IFA-3-SYS : Systèmes d'exploitation 17 mars 2021

NOM Prénom :

Consignes

- L'examen dure 1h30. Ce sujet comporte 15 questions sur 7 pages.
- Les réponses sont 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 autorisés, mais pas de communication entre vous.
- Pour les calculs en binaire, aidez-vous du tableau page 7.
- Dans les questions vrai/faux, les erreurs sont décomptées : ne répondez pas au hasard.

1 Questions de cours

Dans cette première partie, les questions sont indépendantes les unes des autres.

Question 1 Pour chaque affirmation ci-dessous, indiquez si elle est vraie (entourez V) ou bien si elle est fausse/absurde (entourez F).

V F Au démarrage de la machine, le CPU est par défaut en «mode superviseur».

V F Dans un système multitâche, chaque processus dispose d'un noyau distinct.

V F L'appel système `exit()` permet de renvoyer au shell une valeur arbitraire.

V F Un processus qui crée plusieurs threads aura plusieurs PID.

Question 2 On s'intéresse aux trois programmes ci-dessous, compilés sous forme de fichiers exécutables `a`, `b`, et `c`, tous dans le répertoire courant.

a.c

```
main()
{
   int r=fork();
   if(r == 0) {
      exec("./b");
   }
   else {
      wait(NULL);
   }
   printf("A");
}
```

b.c

```
main()
{
   exec("./c");

   fork();

   printf("B");
}
```

c.c

```
main()
{
   int r = fork();
   if(r == 0) {
      printf("Z");
   }
   else {
      printf("C");
   }
}
```

Sur la ligne de commande, l'utilisateur tape `./a`. Indiquez combien de fois chaque lettre est affichée au total.

A	B	C	Z

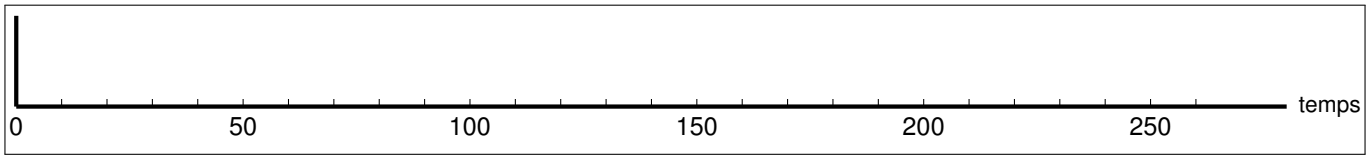
Question 3 Dans un système multitâche, lorsque l'ordonnanceur choisit un processus à exécuter, celui-ci «reprend où il en était» sans recommencer son exécution depuis le début. Mais comment fait-il pour savoir «où il en était» ?

Question 4 On suppose dans cette question un système mono-processeur avec un ordonnanceur multi-niveaux gérant 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

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 5 Décrivez une situation du monde réel (i.e. en dehors du monde de l'informatique) qui implique une forme d'ordonnancement, avec les restrictions suivantes :

— il ne s'agit pas d'ordonnancement «a priori», c'est à dire que l'ensemble des «tâches» n'est pas connu à l'avance. Au contraire, elles «arrivent» au fur et à mesure du temps.

— la stratégie d'ordonnancement n'est *pas* la stratégie *First come, first served*.

S'agit-il d'ordonnancement préemptif, ou coopératif ? Quels sont les critères de choix utilisés ?

Question 6 On considère dans cette question un système avec mémoire virtuelle dont les caractéristiques sont les suivantes. Les adresses virtuelles sont encodées sur 24 bits, et la taille de page est de 512 octets. Combien de bits sont alors nécessaires pour encoder un numéro de page virtuelle ? Pour les calculs en binaire, aidez-vous du tableau page 7.

Question 7 Toujours en supposant une taille de page de 512 octets, on s'intéresse à un processus avec la table de pages ci-dessous (où le symbole ∅ représente un PTE invalide).

VPN	0	1	2	3	4	5	6	7
PPN	∅	3	1	∅	5	1	2	∅

Quelle est l'adresse physique correspondant à l'adresse virtuelle 1050 ? La question est en décimal, répondez également en décimal.

