IF-3-SYS : Systèmes d'exploitation 9 juin 2026

NOM Prénom :

Consignes

- L'examen dure 1h30. Prenez le temps de lire le sujet en entier (24 questions sur 11 pages).
- Les réponses seront à inscrire 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 le binaire et l'hexadécimal, aidez-vous des tableaux page 11.

1 Questions de cours

Dans cette première partie, chaque question est indépendante. Dans les questions avec plusieurs réponses, les erreurs sont décomptées : ne répondez pas au hasard.

Noyau, processus, appels système

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

INSA	Institut National des Sciences Appliquées
DMA	
ISR	
PCB	
POSIX	

Question 2 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

V F L'appel `exec()` permet de créer un nouveau processus qui exécute un nouveau programme.
V F Pour savoir ce que l'utilisateur tape sur le clavier, le shell doit faire un *appel système*.
V F Seul le noyau (et aucun processus) est autorisé à exécuter des instructions dites *privilégiées*.
V F Après un `fork()`, les processus parent et enfant partagent le même espace d'adressage virtuel.

Multitâche et ordonnancement

Question 3 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

V F Dans un scénario avec uniquement des tâches calculatoires de même durée, un ordonnancement Round Robin (avec un quantum nettement plus court que la durée d'une tâche) obtiendra un meilleur temps de séjour moyen qu'un ordonnancement FCFS.

V F Lorsqu'une une tâche est en section critique, l'ordonnanceur n'a pas le droit de l'interrompre pour exécuter une autre tâche.

V F Si l'ordonnanceur est de type préemptif, alors par définition il ne peut pas y avoir de problème de famine.

V F L'exécution d'un programme interactif produira typiquement un comportement que l'ordonnanceur classifiera comme «CPU-bound».

Question 4 On s'intéresse dans cette question à un système multitâche dans lequel les processus calculent en moyenne pendant une durée T entre deux requêtes d'entrées-sorties. Chaque fois qu'un processus se bloque, l'ordonnanceur bascule vers un processus prêt (on suppose qu'il y en a toujours). On note S ce temps de *context-switch* passé à exécuter du code noyau. L'ordonnanceur applique une stratégie Round Robin avec un quantum de durée Q, et on suppose que $S < Q < T$.
La question consiste à évaluer le taux d'utilisation du processeur c'est à dire la fraction du temps effectivement passée à exécuter du code applicatif. Donnez une formule pour U en fonction des différents paramètres du modèle :

Mémoire virtuelle

Question 5 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

V F Implémenter la mémoire virtuelle ne servirait à rien dans un système où la mémoire physique serait plus vaste que l'espace d'adressage virtuel.

V F Deux processus distincts qui accèdent à la même adresse physique verront la même donnée.

V F Sur certaines architectures, la taille d'une page physique et d'une page virtuelle sont différentes.

V F Avec une taille de page de 4kio, si une adresse physique est codée sur 32bits alors les 18 bits de poids fort codent le numéro de page physique.

Question 6 Dans cette question on s'intéresse au TP sur la mémoire virtuelle. Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

V F Dans l'exercice sur les nombres premiers, on a utilisé `fork()` et le copy-on-write pour obtenir plusieurs exemplaires de notre crible d'Ératosthène.

V F Dans l'exercice sur les nombres premiers, on a utilisé `mmap()` pour allouer le tableau, mais on aurait tout aussi bien pu utiliser `malloc()`.

V F Dans l'exercice sur les fichiers, on a utilisé `mmap()` pour consulter le contenu d'un fichier sans avoir à appeler `read()`, mais on a quand même dû appeler `open()` d'abord.

V F Dans l'exercice sur les fichiers, on a appelé `write()` pour afficher des données à l'écran, mais avec les bons paramètres on aurait pu utiliser `mmap()` à la place.

Allocation dynamique

Question 7 Pour chaque programme ci-dessous, entourez V s'il est correct, ou entourez F si le code est invalide et/ou risque de poser problème à l'exécution.

