S. Lavirotte
Polytech Nice Sophia
SI4
2020-2021

TD n° 3
Bibliothèques et Outils pour les Systèmes Embarqués

Le but de ce TD est de vous familiariser avec un certain nombre de bibliothèques et logiciels libres pour la création complète d'un système de fichiers « *From Scratch* » pour des systèmes embarqués.

1 Configuration d'un Noyau pour le Système Embarqué

1.1 Noyau Linux

Pour réaliser ce TD, nous allons avoir besoin d'un noyau pour faire fonctionner notre système. Cela tombe bien car nous en avons produit un lors des TDs précédents. Donc vous allez pouvoir récupérer votre noyau `5.4.91` compilé lors de la précédente séance.

En cas de problème, vous pouvez récupérer ce noyau à l'adresse suivante pour faire vos tests pour ce TD :

```
http://trolen.polytech.unice.fr/cours/isle/td03/vmlinuz-5.4.91
```

1.2 Installation du BusyBox

Nous avons préparé un disque virtuel contenant les sources de `busybox 1.32.1`. Pour en bénéficier, il vous suffit de télécharger :

```
http://trolen.polytech.unice.fr/cours/isle/td03/sdc-busybox.7z
```

Après avoir ajouté le disque virtuel à votre machine virtuelle, vous accéderez au disque de la manière suivante :

```
mkdir /work/td03
mount /dev/sdc1 /work/td03
```

2 Créer un Système de Fichiers

Après son initialisation, le noyau monte la partition racine qui contient le système de fichiers de votre système. C'est ce que nous allons maintenant créé manuellement afin de comprendre comment cela fonctionne.

2.1 Création d'un système de fichier racine pour la cible

Créez un fichier de `2Mo` qui vous permettra de stocker l'image de notre système de fichier (appelons le `root.img`). Faites le formatage de ce disque virtuel en `ext2`. Vous prendrez soin de monter ce fichier créé dans votre arborescence principale afin de permettre les échanges de données.

```
dd if=/dev/zero of=root.img bs=1024 count=2048
mke2fs –m 0 –i 1024 –F root.img
mount –o loop root.img /mnt/
```

2.2 Configuration de Busybox

Commencez par créer une configuration la plus légère/compacte possible permettant de compiler (rappel : nous avions fait cela pour avoir un noyau minimaliste aussi, donc…).
Pour éviter de faire une configuration complète de `busybox`, ce qui prendrait un peu trop de temps, vous allez récupérer le fichier de configuration suivant et le copier à la racine des sources de `busybox` :

```
wget http://trolen.polytech.unice.fr/cours/isle/td03/config-busybox-1.32.1
cp config-busybox-1.32.1 .config
```

Cette configuration inclus les outils pour réaliser les actions suivantes sur la cible :

- Création, suppression et visualisation de répertoires, de fichiers et de fichiers spéciaux
- Autoriser le nettoyage de la console, le changement de clavier (`clear`, `dumpkmap`, `loadkmap` et `reset`)
- Montage de démontage de partitions
- Configuration réseau de la cible (`ifconfig`, `route`, `ping`, `udhcpc`)

- Permettre l'arrêt et le redémarrage de la machine
- Lister et Arrêter un processus, la commande `uptime` pour savoir depuis combien de temps le système a démarré

Comme vous récupérez une configuration existante, ne pas oublier de faire le `make` qui convient pour vérifier cette configuration et repartir de celle-ci.

```
make oldconfig
```

Vous devez maintenant uniquement modifier la configuration pour activer les fonctionnalités suivantes :

- Configurer Busybox pour être un exécutable compilé statiquement (`Settings`)
- Lancer un processus `init` au démarrage du système (`Init Utilities`)
- Activer un serveur Web avec le support pour les scripts CGI (`Networking Utilities`)
- Disposer d'un shell (`ash`) et faire en sorte que celui-ci soit le Shell par défaut (`sh` et `bash`) (`Shells`)
- Ajouter le support math dans tous les Shell (POSIX) (`Shells`)

Une fois ces quelques modifications réalisées, vous devez compilez `BusyBox` pour `x86`.

Copiez l'ensemble des fichiers obtenus après compilation (ceux-ci sont dans le dossier `_install`) dans votre système de fichiers pour la cible. Vérifiez que toutes les commandes sont des liens sur l'exécutable compilé statiquement. A votre avis pourquoi utiliser des liens ?

