IF-3-SYS : Systèmes d'exploitation 25 mai 2021

NOM Prénom :

Consignes

- L'examen dure 1h30. Prenez le temps de lire le sujet en entier (17 questions sur 8 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).**
- Appareils électroniques et documents interdits, sauf une feuille A4 recto-verso manuscrite.
- Pour les calculs en binaire, aidez-vous des tableaux page 8

1 Questions de cours

Dans cette première partie, les questions sont indépendantes les unes des autres. 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 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte ou entourez F si elle est incorrecte ou absurde.

V F Un même programme peut être exécuté par plusieurs processus «en même temps».
V F Un même processus peut exécuter plusieurs programmes «en même temps».
V F Un même processus peut contenir plusieurs threads «en même temps».
V F Un même thread peut contenir plusieurs processus «en même temps».

Question 2 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte ou entourez F si elle est incorrecte ou absurde.

V F Les systèmes d'exploitation Android et Ubuntu sont tous les deux basés sur le même noyau Linux, donc ils offrent les mêmes appels système.
V F Pour des raisons de performance, le shell est en général implémenté comme un composant du noyau.
V F Si tous les processus du système sont suspendus, alors il faut rebooter la machine car aucun processus ne pourra plus être «réveillé».
V F Un changement de contexte est forcément causé par une trappe ou une interruption matérielle.

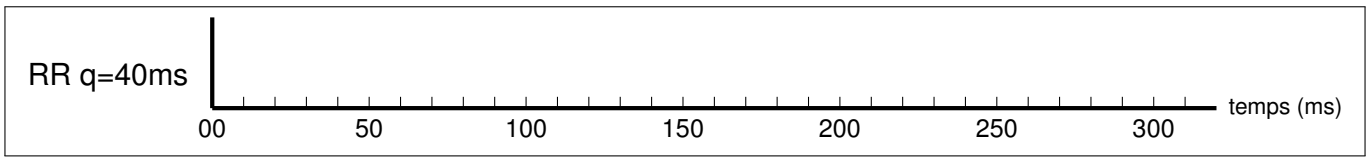
Multitâche et ordonnancement

Question 3 On s'intéresse dans cette question à l'ordonnancement de deux processus A et B sur un même processeur.

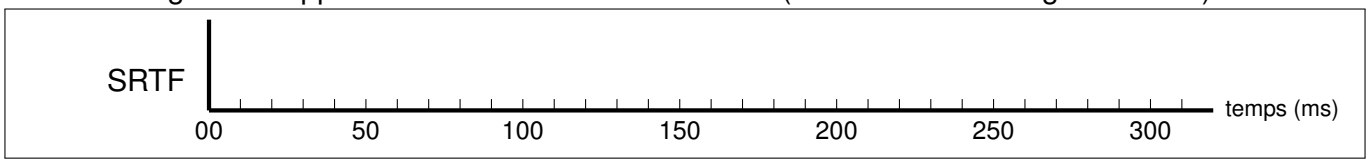
Le processus A doit calculer (c'est à dire s'exécuter sur le CPU) pendant 80ms en tout, mais après chaque 20ms de calcul il fait une requête d'entrées-sorties qui dure 10ms. Il lui faudra donc au minimum 120ms pour se terminer. On suppose que A n'a rien à faire après sa dernière requête d'entrées-sorties.

Le processus B doit calculer durant 200ms et ne fait pas d'entrées-sorties. On suppose que B devient prêt juste après A.

Sur une feuille de brouillon, dessinez un chronogramme indiquant la succession des tâches sur le CPU, en supposant un ordonnancement Round Robin avec un quantum de 40 millisecondes. Recopiez ensuite votre réponse au propre dans le cadre ci-dessous.



Même consigne en supposant un ordonnancement SRTF (Shortest Remaining Time First).



Question 4 Dans la question précédente, la stratégie SRTF donne de bien meilleurs résultats que Round Robin. Pourtant, dans les OS généralistes (Linux, Windows...) les ordonnanceurs sont toujours basés sur la stratégie Round Robin. Justifiez en quelques phrases ce paradoxe apparent.

Gestion mémoire

Question 5 Les affirmations ci-dessous se rapportent à la notion de «pile d'exécution» dans un processus. Pour chaque proposition, entourez V si elle est correcte ou entourez F si elle est incorrecte ou absurde.

V F La pile est typiquement utilisée pour le stockage des instructions du programme.
V F La pile est typiquement utilisée pour le stockage des variables locales.
V F La pile est typiquement utilisée pour permettre aux différents threads de partager leurs variables.
V F La pile est typiquement utilisée pour offrir une forme restreinte d'allocation dynamique.

Question 6 Pour allouer dynamiquement de la mémoire dans un programme C, on utilise en général `malloc()` et `free()`. La fonction `malloc()` prend en argument une taille, puis réserve une zone mémoire de la bonne taille, et renvoie son adresse. Symétriquement, la fonction `free()` prend en argument une adresse de zone à libérer, mais pas d'information de taille. Dans ces conditions, comment l'allocateur peut-il savoir la taille de la zone à libérer ? Ne serait-il pas plus simple pour tout le monde d'ajouter à `free()` un paramètre de taille ?

