IFA-3-SYS : Systèmes d'exploitation 20 mars 2024

NOM Prénom :

Consignes

- Durée : 1h30. Lisez le sujet en entier (16 questions sur 8 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 8.
- Dans les questions vrai/faux, les erreurs sont décomptées : ne répondez pas au hasard.

1 Noyau et processus

Question 1 Pour chacun des mécanismes proposés dans la liste ci-dessous, entourez V si ce mécanisme est implémenté à l'intérieur du noyau, entourez F sinon.

V F Ordonnanceur
V F Fonction `printf()`
V F Cycle de Von Neumann
V F Interrupt Service Routine
V F Fonction `exec()`
V F Context Switch
V F Émulateur de terminal
V F Shell

Question 2 Qu'appelle-t-on une «trappe» ? Expliquez succintement de quoi il s'agit, et en quoi c'est utile dans un système d'exploitation.

Question 3 On s'intéresse dans cette question au programme ci-dessous qui utilise comme en TP les appels système `fork()`, `exec()`, `sleep()`, ainsi que la fonction `int atoi(char *)` pour interpréter une chaîne de caractères en valeur entière. On compile ce programme en un exécutable `truc` puis on l'invoque avec la commande `./truc`.

Dans le cadre ci-dessous, donnez un exemple d'affichage que peut produire cette commande :

```
int main(int argc, char** argv)
{
  if(argc==2)
  {
    int t = atoi(argv[1]);
    sleep(t);
    printf("%d\n", t);
  }
  else
  {
    char * param[] = {"3","6","1","5","4","2"};
    for(int i=0; i<6; i++)
    {
      if(!fork())
      {
         execl(argv[0],param[i],param[i],NULL);
         printf("%d\n", i);
         wait(NULL);
      }
    }
  }
}
```

2 Multitâche et ordonnancement

Question 4 Lorsque le noyau fait un changement de contexte entre deux processus, il doit sauvegarder/restaurer l'état interne de certains composants matériels. Pour chaque élément ci-dessous, entourez V si le noyau doit en faire la sauvegarde/restauration, ou entourez F s'il peut en «effacer» (aka «invalider») sereinement le contenu.

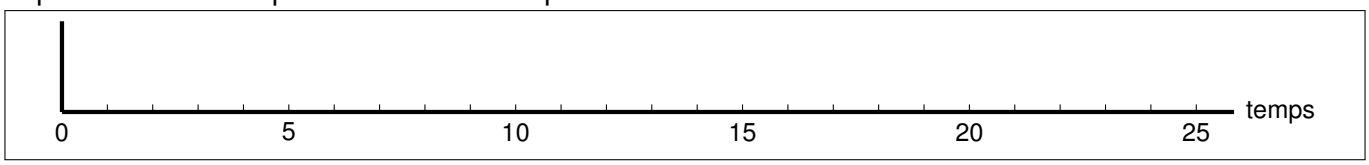
V F Cache CPU : L1, L2...
V F Registres CPU généraux : R0, R1...
V F Registres CPU spécialisés : PC, SP...
V F Translation Lookaside Buffer

Question 5 Que signifie l'expression «attente active» ? En quoi est-ce généralement considéré comme une «mauvaise» façon d'attendre ?

Question 6 On s'intéresse dans cette question à trois tâches avec les caractéristiques suivantes :

nom	A	B	C
instant d'arrivée	0	4	7
durée d'exécution	12	9	4

On suppose que l'ordonnanceur applique la stratégie SRTF. La durée des changements de contexte est supposée négligeable. Au brouillon, dessinez un chronogramme indiquant la succession des tâches sur le processeur. Recopiez ensuite votre réponse dans le cadre ci-dessous.



3 Mémoire virtuelle

Question 7 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.

V F En général, l'espace d'adressage virtuel est plus petit que la mémoire physique.
V F L'adresse physique d'une même page virtuelle peut changer au cours du temps.
V F Grâce à la mémoire virtuelle, les accès à la mémoire principale sont, en moyenne, plus rapides.
V F La taille des pages virtuelles est toujours la même que la taille des pages physiques.

Question 8 On suppose dans cette question un système avec des adresses virtuelles sur 16 bits, dont 7 bits pour le numéro de page. Si l'utilisateur fait un appel `mmap(5000)`, combien de pages lui seront allouées par le noyau ? Vous pouvez vous aider du tableau page 8 pour les calculs en binaire.

Le noyau va allouer [ ] pages.

4 Allocation dynamique

Question 9 Le programme ci-dessous alloue plusieurs tableaux. Dans quelle section de son espace d'adressage sera placé le tableau `tabA` ?

