Esprit : Ecole Sup Privée d'Ingénierie et de Technologies

Rapport de projet Linux Embarqué

Encadrant Académique :
Mme. Souha Bousselmi

Entreprise d'accueil CAPGEMINI ENGINEERING :

Présenté par : **Oumayma Guizeni**

Année Universitaire 2023-2024

Sommaire

Tâche n°1 : Installation et Utilisation d'un Compilateur Croisé pour ARM

Introduction

Ce rapport explique les commandes nécessaires pour installer et utiliser un compilateur croisé pour ARM sur un système Ubuntu. Chaque commande sera accompagnée d'une explication détaillée, et une place sera laissée pour les captures d'écran correspondantes.

I. Installation du Compilateur Croisé

1) Mise à jour de la liste des paquets:

Commande: sudo apt-get update

Cette commande met à jour la liste des paquets disponibles pour installation ou mise à jour. Elle synchronise la liste des paquets disponibles à partir des dépôts configurés sur le système.

```
\Oumayma@\Localhost:~$ sudo apt-get update
Hit:1 http://ppa.launchpad.net/tomtomtom/woeusb/ubuntu focal InRelease
Hit:2 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:3 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:4 http://tn.archive.ubuntu.com/ubuntu focal InRelease
Get:5 http://tn.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
Hit:6 http://tn.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 128 kB in 2s (65.1 kB/s)
Reading package lists... Done
\Oumayma@\Localhost:~$
```

2) Installation des outils de construction et du compilateur ARM:

Commande: sudo apt-get install build-essential gcc-arm-linux-gnueabihf

Cette commande installe le paquet `build-essential`, qui contient des outils de développement essentiels, comme `gcc` ainsi que le compilateur croisé `gcc-arm-linux-gnueabihf`, qui permet de compiler des programmes pour une architecture ARM sur une machine x86.

```
\Oumayma@\Localhost:~$ sudo apt-get install build-essential gcc-arm-linux-gnueabihf
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc-arm-linux-gnueabihf is already the newest version (4:9.3.0-1ubuntu2).
build-essential is already the newest version (12.8ubuntu1.1).
The following packages were automatically installed and are no longer required:
  blender-data efibootmgr fonts-dejavu gdal-data libarmadillo9 libblosc1 libboost-filesystem1.71.0 libboost-locale1.71.0 libcfitsio8
  libcharls2 libdap25 libdapclient6v5 libdcmtk14 libepsilon1 libfreexl1 libfyba0 libgdal26 libgdcm3.0 libgeos-3.8.0 libgeos-c1v5 libgeotiff5
  libglew2.1 libhdf4-0-alt libjemalloc2 libkmlbase1 libkmldom1 libkmlengine1 libminizip1 libnetcdf15 libnvidia-cfg1-515 libnvidia-common-515
  libnvidia-common-525 libnvidia-compute-515 libnvidia-decode-515 libnvidia-encode-515 libnvidia-extra-515 libnvidia-fbc1-515
  libnvidia-gl-515 libogdi4.1 libopencolorio1v5 libopencv-core4.2 libopencv-imgcodecs4.2 libopencv-imgproc4.2 libopencv-videoio4.2
  libopenimageio2.1 libopenvdb6.2 libosdcpu3.4.0 libosdgpu3.4.0 libpq5 libproj15 libraw19 libsocket++1 libspatialite7 libspnav0 libsquish0
  libsuperlu5 libtbb2 libtinyxml2.6.2v5 liburiparser1 libxerces-c3.2 libyaml-cpp0.6 nvidia-compute-utils-515 nvidia-kernel-common-515
  nvidia-kernel-source-515 nvidia-utils-515 odbcinst odbcinst1debian2 proj-bin proj-data xserver-xorg-video-nvidia-515
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
\Oumayma@\Localhost:~$
```

3) Création et édition du fichier source:

Commande: gedit prog.c

Cette commande ouvre l'éditeur de texte `gedit` pour créer et éditer un fichier source C nommé `prog.c`. Le contenu du fichier pourrait être un simple programme "Hello, World!" :

```c
#include <stdio.h>
int main() {
   // printf() displays the string inside quotation
   printf("Hello, World!");
   printf("\n");
   return 0;
}
```

II. Tester avec l'émulateur QEMU

1) Compilation avec le Compilateur Croisé:

Commande: arm-linux-gnueabihf-gcc prog.c -o helloarm

Cette commande utilise le compilateur ARM pour compiler `prog.c` en un exécutable nommé `helloarm` qui est adapté à l'architecture ARM.

2) Installation de l'émulateur QEMU:

Commande: sudo apt install qemu-user

Cette commande installe QEMU, un émulateur qui permet d'exécuter des programmes compilés pour différentes architectures. Ici, `qemu-user` est utilisé pour exécuter des binaires ARM sur une machine x86.

```
\Oumayma@\Localhost:~$ arm-linux-gnueabihf-gcc prog.c -o helloarm
\Oumayma@\Localhost:~$ ls | grep helloarm
helloarm
\Oumayma@\Localhost:~$
```

3) Exécution du programme compilé:

Commande: qemu-arm helloarm

Cette commande tente d'exécuter le programme ARM `helloarm` en utilisant QEMU. À ce stade, une erreur est attendue :

```
\Oumayma@\Localhost:~$ qemu-arm helloarm
helloarm: error while loading shared libraries: libc.so.6: wrong ELF class: ELFCLASS64
\Oumayma@\Localhost:~$
```

Résolution des Erreurs de Bibliothèque:

a. Installation des bibliothèques ARM 32 bits: Assurez-vous que toutes les bibliothèques ARM 32 bits nécessaires sont installées par la commande:
sudo apt-get install libc6-armhf-cross libc6-dev-armhf-cross

b. Configuration du chemin des bibliothèques: Ajoutez le chemin des bibliothèques ARM 32 bits à la variable d'environnement `LD_LIBRARY_PATH` par la commande:
export LD_LIBRARY_PATH=/usr/arm-linux-gnueabihf/lib:/lib/arm-linux-gnueabihf

c. Création du lien symbolique correct pour `libc.so.6`: vérifiez où se trouve libc.so.6 pour ARM et créez un lien symbolique par la commande:
sudo ln -sf /usr/arm-linux-gnueabihf/lib/libc.so.6 /lib/arm-linux-gnueabihf/

d. Re-exécution du programme:
qemu-arm helloarm

et le message “Hello, World !” s’affiche.

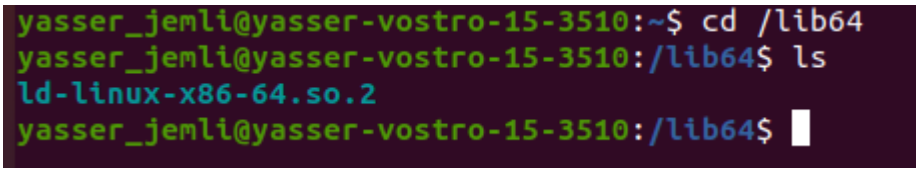
```
\Oumayma@\Localhost:~$ sudo apt-get install libc6-armhf-cross libc6-dev-armhf-cross
Reading package lists... Done
Building dependency tree
Reading state information... Done
libc6-armhf-cross is already the newest version (2.31-0ubuntu9.9cross1).
libc6-armhf-cross set to manually installed.
libc6-dev-armhf-cross is already the newest version (2.31-0ubuntu9.9cross1).
libc6-dev-armhf-cross set to manually installed.
The following packages were automatically installed and are no longer required:
  blender-data efibootmgr fonts-dejavu gdal-data libarmadillo9 libblosc1 libboost-filesystem1.71.0 libboost-locale1.71.0 libcfitsio8
  libcharls2 libdap25 libdapclient6v5 libdcmtk14 libepsilon1 libfreexl1 libfyba0 libgdal26 libgdcm3.0 libgeos-3.8.0 libgeos-c1v5 libgeotiff5
  libglew2.1 libhdf4-0-alt libjemalloc2 libkmlbase1 libkmldom1 libkmlengine1 libminizip1 libnetcdf15 libnvidia-cfg1-515 libnvidia-common-515
  libnvidia-common-525 libnvidia-compute-515 libnvidia-decode-515 libnvidia-encode-515 libnvidia-extra-515 libnvidia-fbc1-515
  libnvidia-gl-515 libogdi4.1 libopencolorio1v5 libopencv-core4.2 libopencv-imgcodecs4.2 libopencv-imgproc4.2 libopencv-videoio4.2
  libopenimageio2.1 libopenvdb6.2 libosdcpu3.4.0 libosdgpu3.4.0 libpq5 libproj15 libraw19 libsocket++1 libspatialite7 libspnav0 libsquish0
  libsuperlu5 libtbb2 libtinyxml2.6.2v5 liburiparser1 libxerces-c3.2 libyaml-cpp0.6 nvidia-compute-utils-515 nvidia-kernel-common-515
  nvidia-kernel-source-515 nvidia-utils-515 odbcinst odbcinst1debian2 proj-bin proj-data xserver-xorg-video-nvidia-515
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
\Oumayma@\Localhost:~$ export LD_LIBRARY_PATH=/usr/arm-linux-gnueabihf/lib:/lib/arm-linux-gnueabihf
\Oumayma@\Localhost:~$ sudo ln -sf /usr/arm-linux-gnueabihf/lib/libc.so.6 /lib/arm-linux-gnueabihf/
\Oumayma@\Localhost:~$ qemu-arm helloarm
Hello, World!
```