2.3 Premières configurations du système

Avant de pouvoir démarrer avec le noyau en utilisant le système de fichier minimaliste que nous avons créé, il est nécessaire de réaliser quelques étapes supplémentaires de configuration.

2.3.1 Peuplement de /dev manuellement et de manière minimaliste

Ajoutez dans votre système de fichier racine une entrée `dev/console` afin de pouvoir interagir avec votre système via un Shell. Pour ceci vous devez consulter votre système Linux pour savoir quels sont les numéros majeurs et mineur de `/dev/console`.

```
mknod dev/console c 5 1
```

Vous devrez aussi créer les entrées `tty` et `tty[0-5]` nécessaires pour que le système ne vous envoie pas de messages d'erreur. Vous irez voir les numéros majeurs et mineurs sur le système avec lequel vous travaillez.

2.3.2 Configuration du clavier (pour ne pas s'arracher les cheveux sur votre système cible)

Pour configurer le clavier, vous devrez utiliser `loadkmap` sur votre machine cible. Il faudra lui passer un fichier sur l'entrée standard pour le configurer.

```
loadkmap < /etc/mykbd.kmap
```

Pour créer ce fichier à partir de votre machine de travail, vous exécuterez les commandes suivantes :

```
apt install console-data
loadkeys fr-latin1 (le fichier correspondant à la configuration de votre clavier)
busybox dumpkmap > /tmp/mykbd.kmap
cp /tmp/mykbd.kmap [sur le /etc sur le système cible]
```

2.4 Démarrage du système avec le nouveau système de fichier

Attention à la manière dont vous faites les écritures sur le système de fichier. Il est **interdit d'avoir deux systèmes accédant au même système de fichiers en écriture par montage** (à la fois la machine hôte et la machine cible),

sous peine d'avoir un système de fichier incohérent par rapport aux modifications faites. Pensez donc bien à démonter le fichier avant de l'utiliser avec la machine virtuelle. Dans un premier temps, votre système de fichier sera monté en lecture seule donc ce n'est pas obligatoire, mais cela va rapidement changer.

Testez le lancement de votre système dans une machine virtuelle (`qemu`). `qemu` a été installé dans votre machine virtuelle de travail. Cela ne sera pas forcément des plus performant, mais lorsque nous simulons un système embarqué, celui-ci n'est de toute façon pas aussi performant que votre machine de travail, donc ce n'est pas un souci. Pour démarrer `qemu`, vous devez être sous environnement graphique car `qemu` ouvre une fenêtre correspondant à l'écran de la machine émulée.

```
qemu-system-x86_64 –m 48 –kernel vmlinuz-5.4.91 -drive format=raw,file=root.img
–append "root=/dev/hda"
```

A cette étape, vous devriez accéder à une console et être capable de lancer les commandes que vous avez compilées avec `BusyBox`. Faites un test en exécutant quelques-unes d'entre elles (création de dossier, de fichier, …). Pour pouvoir remonter le système de fichier racine en lecture écriture sur votre système embarqué :

Attention à ne pas monter deux fois le système de fichiers en écriture (une fois sur la machine Host et une fois sur la machine Guest). Donc bien penser à démonter sur la machine host si vous montez en écriture sur la machine Guest.

```
mount –o remount,rw /dev/root /
```

3 Configuration « avancée »

Lors de votre premier démarrage, vous avez surement constaté que certaines actions ne se sont pas bien déroulées, par exemple lors de l'arrêt de la machine ou bien encore au démarrage de celle-ci pour charger la configuration clavier ou faire les quelques opérations de base nécessaires (vous ne pouvez pas encore monter votre système de fichier en lecture/écriture). Nous allons dans cette section mettre en place ces éléments indispensables au bon fonctionnement du système.

3.1 Systèmes de fichiers racine de la cible

Depuis votre machine de travail, créez les entrées `/proc` sur le système de fichier racine de votre cible.

Puis depuis votre machine cible, montez le système de fichier virtuel `/proc`.

```
mount –t proc none /proc
```

Maintenant que `/proc` est disponible, vous pouvez constater que vous pouvez éteindre proprement votre cible à l'aide de la commande `halt`. Ceci vous permettra ainsi de vous assurer que l'ensemble des modifications sur le système de fichier ont été sauvegardées.