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

```
#include <stdlib.h>

int tabA[]={1,2,3,4};

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

Question 10 Cette question porte sur un allocateur dynamique (similaire à celui implémenté en TP). Pour chaque proposition, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.
Lors d'une allocation :
V F découper un bloc (pour n'en allouer qu'une partie) sert à réduire la fragmentation externe.
V F découper un bloc (pour n'en allouer qu'une partie) sert à réduire la fragmentation interne.
Lors d'une désallocation :
V F fusionner plusieurs blocs voisins sert à réduire la fragmentation externe.
V F fusionner plusieurs blocs voisins sert à réduire la fragmentation interne.

Question 11 Cette question porte sur la stratégie d'allocation dynamique *best-fit*. Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.
V F Cette stratégie n'est pas compatible avec les mécanismes de découpage/fusion de blocs.
V F Cette stratégie n'est jamais victime du problème de fragmentation du tas.
V F Cette stratégie nécessite d'initialiser le tas avec des blocs libres de tailles bien choisies.
V F Cette stratégie cherche à exploiter en priorité les blocs libres les plus petits.

5 Concurrence et synchronisation

Question 12 Pour chaque proposition ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.
V F Différents threads du même processus ont la même pile d'exécution.
V F Différents threads du même processus ont la même table de pages.
V F Différents threads du même processus ont le même CPU.
V F Différents threads du même processus ont les mêmes sections critiques.

Exercice : synchronisation par sémaphores

Dans cet exercice, on s'intéresse à un programme concurrent similaire à ceux vus en cours et en TD, où plusieurs threads se partagent l'accès à une ressource critique. Il y a deux catégories de threads : les *lecteurs* qui accèdent à cette ressource uniquement en lecture, et les threads *rédacteurs* qui peuvent également en modifier le contenu. Plusieurs lecteurs peuvent lire simultanément sans que cela pose de problème. Par contre, pendant qu'un rédacteur est en cours de modification, l'état de la ressource n'est pas cohérent et on veut donc empêcher tous les autres threads (lecteurs *et* rédacteurs) d'y accéder.

Autrement dit, on s'intéresse à un programme dans lequel une quantité arbitraire de threads exécutent chacun l'un ou l'autre des comportements ci-dessous, et on veut garantir les trois contraintes de synchronisation suivantes :
— il est permis à plusieurs lecteurs de `lire()` simultanément
— il est interdit à plusieurs rédacteurs de `modifier()` simultanément
— il est interdit aux lecteurs de `lire()` pendant qu'un rédacteur est en train de `modifier()`

threads lecteurs

```
while(true)
{
  // A?
  lire();
  // B?
}
```

threads rédacteurs

```
while(true)
{
  // C?
  modifier();
  // D?
}
```

Dans les questions suivantes, votre travail va consister à implémenter les synchronisations (points notés A, B, C et D) à l'aide de sémaphores et/ou de variables partagées.

Question 13 Une approche simpliste est de restreindre l'accès à la ressource à un seul thread au maximum, sans tenir compte des deux catégories de threads. Implémentez ci-dessous les synchronisations nécessaires.
Remarques :

- n'oubliez pas de préciser la valeur initiale de vos sémaphores et/ou variables partagées.
- vous pouvez utiliser autant de code et/ou d'appels à P() et/ou à V() que vous jugerez utile.
- vous pouvez aussi laisser un cadre vide pour dire «aucune synchronisation n'est nécessaire ici».

Conditions initiales

lecteur : synchro A

rédacteur : synchro C

lecteur : synchro B

rédacteur : synchro D

N'hésitez pas à ajouter des commentaires pour expliquer votre démarche :

Question 14 Vous remarquez que cette solution simpliste est certes correcte en termes de synchronisation, mais qu'elle aura probablement des performances assez mauvaises. Proposez une nouvelle solution qui autorise des lecteurs multiples à accéder simultanément à la ressource.

Conditions initiales

lecteur : synchro A

rédacteur : synchro C

lecteur : synchro B

rédacteur : synchro D

N'hésitez pas à ajouter des commentaires pour expliquer votre démarche :

Question 15 Vous remarquez que cette nouvelle solution souffre d'un risque de famine. En réalité, ce sont nos trois contraintes de synchronisation (cf page 4) qui, à elles seules, sont insuffisantes pour garantir un comportement «raisonnable» du système.
Décrivez un scénario d'exécution dans lequel une situation de famine se produit.

Question 16 Proposez une nouvelle solution évitant ce risque de famine.

Conditions initiales

lecteur : synchro A

rédacteur : synchro C

lecteur : synchro B

rédacteur : synchro D

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.