III. Partie de Q/A:

- **Quelles sont les étapes de compilation?**
  1. Prétraitement (Preprocessing): Cette étape traite les directives de préprocesseur (comme `#include`, `#define`, etc).Le préprocesseur génère un fichier de code source modifié qui est ensuite passé à l'étape suivante.
     Commande: gcc -E source.c -o source.i
  2. Compilation: Le compilateur traduit le code source prétraité en code assembleur spécifique à l'architecture du processeur cible. Le fichier résultant est un fichier en langage assembleur (.s ou .asm).
     Commande: gcc -S source.i -o source.s
  3. Assemblage: L'assembleur convertit le code assembleur en code machine, produisant un fichier objet.
     Commande: gcc -c source.s -o source.o
  4. Édition des liens: L'éditeur de liens combine un ou plusieurs fichiers objets en un exécutable ou une bibliothèque.
     Commande: gcc source.o -o executable

- **Définit les bibliothèques statiques et les bibliothèques dynamiques.**
  1. **Les bibliothèques statiques** sont des collections de fonctions et de routines compilées qui sont liées directement aux programmes lors de la phase de compilation. Elles sont intégrées dans l'exécutable final, ce qui signifie que toutes les parties du code nécessaires de la bibliothèque sont copiées dans l'exécutable.
  2. **Les bibliothèques dynamiques**, également appelées bibliothèques partagées, sont des collections de fonctions et de routines qui sont liées aux programmes au moment de l'exécution (dynamique) ou au chargement (runtime). Elles ne sont pas intégrées dans l'exécutable final mais restent indépendantes et sont chargées par le système d'exploitation lors de l'exécution du programme.

- **Qu'est ce qu'un chargeur dynamique?**
  Un chargeur dynamique (ou chargeur de bibliothèques dynamiques) est un composant du système d'exploitation qui gère le chargement et le lien des bibliothèques dynamiques (ou bibliothèques partagées) au moment de l'exécution d'un programme. Il permet à une application de référencer des fonctions et des routines dans des bibliothèques dynamiques sans les inclure directement dans l'exécutable.



Conclusion

Ce rapport a détaillé les étapes nécessaires pour installer et utiliser un compilateur croisé pour ARM sur une machine x86. Les étapes sont illustrées par les captures d'écrans ci-dessus.

Tâche n°2 : Installation et Configuration de Crosstool-NG pour Compiler une Toolchain ARM

Introduction

Ce rapport détaille les étapes nécessaires pour installer et configurer Crosstool-NG, un outil de construction de toolchains croisés, pour créer une toolchain ARM. Chaque commande sera expliquée en détail, et une place sera laissée pour les captures d'écran correspondantes.

I. Mise à Jour et Installation des Dépendances:

1) Mise à jour de la liste des paquets:

Commande: sudo apt-get update

Cette commande met à jour la liste des paquets disponibles pour installation ou mise à jour. Elle synchronise la liste des paquets disponibles à partir des dépôts configurés sur le système.

2) Mise à niveau des paquets installés:

Commande: sudo apt-get upgrade

Cette commande met à niveau tous les paquets installés sur le système vers les versions les plus récentes disponibles, en tenant compte des changements de dépendances.

```
\Oumayma@\Localhost:~$ sudo apt-get upgrade
[sudo] password for yasser:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  blender-data efibootmgr fonts-dejavu gdal-data libarmadillo9 libblosc1 libboost-filesystem1.71.0 libboost-locale1.71.0 libcfitsio8
  libcharls2 libdap25 libdapclient6v5 libdcmtk14 libepsilon1 libfreexl1 libfyba0 libgdal26 libgdcm3.0 libgeos-3.8.0 libgeos-c1v5 libgeotiff5
  libglew2.1 libhdf4-0-alt libjemalloc2 libkmlbase1 libkmldom1 libkmlengine1 libminizip1 libnetcdf15 libnvidia-cfg1-515 libnvidia-common-515
  libnvidia-common-525 libnvidia-compute-515 libnvidia-decode-515 libnvidia-encode-515 libnvidia-extra-515 libnvidia-fbc1-515
  libnvidia-gl-515 libogdi4.1 libopencolorio1v5 libopencv-core4.2 libopencv-imgcodecs4.2 libopencv-imgproc4.2 libopencv-videoio4.2
  libopenimageio2.1 libopenvdb6.2 libosdcpu3.4.0 libosdgpu3.4.0 libpq5 libproj15 libraw19 libsocket++1 libspatialite7 libspnav0 libsquish0
  libsuperlu5 libtbb2 libtinyxml2.6.2v5 liburiparser1 libxerces-c3.2 libyaml-cpp0.6 nvidia-compute-utils-515 nvidia-kernel-common-515
  nvidia-kernel-source-515 nvidia-utils-515 odbcinst odbcinst1debian2 proj-bin proj-data xserver-xorg-video-nvidia-515
Use 'sudo apt autoremove' to remove them.
The following packages will be upgraded:
  libgif7 libtiff5 libtiff5:i386
3 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

3) Installation des paquets nécessaires:

Commande: sudo apt install build-essential git autoconf bison flex texinfo help2man gawk libtool-bin libncurses5-dev unzip python-dev-is-python3

Cette commande installe les outils de développement et les bibliothèques nécessaires pour compiler et configurer Crosstool-NG.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ sudo apt install build-essential git autoconf bison flex texinfo help2man gawk libtoo
l-bin libncurses5-dev unzip python-dev-is-python3
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version (2.69-11.1).
bison is already the newest version (2:3.5.1+dfsg-1).
flex is already the newest version (2.6.4-6.2).
libtool-bin is already the newest version (2.4.6-14).
python-dev-is-python3 is already the newest version (3.8.2-4).
help2man is already the newest version (1.47.13).
texinfo is already the newest version (6.7.0.dfsg.2-5).
build-essential is already the newest version (12.8ubuntu1.1).
gawk is already the newest version (1:5.0.1+dfsg-1ubuntu0.1).
git is already the newest version (1:2.25.1-1ubuntu3.12).
```

II. Clonage et Installation de Crosstool-NG:

1) Clonage du dépôt Crosstool-NG:

Commande: git clone https://github.com/crosstool-ng/crosstool-ng

Cette commande clone le dépôt Git de Crosstool-NG dans le répertoire actuel.J'ai créé un dossier nommé Oumayma_tache2 dans lequel je veux faire le clonage.

```
\Oumayma@\Localhost:~$ cd Oumayma_tache2/
\Oumayma@\Localhost:~/Oumayma_tache2$ ls | grep cross*
crosstool-ng-master
\Oumayma@\Localhost:~/Oumayma_tache2$
```

2) **Navigation vers le répertoire Crosstool-ng-master:**
Commande: cd crosstool-ng/
Cette commande change le répertoire de travail actuel pour le répertoire `crosstool-ng-master` cloné.

```
\Oumayma@\Localhost:~/Oumayma_tache2$ cd crosstool-ng-master/
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$
```

