IFA-3-SYS : Systèmes d'exploitation

NOM Prénom :

Consignes

- L'examen dure 1h30. Prenez le temps de lire le sujet en entier (15 questions sur 7 pages)
- 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 le binaire et l'hexadécimal, aidez-vous des tableaux page 7.
- Dans les questions vrai/faux, les erreurs sont décomptées : ne répondez pas au hasard.

Échauffement

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

INSA	Institut National des Sciences Appliquées
FCFS	
ISR	
MMU	
PID	

1 Noyau et processus

Question 2 Lorsqu'un processus lit des données sur le disque, par exemple en invoquant l'appel système `read()`, comment fait-il pour savoir à quel moment les données sont arrivées ?

Question 3 Si on exécute le programme ci-dessous, combien de fois la lettre X sera-t-elle affichée ?

```
1  void main()
2  {
3    for (int i=0; i<10; i++)
4    {
5      if ((i & 1) == 1)
6      {
7        fork();
8      }
9    }
10   print("X");
11 }
```

La lettre X sera affichée [ ] fois en tout.

2 Ordonnancement

Question 4 Pour chaque affirmation ci-dessous, entourez V si elle vous parait correcte, ou entourez F si elle est fausse/absurde.

V F Par définition, les stratégies d'ordonnancement préemptives sont sans risque de famine.

V F Dans un système interactif, par ex. un téléphone, la majorité des processus sont *IO-bound*.

V F Le *waiting time* d'un processus n'inclut pas le temps passé dans des syscalls bloquants.

V F Si à un instant donné tous les processus du système sont suspendus, alors il faudra redémarrer la machine car aucun processus ne pourra plus être réveillé.

Question 5 On s'intéresse dans cette question à trois processus A, B, et C qui se partagent un verrou mutex `M`. Le code des processus est donné ci-dessous, avec des commentaires indiquant les temps de calcul de chaque fonction. Par exemple, s'il était exécuté seul, le processus A se terminerait après 100 millisecondes (resp. 50 ms pour B et 70 ms pour C). Mais à cause des synchronisations, il faudra peut-être devoir attendre avant de pouvoir entrer en section critique.

a.c

```
main()
{
  calculer_a1(); // 30 ms
  lock(M);
    calculer_a2(); // 60 ms
  unlock(M);
  calculer_a3(); // 10 ms
  exit();
}
```

b.c

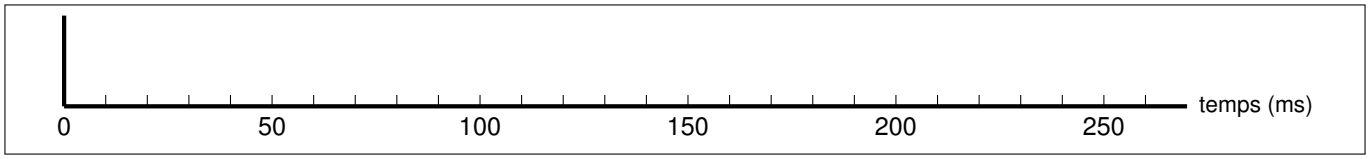
```
main()
{
  calculer_b1(); // 20 ms
  lock(M);
    calculer_b2(); // 20 ms
  unlock(M);
  calculer_b3(); // 10 ms
  exit();
}
```

c.c

```
main()
{
  calculer_c1(); // 70 ms

  exit();
}
```

On suppose un ordonnanceur circulaire (Round Robin) avec un quantum de 40 ms. Le processus A est prêt à l'instant initial, B arrive à t=10ms et C arrive à t=20ms. Sur une feuille de brouillon, dessinez un chronogramme indiquant la succession des tâches sur le processeur. Recopiez ensuite votre réponse dans le cadre ci-dessous. Faites apparaître les moments où un processus est en section critique (par exemple avec des hachures).



Question 6 Dans le cadre ci-dessous, donnez le diagramme états/transitions d'un processus, en supposant un ordonnanceur non-préemptif.

3 Mémoire virtuelle

Question 7 Pour chaque affirmation ci-dessous, entourez V si elle vous parait correcte, ou entourez F si elle est fausse/absurde.

