Nom : AIDER
Prénom : Smail
N°Etudiant : 3603379
Parcours : SAR
Responsable : M Alain Greiner

Compte-Rendu TP9
Application multi-tâches coopératives

C) *Application producteur / consommateur*

Question C1

Les deux taches travaillent sur une variable partagée « *BUFFER* » sans aucune synchronisation, donc on ne peut pas prévoir un comportement précis.
Comme les deux taches s'exécute sur des processeurs différent et que leurs débits de production/consommation est identique, on peut dire que le consommateur va suivre le même rythme que le producteur.

Question C3

On observe que le consommateur ne consomme pas toutes les valeurs produites par le producteur. Le déférence de débit de production/consommation change le comportement des deux programmes en l'absence de synchronisation.

D) *Synchronisation par bascule SET/RESET*

Question D1

La variable *SYNC* peut être modifier par les deux taches producteur et consommateur sauf qu'il n y a pas de risque d'incohérence car il y a qu'un seul propriétaire à un instant 't'. Pas de concurrence d'accès en cas du respect des 3 conditions suivantes :
- 1 seul producteur
- 1 seul consommateur
- 1 seule case dans le tampon

Question D2

En introduisant la variable *SYNC*, la synchronisation est bien respectée. Le débit de production et de consommation dépend du délai d'attente maximal entre les deux taches, Max *(consumer_delay, producer_delay)*.
La durée d'exécution avec synchronisation est presque identique entre les deux taches.

Question D3

Cette cachabilté introduit un risque de dysfonctionnement à cause de **l'obsolescence du cache**. Comme les deux variables sont stockées dans un segment cachable, les modifications apportées par un processeur ne seront pas visible par l'autre.
Dans notre cas, le *SNOOP* est activé, donc la copie qui se trouve dans le cache est toujours invalidée lors d'une écriture sur la même adresse.

Question D4

En désactivant le mécanisme de SNOOP, le programme se bloque parce que les deux taches (processeurs) travaillent sur des copies différentes, chacun dans son cache.

Question D5

Le coût matériel causé par le mécanisme de *SNOOP* n'est pas du à l'implémentation de l'automate *FSM_SNOOP* (négligeable).
Ce mécanisme nécessite d'avoir une mémoire double accès « ***Dual Port Ram*** » pour permettre deux accès simultanés (un par le processeur et l'autre par le snoop) dans le répertoire du cache.

E) *Problèmes de synchronisation liés au compilateur*

Question E1

Introduction des macros « ***__sync_synchronize()*** » pour garantir l'ordre d'accès aux variables partagées :

```
BUF = n;
__sync_synchronize();
SYNC = 1;
```

```
val = BUF;
__sync_synchronize();
SYNC = 0;
```

Question E2

Le code binaire avant et après l'introduction de la macro :

```
lui     v1,0x100
lw      v0,24(s8)
sw      v0,408(v1)
lui     v1,0x100
li      v0,1
sw      v0,412(v1)
```

=>

```
lui     v1,0x100
lw      v0,24(s8)
sw      v0,408(v1)
sync
lui     v1,0x100
li      v0,1
sw      v0,412(v1)
```

Le temps exécution a diminué de 1171 cycles après avoir introduit cette macro.

F) *Synchronisation par FIFO logicielle*

Question F1

La structure représentant le canal de communication est déclarée comme une variable globale car c'est une variable partagée.
Les différents champs de la structure :
- buf : un tableau circulaire représentant la FIFO
- ptr : numéro de la première case pleine à lire
- ptw : numéro de la première case libre à écrire
- sts : l'état de la FIFO
- depth : la taille de la FIFO
- lock : un verrou pour garantir un accès exclusif à la FIFO

Question F2

Il est préférable d'utiliser un verrou à ticket plutôt qu'un verrou simple pour éviter la **famine**. En effet, on a utilisé un verrou simple dans **l'espace système** lors du TP8 car c'est au système d'exploitation de gérer l'équité.

Question F3

Les arguments des deux fonctions *lock_aquire()* et *lock_release()* :
- lock_t * : un pointer vers le verrou

Question F4

- La fonction *lock_acquire()* permet de demander un *'ticket'* puis d'incrémenter la variable « *lock_t→free* » de façon **atomique** grâce à la fonction *atomic_increment()*. Elle rentre ensuite dans une attente active jusqu'à ce que la ressource soit disponible. Cette attente est modélisée sous forme d'une boucle en comparant la valeur du '*ticket*' avec la variable « *lock_t→current* » qui sert à établir l'équité.

- La fonction *atomic_increment()* utilise les deux primitives « Load-Link(LL) et Store-Conditional(SC) » pour réaliser une demande de verrou exclusive. Elle renvoie la valeur courante de la variable « *lock_t→free* » puis elle l'incrémente.

Question F5

La fonction *lock_release()* permet de libérer le verrou en faisant une écriture sur la variable « *lock_t→current* ». cette fonction n'a pas besoin d'être écrite en assembleur car on n'a pas besoin d'utiliser les primitives offertes par le matériel pour relâcher le verrou. En effet, la tache qui exécute cette fonction est la seule à détenir le verrou.

Question F6

Une tache qui ne peut effectuer son transfert à cause de l'état de la FIFO (pleine/vide) doit impérativement relâcher le verrou pour laisser à l'autre tache de prendre le verrou afin d'écrire/lire dans la FIFO.
Si le verrou n'est pas relâché, dans ce cas, le programme se bloque.

Question F8

- 50 Itérations =>

DEPTH	1	2	4	8
Producer	529211	555561	551685	520659
Consumer	538882	553285	550183	546873

Le temps d'exécution reste presque identique.
La taille de la FIFO ne sert pas à augmenter le parallélisme (le temps d'exécution) du programme. Sa profondeur sert à encaisser les variations instantanées du débit, Encaisser les rafales (burst).

Question F9

- 1000 Itérations =>

DEPTH	1	2	4	8
Producer	304603	298109	259631	261619
Consumer	304732	298239	259760	261750

G) *Application logicielle multi-taches*

Question G1

Dans le « reset.s », il faut définir les applications utilisateurs à exécuter sur chaque tache en mettant l'adresse de la fonction « router() » dans le registre EPC des processeurs 2 à 5.

Question G2

Chaque canal de communication doit posséder son propre verrou d'accès exclusif car on ne veut pas que le consommateur soit bloquer pour accéder aux données qui sont déjà disponible, ou que le producteur soit bloquer en attendant le verrou pour écrire dans la FIFO qui n'est pas encore pleine.