3) **Préparation du build:**
Commande: ./bootstrap
Cette commande initialise les fichiers nécessaires à l'installation de Crosstool-NG.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ls
aclocal.m4        build.log     config.h.in~   configure~    ct-ng      include           licenses.d  Makefile.am  paths.sh.in  stamp-h1
autom4te.cache    config        config.log     configure.ac  ct-ng.in   issue_template.md m4          Makefile.in  README.md    testing
bash-completion   config.h      config.status  contrib       debian     kconfig           maintainer  packages     samples      TODO
bootstrap         config.h.in   configure      COPYING       docs       LICENSE           Makefile    paths.sh     scripts      verbatim-data.mk
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./bootstrap
INFO  :: *** Generating package version descriptions
INFO  :: Master packages: autoconf automake avr-libc binutils bison cloog dtc duma elf2flt expat gcc gdb gettext glibc gmp gnuprumcu isl libelf
 libiconv libtool linux ltrace m4 make mingw-w64 mold moxiebox mpc mpfr musl ncurses newlib-nano newlib picolibc strace uClibc-ng zlib zstd
INFO  :: Generating 'config/versions/autoconf.in'
INFO  :: Generating 'config/versions/automake.in'
INFO  :: Generating 'config/versions/avr-libc.in'
INFO  :: Generating 'config/versions/binutils.in'
INFO  :: Generating 'config/versions/bison.in'
INFO  :: Generating 'config/versions/cloog.in'
INFO  :: Generating 'config/versions/dtc.in'
INFO  :: Generating 'config/versions/duma.in'
INFO  :: Generating 'config/versions/elf2flt.in'
INFO  :: Generating 'config/versions/expat.in'
INFO  :: Generating 'config/versions/gcc.in'
INFO  :: Generating 'config/versions/gdb.in'
```

4) **Configuration du build:**
Commande: ./configure --enable-local
Cette commande configure Crosstool-NG pour une installation locale, permettant d'exécuter l'outil depuis le répertoire actuel.

```
INFO  :: *** Done!
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./configure --enable-local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking how to create a pax tar archive... gnutar
checking whether make supports nested variables... (cached) yes
checking whether to enable maintainer-specific portions of Makefiles... yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking whether ln -s works... yes
```

5) **Compilation de Crosstool-NG:**
Commande: make
Cette commande compile les sources de Crosstool-NG.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ make
/usr/bin/make  all-recursive
make[1]: Entering directory '/home/yasser/Oumayma_tache2/crosstool-ng-master'
Making all in kconfig
make[2]: Entering directory '/home/yasser/Oumayma_tache2/crosstool-ng-master/kconfig'
/usr/bin/make  all-am
make[3]: Entering directory '/home/yasser/Oumayma_tache2/crosstool-ng-master/kconfig'
make[3]: Nothing to be done for 'all-am'.
make[3]: Leaving directory '/home/yasser/Oumayma_tache2/crosstool-ng-master/kconfig'
make[2]: Leaving directory '/home/yasser/Oumayma_tache2/crosstool-ng-master/kconfig'
make[2]: Entering directory '/home/yasser/Oumayma_tache2/crosstool-ng-master'
  GEN      ct-ng
  GEN      bash-completion/ct-ng
  GEN      docs/ct-ng.1
make[2]: Leaving directory '/home/yasser/Oumayma_tache2/crosstool-ng-master'
make[1]: Leaving directory '/home/yasser/Oumayma_tache2/crosstool-ng-master'
```

III. Configuration de Crosstool-NG:

1) **Affichage des configurations disponibles:**
   Commande: `./ct-ng list-samples`
   Cette commande affiche une liste des configurations de toolchain prêtes à l'emploi disponibles dans Crosstool-NG.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./ct-ng list-samples
Status  Sample name
[L...]   aarch64-ol7u9-linux-gnu
[L...]   aarch64-ol8u6-linux-gnu
[L...]   aarch64-ol8u7-linux-gnu
[L...]   aarch64-rpi3-linux-gnu
[L...]   aarch64-rpi4-linux-gnu
[L...]   aarch64-unknown-linux-gnu
[L...]   aarch64-unknown-linux-musl
[L...]   aarch64-unknown-linux-uclibc
[L...]   alphaev56-unknown-linux-gnu
[L...]   alphaev67-unknown-linux-gnu
```

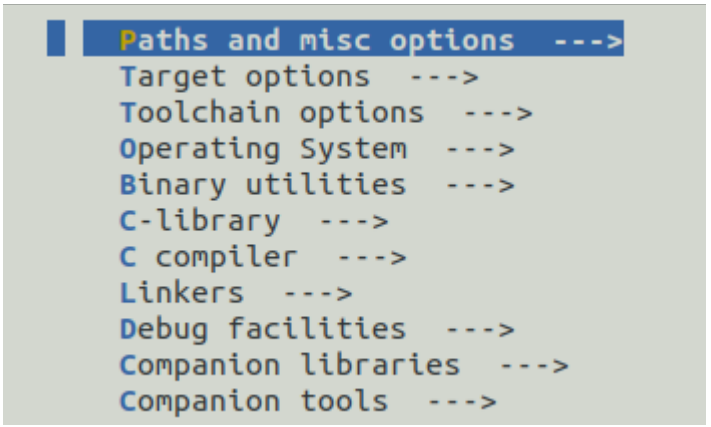
2) **Sélection de la configuration AArch64 pour Raspberry Pi 3:**
   Commande: `./ct-ng aarch64-rpi3-linux-gnu`
   Cette commande sélectionne la configuration prédéfinie pour compiler une toolchain pour un Raspberry Pi 3 avec architecture AArch64.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./ct-ng aarch64-rpi3-linux-gnu
  CONF   aarch64-rpi3-linux-gnu
#
# configuration written to .config
#

*****************************************************

Initially reported by: Bryan Hundven
URL:

Comment:
Raspberry PI 3 aarch64

*****************************************************

Now configured for "aarch64-rpi3-linux-gnu"
```

3) **Lancement de la configuration interactive:**
   Commande: `./ct-ng menuconfig`
   Cette commande lance l'interface de configuration basée sur un menu texte pour personnaliser les options de la toolchain.

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./ct-ng menuconfig
  CONF   menuconfig


*** End of the configuration.
*** Execute 'ct-ng build' to start the build or try 'ct-ng help'.
```

4) **Personnalisation de la Configuration:**
   Nous pouvons définir des paramètres personnalisés en fonction de nos besoins : Target options comme ARM dans notre cas, Toolchain options, Operating System.., etc.



5) **Installation de python-dev-is-python3**
   Commande: sudo apt install python-dev-is-python3

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ sudo apt install python-dev-is-python3
[sudo] password for yasser:
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-dev-is-python3 is already the newest version (3.8.2-4).
The following packages were automatically installed and are no longer required:
  blender-data efibootmgr fonts-dejavu gdal-data libarmadillo9 libblosc1 libboost-filesystem1.71.0 libboost-locale1.71.0 libcfitsio8
  libcharls2 libdap25 libdapclient6v5 libdcmtk14 libepsilon1 libfreexl1 libfyba0 libgdal26 libgdcm3.0 libgeos-3.8.0 libgeos-c1v5 libgeotiff5
  libglew2.1 libhdf4-0-alt libjemalloc2 libkmlbase1 libkmldom1 libkmlengine1 libminizip1 libnetcdf15 libnvidia-cfg1-515 libnvidia-common-515
  libnvidia-common-525 libnvidia-compute-515 libnvidia-decode-515 libnvidia-encode-515 libnvidia-extra-515 libnvidia-fbc1-515
  libnvidia-gl-515 libogdi4.1 libopencolorio1v5 libopencv-core4.2 libopencv-imgcodecs4.2 libopencv-imgproc4.2 libopencv-videoio4.2
  libopenimageio2.1 libopenvdb6.2 libosdcpu3.4.0 libosdgpu3.4.0 libpq5 libproj15 libraw19 libsocket++1 libspatialite7 libspnav0 libsquish0
  libsuperlu5 libtbb2 libtinyxml2.6.2v5 liburiparser1 libxerces-c3.2 libyaml-cpp0.6 nvidia-compute-utils-515 nvidia-kernel-common-515
  nvidia-kernel-source-515 nvidia-utils-515 odbcinst odbcinst1debian2 proj-bin proj-data xserver-xorg-video-nvidia-515
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

IV. Produce the toolchain:

1) **Run**:
   Commande: ./ct-ng build aarch64-rpi3-linux-gnu
   Cette commande est utilisée pour construire une toolchain croisée (cross-toolchain) en utilisant la configuration spécifiée.À ce stade, une erreur est attendue:

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./ct-ng build aarch64-rpi3-linux-gnu
[INFO ]  Performing some trivial sanity checks
[ERROR]  Don't set LD_LIBRARY_PATH. It screws up the build.
[ERROR]
[ERROR]  >>
[ERROR]  >>  Build failed in step '(top-level)'
[ERROR]  >>
[ERROR]  >>  Error happened in: CT_Abort[scripts/functions@489]
[ERROR]  >>        called from: CT_TestAndAbort[scripts/functions@509]
[ERROR]  >>        called from: main[scripts/crosstool-NG.sh@58]
```