Question 8 Dans le contexte de la mémoire virtuelle, qu'est-ce que la technique dite de Copy-On-Write ? Décrivez en quelques mots sur quels mécanismes (matériels et logiciels) elle repose, et aussi à quoi elle est utile.

Question 9 L'outil `objdump` permet d'examiner le contenu d'un fichier exécutable. En particulier, il permet de consulter le détail de :

V F la section `.text`
V F la section `.data`
V F la section `.stack`
V F la section `.heap`

Question 10 On veut écrire une fonction permettant d'échanger les valeurs de deux variables entières. Complétez le code ci-dessous afin que le programme affiche «x=17, y=25». C'est à vous de choisir la signature de la fonction `swap()`, et donc d'appeler correctement cette fonction depuis `main()`.

```
main()
{
  int x=25;
  int y=17;

  swap(

  printf("x=%d, y=%d\n", x, y);
}
```

```
void swap(
{




}
```

Question 11 En général, les primitives de synchronisation comme les verrous mutex et les sémaphores sont implémentés non pas comme des fonctions userland ordinaires, mais comme des appels système. Mais pour quelle(s) raison(s) ? Pour chaque explication ci-dessous, entourez V si elle vous parait correcte, ou entourez F si elle est fausse/absurde.

V F Car du code noyau s'exécutera avec de meilleures performances que du code userland.

V F Car le noyau peut suspendre l'exécution des processus, ce qui améliore les performances.

V F Car une synchronisation nécessite de désactiver les interruptions, ce qui est réservé au noyau.

V F Car une synchronisation nécessite des instructions atomiques, qui sont réservées au noyau.

2 Problème : la barrière de rendez-vous synchrone

Dans cette seconde partie du sujet, on va étudier plusieurs variantes d'une primitive déjà abordée en TD, la barrière de synchronisation. Rappel : lorsqu'un thread atteint une barrière, il doit attendre que tous les autres threads du programme aient également atteint leur barrière. Une fois que tout le monde est arrivé, les threads peuvent «franchir la barrière» et poursuivre leur exécution.

Question 12 On s'intéresse dans un premier temps à un programme avec seulement deux threads :

```
threadA()
{
   C1();
   barriereA();
   C2();
}
```

```
threadB()
{
   C3();
   barriereB();
   C4();
}
```

Les fonctions C1..C4 sont supposées purement calculatoires, c'est à dire qu'elles ne comportent que des instructions ordinaires. Votre travail consiste à implémenter les fonctions `barriereA` et `barriereB` de façon à garantir que C1 et C3 seront terminées avant que C2 et C4 ne soient exécutées.
Vous pouvez utiliser un ou plusieurs sémaphores, et/ou des instructions et variables ordinaires. Dans tous les cas, n'oubliez pas d'en indiquer les valeurs initiales.

Variables partagées

`barriereA()`

`barriereB()`

Question 13 On cherche maintenant une solution générique pour un programme à N threads. Chaque thread est identifié par son numéro `i` (avec $1 \leqslant$ `i` $\leqslant$ N) et exécute le code suivant :

```
thread(int i)
{
   C(2*i-1);
   barriere(i);
   C(2*i);
}
```

Votre travail consiste à implémenter la fonction `barriere(int i)`. N'oubliez pas de déclarer explicitement tous vos sémaphores et variables partagés ainsi que leur valeur initiale.

Variables partagées

`barriere(int i)`

Question 14 La barrière que vous venez d'implémenter est-elle cyclique ? Autrement dit, on se demande si elle fonctionnerait correctement pour les threads de la forme ci-dessous :

```
thread(int i)
{
   while(true)
   {
      C(i);
      barriere(i);
   }
}
```

Si vous répondez que votre barrière est cyclique, donnez un argument pour justifier votre affirmation. Au contraire, si vous répondez qu'elle ne l'est pas, donnez un exemple de scénario causant un problème.

Question 15 Proposez une nouvelle version de `barrière(i)` satisfaisant ces deux propriétés :

1. Le nombre de sémaphores est indépendant du nombre de threads.
2. La barrière est cyclique, au sens de la question précédente.

Variables partagées

`barrière(int i)`

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.