Examen de Système d'Exploitation et Programmation Concurrente

2ème année ENSIMAG
Décembre 2013

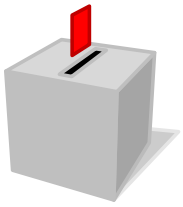
Documents autorisés
Durée : 3 heures

Il est demandé de rédiger les solutions dans un pseudo-langage dans le style de ce qui a été fait en cours et en TD et en respectant les notations de l'énoncé.
La clarté et la concision seront des éléments importants d'appréciation, lorsque des programmes sont demandés, ceux-ci sont en général suffisants et d'éventuelles explications supplémentaires devront être concises et porter sur le principe de la solution.

Les deux parties doivent être rédigées sur deux copies différentes

1 Synchronisation

Rappel : les deux parties doivent être rédigées sur deux copies différentes.
English translation of French voting vocabulary is given in the annexe B.



Plusieurs processus votent. Il faut les synchroniser. Pour voter, un processus doit passer dans un isoloir. Il y a N isoloirs. Il doit y avoir au plus 1 processus dans un isoloir à 1 instant donné.
On modélise les isoloirs par un tableau de booléen **isoloir_occupe**, avec la sémantique suivante :

- **isoloir_occupe [i] == faux** : l'isoloir est vide.
- **isoloir_occupe [i] == vrai** : un processus est dans l'isoloir.

Les isoloirs sont vides à l'origine, c'est-à-dire **bool isoloir [N] = {faux, ..., faux};**.

Question 1 *Écrivez un* moniteur *avec les primitives de synchronisations suivantes :*

- ***entrer_isoloir (int *i)***
- ***sortir_isoloir (int i)***

où i est un index d'isoloir. ***entrer_isoloir*** *permet au processus de rentrer dans un isoloir libre, et lui retourne l'index de l'isoloir.* ***sortir_isoloir*** *permet au processus de quitter l'isoloir i dans lequel il était entré. On considère qu'un processus appellera toujours* ***sortir_isoloir*** *avec l'index qu'il a récupéré avec* ***entrer_isoloir*** *.*
Vous devez garantir qu'il ne peut y avoir au maximum qu'un seul processus par isoloir.
Vous devez définir les variables nécessaires à l'écriture du moniteur.

On considère maintenant qu'il y a des bulletins dans chaque isoloir. Ils sont représentés par un tableau d'entiers, il y a au départ B bulletins dans chaque isoloir.
int bulletins [N] = {B, ..., B};
A chaque fois qu'un processus vote, il consomme un bulletin dans l'isoloir où il se trouve. Un processus qui vote a donc le comportement suivant :

```
void vote(void) {
    int i;

    entrer_isoloir(&i);
    bulletins[i]--;
    /* ... choisi son candidat ... */
    sortir_isoloir(i);
}
```

De plus, il existe un processus chargé de remettre des bulletins dans les isoloirs. Ce processus entre dans un isoloir vide où il n'y a plus de bulletins, y dépose B bulletins, et sort de l'isoloir. Le comportement de ce processus est le suivant, **trouver_isoloir_sans_bulletin** étant une nouvelle primitive de synchronisation du moniteur.

```
void remplir_bulletins() {
    int i;

    while(true) {
        trouver_isoloir_sans_bulletin(int *i);
        bulletins[i] = B;
        sortir_isoloir(i);
    }
}
```

Question 2 *Modifier* ***entrer_isoloir*** *et / ou* ***sortir_isoloir*** *de telle sorte que :*

- *un processus qui veux voter ne peut pas entrer dans un isoloir où il n'y a pas de bulletins.*
- *un processus qui vote et qui prend le dernier bulletin signale qu'il y a un isoloir à remplir.*

Écrivez aussi la primitive ***trouver_isoloir_sans_bulletin*** *. Quand aucun isoloir n'a besoin d'être rempli, le processus se placera en attente. Attention, le processus qui veut remplir des bulletins ne doit pas entrer dans un isoloir qui n'est pas vide.*

Une fois qu'un processus votant a rempli son bulletin et quitté l'isoloir, il doit mettre son bulletin dans l'urne.
On veut garantir qu'il n'y a pas de fraude quand les processus qui votent mettent leur bulletin dans l'urne.
Pour garantir qu'il n'y a pas de fraude, un processus doit mettre son bulletin dans l'urne uniquement si il y a au moins 2 autres processus prêts à mettre leur bulletin dans l'urne. Quand un groupe de processus a commencé de mettre ses bulletins dans l'urne, les autres processus doivent attendre qu'ils aient fini avant de se présenter devant l'urne.
Voici un petit exemple qui illustre ce mécanisme :

- au départ, aucun processus n'est prêt à déposer son bulletin.
- les processus $p1$ puis $p2$ arrivent devant l'urne et veulent mettre leur bulletin, mais ils ne peuvent pas car ils ont besoin d'être au moins 3.
- le processus $p3$ arrive devant l'urne, il va pouvoir permettre à $p1$, $p2$ et lui-même $p3$ à mettre leur bulletin dans l'urne.