Résolution d' Erreur:

Pour résoudre cela, nous devons faire un Reset a la variable d'environnement que nous avons utilisé dans la tâche 1.

```
[00:00] / make: *** [ct-ng:261: build] Error 1
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ unset LD_LIBRARY_PATH
```

2) **Rerun:**

```
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ unset LD_LIBRARY_PATH
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$ ./ct-ng build aarch64-rpi3-linux-gnu
[INFO ]  Performing some trivial sanity checks
[WARN ]  Number of open files 1024 may not be sufficient to build the toolchain; increasing to 2048
[INFO ]  Build started 20240611.100154
[INFO ]  Building environment variables
[WARN ]  Directory '/home/yasser/src' does not exist.
[WARN ]  Will not save downloaded tarballs to local storage.
[EXTRA]  Preparing working directories
[EXTRA]  Installing user-supplied crosstool-NG configuration
[EXTRA]  ===================================================================
[EXTRA]  Dumping internal crosstool-NG configuration
[EXTRA]    Building a toolchain for:
[EXTRA]      build  = x86_64-pc-linux-gnu
[EXTRA]      host   = x86_64-pc-linux-gnu
[EXTRA]      target = aarch64-rpi3-linux-gnu
[EXTRA]  Dumping internal crosstool-NG configuration: done in 0.08s (at 00:02)
[INFO ]  ===================================================================
[INFO ]  Retrieving needed toolchain components' tarballs
[INFO ]  Retrieving needed toolchain components' tarballs: done in 0.46s (at 00:02)
[INFO ]  ===================================================================
[INFO ]  Extracting and patching toolchain components
[INFO ]  Extracting and patching toolchain components: done in 0.55s (at 00:03)
```

et ça fonctionne et le build de toolchain a commencé.

```
ol-ng-master/.build/aarch64-elf/src
[EXTRA]    Put the license information to: /home/yasser/x-tools/aarch64-elf/shar
e/licenses
[INFO ]  Finalizing the toolchain's directory: done in 5.57s (at 75:26)
[INFO ]  Build completed at 20240614.020431
[INFO ]  (elapsed: 75:25.13)
[INFO ]  Finishing installation (may take a few seconds)...
[75:26] /   CONF  aarch64-rpi3-linux-gnu
#
# configuration written to .config
#

*****************************************************

Initially reported by: Bryan Hundven
URL:

Comment:
Raspberry PI 3 aarch64

*****************************************************

Now configured for "aarch64-rpi3-linux-gnu"
\Oumayma@\Localhost:~/Oumayma_tache2/crosstool-ng-master$
```

et maintenant la Toolchain est prête.

V. Partie de Q/A:

- **Qu'est ce qu'un toolchain?**
  Un toolchain, ou chaîne d'outils, est un ensemble d'outils logiciels utilisés pour développer des applications, généralement pour des systèmes embarqués ou des plateformes spécifiques. Chaque outil dans une toolchain a une fonction particulière dans le processus de développement, allant de la compilation du code source à la création de l'exécutable final. Les toolchains sont cruciaux pour le développement d'applications qui doivent être exécutées sur des architectures matérielles spécifiques, souvent différentes de l'architecture de développement.

- **Quels sont les outils de toolchain?**
  Une toolchain typique comprend plusieurs composants essentiels :

  1. Compilateur (Compiler)
     - **Rôle :** Convertir le code source écrit dans un langage de programmation (comme C, C++, etc.) en code machine ou en un langage intermédiaire.
     - **Exemple :** `gcc` (GNU Compiler Collection)
  2. Assembleur (Assembler)
     - **Rôle :** Convertir le code assembleur en code machine.
     - **Exemple :** `as` (GNU Assembler)
  3. Éditeur de Liens (Linker)
     - **Rôle :** Combine un ou plusieurs fichiers objets et bibliothèques en un seul exécutable ou une bibliothèque partagée.
     - **Exemple :** `ld` (GNU Linker)
  4. Bibliothèques (Libraries)
     - **Rôle :** Fournit des collections de fonctions et de routines pré-compilées que les programmes peuvent utiliser.
     - **Exemple :** `libc` (la bibliothèque standard C)
  5. Débogueur (Debugger)
     - **Rôle :** Permet de tester et de déboguer le programme en cours de développement.
     - **Exemple :** `gdb` (GNU Debugger)
  6. Outils de Construction (Build Tools)
     - **Rôle :** Automatisent le processus de compilation et de lien.
     - **Exemple :** `make` (GNU Make)
  7. Autres Utilitaires (Other Utilities)
     - **Rôle :** Incluent divers outils pour des tâches spécifiques comme l'optimisation du code, l'analyse statique, etc.
     - **Exemple :** `objcopy`, `strip`, `nm`

- **Qu'est une variable d'environnement?**
  Définition: Les variables d'environnement constituent un moyen d'influencer le comportement des logiciels sur votre système
  Ajouter une variable d'environnement d'un façon temporaire
  Pour ajouter une variable d'environnement de manière temporaire (valide seulement pour la session actuelle du terminal), utilisez la commande `export` :
  Commande: export NOM_VARIABLE=valeur

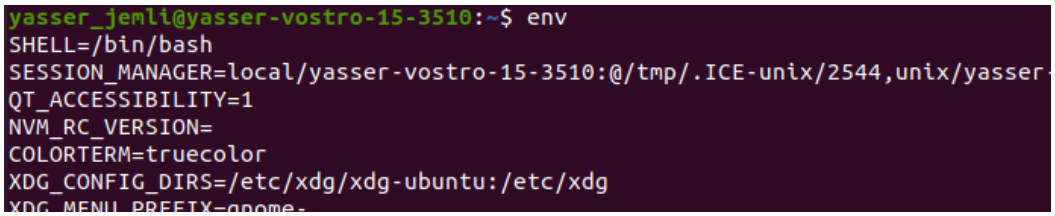
  Ajouter une variable d'environnement d'un façon permanente
  Pour ajouter une variable d'environnement de manière permanente, vous devez modifier les fichiers de configuration du shell. Les fichiers couramment modifiés sont `.bashrc`,

`.bash_profile`, ou `.profile` (pour Bash).
Commandes: nano ~/.bashrc

export NOM_VARIABLE=valeur

source ~/.bashrc

Lister les variable d'environnement
Pour lister toutes les variables d'environnement, utilisez la commande `printenv` ou `env`**:**



Définir la variable PATH
La variable PATH est une variable d'environnement qui spécifie une liste de répertoires que le système d'exploitation doit rechercher pour trouver des exécutables lorsque des commandes sont exécutées.Pour ajouter un répertoire à la variable `PATH` :
Commande: export PATH=$PATH:/nouveau/répertoire
A quoi sert LD_LIBRARY_PATH?
`LD_LIBRARY_PATH` est une variable d'environnement utilisée sous Unix et Linux pour spécifie une liste de répertoires où le chargeur de bibliothèques dynamiques doit chercher des bibliothèques partagées avant de chercher dans les répertoires par défaut (`/lib` et `/usr/lib`).
Commande:
export LD_LIBRARY_PATH=/chemin/vers/lib1:/chemin/vers/lib2:$LD_LIBRARY_PATH

Conclusion

Ce rapport a détaillé les étapes nécessaires pour installer et configurer Crosstool-NG afin de créer une toolchain ARM personnalisée. Les captures d'écran de chaque étape sont fournies ci-dessus.

