Master 1 – Informatique

TP 1
Namespaces et cgroups

Attention aux copiés-collés des commandes sur un terminal. Certains caractères word peuvent ne pas correspondre aux caractères ASCII attendu par le terminal (les « , les -, etc.).

Exercice 1 : namespace pid et mount

Namespace pid

1. Lancez un nouveau bash qui aura les mêmes namespaces que son père sauf pour « pid » (l'option --fork indique que le programme sera exécuté dans un processus fils – c'est unshare qui se fork).

```
sudo unshare  --fork --pid bash
```

2. Indiquez le pid du processus dans le nouveau bash (echo $$) et dans le host (ouvrez une autre console pour ce faire et utilisez la commande ps a). Nous appellerons cette deuxième console la console host dans la suite.
3. Listez tous les processus dans les 2 consoles (ps aux | grep bash). La commande ps utilisant la liste des processus se trouvant dans le répertoire /proc, les deux processus doivent afficher la même chose.
4. Les namespace sont ils les mêmes (commande `lsns –p <pid>` ou directement dans `/proc/<pid>/ns/`) ? Il faut regarder les numéros d'inode pour vérifier quelles sont les namespace commun. Autre méthode, vous pouvez aussi lire les liens symboliques dans chacune des deux consoles (readlink /proc/**pid**/ns/*). Quittez (exit).

Namespace mount

5. Lancez un nouveau bash qui aura les mêmes namespace que son père sauf le namespace « mount »

```
sudo unshare  --fork --mount bash
```

6. Identifiez un dossier avec du contenu (disons « folder »). Et dans le bash de namespace différent tapez la commande :

```
sudo mount --bind folder toto
```

7. Testez le contenu du dossier toto dans les deux bash/consoles.

8. Afin d'isoler la vue des processus, il est possible de combiner le namespace « pid » avec le montage d'un /proc qui lui est propre. Tapez la commande

```
sudo unshare --pid  --fork --mount-proc bash
```

9. Testez ps dans les deux consoles (celle qui est « unshare » et la console host).
10. Listez les processus en cours d'exécution en parcourant le dossier /proc. Comme indiqué plus haut vous avez un dossier par processus (le nom du dossier est le pid du processus). Faites ce « ls » dans les deux consoles.

11. Comparez les namespace des deux bash (unshared et classique). Cela se fait dans la console host : identifiez les pid des deux bash, consultez le dossier `/proc/<pid>/ns` (commande `ls –l` pour voir les identifiants des inodes). Vous devez observer qu'en plus du namespace pid et pid_children qui est différent, le montage d'un /proc différent à nécessité la création d'un namespace mount spécifique pour le nouveau bash.
12. Quittez.

Exercice 2 : un conteneur simple en C

Il vous est d'exécuter un programme C simple qui créer un conteneur. Il y a quelques étapes préalables à son fonctionnement.

1. Décrivez, à partir du code, les différentes étapes pour créer ce conteneur.
2. Ce conteneur exécute un bash qui est isolé. Pour le tester vous devez d'abord créer un nouveau répertoire. Vous mettrez le code dedans. Vous devez créer 4 sous dossiers : bin, lib, lib64 et proc.
3. Vous devez ensuite copier le binaire bash dans le bin local. Vous devez faire de même avec ses librairies. Pour les connaitre : ldd /bin/bash. Puis pour chacune de ses librairies les copier-coller dans les répertoires correspondants. Il devrait y en avoir 4 : libtinfo.so.6, libdl.so.2, libc.so.6, et ld-linux-x86-64.so.2.
4. Compilez votre conteneur, et lancez-le dans le dossier où il y a tout (votre code, les dossiers bin, etc.).
5. Votre bash fonctionne et il est isolé. Il n'y a aucune commande autre que le bash puisque votre processus est isolé. Pour avoir les autres commandes, il faudrait les copier-coller dans votre dossier ainsi que les librairies associées.

Exercice 3 (facultatif):

Cet exercice va essentiellement utiliser la commande ip combiné à l'option netns qui permet de manipuler directement les namespace network. La commande unshare aurait pu aussi être utilisée pour créer ces namespace networks (mais moins pratique ici).

Communication entre un namespace network et le namespace root.

