L2 Informatique — Université de Nantes
X4I0010 – Système d'exploitation — 2015/2016

Contrôle continu
Contrôle de travaux pratiques – Contrôle final

Épreuve de contrôle continu du 28 avril 2016 (groupe 402)
Tous documents autorisés

Déposez votre devoir sous la forme d'une seule archive compressée sur Madoc dans l'espace prévu à cet effet (Dépôt des compte-rendus de TP CC → Groupe 402 : Contrôle continu de TP – Contrôle final). Veillez bien à ce que votre archive comporte les noms du binôme de TP (ex : dupont-durant_TPCC_Controle_final.zip). Le contenu de l'archive sera constituée des fichiers sources commentés des 3 programmes.

Exercice 1 (Les processus)

Écrire un programme C (ou C++) qui crée deux processus fils affichant toutes les secondes leur PID ainsi que la valeur d'un compteur incrémenté de un chaque seconde. De son coté, le processus père attendra dix secondes avant d'envoyer un signal qui devra "tuer" le 1er fils. Il attendra encore dix secondes pour faire la même chose à son second fils.

Exercice 2 (Les threads et les sémaphores)

Nous voulons écrire un programme qui crée deux threads en plus du thread principal (3 threads en tout). Les threads créés devront accéder à une ressource commune constituée d'une variable entière **x** globale à l'application. Un des deux threads auxiliaires devra sans cesse incrémenter la variable **x** de un. L'autre thread devra sans cesse la décrémenter de un. De plus, chaque thread conservera le nombre de fois qu'il a modifié **x**. Ainsi, normalement le contenu de **x** devrait être identique à la différence du nombre d'incrémentations de **x** et du nombre de décrémentations de **x**. Le thread principal devra afficher la valeur de la variable **x** toutes les secondes. Il affichera aussi la différence entre le nombre d'opérations effectuées par le premier thread et celui du second thread (pour cette raison, ces deux compteurs seront aussi implémentés comme des variables globales partagées par les 3 threads).

1. Écrire un premier programme sans prendre de précaution particulière pour l'accès aux ressources partagées par les 3 threads. Normalement, l'exécution de ce programme devrait montrer rapidement une divergence entre le contenu variable **x** et la différence des comptages des opérations d'incrémentation et de décrémentation.
2. Écrire un second programme utilisant un sémaphore pour créer un "mutex" permettant de protéger l'accès concurrent aux ressources partagées. L'exécution de ce programme ne devrait pas montrer de divergence entre le contnu de **x** et la différence des comptages des opérations effectuées par les deux threads auxiliaires.