Tâche n°3 : Adding a System Call to Linux Kernel

Introduction

Ce rapport explore le processus d'ajout d'un nouvel appel système au noyau Linux. En détaillant les étapes nécessaires depuis la modification du code source jusqu'à la validation du nouvel appel système, il vise à fournir une compréhension claire et pratique de cette procédure cruciale pour l'extension des fonctionnalités du système d'exploitation.

I. Préparation de l'Environnement de Développement

1) Mise à jour et mise à niveau des paquets système

Commande: `sudo apt update && sudo apt upgrade`

Cette commande met à jour la liste des paquets disponibles pour installation ou mise à jour. Elle synchronise la liste des paquets disponibles à partir des dépôts configurés sur le système.

2) Installation des outils de développement nécessaires

Commande:
```
sudo apt install gcc build-essential libncurses-dev libssl-dev libelf-dev bison flex -y
sudo apt install dwarves
sudo apt clean && sudo apt autoremove -y
```

Ces commandes permettent d'installer les outils de compilation et les dépendances nécessaires pour construire le noyau Linux.

II. Téléchargement et Extraction du Noyau Linux

1) Téléchargement du noyau Linux version 5.19.1

Commande:
```
wget -P ~/ https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.19.1.tar.xz
tar -xvf ~/linux-5.19.1.tar.xz -C ~/
```

Cette commande télécharge le noyau Linux version 5.19.1 et l'extrait dans le répertoire personnel de l'utilisateur(home directory) mais dans ce cas j'ai changer le path d'extraction mentionné ci-dessous:

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3$ ls
linux-5.19.1
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3$ cd linux-5.19.1/
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ ls
arch            helloworld  MAINTAINERS              README      vmlinux
block           include     Makefile                 samples     vmlinux-gdb.py
certs           init        mm                       scripts     vmlinux.map
COPYING         ipc         modules.builtin          security    vmlinux.o
CREDITS         Kbuild      modules.builtin.modinfo  sound       vmlinux.symvers
crypto          Kconfig     modules-only.symvers     System.map
Documentation   kernel      modules.order            tools
drivers         lib         Module.symvers           usr
fs              LICENSES    net                      virt
```

III. Configuration et Installation du Noyau

1) Redémarrage du système pour utiliser le nouveau noyau

Commande: `reboot`

Cette commande redémarre le PC pour appliquer les modifications du noyau.

2) Configuration du noyau

Commande:
```
cd linux-5.19.1
mkdir helloworld
gedit helloworld/helloworld.c
```

Ces commandes permettent de naviguer vers le répertoire du noyau, de créer un répertoire pour le nouvel appel système et d'ouvrir un éditeur pour écrire le code de l'appel système.

3) Édition du `Makefile` dans `helloworld`

Commande: `gedit helloworld/Makefile`

Cette commande ouvre le fichier Makefile pour configurer la compilation du nouvel appel système.

4) **Modification du `Makefile` principal**
   Commande: gedit Makefile
   Cette commande ouvre le Makefile principal pour ajouter le nouvel appel système à la liste des modules à compiler.

IV. Modification des Fichiers Système Requis

1) **Édition de include/linux/syscalls.h**
   Commande: gedit include/linux/syscalls.h
   Cette commande permet d'ouvrir le fichier syscalls.h afin de le modifier.
2) **Ajout avant #endif**
   asmlinkage long sys_helloworld(void);
   Cette ligne déclare le nouvel appel système.
3) **Édition de arch/x86/entry/syscalls/syscall_64.tbl**
   Commande: gedit arch/x86/entry/syscalls/syscall_64.tbl
   Cette commande permet d'ouvrir le fichier syscall_64.tbl afin de le modifier.
4) **Ajout à la fin de la section appropriée**
   451 common helloworld sys_helloworld
   Cette ligne ajoute le nouvel appel système à la table des appels système pour l'architecture x86_64.

V. Compilation

1) **Configuration du noyau via l'interface graphique**
   Commande: sudo make menuconfig
   Cette commande lance une interface graphique pour configurer le noyau.

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ sudo make menuconfig


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
```

2) **Édition du fichier `.config`**
   Commande: gedit .config
   Cette commande ouvre le fichier de configuration du noyau pour toute modification nécessaire.

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ gedit .co
.cocciconfig  .config
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ gedit .config
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ sudo mak
make                        makeinfo                    make_strings
make-first-existing-target  make-ssl-cert               mako-render
```

3) **Compilation du noyau et des modules**
   Commande: sudo make -j8
   sudo make modules_install -j8
   sudo make install -j4
   Ces commandes compilent le noyau et ses modules, puis installent le noyau compilé.

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19.1$ sudo make -j8
  DESCEND objtool
  DESCEND bpf/resolve_btfids
  HOSTCC  /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/fixdep.o
  HOSTCC  /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/fixdep.o
  HOSTLD  /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/fixdep-in.o
  HOSTLD  /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/fixdep-in.o
  LINK    /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/fixdep
  LINK    /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/fixdep
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/main.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/rbtree.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/zalloc.o
  CALL    scripts/atomic/check-atomics.sh
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/bpf/resolve_btfids/libsubcmd/exec-cmd.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/exec-cmd.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/help.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/arch/x86/special.o
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/pager.o
  CALL    scripts/checksyscalls.sh
  CC      /home/yasser_jemli/embedded_operating_system/task_3/linux-5.19.1/tools/objtool/arch/x86/decode.o
```

fin de compilation est comme suit:

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$ sudo make install -j4
[sudo] password for yasser_jemli:
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/dkms 5.19.1 /boot/vmlinuz-5.19.1
 * dkms: running auto installation service for kernel 5.19.1
Kernel preparation unnecessary for this kernel.  Skipping...
```

et finalement notre noyau est prêt:

```
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$
```

```
Found linux image: /boot/vmlinuz-5.15.0-92-generic
Found initrd image: /boot/initrd.img-5.15.0-92-generic
Found linux image: /boot/vmlinuz-5.19.1
Found initrd image: /boot/initrd.img-5.19.1
Adding boot menu entry for UEFI Firmware Settings
done
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$
```

VI. Validation du Nouvel Appel Système

1) Sélection du nouveau noyau via Grub dans les options avancées

Commande: `sudo update-grub`
`sudo shutdown -r now`

Ces commandes mettent à jour le gestionnaire de démarrage Grub et redémarrent le système. Mais avant de mettre à jour Grub, je souhaite modifier certains paramètres dans le fichier de configuration de Grub. Je vais définir `GRUB_TIMEOUT_STYLE` sur `menu` et définir `GRUB_TIMEOUT` pour spécifier un délai d'attente, permettant ainsi à l'utilisateur de choisir le noyau qu'il souhaite utiliser.

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$ sudo nano /etc/default/grub
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$
```

```
GRUB_DEFAULT=saved
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$ sudo update-grub
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Sourcing file `/etc/default/grub.d/oem-flavour.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.15.0-107-generic
Found initrd image: /boot/initrd.img-5.15.0-107-generic
Found linux image: /boot/vmlinuz-5.15.0-92-generic
Found initrd image: /boot/initrd.img-5.15.0-92-generic
Found linux image: /boot/vmlinuz-5.19.1
Found initrd image: /boot/initrd.img-5.19.1
Adding boot menu entry for UEFI Firmware Settings
done
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$
```

et maintenant on va arrêter le PC pour booter sur le nouveau kernel:

```
done
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_3/linux-5.19
.1$ sudo shutdown -r now
```

VII. Création et Exécution d'un Programme Utilisateur

1) **Création d'un programme en C pour tester le nouvel appel système**
   Commande: `nano programme.c`
   Cette commande ouvre un éditeur pour écrire un programme C testant le nouvel appel système.
2) **Compilation et exécution du programme**
   Commande: `gcc programme.c`
   `./a.out`
   Ces commandes compilent et exécutent le programme de test.