1. Créer un nouveau namespace toto :
   ```
   ip netns add toto
   ```
2. Créez deux interfaces virtuelles et connectez les:
   `ip link add veth-root type veth peer name veth-toto` //Créer l'interface virtuelle et son extrémité (créer aussi l'autre interface veth-toto)
3. Effectuez un « ip link ». Vous voyez les deux nouvelles interfaces nouvellement créées.
4. Associez l'une des deux interfaces dans le namespace toto. L'autre restera dans le namespace root.
   `ip link set veth-toto netns toto` //associe l'interface veth-tot au namespace toto
5. Effectuez un « ip link », l'interface veth-toto a disparu (car rattachée à l'autre namespace).
6. Associez une adresse ip à chacune des interfaces :
   ```
   ip addr add 192.168.0.2/24 dev veth-root
   ip –n toto addr add 192.168.0.1/24 dev veth-toto
   ```
7. On active les interfaces

```
ip link set veth-root up
ip –n toto link set veth-toto up
```

8. Les deux interfaces peuvent se pinguer:
   `ping 192.168.0.1` //le namespace root ping le namepsace toto
   `ip netns exec toto ping 192.168.0.2` //La commande ping est effectué dans le namespace toto et ping le namespace root
9. Pour la connection à Internet cela se fait en plusieurs étapes (voir exercice plus bas). Cependant, si vous aviez mis des adresses IP publiques valides à vos deux interfaces veth il suffirait de rajouter une route par défaut :
   `ip netns exec toto ip route add default via <@IP de veth-root>`
   A minima, vous pouvez toujours tester l'accès à l'interface physique :
   `ip netns exec toto ip route add default via 192.168.0.2`
   `ip netns exec toto ping <@IP de votre interface physique>`

Code de l'exercice 2

```
#define _GNU_SOURCE //Doit etre appele en premier (permet d'acceder a des appels systemes non normalise: unshare ici)


#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/mount.h>

#include<unistd.h>

#include<wait.h>

#include<sched.h>




//Attention: doit etre lance en sudo/root


int main()
```

```
{
  pid_t retourFork;
  int flags;

  flags=CLONE_NEWPID;

  if(unshare(flags)<0){ perror("Error unshare"); exit(2);} //Pour NEWPID seuls les enfants sont dans un nouveau namespace PID - c'est pour cette raison qu'on le fait avant le fork
  retourFork=fork();

  //Les autres "unshare" ont lieu apres le fork pour ne pas isoler le processus pere (celui-ci peut-etre vu comme le runtime)

  switch(retourFork)
  {
    case -1: perror("Error fork");
             exit(2);

    case 0:  printf("Je suis le fils apres le unshare - mon pid est %d\n",getpid());

             //On monte un autre dossier /proc car la commande ps s'appuie dessus (et affichera les numeros de processus du namespace parent) si ce namespace n'a pas son propre /proc.
             //Cela se fait en 2 etapes

             //1. D'abord on separe les points de montage de celui du pere (nouveau fs namespace)
             flags=CLONE_FS;
             if(unshare(flags)<0){ perror("Error unshare proc fils"); exit(2);}

             //2. Puis on monte un nouveau /proc qui sera forcement different de celui du pere car nouveau namespace fs
```

```
            //Premier argument (dummy) car il n'y a pas de source (puisque le systeme va creer un nouveau filesystem proc)

            //Deuxieme argument c'est la destination: le dossier local proc

            //Troisieme argument: le type de systeme de fichiers

            if( mount("proc", "proc", "proc", 0, NULL) <0) perror("Error mount proc");


            //A faire dans le dossier courant: creation d'un dossier bin, lib,et lib64

            //Les dossiers bin, lib, et lib64 devront contenir les binaires (au moins ps, bash, ls et kill) et les librairies correspondantes.


            //chroot

            if(chroot("./")==-1) perror("Error chroot");


            //On lance le bash

            if(execlp("/bin/bash","bash",NULL)<0) { perror("Error execlp"); exit(2); }

            break;


    default: printf("Je suis le pere (pid=%d)- j'attends la terminaison du fils de pid=%d.\n",getpid(),retourFork);

            if(wait(NULL)<0) perror("Error wait");

            printf("Conteneur termine\n");

  }


  return(0);

}
```