V F

```
int * f1 (void)
{
  int * px;

  * px = 2026;
  return px;
}
```

V F

```
int * f2 (void)
{
  int x;

  x = 2026;
  return & x;
}
```

V F

```
int * f3 (void)
{
  int * px =
    malloc(sizeof(int));
  px = 2026;
  return px;
}
```

V F

```
int * f4 (void)
{
  int * px =
    malloc(sizeof(int));
  * px = 2026;
  return & px;
}
```

Question 8 Cette question porte sur un allocateur dynamique similaire à celui implémenté en TP. Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

Lors d'une allocation :

V F découper un bloc (pour n'en allouer qu'une partie) aide à réduire la fragmentation externe.

V F découper un bloc (pour n'en allouer qu'une partie) aide à réduire la fragmentation interne.

Lors d'une désallocation :

V F fusionner plusieurs blocs voisins aide à réduire la fragmentation externe.

V F fusionner plusieurs blocs voisins aide à réduire la fragmentation interne.

Stockage et systèmes de fichiers

Question 9 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse et/ou absurde.

V F Le nombre 2^{2026} (2 à la puissance 2026) est beaucoup plus grand que le nombre 10^{600}.

V F Le nombre 10^{2026} (10 à la puissance 2026) est beaucoup plus grand que le nombre 2^{6000}.

V F Dans un système de fichiers de type Unix, les fichiers sont représentés par des inodes, alors que les répertoires ne le sont pas.

V F Quand on parle du «driver» d'un disque dur, on parle d'un composant matériel chargé de réaliser les opérations de lecture et d'écriture.

Question 10 Dans la commande `cd ..` que signifient les deux points ? Par quel(s) composant(s) matériel(s) et/ou logiciel(s) sont-ils interprétés ?

2 Exercice : Programmation concurrente

Dans cette seconde partie de l'examen (questions 13 à 24) on s'intéresse à différents algorithmes d'exclusion mutuelle. À chaque exercice, on suppose un programme de la forme ci-dessous, où la «section critique» et la «section restante» sont illustrées par des appels de fonction. Le contenu des fonctions n'est pas important pour nous : on s'intéresse seulement à la synchronisation, c'est à dire à l'implémentation des sections d'entrée (verrouillage) et de sortie (déverrouillage).

thread A

```
1 while(true) {
2     // entry section
3
4     critical();
5
6     // exit section
7
8     remainder();
9 }
```

thread B

```
1 while(true) {
2     // entry section
3
4     critical();
5
6     // exit section
7
8     remainder();
9 }
```

Hypothèses

- Mêmes conventions qu'en cours et en TD : notation pseudo-code avec exécution séquentielle concurrente, lecture et écriture mémoire atomiques pour les types simples (entier, booléen).
- L'attente en section d'entrée sera implémentée par une boucle d'attente active : on ne se préoccupera pas ici de suspendre l'exécution.
- Vous pouvez supposer que chaque section critique a toujours un temps d'exécution fini, mais que l'exécution d'une section restante peut durer indéfiniment.

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

Propriétés (rappel de cours)

- On dit qu'un algorithme garantit l'*exclusion mutuelle* s'il assure que deux threads ne peuvent pas se trouver simultanément en section critique.
- On dit qu'un algorithme garantit la *progression* s'il assure que, lorsque un ou plusieurs threads sont en section d'entrée, au moins un pourra entrer en section critique après un temps fini.
- On dit qu'un algorithme garantit l'*équité* s'il assure que, une fois qu'un thread est en section d'entrée, il existe une borne sur le nombre de fois où d'autres threads peuvent entrer en section critique avant lui.

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

Question 11 Si un algorithme d'exclusion mutuelle ne garantit pas la progression, quel problème peut se produire ? Donnez le nom de ce problème, et un exemple illustratif.

Question 12 Si un algorithme d'exclusion mutuelle ne garantit pas l'équité, quel problème peut se produire ? Donnez le nom de ce problème, et un exemple illustratif.

