TD Linux Embarqué

1 Machine virtuelle, fichiers

La machine virtuelle que nous utiliserons en TD et en TP est disponible à cette adresse : `https://cloud.ensea.fr/index.php/s/nOGKPPOei8LDz0F`

1.1 Machine virtuelle

1. Télécharger le fichier `VM-SOC-2019.ova`
2. Installer Virtualbox. Installer également l'*extension pack* correspondant à la version de Virtualbox choisie
3. Importer la VM `VM-SOC-2019.ova`. Mettre à jour le chemin du dossier partagé
4. Lancer la VM. Le login est `ensea`, le mot de passe est `ensea`

1.2 Fichiers

```
struct asserv {
    char version[10];
    float taille_roue;
    double entraxe;
    unsigned char ratio;
    unsigned short p;
    unsigned int i;
    int d;
};
```

1. Quelle est la taille, en octets, utilisée par cette structure ?
2. En quoi cette taille peut elle varier d'une machine à l'autre ? Modifier la structure pour la rendre plus portable.
3. Écrire un programme permettant à l'utilisateur de remplir cette structure.
4. Modifier le programme pour pouvoir sauvegarder la structure dans un fichier binaire. Vous utiliserez, entre autres, les fonctions `open`, `read` et `write`.
5. Ajouter un autre système de sauvegarde sous forme de fichier texte, où les valeurs seraient compréhensible par un être humain. Vous utiliserez, entre autres, les fonctions `fopen`, `fprintf` et `fscanf`.

2 Processus et IPC

Ce TD peut se faire sous n'importe quelle machine Linux. Toutefois, il y a un risque de faire planter la machine (d'ailleurs, on vous le demandera explicitement, cf. question 3). Il serait plus sage de travailler sur une machine virtuelle.

2.1 Les Processus

1. Créer un processus fils qui s'endort pendant 10 secondes et rend la main à son père.
   — Quel est le statut retourné quand tout se passe normalement ?
   — Quel est le statut retourné quand vous tuez le fils prématurément par le biais d'un signal de votre choix ?
   — Que se passe-t-il lorsque le père qui est tué prématurément. Utiliser les primitives `getpid()` et `getppid()`.
2. Générer par une boucle n processus issus du même père. Mettre le père en attente de tous les fils. Afficher les statut des fils au fur et à mesure de leur disparition. Les fils font un `sleep(`s`)` avec des valeurs de s différentes.
3. Générer un nombre de processus illimité. Existe-t-il une limite ? Pensez à sauvegarder avant, on ne sait jamais.
4. Créer un programme capable de lancer différentes commandes unix en donnant la chaîne de caractère correspondante.

2.2 IPC

1. Mettre en place un pipe entre un processus père et son fils. Le père lit sur l'entrée standard et écrit dans le pipe, le fils lit dans le pipe et écrit sur la sortie standard.
2. Quand le père reçoit la commande `exit`, il envoie un signal SIGUSR1 au fils qui termine, puis attend la mort du fils.

3 Les modules

Ce TD peut se faire sous n'importe quelle machine Linux. Toutefois, il y a un risque de faire planter (involontairement, par erreur) la machine. Il serait plus sage de travailler sur une machine virtuelle.

1. Créez un module simple permettant d'écrire du texte dans le journal du noyau au chargement et au déchargement. Utilisez la fonction `printk()`, et la commande `sudo dmesg` pour voir les messages.
2. Testez les commandes `modinfo`, `insmod`, `rmmod`, `lsmod` et `dmesg`.
3. Trouvez un numéro de majeur "libre". Pour ce faire, tapez la commande : `less /proc/devices`.
4. Créez un fichier spécial `/dev/le_driver` (avec la commande `mknod`) Quelles sont les valeurs retournées par la commande `ls -l /dev/le_driver` ?
5. Dans la fonction d'initialisation du module précédent, enregistrez le driver à l'aide de la fonction `register_chrdev()`. Dans un premier temps la structure **`struct file_operations`** ne sera pas configurée. Pensez à désenregistrer le driver avec la fonction `unregister_chrdev`.
6. Écrivez la fonction `write()` de telle manière à ce qu'elle affiche le message reçu dans le journal.
7. Écrivez la fonction `read()` pour qu'elle renvoie le nombre de fois que la fonction `write()` a été appelée.
8. Tester avec `echo` et `cat`, puis avec un programme C.
9. (Optionnel) Modifier le module précédent pour rajouter un timer. Les écritures dans le journal se font toutes les secondes. La fonction `read()` renvoie le nombre de secondes écoulées.
10. (Optionnel) Réalisez le même exercice en créant une entrée dans /proc/