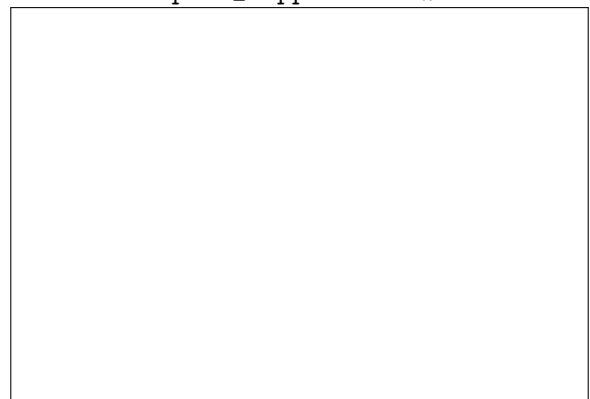
`apres_insertion()`



`avant_suppression()`



`apres_suppression()`



N'hésitez pas à ajouter des explications page suivante pour nous aider à comprendre votre approche :

Annexe : quelques puissances de 2

$2^0 = 1$	$2^{16} = 65\,536$	$2^{32} = 4\,294\,967\,296$	$2^{48} = 281\,474\,976\,710\,656$
$2^1 = 2$	$2^{17} = 131\,072$	$2^{33} = 8\,589\,934\,592$	$2^{49} = 562\,949\,953\,421\,312$
$2^2 = 4$	$2^{18} = 262\,144$	$2^{34} = 17\,179\,869\,184$	$2^{50} = 1\,125\,899\,906\,842\,624$
$2^3 = 8$	$2^{19} = 524\,288$	$2^{35} = 34\,359\,738\,368$	$2^{51} = 2\,251\,799\,813\,685\,248$
$2^4 = 16$	$2^{20} = 1\,048\,576$	$2^{36} = 68\,719\,476\,736$	$2^{52} = 4\,503\,599\,627\,370\,496$
$2^5 = 32$	$2^{21} = 2\,097\,152$	$2^{37} = 137\,438\,953\,472$	$2^{53} = 9\,007\,199\,254\,740\,992$
$2^6 = 64$	$2^{22} = 4\,194\,304$	$2^{38} = 274\,877\,906\,944$	$2^{54} = 18\,014\,398\,509\,481\,984$
$2^7 = 128$	$2^{23} = 8\,388\,608$	$2^{39} = 549\,755\,813\,888$	$2^{55} = 36\,028\,797\,018\,963\,968$
$2^8 = 256$	$2^{24} = 16\,777\,216$	$2^{40} = 1\,099\,511\,627\,776$	$2^{56} = 72\,057\,594\,037\,927\,936$
$2^9 = 512$	$2^{25} = 33\,554\,432$	$2^{41} = 2\,199\,023\,255\,552$	$2^{57} = 144\,115\,188\,075\,855\,488$
$2^{10} = 1\,024$	$2^{26} = 67\,108\,864$	$2^{42} = 4\,398\,046\,511\,104$	$2^{58} = 288\,230\,376\,151\,711\,744$
$2^{11} = 2\,048$	$2^{27} = 134\,217\,728$	$2^{43} = 8\,796\,093\,022\,208$	$2^{59} = 576\,460\,752\,303\,423\,488$
$2^{12} = 4\,096$	$2^{28} = 268\,435\,456$	$2^{44} = 17\,592\,186\,044\,416$	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
$2^{13} = 8\,192$	$2^{29} = 536\,870\,912$	$2^{45} = 35\,184\,372\,088\,832$	$2^{61} = 2\,305\,843\,009\,213\,693\,952$
$2^{14} = 16\,384$	$2^{30} = 1\,073\,741\,824$	$2^{46} = 70\,368\,744\,177\,664$	$2^{62} = 4\,611\,686\,018\,427\,387\,904$
$2^{15} = 32\,768$	$2^{31} = 2\,147\,483\,648$	$2^{47} = 140\,737\,488\,355\,328$	$2^{63} = 9\,223\,372\,036\,854\,775\,808$
			$2^{64} = 18\,446\,744\,073\,709\,551\,616$

On rappelle également que :

- 1 kio = 1024 octets,
- 1 Mio = 1024 Kio,
- 1 Gio = 1024 Mio,
- 1 Tio = 1024 Gio,
- etc. (avec dans l'ordre : Pio, Eio, Zio, Yio)

En cas de doute sur les unités de mesure, n'hésitez pas à demander des précisions.