— pendant que $p1$, $p2$ et $p3$ mettent leur bulletin, les processus $p4$, $p5$, $p6$ et $p7$ arrivent devant l'urne. Ils doivent attendre que $p1$, $p2$ et $p3$ aient fini avant de savoir s'ils peuvent mettre leur bulletins.
— une fois que $p1$, $p2$ et $p3$ ont fini, $p4$, $p5$, $p6$ et $p7$ constatent qu'ils sont suffisamment nombreux pour déposer leur bulletins. Ils commencent à déposer leur bulletin. Les processus $p8$ et $p9$ arrivent et se mettent en attente.
— $p4$, $p5$, $p6$ et $p7$ terminent de déposer leur bulletin et s'en vont. $p8$ et $p9$ ne sont pas assez nombreux pour déposer leur bulletins, ils attendent.

On considère qu'à la fin de la journée, des processus en charge de l'organisation des votes viendront compléter les derniers processus en attente de dépose de bulletins.
Pour résumer, le comportement d'un processus est maintenant le suivant :

```
vote() {
    int i;

    entrer_isoloir(&i);
    bulletins[i]--;
    /* ... choisi son candidat ... */
    sortir_isoloir(i);

    attente_devant_urne();
    /* ... depose son bulletin dans l'urne ... */
    quitter_urne();
}
```

On a un nouveau moniteur qui protège l'accès à l'urne.

Question 3 *Écrivez les primitives de moniteur* ***attente_devant_urne*** *et* ***quitter_urne*** *de manière à ce que les processus attendent d'être au moins 3 pour pouvoir déposer leur bulletin. Vous pouvez utiliser les variables globales que vous désirez.*

2 Mémoire et ordonnancement

Rappel : les deux parties doivent être rédigées sur deux copies différentes.

NB : Une table avec les valeurs des puissances de 2 (de 2^0 à 2^{64}) est donnée en annexe page 6.

2.1 Adressage 64 bits et les processeurs ARM

L'architecture ARM est l'architecture dominante parmi les smartphones. Dans ses dernières incarnations, elle a tous les attributs des processeurs issus du monde PC (extensions vectorielles, parallélisme, GPGPU), avec une performance pure plus faible au profit de la consommation d'énergie.
La dernière évolution, l'architecture ARMv8 a un espace d'adressage de 64 bits. Les adresses exprimées sur 64 bits sont donc découpées et traduites par la MMU. Il y a 3 découpages utilisables. Deux des découpages possibles sont représentés par le schéma suivant 1. Le troisième est un peu plus complexe et ne sera pas utilisé dans cet exercice.

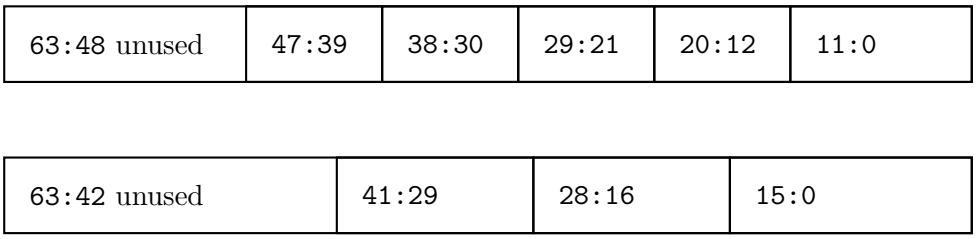


FIGURE 1 – Deux découpages des adresses virtuelles en ARMv8

Question 4 *Quelle est la véritable taille de l'espace d'adressage virtuel (la taille de l'espace virtuel qui peut réellement être traduite) dans les deux cas ? Avec quelles tailles de pages ?*

Question 5 *Dessinez un schéma représentant les deux tables des pages associées aux deux découpages. Vous indiquerez en particulier sur le schéma :*

- *Le nombre d'entrées dans chaque niveau de la table*
- *La taille des tables intermédiaires de chaque niveau (en Ko)*

2.2 Pagination et ordonnancement

Les processus et les threads (processus lègers) d'un système ont trois états majeurs : *Actif*, *Prêt* et *Bloqué* (sur une entrée-sortie).

