CSEL1 (construction systèmes embarqués sous Linux) – Travail pratique

Programmation système : Multiprocessing

Processus, signaux et communication

1. Concevez et développez une petite application mettant en œuvre un des services de communication proposé par Linux (p.ex. « socketpair ») entre un processus parent et un processus enfant.
   Le processus enfant devra émettre quelques messages sous forme de texte vers le processus parent, lequel les affichera sur la console. Le message « exit » permettra de terminer l'application.
   Cette application devra impérativement capturer tous les signaux et les ignorer. Seul un message d'information sera affiché sur la console.
   Chacun des processus devra utiliser son propre cœur, p.ex. core 0 pour le parent, et core 1 pour l'enfant.

CGroups

2. Concevez une petite application permettant de valider la capacité des groupes de contrôle de limiter l'utilisation de la mémoire.

   Quelques indications pour la création du programme :
   a. Allouer un nombre défini de blocs de mémoire d'un mégabyte, p.ex. 50
   b. Tester si le pointeur est non nul
   c. Remplir le bloque avec des 0

   Quelques indications pour monter les CGroups :
   a. `$ mount -t tmpfs none /sys/fs/cgroup`
   b. `$ mkdir /sys/fs/cgroup/memory`
   c. `$ mount -t cgroup -o memory memory /sys/fs/cgroup/memory`
   d. `$ mkdir /sys/fs/cgroup/memory/mem`
   e. `$ echo $$ > /sys/fs/cgroup/memory/mem/tasks`
   f. `$ echo 20M > /sys/fs/cgroup/memory/mem/memory.limit_in_bytes`

   Quelques questions :
   a. Quel effet a la commande « `echo $$ > …` » sur les cgroups ?
   b. Quel est le comportement du sous-système « memory » lorsque le quota de mémoire est épuisé ? Pourrait-on le modifier ? Si oui, comment ?
   c. Est-il possible de surveiller/vérifier l'état actuel de la mémoire ? Si oui, comment ?

CSEL1 (construction systèmes embarqués sous Linux) – Travail pratique

Programmation système : Multiprocessing

3. Afin de valider la capacité des groupes de contrôle de limiter l'utilisation des CPU, concevez une petite application composée au minimum de 2 processus utilisant le 100% des ressources du processeur.

   Quelques indications pour monter les CGroups :

   a. Si pas déjà effectuer, monter le cgroup de l'exercice précédent.
   b. `$ mkdir /sys/fs/cgroup/cpuset`
   c. `$ mount -t cgroup -o cpu,cpuset cpuset /sys/fs/cgroup/cpuset`
   d. `$ mkdir /sys/fs/cgroup/cpuset/high`
   e. `$ mkdir /sys/fs/cgroup/cpuset/low`
   f. `$ echo 3 > /sys/fs/cgroup/cpuset/high/cpuset.cpus`
   g. `$ echo 0 > /sys/fs/cgroup/cpuset/high/cpuset.mems`
   h. `$ echo 2 > /sys/fs/cgroup/cpuset/low/cpuset.cpus`
   i. `$ echo 0 > /sys/fs/cgroup/cpuset/low/cpuset.mems`

   Quelques questions :

   a. Les 4 dernières lignes sont obligatoires pour les prochaines commandes fonctionnent correctement. Pouvez-vous en donner la raison ?
   b. Ouvrez deux shells distinctes et placez une dans le cgroup high et l'autre dans le cgroup low, p. ex.

      ```
      # ssh root@192.168.0.14
      $ echo $$ > /sys/fs/cgroup/cpuset/low/tasks
      ```

      Lancez ensuite votre application dans chacune des shells.
      Quel devrait être le bon comportement ? Pouvez-vous le vérifier ?
   c. Sachant que l'attribut `cpu.shares` permet de répartir le temps CPU entre différents cgroups, comment devrait-on procéder pour lancer deux tâches distinctes sur le cœur 6 de notre processeur et attribuer 75% du temps CPU à la première tâche et 25% à la deuxième ?