Question 7 On s'intéresse dans cette question aux différentes stratégies d'allocation dynamique : first-fit, best-fit et worst-fit. La *freelist* de l'allocateur contient initialement les blocs libres suivants, chaînés dans cet ordre : bloc A (100 octets), bloc B (300 octets) puis bloc C (200 octets). On suppose ici qu'un bloc de taille N peut servir à allouer une zone de taille $T \leqslant N$, laissant le cas échéant un bloc libre de taille $N-T$ (ce qui revient à négliger l'espace occupé par les méta-données).

— L'application demande une zone de taille 150, que l'allocateur prend dans le bloc B. Elle demande ensuite 110, que l'allocateur prend aussi dans B, puis 40, qu'il prend dans le bloc A. Quelle stratégie est-il en train d'appliquer ? Entourez l'un des chiffres ci-dessous.

1 first-fit 2 best-fit 3 worst-fit

— Même question sur un autre scénario : L'application demande 150 octets, que l'allocateur prend dans le bloc C. Puis elle demande 50 octets, que l'allocateur prend aussi dans C. Quelle stratégie est-il en train d'appliquer ? Entourez l'un des chiffres ci-dessous.

1 first-fit 2 best-fit 3 worst-fit

— L'application demande 100 octets, que l'allocateur prend dans le bloc A. Puis elle demande 200 octets, que l'allocateur prend dans B. Puis 50, qu'il prend aussi dans B. Quelle stratégie est-il en train d'appliquer ? Entourez l'un des chiffres ci-dessous.

1 first-fit 2 best-fit 3 worst-fit

— L'application demande 80 octets, que l'allocateur prend dans le bloc A. Puis elle demande 20 octets, que l'allocateur prend aussi dans A. Puis 50, qu'il prend dans le bloc B. Quelle stratégie est-il en train d'appliquer ? Entourez l'un des chiffres ci-dessous.

1 first-fit 2 best-fit 3 worst-fit

Concurrence et synchronisation

Question 8 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte ou entourez F si elle est incorrecte ou absurde.

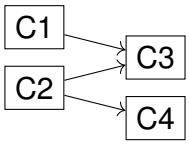
V F Une fois qu'un thread a verrouillé un mutex pour rentrer en section critique, il est interdit à l'ordonnanceur de lui retirer le processeur.

V F Un «sémaphore binaire» est essentiellement équivalent à un verrou mutex.

V F On dit qu'une opération est «atomique» lorsqu'elle est réalisée par le noyau et non pas par un processus ordinaire.

V F Les problèmes de concurrence posés par le multitâche sont essentiellement les mêmes, qu'on s'intéresse à des threads ou à des processus.

Question 9 On s'intéresse à un programme composé de quatre threads concurrents T1 à T4, où chaque thread Ti exécute un unique calcul Ci.
On veut que l'exécution de ce programme respecte les contraintes de précédence représentées dans le schéma [1] ci-contre. Complétez le code ci-dessous avec des synchronisations pour réaliser ce comportement. N'oubliez pas d'indiquer la valeur initiale de chaque sémaphore.



Conditions initiales

```
```

Thread T1

```
C1;
```

Thread T2

```
C2;
```

Thread T3

```
C3;
```

Thread T4

```
C4;
```

Systèmes de fichiers

Question 10 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte, ou entourez F si elle est incorrecte ou absurde.

V F Dans un système basé sur les *inodes*, il n'y a pas de limite supérieure pour la taille d'un fichier, sauf bien sûr la capacité de stockage du disque.

V F Dans l'appel système `read()`, le deuxième argument `void* buf` est un pointeur qu'on passe au noyau pour lui indiquer où il doit *écrire*.

V F Dans un système basé sur les *inodes*, les fichiers sont représentés par des *inodes* mais pas les répertoires.

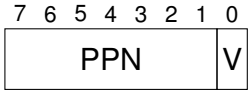
V F Dans un système basé sur une FAT (*File Allocation Table*) il y a une entrée dans la FAT pour chaque processus.

1. Rappel : une flèche entre Ci et Cj indique que Ci doit *précéder* Cj, c'est à dire que Ci doit être terminé avant que Cj ne puisse commencer.

2 Problème : implémentation de la table de pages

Dans cette partie, on va s'intéresser à un système «jouet» avec mémoire virtuelle paginée. L'adressage se fait par octet, et la taille des pages est de 16 octets. Le CPU manipule des adresses virtuelles codées sur 16 bits mais par convention, les adresses situées au-dessus d'un certain seuil sont interdites. [2]

Chaque processus est doté d'un espace d'adressage virtuel de 256 octets. [3] La table des pages est implémentée comme un simple tableau contigu, indexé par le numéro de page virtuelle, et stocké en RAM. Le noyau indique à la MMU l'adresse (physique) de la table de page courante grâce à un registre qu'on appellera le *Page Table Pointer*. Chaque entrée de la table de page (PTE) occupe exactement un octet, dont le format est illustré ci-dessous. Les sept bits de poids fort encodent le numéro de page physique (PPN) et le bit de poids faible est le *valid bit*.