3.2 Configuration du système pour le démarrage

Nous allons maintenant mettre en place une configuration pour le système de démarrage de la machine. Créez le fichier `/etc/inittab`. Vous pourrez trouver de la documentation sur la syntaxe supportée par `BusyBox` sur le site Web de `BusyBox`.

```
# Ceci est un script pour init
::sysinit:/etc/init.d/rcS
# Démarrer un shell "askfirst" sur la console
::askfirst:-/bin/sh
tty2::askfirst:-/bin/sh
tty3::askfirst:-/bin/sh
tty4::askfirst:-/bin/sh
# Invocation de /sbin/getty pour les ttys suivants
```

```
tty4::respawn:/sbin/getty 38400 tty5
tty5::respawn:/sbin/getty 38400 tty6
# Choses à refaire au redémarrage d'init
::restart:/sbin/init
# Choses à refaire au rédémarrage de la machine
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

Créez le script `/etc/init.d/rcS` utilisé dans le fichier `/etc/inittab`. Dans ce script de configuration, vous veillerez à charger automatiquement la bonne configuration de votre clavier et de monter les systèmes de fichier `/proc`. Pensez bien à donner les droits d'exécution sur ce script Shell `rcS`.

4 Un serveur web sur la machine cible

Maintenant que nous disposons d'un système correctement configuré, nous allons tenter de mettre en place un serveur Web sur notre système qui sera embarqué. Voici les opérations principales à réaliser.

4.1 Configuration réseau

Vous devez relancer votre machine virtuelle de votre système embarqué (celle lancée avec `qemu`) avec les paramètres supplémentaires suivants :

```
-netdev user,id=mynet0,net=192.168.10.0/24,dhcpstart=192.168.10.10,hostfwd=tcp::5555-:80
-device e1000,netdev=mynet0
```

Ces options permettent de définir la plage IP pour le réseau créé, les paramètres pour le DHCP, de mettre en place une redirection du port `80` de votre machine virtuelle (Guest OS) via le port `5555` de votre Host OS (en TCP bien entendu) et d'activer une carte réseau de type `e1000` sur la machine virtuelle.

Sur votre Guest OS, pour configurer l'interface réseau, nous allons utiliser un client DHCP (nous avons inclus dans la configuration `udhcpc`). Copier les fichiers présents dans le dossier `examples/udhcp/` des sources de BusyBox sur le système de fichiers embarqué dans le dossier `usr/share/udhcpc/`. Donner le droit d'exécution sur `sample.script` et ajouter un lien vers ce fichier en le nommant `default.script`. Ensuite, ajouter l'appel à `/sbin/udhcpc` dans votre script de démarrage `etc/init.d/rcS`. Enfin, créer le dossier `etc/udhcpc` nécessaire pour stocker la configuration qui sera mise en place.

Vous pourrez vérifier votre configuration à l'aide des commandes `ifconfig` et `route` pour vous assurer que la configuration réseau est bien en place.

4.2 Configuration d'un serveur web sur la cible

Créez le dossier `www` sur le système de fichier de la cible. Ajoutez-y un fichier `index.html` de votre choix (un simple titre et texte suffisent pour tester).

Lancez le serveur http de `BusyBox` via la ligne de commande suivante :

```
Guest OS# /usr/sbin/httpd -h /www/ &
```

Testez la connexion depuis votre machine hôte avec un navigateur, et le tour est joué !

```
Host OS# lynx http://localhost:5555/
```

5 Conclusion

Vous venez de créer un système embarqué « *From Scratch* » qui fonctionne avec un système de fichier de moins de 2Mo. Attention toutefois, celui-ci ne contient pas le bootloader, ni le noyau car celui-ci est passé en paramètre à `qemu`. Il faudrait donc un système de fichier de l'ordre de 4Mo pour avoir une cible complète (`2.6`Mo pour le noyau

et environ 1.3Mo pour le système de fichier). Ceci laisse toute de même pas mal de place pour ajouter d'autres fonctionnalités (le code métier) à votre système embarqué s'il dispose par exemple de 4 ou 8Mo.

6 Pour aller plus loin (non obligatoire)

Créez le `www/cgi-bin` sur le système de fichier de la cible. Vous placerez le fichier `uptime` suivant dans le dossier `www/cgi-bin/` :

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<html>"
echo "<header></header>"
echo "<body>"
echo "<h1>Durée de fonctionnement:</h1>"
echo "Votre systeme embarque tourne depuis: <pre>"
echo `uptime`
echo "</pre>"
echo "</body></html>"
```

N'oubliez pas de donner les droits d'exécution sur ce script `uptime`.

Intégrez le démarrage automatique du serveur Web au script de démarrage `rcS` de votre cible embarquée.