Version n° 1

Dans le code ci-dessous, l'état du verrou (libre ou verrouillé) est implémenté par une variable booléenne. Ainsi, la section d'entrée consiste simplement à attendre que le verrou soit libre, puis à le marquer comme verrouillé.

Conditions initiales

```
bool locked = false;
```

thread A

```
1  while(true) {
2      while(locked)
3      { /* busy wait */ }
4      locked = true ;
5
6      critical();
7
8      locked = false;
9
10     remainder();
11 }
```

thread B

```
1  while(true) {
2      while(locked)
3      { /* busy wait */ }
4      locked = true;
5
6      critical();
7
8      locked = false;
9
10     remainder();
11 }
```

Question 13 Est-ce que le code de la version n° 1 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution [1] menant à un problème.

1. Un trace d'exécution est une séquence de la forme «A1, B1, A2, etc» qui se lit «A exécute sa ligne n° 1, puis B exécute sa ligne n° 1, puis A exécute sa ligne n° 2, etc»

Question 14 Toujours sur la version nº 1, est-ce que cette implémentation garantit la progression ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 15 Toujours sur la version nº 1, est-ce que cette implémentation garantit l'équité ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Version nº 2

On s'intéresse maintenant à une seconde implémentation illustrée ci-dessous, dans laquelle les threads utilisent une variable entière pour savoir «à qui le tour». Ainsi la section d'entrée consiste maintenant à attendre que ce soit mon tour :

Conditions initiales

```
int turn = 0;
```

thread A

```
1  while(true) {
2      while(turn==1)
3      { /* busy wait */ }
4
5      critical();
6
7      turn = 1;
8
9      remainder();
10 }
```

thread B

```
1  while(true) {
2      while(turn==0)
3      { /* busy wait */ }
4
5      critical();
6
7      turn = 0;
8
9      remainder();
10 }
```

Question 16 Est-ce que la version n° 2 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 17 Est-ce que la version n° 2 garantit la progression ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 18 Est-ce que la version n° 2 garantit l'équité ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Version n° 3

Dans cette troisième version, on se donne une paire de drapeaux booléens pour dire «je veux entrer». Par courtoisie, chaque thread attend que l'autre ait quitté la section critique (et baissé son drapeau) avant d'entrer à son tour.

Conditions initiales

```
bool flag[2] = {false, false};
```

thread A

```
1  while(true) {
2      flag[0] = true;
3      while(flag[1])
4      { /* busy wait */ }
5
6      critical();
7
8      flag[0] = false;
9
10     remainder();
11 }
```

thread B

```
1  while(true) {
2      flag[1] = true;
3      while(flag[0])
4      { /* busy wait */ }
5
6      critical();
7
8      flag[1] = false;
9
10     remainder();
11 }
```

Question 19 Est-ce que la version n° 3 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 20 Est-ce que la version n° 3 garantit la progression ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 21 Est-ce que la version n° 3 garantit l'équité ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Version n° 4

Dans cette dernière implémentation, on combine les deux versions précédentes, en utilisant à la fois une paire de booléens et une variable entière.

Conditions initiales

```
int turn = 0;
bool flag[2] = {false, false};
```

thread A

```
1  while(true) {
2      flag[0] = true;
3      turn = 1;
4      while(flag[1] && turn==1)
5      { /* busy wait */ }
6
7      critical();
8
9      flag[0] = false;
10
11     remainder();
12 }
```

thread B

```
1  while(true) {
2      flag[1] = true;
3      turn = 0;
4      while(flag[0] && turn==0)
5      { /* busy wait */ }
6
7      critical();
8
9      flag[1] = false;
10
11     remainder();
12 }
```

Question 22 Est-ce que la version n° 4 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 23 Est-ce que la version n° 4 garantit la progression ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Question 24 Est-ce que la version n° 4 garantit l'équité ? Si votre réponse est «oui», donnez une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

