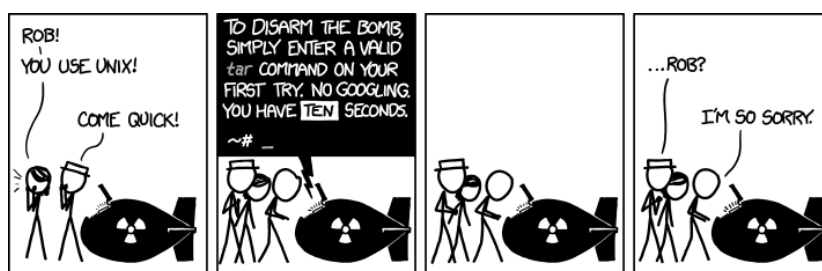


NOM Prénom :

Consignes

- Durée : 1h30. Lisez le sujet en entier (20 questions sur 9 pages) avant de commencer.
- Écrivez lisiblement et surtout sans ratures. Utilisez un brouillon (vraiment).
- Les réponses seront à inscrire sur le sujet. Commencez par écrire votre nom ci-dessus.
- Documents et appareils interdits, sauf une feuille A4 recto-verso manuscrite.
- Pour les calculs en binaire, vous pouvez vous aider des tableaux donnés en page 9.
- Sauf mention contraire, les questions sont indépendantes les unes des autres.
- Dans les questions vrai/faux, les erreurs sont décomptées : ne répondez pas au hasard.

Usage du shell



Question 1 Donnez une ligne de commande shell permettant d'extraire dans le répertoire courant tous les fichiers et dossiers contenus dans une archive SYS-TP.tgz

Question 2 Donnez une séquence de lignes de commandes shell permettant de créer l'arborescence illustrée ci-dessous. À l'instant initial, le dossier courant est «racine», supposé vide. Dans votre réponse, aucune ligne ne peut comporter un slash « / » ni utiliser la commande cd.

```

racine
├── dossier1
│   ├── fichierA
│   └── dossier2
│       └── fichierB
└── dossier3
    ├── dossier4
    └── fichierC
  
```

1 Noyau, processus, threads

Question 3 On s'intéresse à un processus parent qui vient de créer un ou plusieurs enfants, et on suppose que, pour l'instant, ni parent ni enfant(s) n'ont invoqué d'autre appel système. Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.

- ☐ V ☐ F Le parent exécute le même programme que ses enfants.
- ☐ V ☐ F Le parent partage son espace d'adressage avec ses enfants.
- ☐ V ☐ F Le parent connaît les numéros (pid) de ses enfants.
- ☐ V ☐ F Si le parent meurt, ses enfants restent en vie.

Question 4 Sous Linux, comment s'appelle l'utilitaire qui permet d'afficher le contenu d'un fichier exécutable, en particulier les instructions machine ?

Question 5 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte ou entourez F si elle est incorrecte et/ou absurde.

- ☐ V ☐ F Les différents threads d'un même processus ont la même pile d'exécution.
- ☐ V ☐ F Les différents threads d'un même processus ont la même table de pagination.
- ☐ V ☐ F Les différents threads d'un même processus ont le même CPU.
- ☐ V ☐ F Les différents threads d'un même processus ont les mêmes sections critiques.

2 Multitâche et ordonnancement

Question 6 Dans le cadre ci-dessous, dessinez le diagramme états/transitions d'un PCB dans l'ordonnanceur. Indiquez à l'aide d'un «P» la (ou les) transition(s) correspondant à la *préemption*.

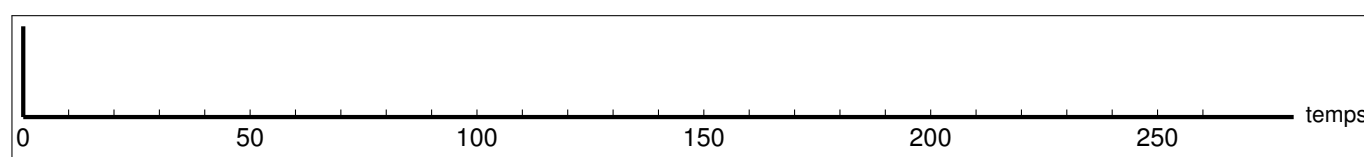
Question 7 Que signifie le terme de «PCB» ?

Question 8 On suppose dans cette question un ordonnanceur avec trois niveaux de priorité. La priorité (haute, moyenne, ou basse) de chaque processus est fixée lorsque celui-ci est créé. Entre des niveaux distincts, l'ordonnancement est strictement préemptif. Parmi les processus d'un même niveau, l'ordonnancement est circulaire (Round Robin) avec un quantum de 20 millisecondes. La durée des commutations de contexte est supposée négligeable.