VIII. Vérification des Messages du Noyau

1) **Vérification des messages du noyau pour voir la sortie de l'appel système**
   Commande: `dmesg`
   Cette commande affiche les messages du noyau pour vérifier la sortie de l'appel système.

```
oumayma@oumayma-VirtualBox:~/embedded_operating_system/task_3/test_sysca
ll$ sudo dmesg
[sudo] Mot de passe de oumayma :
[    0.000000] Linux version 5.15.441000 (root@oumayma-VirtualBox) (gcc
(Ubuntu 13.2.0-23ubuntu4) 13.2.0, GNU ld (GNU Binutils for Ubuntu) 2.42)
 #2 SMP Mon Jul 1 17:58:35 CET 2024
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.441000 root=U
UID=c3185fb2-f5be-459c-9455-a36907c5f309 ro recovery nomodeset dis_ucode
_ldr
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
IED" operation="sendmsg" profile="snap-update-ns.firefox" pid=2871 comm=
"5" family="unix" sock_type="stream" protocol=0 requested_mask="send" de
nied_mask="send"
[  115.036362] audit: type=1400 audit(1719854026.449:265): apparmor="DEN
IED" operation="open" profile="snap-update-ns.firefox" name="/proc/2877/
maps" pid=2877 comm="5" requested_mask="r" denied_mask="r" fsuid=1000 ou
id=0
[  432.683205] Hello World.
oumayma@oumayma-VirtualBox:~/embedded_operating_system/task_3/test_sysca
ll$
```

2) **Nettoyage des fichiers temporaires après l'installation**

Commande: sudo make clean -j4

Cette commande nettoie les fichiers temporaires créés pendant la compilation.J'ai utiliser un seul core dans ce cas comme mentionné ci-dessous:

```
oumayma@oumayma-VirtualBox:~/embedded_operating_system/task_3/linux-5.15
.44$ sudo make clean -j1
  CLEAN   arch/x86/entry/vdso
  CLEAN   arch/x86/kernel/cpu
  CLEAN   arch/x86/kernel
  CLEAN   arch/x86/purgatory
  CLEAN   arch/x86/realmode/rm
  CLEAN   arch/x86/lib
```

IX. Partie de Q/A:

Qu'est ce qu'un appel système? et quels sont les types les plus utilisés?

Les appels système sont des interfaces fournies par le noyau d'un système d'exploitation pour permettre aux programmes en espace utilisateur d'interagir avec les ressources matérielles et logicielles du système. Voici les principaux types d'appels système, classés en fonction de leur fonctionnalité :

1. Gestion des fichiers

- open : Ouvrir un fichier.
- close : Fermer un fichier.
- read : Lire à partir d'un fichier.
- write : Écrire dans un fichier.
- lseek : Déplacer le pointeur de fichier.
- stat : Obtenir les informations sur un fichier.
- unlink : Supprimer un fichier.
- mkdir : Créer un répertoire.

- rmdir : Supprimer un répertoire.

2. Gestion des processus

- fork : Créer un nouveau processus en copiant le processus actuel.
- execve : Remplacer l'image du processus courant par une nouvelle image de processus.
- exit : Terminer un processus.
- wait : Attendre la fin d'un processus enfant.
- getpid : Obtenir l'ID du processus courant.
- getppid : Obtenir l'ID du processus parent.
- kill : Envoyer un signal à un processus.
- nice : Changer la priorité d'un processus.

3. Gestion de la mémoire

- brk : Modifier la fin de la zone de données du processus.
- mmap : Mapper des fichiers ou des dispositifs en mémoire.
- munmap : Démapper des fichiers ou des dispositifs de la mémoire.

4. Gestion des communications inter-processus (IPC)

- pipe : Créer un canal de communication entre deux processus.
- shmget : Allouer un segment de mémoire partagée.
- shmat : Attacher un segment de mémoire partagée.
- shmdt : Détacher un segment de mémoire partagée.
- msgget : Créer ou obtenir l'accès à une file de messages.
- msgsnd : Envoyer un message à une file de messages.
- msgrcv : Recevoir un message d'une file de messages.
- semget : Créer ou obtenir l'accès à un ensemble de sémaphores.
- semop : Effectuer des opérations sur un sémaphore.

5. Gestion des périphériques

- ioctl : Contrôler des périphériques.
- read : Lire à partir d'un périphérique.
- write : Écrire dans un périphérique.

6. Gestion des réseaux

- socket : Créer un point de communication réseau.
- bind : Associer une adresse à un socket.
- listen : Mettre un socket en mode écoute.
- accept : Accepter une connexion entrante sur un socket.
- connect : Établir une connexion à un socket distant.
- send : Envoyer des données sur un socket.
- recv : Recevoir des données sur un socket.
- shutdown : Fermer une partie ou l'ensemble d'une connexion socket.

7. Gestion des informations du système

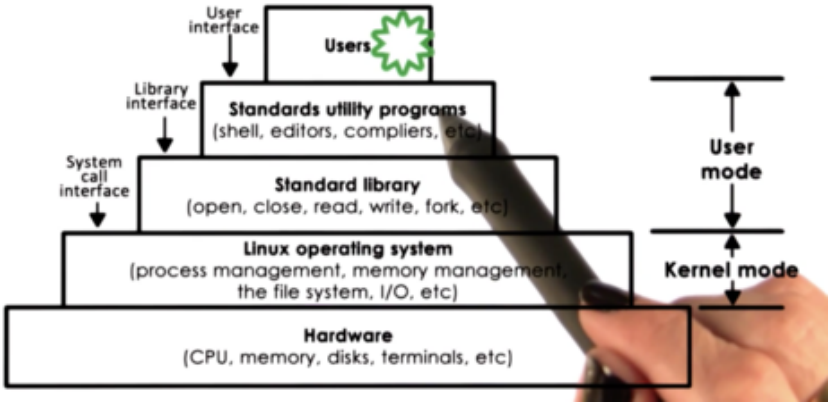
- uname : Obtenir des informations sur le système d'exploitation.
- gettimeofday : Obtenir l'heure actuelle.
- settimeofday : Définir l'heure actuelle.

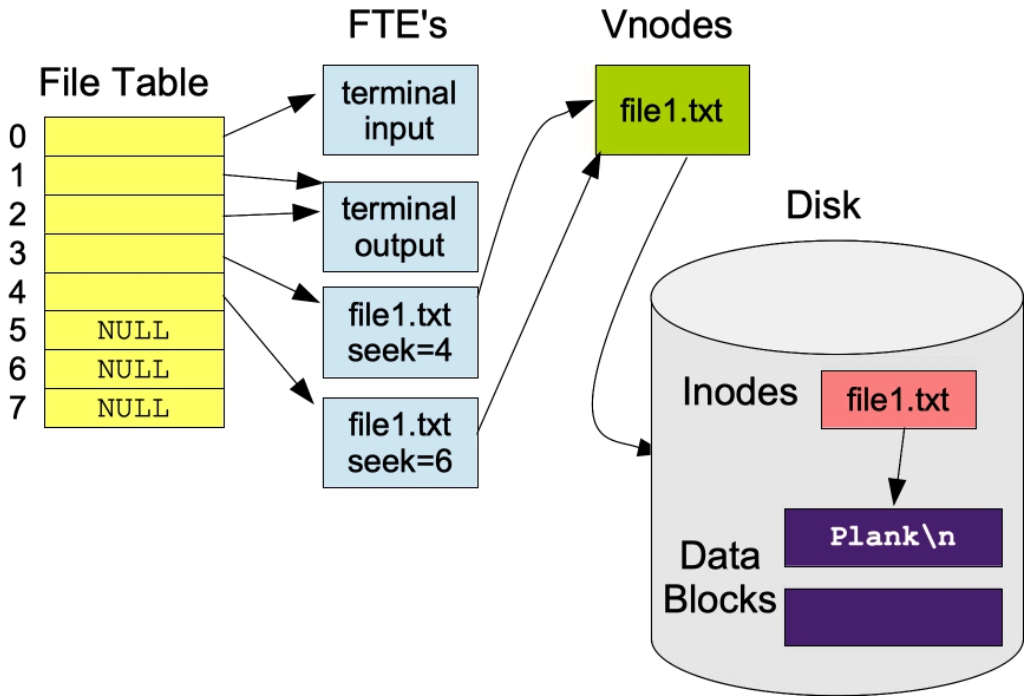
- sysinfo : Obtenir des informations sur le système (mémoire, chargement, etc.).

