TD n° 2

Système temps-réel

I) Application temps réel embarquée de la mission Pathfinder

Présentation de la mission

Mars Pathfinder est une sonde spatiale développée par la NASA. C'est la première sonde à se poser sur le sol de la planète Mars le 4 juillet 1997. Cette sonde a libérée un robot mobile *Sojourner* chargé de faire des mesures (photo, météo, etc) et des prélèvements. Le robot est relié à la sonde Pathfinder par une liaison UHF (Ultra Hautes Fréquences) et en reçoit des commandes de pilotage.

Après quelques jours de fonctionnement correct, le calculateur embarqué s'est réinitialisé à plusieurs reprises conduisant à la perte de nombreuses informations. Après une analyse détaillée du problème et une série de tests sur terre, une erreur dans l'implémentation multitâche s'est avéré être la source du problème. Cette erreur est liée à la gestion d'une ressource critique. Une fois l'erreur repérée et la solution trouvée, le code a été modifié et transféré depuis la terre.

Architecture logicielle

L'architecture logicielle est une architecture multitâche gérée par le noyau temps réel VxWorks (Wind River System). L'ensemble de l'application comprend plus de 25 tâches. Ces tâches sont de type périodique et apériodique. La mission a des phases d'activités très différentes (vol interplanétaire, pose de la sonde sur Mars, exploration par le robot du sol) et toutes les tâches ne sont pas utiles dans une phase donnée.

Nous nous intéresserons dans cette étude à la phase d'exploration et donc à la configuration de tâches associée à ce mode. L'architecture logicielle simplifiée de l'application est donc réduite à la liste des tâches suivantes par ordre de priorité :

- `Ordo_bus` : (tâche la plus prioritaire) tâche ordonnanceur du bus 1553 vérifiant que le trasfert des données a été correctement effectué et prépare le nouveau cycle de transert ;
- `Distribution_donnees` : tâche de distribution des données du bus 1553 ;
- `Pilotage` : tâche de pilotage du robot ;
- `Radio` : tâche de gestion des communications radio ;
- `Camera` : tâche de gestion de la caméra ;
- `Mesures` : tâche dédiée aux mesures ;
- `Meteo` : (tâche la moins prioritaire)tâche de gestion des données météo.

Les caractéristiques temporelles des tâches (classées par ordre de priorité) sont données dans le tableau ci-dessous :

Tâche	Priorité	C_i	P_i	Temps utilisation ressource
Ordo_bus	7	1	5	-
Distribution_donnees	6	1	5	1
Pilotage	5	1	10	1
Radio	4	1	10	-
Camera	3	1	10	-
Mesures	2	2	200	2
Meteo	1	{2,3}	200	{2,3}

A noter que la durée d'exécution de la tâche `Meteo` peut avoir deux valeurs (2 ou 3) correspondant à des échanges de données plus ou moins important.

Pour simplifier l'étude :

- la durée d'exécution C_i, la période P_i et le temps d'utilisation de la ressource critique ont été réduites avec une granularité de 25ms. Par exemple, la tâche `Ordo_bus` dure 25ms et sa période est de 125ms sur le système réel.
- La durées des exécutions des tâches sont des valeurs fixes estimées dans le pire cas (WCET), excepté pour la tâche `Meteo` qui peut avoir deux valeurs (2 ou 3) correspondant à des échanges de données plus ou moins important.
- les temps d'accès à la ressource commune correspondent à la totalité de la durée d'exécution de la tâche La tâche `Pilotage` prend donc la ressource (i.e. un mutex) au début de son exécution et la libère à la fin de son exécution. Ainsi, une tâche utilisant cette ressource partagée ne pourra pas commencer son exécution si une autre tâche est en posséssion de la ressource.

Supposons que les tâches soient indépendantes (pas de ressource partagée).

Question 1. Quelle est le facteur d'utilisation W de cette configuration de tâche sachant que U(7) = 0,729 ? Conclure sur l'optimalité de l'algorithme RMS pour $C_{Meteo} = 2$ *et* $C_{Meteo} = 3$.

Question 2. Sur quelle période d'étude faudrait-il tracer l'ordonnancement pour vérifier l'optimalité de RMS sur ce système de tâche ? Vérifier que RMS fonctionne bien sur l'intervalle de temps [0 - 25] en supposant toujours que les tâches soient indépendantes.

L'interdépendance entre les tâches, due au partage de la ressource critique, ne permet plus d'appliquer la condition analytique précédente et conduit donc à tracer la séquence d'exécution de l'application pour vérifier l'optimilaté de l'ordonnancement. **On se limitera à une période d'étude [0 - 25] pour tous les chronogrammes dans la suite.**

Question 3. Tracer la séquence d'exécution (sur [0 - 25]) suivant l'algorithme d'ordonnancement basé sur les priorités données dans le tableau précédent dans le cas ou $C_{Meteo} = 2$. L'ordonnancement est-il optimal ?

Question 4. Tracer la séquence d'exécution (sur [0 - 25]) suivant l'algorithme d'ordonnancement basé sur les priorités données dans le tableau précédent dans le cas ou $C_{Meteo} = 3$. L'ordonnancement est-il optimal ?

Question 5. Comment nomme-t-on ce type de problème et quel protocole (en général) est mis en place pour éviter ce phénomène ? Tracer la séquence d'exécution (sur [0 - 25]) avec ce protocole (on garde les mêmes priorités de base pour les tâches).

Pour éviter ce problème, il est aussi possible d'utiliser le protocole de gestion de ressource "super-priorité". Ce protocole attribut à une tâche en section critique une priorité maximale. Ce qui conduit à interdire les communtations de tâches pendant les sections critiques.

Question 6. Tracer la séquence d'exécution (sur [0 - 25]) avec le protocole "super-priorité" (on garde les mêmes priorités de base pour les tâches).