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

Compte-Rendu TP10
Partage du processeur / Commutation de taches

C) *Contexte d'exécution d'une tache*

Question C1

Dans une architecture mono-processeur, les deux taches T0 et T1 doivent partager le processeur. Le temps est decoupé en tranches et une seule tache est en cours d'execution à un instant 't'.
Mecanisme de commutation de taches :
1. La tache T0 s'excute en premier
2. Une interruption est générée par le timer
3. Le gestionnaire d'interruption determine le numéro de l'interruption active en consultant le composant ICU, et appelle ensuite la fonction _isr_switch()
4. L'ISR commence par acquitter l'interruption du timer et appelle la fonction _ctx_switch() qui determine la tache interrumpue et la tache à executer. Sauvegarde le contexte d'execution matériel de la tache T0 et charge le contexte d'execution de la tache T1.
5. Le processeur execute le tache T1
6. Une nouvelle interruption arrive
7. Se brancher à l'ISR _isr_switch()
8. Sauvegarde du contexte de T1, resturation de celui de T0
9. Exécution de la tache T0 …

Question C2

Le contexte d'une tache est sauvegardé dans le tableau _TASK_CONTEXT_ARRAY qui est une variable globale du noyau.
Il n'est pas nécessaire de sauvegarder les valeurs stockées dans la pile car chaque tache possede sa propre pile donc la sauvegarde de pointeur de pile suffit.

Question C3

Pour mémoriser le placement des taches sur les processeurs, le GIET dispose d'un tableau de taille NB_PROCS, chaque case contient les task_id des taches sur chaque processeur. Il est donc possible d'utiliser la fonction _procid() pour indexer ce tableau.

Question C4

Pour mémoiser quelle tache est en cours d'exécution sur chacun des processeurs à un instant donné, le GIET utilise le tableau _CURRENT_TASK_ARRAY.

Question C5

- Parmi les 32 registres, il n'est pas nécessaire de sauvegarder $0, $26 et $27 lors d'un changement de contexte.
- Il faut sauvegarder les registres HI et LO car on peut avoir une situation ou la multiplication a terminé mais le resultat n'est pas encore copié dans un registre utilisable lors d'un changement de contexte.

- Il est indispensable de sauvegarder SR, EPC et CR car ils sont des registres systemes permettant la gestion des interruptions et des exceptions.

Question C6

La fonction _TASK_SWITCH() a commme argument l'adresse de début du contexte de la tache courante et celui de la prochaine tache. Elle reçoit ses arguments via les registres $4 et $5 et elle posséde pas de valeur de retour.

Question C7

La fonction _TASK_SWITCH est découpée en deux parties :
- Sauvegarde du contexte de la tache courante
- Restitution du contexte de la prochaine tache
Cette fonction peut modifer les registres $26, et $27 dans la phase de auvegarde du contexte de la tache sortante.

Question C8

La fonction _TACK_SWITCH() est toujours écrite en assembleur car elle sauvegarde et récupere les contextes matiériels des taches en interagissant avec les registres du processeur, ce qui est possible qu'avec des intructions en assembleur.

Question C9

La fonction _TASK_SWITCH() se branche à l'adresse contenue dans le registre $31 de la nouvelle tache. Il faut initialiser la case contenant le registre $31 du tableau de contexte de la tache pour savoir le point de retour apres le changement de contexte.

Question C10

La politique d'orfonnancement implémtée par la fonction _CTX_SWITCH() est round-robin.

D) *Création et lancement des taches*

Question D1

Les 3 registres qu'il faut initialiser avant de lancer la tache T(p,0) :
- SP qui prend comme valeur l'adresse de la pile de la tache T(p,0)
seg_stack_base + (4 * pid + 1) * 64K
- EPC qui prend comme valeur l'adresse de la fonction à executer par T(p,0)
seg_entry_point[4 * pid]
- SR qui vaut **0XFF13**

Question D2

Les cases du tableau qui doivent impérativement etre initiliser pour chacune des taches T(n,k) sont :
- 0 (SR) qui prend comme valeur 0XFF13
- 32 (EPC) qui prend comme valeur **task_entry_point[4 * pid + k]**
- 29 (SP) qui prend comme valeur **seg_stack_base + (4 * pid + k + 1) * 64K**
- 32 (RA) qui prend comme valeur l'adresse de l'instruction ERET pour sauter à l'adresse du programme utilisateur contenue dans EPC

Question D6

La taille du segment de pile doit etre minumaum 16 * 64K soit : **0x100000**.
Il faut initialiser le pointer de pile de chacune des taches T(n,k) à la valeur :
seg_stack_base + (4 * n + k + 1) * 64K

Question D7

Pour assurer l'equitabilité entre les differentes taches, c'est le systeme d'exploitation qui doit decider de la périodicité des changements de contexte.

E) *Fonctionnement multi-taches sur mono-processeur*

Question E2

F) *Fonctionnement multi-taches sur multi-processeurs*

Question F2

Le fonctionnement bi-processeur augmente la performance du système : une tâche ne doit partager les ressources de son processeur qu'avec une autre tâche, autrement dit le deuxième processeur a reduit la charge du premier processeur, d'où l'intérêt d'ajouter le nombre de processeurs dans une machine.