On s'intéresse aux tâches ci-dessous (les temps sont indiqués en millisecondes) :

	A	B	C	D	E	F	G
durée d'exécution	70	40	40	10	20	20	10
instant d'arrivée	0	0	9	9	9	19	19
priorité	moy	haute	basse	moy	haute	basse	moy

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



3 Mémoire virtuelle

Question 9 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 9.

VA	789	4A5	5A5	31F
PA				

Question 10 On s'intéresse dans cette question à un processus qui vient de rencontrer un défaut de page. Dans quel état (au sens de la question 6) est ce processus vis-à-vis de l'ordonnanceur ?

4 Allocation dynamique

Question 11 Pour chaque affirmation ci-dessous, entourez V si elle est correcte, entourez F si elle est fausse et/ou absurde.

- ☐ V ☐ F L'allocation statique consiste à déléguer au noyau le placement en mémoire des données.
- ☐ V ☐ F L'allocation statique consiste à déléguer au noyau le placement en mémoire des instructions.
- ☐ V ☐ F La pile d'exécution contient les instructions que le programme doit exécuter.
- ☐ V ☐ F La pile d'exécution contient les variables locales des fonctions du programme.

Question 12 On s'intéresse dans cette question à un gestionnaire de mémoire dynamique appliquant la stratégie worst-fit. À l'instant initial, la *freelist* contient les blocs suivants, chaînés dans cet ordre :

A (9 kio) → B (4 kio) → C (20 kio) → D (18 kio) → E (7 kio)

On suppose que l'application demande successivement trois allocations pour des tableaux qu'on appellera X, Y, et Z, dont les tailles respectives sont 10 kio, 12 kio, et 9 kio.

Le tableau X sera alloué dans le bloc

Le tableau Y sera alloué dans le bloc

Le tableau Z sera alloué dans le bloc

5 Programmation concurrente : l'algorithme de Peterson

Cette dernière partie de l'examen (questions 13 à 20) porte sur des algorithmes d'exclusion mutuelle. Dans chaque exercice, on s'intéresse à un programme de la forme ci-dessous, où la «section critique» est représentée par un appel de fonction `critical()`, et la «section restante» est illustrée par un appel de fonction `remainder()`.

Le contenu de ces fonctions n'est pas important pour nous ; on s'intéressera uniquement à la synchronisation, c'est à dire à l'implémentation de la «section d'entrée» et de la «section de sortie».

thread A	thread B
<pre> 1 while(true) { 2 // entry section 3 4 critical(); 5 6 // exit section 7 8 remainder(); 9 }</pre>	<pre> 1 while(true) { 2 // entry section 3 4 critical(); 5 6 // exit section 7 8 remainder(); 9 }</pre>

Hypothèses

- On fait l'hypothèse que la section critique a toujours un temps d'exécution fini, mais que la section restante peut durer indéfiniment.
- L'attente à l'entrée sera implémentée par une boucle d'attente active, aka *spinlock*. On ne se préoccupe pas ici de la suspension des threads.
- Mêmes conventions qu'en cours et en TD : notation pseudo-code avec exécution séquentielle, lecture et écriture mémoire atomiques pour les types simples (entier, booléen) mais pas plus.

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

Propriétés

- On dit qu'un algorithme satisfait la propriété d'**exclusion mutuelle** lorsqu'il garantit que deux threads ne peuvent pas se trouver simultanément à l'intérieur de (c.à.d. en train d'exécuter le code de) leurs sections critiques respectives.
- On dit qu'un algorithme satisfait la propriété de **progression** lorsqu'il garantit l'absence d'interblocage (deux threads ne doivent pas s'attendre mutuellement) et l'absence d'interférence (un thread qui s'arrête en section restante ne doit pas avoir d'effet sur les autres).

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-en une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution¹ menant à un problème.

Question 14 Toujours sur la version n° 1, est-ce que cette implémentation garantit la progression ? Si votre réponse est «oui», donnez-en une justification en quelques phrases. Si votre réponse est «non», donnez une trace d'exécution menant à un problème.

1. Formulez votre réponse sous la forme «A1, B1, A2, etc» pour dire «A exécute sa ligne n° 1, puis B exécute sa ligne n° 1, puis A exécute sa ligne n° 2, etc»

Version n° 2 On s'intéresse maintenant à une seconde implémentation illustrée ci-dessous, dans laquelle les threads utilisent une variable entière pour dire «à qui le tour». La section d'entrée consiste maintenant à attendre que ce soit mon tour avant de rentrer :

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

Question 16 Est-ce que la version n° 2 garantit la progression ? Si votre réponse est «oui», donnez-en 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 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 17 Est-ce que la version n° 3 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez-en 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° 3 garantit la progression ? Si votre réponse est «oui», donnez-en 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 version 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 19 Est-ce que la version n° 4 garantit l'exclusion mutuelle ? Si votre réponse est «oui», donnez-en 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° 4 garantit la progression ? Si votre réponse est «oui», donnez-en 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, hexadécimal, et 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, notées en décimal :

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

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 ces unités, n'hésitez pas à demander des précisions.