Annexe : aide pour les calculs en binaire

Les premiers nombres entiers, notés en décimal, en hexadécimal, et en binaire :

Dec	Hex	Bin
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100

Dec	Hex	Bin
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001

Dec	Hex	Bin
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110

Dec	Hex	Bin
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011

Les premières puissances de 2 :

$2^0 = 1$	$2^{16} = 65\,536$	$2^{32} = 4\,294\,967\,296$	$2^{48} = 281\,474\,976\,710\,656$
$2^1 = 2$	$2^{17} = 131\,072$	$2^{33} = 8\,589\,934\,592$	$2^{49} = 562\,949\,953\,421\,312$
$2^2 = 4$	$2^{18} = 262\,144$	$2^{34} = 17\,179\,869\,184$	$2^{50} = 1\,125\,899\,906\,842\,624$
$2^3 = 8$	$2^{19} = 524\,288$	$2^{35} = 34\,359\,738\,368$	$2^{51} = 2\,251\,799\,813\,685\,248$
$2^4 = 16$	$2^{20} = 1\,048\,576$	$2^{36} = 68\,719\,476\,736$	$2^{52} = 4\,503\,599\,627\,370\,496$
$2^5 = 32$	$2^{21} = 2\,097\,152$	$2^{37} = 137\,438\,953\,472$	$2^{53} = 9\,007\,199\,254\,740\,992$
$2^6 = 64$	$2^{22} = 4\,194\,304$	$2^{38} = 274\,877\,906\,944$	$2^{54} = 18\,014\,398\,509\,481\,984$
$2^7 = 128$	$2^{23} = 8\,388\,608$	$2^{39} = 549\,755\,813\,888$	$2^{55} = 36\,028\,797\,018\,963\,968$
$2^8 = 256$	$2^{24} = 16\,777\,216$	$2^{40} = 1\,099\,511\,627\,776$	$2^{56} = 72\,057\,594\,037\,927\,936$
$2^9 = 512$	$2^{25} = 33\,554\,432$	$2^{41} = 2\,199\,023\,255\,552$	$2^{57} = 144\,115\,188\,075\,855\,488$
$2^{10} = 1024$	$2^{26} = 67\,108\,864$	$2^{42} = 4\,398\,046\,511\,104$	$2^{58} = 288\,230\,376\,151\,711\,744$
$2^{11} = 2048$	$2^{27} = 134\,217\,728$	$2^{43} = 8\,796\,093\,022\,208$	$2^{59} = 576\,460\,752\,303\,423\,488$
$2^{12} = 4\,096$	$2^{28} = 268\,435\,456$	$2^{44} = 17\,592\,186\,044\,416$	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
$2^{13} = 8\,192$	$2^{29} = 536\,870\,912$	$2^{45} = 35\,184\,372\,088\,832$	$2^{61} = 2\,305\,843\,009\,213\,693\,952$
$2^{14} = 16\,384$	$2^{30} = 1\,073\,741\,824$	$2^{46} = 70\,368\,744\,177\,664$	$2^{62} = 4\,611\,686\,018\,427\,387\,904$
$2^{15} = 32\,768$	$2^{31} = 2\,147\,483\,648$	$2^{47} = 140\,737\,488\,355\,328$	$2^{63} = 9\,223\,372\,036\,854\,775\,808$
			$2^{64} = 18\,446\,744\,073\,709\,551\,616$

On rappelle également que :

— 1 kio (un kilooctet) = 1024 octets,
— 1 Mio (un mégaoctet) = 1024 kio,
— 1 Gio (un gigaoctet)= 1024 Mio,
— 1 Tio (un téraoctet)= 1024 Gio,
— et ainsi de suite, avec dans l'ordre : Pio (pétaoctet), Eio (exaoctet), Zio (zettaoctet), Yio (yottaoctet)

En cas de doute sur les tailles ou sur le binaire, n'hésitez pas à demander des précisions.