8. Gestion des utilisateurs et des groupes

- getuid : Obtenir l'ID utilisateur réel du processus courant.
- geteuid : Obtenir l'ID utilisateur effectif du processus courant.
- getgid : Obtenir l'ID de groupe réel du processus courant.
- getegid : Obtenir l'ID de groupe effectif du processus courant.
- setuid : Définir l'ID utilisateur réel ou effectif du processus courant.
- setgid : Définir l'ID de groupe réel ou effectif du processus courant.

Explication de processus from user interface till request from kernel and sending response to user:





Conclusion

Ce rapport a détaillé le processus d'ajout d'un nouvel appel système au noyau Linux, de la préparation de l'environnement de développement à la validation finale. En suivant les étapes décrites, y compris la mise à jour des outils, la configuration, la compilation et les modifications des fichiers système, nous avons montré comment étendre les fonctionnalités du noyau de manière méthodique. Cette procédure permet aux développeurs de personnaliser et d'améliorer le noyau Linux selon leurs besoins spécifiques.

Tâche n°4 : Writing a simple hello world kernel module on Linux

Introduction

Ce rapport décrit le processus de création d'un module de noyau Linux simple “Hello World” sur une distribution Ubuntu. Ce module imprimera un message lors de son chargement et de son déchargement du noyau, démontrant ainsi les bases du développement de modules de noyau.

I. Préparation de l'Environnement de Développement

1) Installation des paquets nécessaires

Commande: `sudo apt-get install build-essential linux-headers-`uname -r``
Pour le développement de modules de noyau, nous avons besoin des paquets build-essential et des en-têtes du noyau Linux correspondants. Ces paquets peuvent être installés sur Ubuntu.

```
oumayma@oumayma-VirtualBox:~/embedded_operating_system/task_4$ sudo apt-
get install build-essential linux-headers-5.15.441000
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
```

2) Passage en mode super-utilisateur

Commande : `sudo -i`
Pour éviter les erreurs de permission lors des opérations suivantes, nous nous connectons en tant que root.

II. Création du Module de Noyau

1) Création du répertoire source

Commande : `mkdir kernel_modules`
`cd kernel_modules`
Nous créons un nouveau répertoire source pour placer le module de noyau "Hello World".

2) Création du fichier source du module

Commande : `touch hello_world_mod.c`

```
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4#
mkdir kernel_modules
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4#
cd kernel_modules/
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4/k
ernel_modules# touch hello_world_mod.c
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4/k
ernel_modules# gedit hello_world_mod.c
```

Nous créons le fichier source du module nommé `hello_world_mod.c`. Voici le contenu de ce fichier:

```c
MODULE_AUTHOR("Bharath Reddy");
MODULE_DESCRIPTION("A simple hello world module");
MODULE_VERSION("0.01");

static int __init hello_mod_init(void)
{
        printk(KERN_ALERT "Hello world from kernel!! \n");
        return 0;
}

static void __exit hello_mod_exit(void)
{
        printk(KERN_ALERT "Exiting hello world module from kernel !!!\n>
}

module_init(hello_mod_init);
module_exit(hello_mod_exit);
```

Ce module contient deux fonctions : `hello_init` et `hello_exit`, qui seront appelées respectivement lors du chargement et du déchargement du module.

3) Création du Makefile

Commande : touch Makefile
Nous créons le fichier Makefile dans le répertoire source. Ce fichier est nécessaire pour construire le module de noyau et générer le fichier .ko (kernel object). Voici le contenu de ce fichier:

```
obj-m += hello_world_mod.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

4) Compilation du Module

Commande : make all
Pour générer le fichier hello_world_mod.ko, nous exécutons la commande ci-dessus dans le répertoire source du module.

```
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4/k
ernel_modules# make all
make -C /lib/modules/5.15.441000/build M=/home/oumayma/embedded_operatin
g_system/task_4/kernel_modules modules
make[1] : on entre dans le répertoire « /home/oumayma/embedded_operating
_system/task_3/linux-5.15.44 »
  CC [M]  /home/oumayma/embedded_operating_system/task_4/kernel_modules/
hello_world_mod.o
  MODPOST /home/oumayma/embedded_operating_system/task_4/kernel_modules/
Module.symvers
  CC [M]  /home/oumayma/embedded_operating_system/task_4/kernel_modules/
hello_world_mod.mod.o
  LD [M]  /home/oumayma/embedded_operating_system/task_4/kernel_modules/
hello_world_mod.ko
make[1] : on quitte le répertoire « /home/oumayma/embedded_operating_sys
tem/task_3/linux-5.15.44 »
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4/k
ernel_modules#
```

III. Chargement et Vérification du Module

1) Effacer les logs dmesg

Commande : `dmesg -C`
Nous effaçons les logs dmesg pour ne voir que les derniers messages provenant de notre module.

2) Insertion du Module

Commande : `insmod hello_world_mod.ko`
Nous insérons le module "Hello World" dans le noyau.

3) Vérification du chargement du Module

Commande : `lsmod | grep "hello"`
Nous vérifions si le module est correctement chargé dans le noyau. La sortie devrait indiquer que `hello_world_mod` est chargé.

4) Vérification des logs dmesg

Commande : `dmesg`
Nous vérifions les logs dmesg pour voir le message de la fonction d'initialisation du module. Voici la sortie:

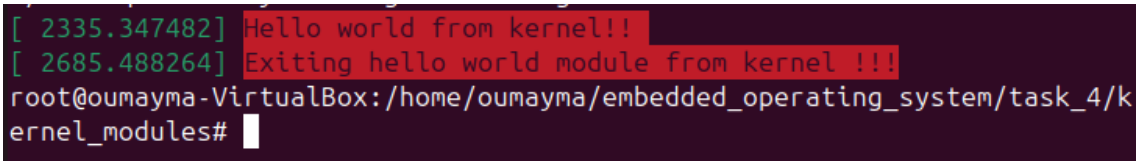
```
[ 2335.347482] Hello world from kernel!!
root@oumayma-VirtualBox:/home/oumayma/embedded_operating_system/task_4/k
ernel_modules#
```

5) Suppression du Module

Commande : `rmmod hello_world_mod`
Nous supprimons le module "Hello World" du noyau.

6) Vérification des logs dmesg après suppression

Commande : `dmesg`
Nous vérifions à nouveau les logs dmesg pour voir le message de la fonction de sortie du module.La sortie devrait contenir :



IV. Partie de Q/A:

Définition des modules de noyaux
Les modules de noyau sont des morceaux de code qui peuvent être chargés et déchargés dans le noyau Linux à volonté. Ils permettent au noyau d'étendre ses fonctionnalités sans avoir besoin de recompiler ou de redémarrer le système. Les modules peuvent inclure des pilotes de périphériques, des systèmes de fichiers ou d'autres fonctionnalités.

Gestion des Modules de Noyau

1) *Chargement des Modules*
   - `insmod` : Utilisé pour insérer (charger) un module dans le noyau; exp:
     `sudo insmod nom_du_module.ko`
   - `modprobe` : Utilisé pour insérer un module ainsi que ses dépendances. Il est préférable d'utiliser modprobe car il gère les dépendances automatiquement; exp:
     `sudo modprobe nom_du_module`

2) *Suppression des Modules*
   - `rmmod` : Utilisé pour supprimer (décharger) un module du noyau; exp:
     `sudo rmmod nom_du_module`
   - `modprobe -r` : Utilisé pour supprimer un module ainsi que ses dépendances; exp:
     `sudo modprobe -r nom_du_module`

3) *Affichage des Modules Chargés*
   - `lsmod` : Affiche la liste des modules actuellement chargés dans le noyau.

4) *Informations sur les Modules*
   - `modinfo` : Affiche les informations sur un module spécifié, comme la version, la licence, les dépendances, etc; exp:
     `modinfo nom_du_module`

Conclusion

Ce rapport décrit en détail le processus de création d'un module de noyau Linux simple "Hello World". En suivant les étapes d'installation des outils nécessaires, de création et de compilation du module, et en utilisant des commandes pour charger et décharger le module, nous avons démontré les bases du développement de modules de noyau sur Linux. Cette expérience fournit une base solide pour explorer des fonctionnalités de noyau plus avancées.

Tâche n°5 : Creating a Minimal Linux System for the BeagleBone Black with Buildroot

Introduction

Ce rapport détaille le processus de création d'un système Linux minimal pour le BeagleBone Black en utilisant Buildroot, un système de construction largement utilisé dans le domaine de Linux embarqué. Ce projet s'inscrit dans le cadre d'une étude indépendante sur le développement Linux embarqué.

I. Préparation de l'Environnement de Développement

1) Récupération de Buildroot

Commande : git clone https://git.buildroot.net/buildroot
cd buildroot git
checkout 2022.11

Tout d'abord, clonons le dépôt Git de Buildroot et passons à la version la plus récente qui est dans ce cas 2022.11

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5$ git clone https://github.com/buildroot/buildroot.git
Cloning into 'buildroot'...
remote: Enumerating objects: 547588, done.
remote: Counting objects: 100% (768/768), done.
remote: Compressing objects: 100% (435/435), done.
remote: Total 547588 (delta 351), reused 653 (delta 329), pack-reused 546820
Receiving objects: 100% (547588/547588), 140.24 MiB | 929.00 KiB/s, done.
Resolving deltas: 100% (352429/352429), done.
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5$ ls
10.png  11.png  12.png  13.png  14.png  1.png  2.png  4.png  5.png  6.png  7.png  8.png  9.png  buildroot
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5$ cd buildroot/
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5/buildroot$
```