V F Dans un système qui implémenterait la mémoire virtuelle mais pas le multitâche, il ne pourrait pas y avoir de faute de page.

V F La mémoire principale a un temps d'accès 5 à 10 fois plus important que le cache processeur.

V F Si deux processus accèdent en lecture à la même adresse physique, ils verront la même valeur.

V F Lorsqu'elle effectue une traduction d'adresse, la MMU peut être amenée à faire plusieurs lectures supplémentaires en mémoire principale.

Question 8 On suppose dans cette question un système avec des adresses virtuelles sur 32 bits dont 20 bits de numéro de page. Si l'utilisateur fait un appel `mmap(10000)`, combien de pages lui seront allouées par le noyau ?

Le noyau va allouer ☐ pages.

Question 9 On suppose dans cette question un système avec des adresses virtuelles sur 16 bits et des pages de 64 octets. Pour chaque adresse virtuelle ci-dessous, indiquez son numéro de page virtuelle et son offset. Répondez en notation hexadécimale. Rappel : vous pouvez vous aider des tableaux page 7.

adresse	n° de page	offset
`0x0001`		
`0x0060`		
`0xABCD`		
`0x5678`		

4 Allocation dynamique

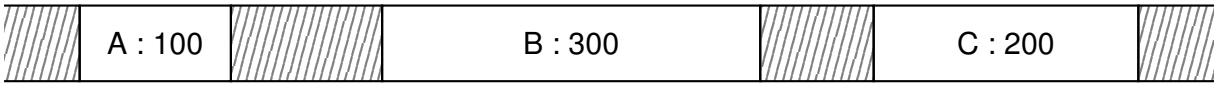
Question 10 Le programme ci-dessous alloue plusieurs tableaux. Dans quelle section de son espace d'adressage sera placé le tableau `tabA` ?

```
int main(void)
{
  int tabA[]={1,2,3,4};

  int *tabB = malloc(100 * sizeof(int));
  for(int i=0; i<100; i++)
  {
    tabB[i] = tabA[i % 4];
  }
  return 0;
}
```

- [V] `.text` (le code)
- [V] `.data` (les données statiques)
- [V] `.heap` (le tas)
- [V] `.stack` (la pile)

Question 11 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 $N-T$ (autrement dit, on néglige 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

Question 12 On souhaite écrire une fonction qui compte le nombre zéros consécutifs à droite dans la représentation binaire d'un entier `n`, supposé strictement positif. Par exemple si `n` vaut 80 on doit retourner 4, car en binaire 80 s'écrit 1010000. Complétez le code ci-dessous. Pour les calculs en binaire, vous pouvez vous aider des tableaux page 7.

```
unsigned int count_trailing_zeros(unsigned int n)
{
   assert(n > 0);









}
```

Question 13 On a parlé en cours d'un phénomène appelé la «fragmentation» de la mémoire. Donnez un exemple de situation «du monde réel» (c.à.d. en dehors de l'informatique et des ordinateurs) où un phénomène analogue se produit.

5 Concurrence et synchronisation

Question 14 Pour affirmation ci-dessous, entourez V si elle vous parait correcte, ou entourez F si elle est fausse/absurde.

V F Il est possible d'implémenter les sémaphores à partir de verrous mutex.
V F Il est possible d'implementer les verrous mutex à partir de sémaphores.
V F Il est possible d'implémenter les verrous mutex à partir d'instructions élémentaires.
V F Les termes «verrou mutex» et «sémaphore» sont synonymes.

Question 15 On s'intéresse dans cette question aux risques d'interblocage (deadlock) dans un programme où des threads concurrents se synchronisent à l'aide de sémaphores. Écrivez ci-dessous le code de trois threads A, B, et C tels que :

— aucun thread ne comporte de boucle ni de structure de contrôle comme `if-then` etc,
— si on exécute les trois threads ensemble, alors un interblocage peut se produire (mais il n'est pas garanti),
— si on exécute seulement deux de ces trois threads (n'importe lesquels) alors aucun interblocage ne peut se produire.

Pour chaque sémaphore que vous utiliserez, pensez bien à préciser sa valeur initiale.

Conditions initiales

thread A

thread B

thread C

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} = 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,
- etc. (avec dans l'ordre : Pio, Eio, Zio, Yio)

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