Remarque : pour les calculs en binaire, et la notation hexadécimale, aidez-vous des tableaux page 8.

Question 11 Combien d'espace en RAM une table de pages occupe-t-elle ? Répondez en octets.

Question 12 Si on néglige la part de l'espace d'adressage physique dédiée aux entrées-sorties, quelle quantité de RAM peut-on installer sur la machine au maximum ? Répondez en octets.

Question 13 On suppose que le processeur n'a pas de cache, et que la MMU n'a pas de TLB. Combien d'accès mémoire sont nécessaires sur cette architecture pour réaliser une opération de lecture ? Même question pour une écriture ?

1 lecture = accès 1 écriture = accès

2. Cette restriction est plausible : par exemple, les architectures x86-64 et ARMv8 manipulent des adresses sur 64 bits, mais ne supportent pas l'adressage au-delà de 48 bits. Autrement dit, seuls les six octets de poids faible d'une adresse virtuelle sont significatifs, et les deux octets de poids fort doivent toujours être zéro.

3. Autrement dit, seul l'octet de poids faible d'une adresse virtuelle est significatif, l'autre l'octet doit toujours être zéro.

Question 14 On suppose que la mémoire physique commence par le contenu illustré ci-dessous. Chaque ligne représente 8 octets notés en hexadécimal. La colonne de gauche indique les adresses, également notées en hexadécimal.

```
0000:  35 fa 12 39 8d 1a c6 46
0008:  94 c9 c2 f2 05 7d ab 5d
0010:  3a 8e ff 78 8c ff 0f c2
0018:  78 3e 66 f7 ef a3 71 a9
0020:  b9 66 54 78 d0 de 44 ff
0028:  87 ff 57 d0 2f eb f6 df
0030:  a9 b4 bf c2 ff d7 fe 03
0038:  fd 09 10 87 39 7f 6e 05
0040:  21 60 a7 c5 7e 8a 5d d3
0048:  e9 c9 59 91 3c 58 74 54
0050:  3a 01 8c 1a 40 03 0f 01
0058:  3e 1b 43 21 04 a3 02 71
0060:  ...
```

Le registre *Page Table Pointer* de la MMU pointe à l'adresse `0x30`. Pour chacune des adresses virtuelles VA ci-dessous, indiquez l'adresse physique PA correspondante (en hexadécimal), ou «invalide» s'il s'agit d'une adresse virtuelle invalide.

VA	0x65	0x0c	0x26	0xa8
PA				

Question 15 Afin de pouvoir manipuler les tables de pages des processus, le noyau a souvent besoin de simuler en logiciel le fonctionnement de la MMU. La fonction ci-dessous prend comme paramètres un pointeur vers une table de pages [4] et une adresse virtuelle à traduire. [5] Elle doit retourner l'adresse physique correspondante, ou retourner la constante `INVALID_ADDR` si VA est invalide. Complétez le corps de la fonction pour implémenter cette traduction.

```
void* translate_virtual_to_physical( uint8_t* PT, void* VA )
{
   int VPN =









}
```

4. Le paramètre PT est déclaré ici comme `uint8_t*`, c'est à dire «pointeur vers un *entier non signé sur 8 bits*», autrement dit un pointeur d'octet. On aurait aussi bien pu écrire `char*` qui est synonyme, mais moralement les entrées de la table de pages ne sont pas des «caractères».

5. Le paramètre VA et le type de retour sont déclarés `void*`, c'est à dire «pointeur générique». Si ça vous chagrine trop, considérez que ce sont des `uint8_t*` (ou des `char*` cf note précédente) et la consigne reste la même.

Question 16 L'architecture étudiée dans les questions précédentes permet-elle de partager de la mémoire entre plusieurs processus ? Si vous répondez «oui», expliquez les opérations nécessaires. Si vous répondez «non», expliquez ce qui manque, et comment il faudrait modifier CPU/MMU/noyau pour supporter cette fonctionnalité.

Question 17 L'architecture étudiée dans les questions précédentes permet-elle de faire du *copy-on-write* ? Si vous répondez «oui», expliquez les opérations nécessaires. Si vous répondez «non», expliquez ce qui manque, et comment il faudrait modifier CPU/MMU/noyau pour supporter cette fonctionnalité.

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 = 1024 octets,
- 1 Mio = 1024 Kio,
- 1 Gio = 1024 Mio,
- 1 Tio = 1024 Gio,
- et ainsi de suite, avec dans l'ordre : Pio, Eio, Zio, Yio

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