Question 6 *Précisez les transitions possibles entre les états.*

Question 7 *Dans quel état est un thread qui provoque un* défaut de page *? Change-t-il d'état ? Pourquoi ?*

Question 8 *Un thread change-t-il d'état lorsque qu'une de ses adresses est traduite en utilisant la TLB ? Lorsqu'une adresse est traduite en utilisant la table des pages ?*

On considère pour la question suivante que le traitement d'un défaut de page coûte :

- $8ms$, si la page est déjà en mémoire et n'a pas été modifiée ou si il faut une nouvelle page vide
- $20ms$, si la page a été modifiée

Les deux cas arrivent avec un probabilité de 50%. Un accès mémoire sur cette machine coûte $100ns$.

Question 9 *Quel est le taux maximal de défaut de page par accès mémoire pour avoir un temps moyen d'accès mémoire inférieur à* $200ns$.

Le processeur de votre ordinateur est utilisé à 20% par votre application tandis que votre disque dur est utilisé à 97%.

Question 10 *Vous soupçonnez que votre application* swappe. *Que changez-vous dans le matériel composant votre ordinateur pour accélérer l'exécution de votre application ? Justifiez.*

Question 11 *Votre application, qui swappe, est composée de deux processus indépendants quasi-identiques. Que devrait faire le système d'exploitation pour accélérer l'exécution de votre application ? Justifiez.*

Question 12 *Peut-on vérifier si une application* swappe *? Comment ?*

2.3 Les liens entre les caches de données et la MMU

La phase de traduction peut se placer à plusieurs niveaux dans la hiérarchie mémoire. Parfois avant les caches. Ils manipulent alors des adresses physiques. Parfois après. Les caches manipulent alors des adresses virtuelles. Pour une même architecture, il y a même eu parfois des changements entre deux générations successives de processeurs.
La figure 2 représente l'architecture physique d'un PC (x86-64). Le processeur est un processeur multi-cœur. Cela a un impact direct sur la traduction.

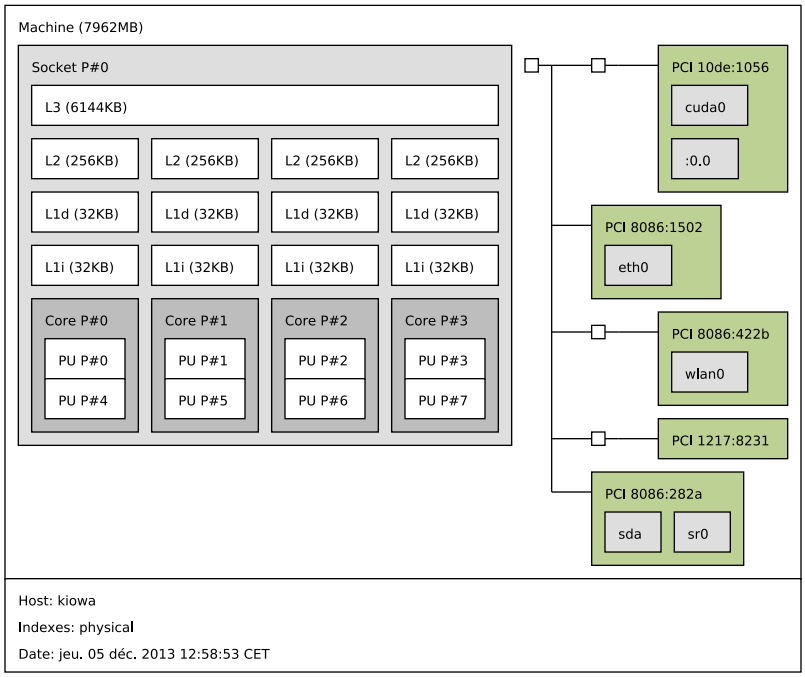


FIGURE 2 – Schéma de l'architecture d'un PC généré par lstopo (hwlock)

Question 13 *Position de la MMU Où placeriez-vous la phase de traduction par la MMU ? Où ne pouvez-vous pas la mettre ? Justifiez !*

A Annexes

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}	=	144 115 188 075 855 872
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

B Translation

isoloir	→	voting cubicle, voting booth
bulletin	→	ballot paper
urne	→	ballot box