L2 TDTP 6 – ThreadsUnix - Mutex

Ce TP se fait sur les machines Unix/Linux. Vous devez rendre un rapport et vos programmes à la fin du cours sur Espadon. Assurez-vous que votre rapport porte un nom du type "1516_L2_TDTP_6_Threads_Unix_Mutex_*votreNom*". Mettez dans le rapport des copies d'écran des résultats d'exécutions ("Applications" → "Accessoires" → "Capture d'écran") que vous ferez régulièrement le long du TP pour montrer la réussite ou l'échec de vos opérations. Rendez le tout dans un fichier Zip de même nom que précédemment. Pour tous les TP, ne validez votre dépôt sur Espadon qu'une fois le TP totalement terminé; à la fin de chaque séance, déposez un zip de vos travaux avec votre rapport en l'état, mettez-le à jour à chaque séance, et une fois le TP fini, validez.

I]Connexion

1) Connectez-vous à votre machine virtuelle Unix/Linux (Utilisez le mot de passe que vous avez défini au premier TP).

2) Dans le répertoire "/home/mathinfo2014/Documents/SE_and_PS", créez un répertoire "Mutex" (commande *mkdir*; ce sera le répertoire de travail du TP).

II] Exercice : La chenille

1) Dans le fichier "laChenille.c", écrivez un programme en C qui déclare une liste chainée à partir d'une structure *Job* qui contient un pointeur vers un *Job*, un entier *jobIndex* indiquant le numéro de 'Job', et deux entiers *a* et *b*, le 'Job' à réaliser étant d'effectuer la somme des deux nombres (qui seront tirés au hasard) et de l'afficher à l'écran.

2) Créez une fonction *createJobs* qui renvoie un pointeur sur un nouveau *Job* tiré au hasard (mais avec un numéro de *Job* qui suit ceux des jobs précédents).

3) Créez une fonction *createJobs* qui génère une liste chainée sur un nombre *n* de jobs passé en paramètre à la fonction et renvoie un pointeur sur le premier élément de la liste.

4) Créez une fonction de thread *providerFunction* qui fait passer un entier variable globale *flag* à *1* une fois qu'elle a entamé son exécution, puis qui attend se manière sécurisée d'être notifiée de l'absence de jobs à traiter, et qui, quand notifiée rajoute un nombre de jobs à la liste compris entre *NB_JOBS_REFILL* et *2xNB_JOBS_REFILL*.

5) Créez une fonction de thread *workerFunction* qui prend de manière sécurisée le prochain job de la liste ; si aucun job n'est disponible, elle notifie le provider et attend d'être notifiée de la présence de jobs dans la liste. Après avoir effectué et être sortie de sa section critique, elle effectue le travail du job et attend pendant un temps compris entre *1* secondes et *1+SLEEP_TIME* secondes avant de recommencer.

6) Créez un programme principal (main) qui initialise les différente structures et variables nécessaires (toutes ne sont pas décrites dans l'énoncé) à l'exécution du programme, puis crée un thread à partir de la fonction *providerFunction* ; puis le main attend que le flag, initialement à *zéro*, passe à *1* pour commencer à générer un nombre *NB_WORKERS* de threads basés sur la fonction *workerFunction* ; enfin, pour la forme, le programme principal attend la fin de l'exécution des threads worker puis du provider.

Un exemple de sortie à l'exécution est donné dans la figure ci-dessous :