maintenant on va passer à la version 2022.11:

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5/buildroot$ git checkout 2022.11
Note: switching to '2022.11'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 40bd4a32aa Update for 2022.11
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5/buildroot$
```

2) Configuration de Buildroot

Commande : make menuconfig

Buildroot utilise `menuconfig` pour la configuration, similaire à celle du noyau Linux et de crosstool. Exécutions la commande suivante pour démarrer la configuration.

Suivons les instructions de la documentation de formation de Buildroot avec les ajustements nécessaires pour un BeagleBone non sans fil, utilisant un arbre de périphérique différent.

3) Configuration Principale

Options Cible :

- Sélectionner ARM (little endian) comme architecture cible.
- Choisir cortex-A8 comme variante d'architecture cible.

```
    Target Architecture (ARM (little endian))  --->
    Target Architecture Variant (cortex-A8)  --->
    Target ABI (EABIhf)  --->
    Floating point strategy (VFPv3-D16)  --->
    ARM instruction set (ARM)  --->
    Target Binary Format (ELF)  --->
```

Chaîne d'Outils :

- Utiliser une chaîne d'outils externe de type ARM 2021.07 pour réduire le temps de compilation.

```
    Toolchain type (External toolchain)  --->
    *** Toolchain External Options ***
    Toolchain (Arm ARM 2021.07)  --->
    Toolchain origin (Toolchain to be downloaded and installed)  --->
[ ] Copy gdb server to the Target (NEW)
    *** Host GDB Options ***
[ ] Build cross gdb for the host (NEW)
    *** Toolchain Generic Options ***
[ ] Copy gconv libraries (NEW)
()  Extra toolchain libraries to be copied to target (NEW)
()  Target Optimizations (NEW)
()  Target linker options (NEW)
```

Configuration Système :

- Définir le nom d'hôte système comme `billbeaglebone`.
- Définir une bannière système, par exemple "Oumayma Embedded Linux".
- Activer la connexion root avec mot de passe.

```
    Root FS skeleton (default target skeleton)  --->
(billbeaglebone) System hostname
(Oumayma's Embedded Linux!) System banner
    Passwords encoding (sha-256)  --->
    Init system (BusyBox)  --->
    /dev management (Dynamic using devtmpfs only)  --->
(system/device_table.txt) Path to the permission tables (NEW)
[ ] support extended attributes in device tables (NEW)
[ ] Use symlinks to /usr for /bin, /sbin and /lib (NEW)
[*] Enable root login with password (NEW)
(12832340) Root password
    /bin/sh (busybox' default shell)  --->
```

Noyau :

- Activer le noyau Linux avec la version par défaut (6.0).
- Utiliser `omap2plus` comme nom de configuration par défaut.
- Utiliser `zImage` comme format binaire du noyau.
- Spécifier le fichier source de l'arbre de périphérique in-tree comme `am335x-boneblack`.
- Activer "Needs host OpenSSL".

```
[*] Linux Kernel
      Kernel version (Latest version (6.0))  --->
()    Custom kernel patches (NEW)
      Kernel configuration (Using an in-tree defconfig file)  --->
(omap2plus) Defconfig name
()    Additional configuration fragment files (NEW)
()    Custom boot logo file path (NEW)
      Kernel binary format (zImage)  --->
      Kernel compression format (gzip compression)  --->
[*]   Build a Device Tree Blob (DTB)
[ ]     DTB is built by kernel itself (NEW)
(am335x-boneblack) In-tree Device Tree Source file names
()      Out-of-tree Device Tree Source file paths (NEW)
[ ]     Keep the directory name of the Device Tree (NEW)
[ ]     Build Device Tree with overlay support (NEW)
[ ]   Install kernel image to /boot in target (NEW)
[*]   Needs host OpenSSL
[ ]   Needs host libelf (NEW)
[ ]   Needs host pahole (NEW)
      Linux Kernel Extensions  --->
      Linux Kernel Tools  --->
```

Paquets Cibles :

- Activer BusyBox.
- Facultatif : Activer GDB, gdbserver et Valgrind pour le débogage et le profilage.
- Facultatif : Activer dropbear pour les applications réseau.

```
-*- BusyBox
(package/busybox/busybox.config) BusyBox configuration file to use?
()    Additional BusyBox configuration fragment files (NEW)
[ ]   Show packages that are also provided by busybox (NEW)
[ ]   Individual binaries (NEW)
[ ]   Install the watchdog daemon startup script (NEW)
    Audio and video applications  --->
    Compressors and decompressors  --->
    Debugging, profiling and benchmark  --->
    Development tools  --->
    Filesystem and flash utilities  --->
    Fonts, cursors, icons, sounds and themes  --->
    Games  --->
```

Chargeurs de Démarrage :

- Activer U-Boot.
- Configurer U-Boot pour `am335x_evm`.
- Utiliser la version U-Boot 2022.04.
- Activer seulement `u-boot.img` comme format binaire U-Boot.
- Installer l'image binaire U-Boot SPL avec le nom `MLO`.

```
[*] U-Boot
      Build system (Kconfig)  --->
      U-Boot Version (2022.04)  --->
()    Custom U-Boot patches
      U-Boot configuration (Using an in-tree board defconfig file)  --->
(am335x_evm) Board defconfig
()    Additional configuration fragment files (NEW)
[ ]   U-Boot needs dtc (NEW)
[ ]   U-Boot needs host python 3.x (NEW)
[ ]   U-Boot needs pylibfdt (NEW)
[ ]   U-Boot needs pyelftools (NEW)
[ ]   U-Boot needs OpenSSL (NEW)
[ ]   U-Boot needs lzop (NEW)
[ ]   U-Boot needs gnutls (NEW)
[ ]   U-Boot needs util-linux (NEW)
      U-Boot binary format  --->
[ ]   produce a .ift signed image (OMAP) (NEW)
[*]   Install U-Boot SPL binary image
(MLO)   U-Boot SPL/TPL binary image name(s)
```

Sauvegarder et quitter après avoir terminé la configuration.

4) Construction du Système

Commande : make 2>&1 | tee build.log
La première construction peut prendre un certain temps. Une fois terminée et après avoir résolu les erreurs éventuelles (comme l'installation de Perl), les fichiers de sortie seront disponibles dans le répertoire `output/images`.

```
yasser_jemli@yasser-vostro-15-3510:~/embedded_operating_system/task_5/buildroot$ make 2>&1 | tee build.log
/usr/bin/make -j1 O=/home/yasser_jemli/embedded_operating_system/task_5/buildroot/output HOSTCC="/usr/bin/gcc" HOSTCXX="/usr/bin/g++" syncconfig
mkdir -p /home/yasser_jemli/embedded_operating_system/task_5/buildroot/output/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" /usr/bin/make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
    obj=/home/yasser_jemli/embedded_operating_system/task_5/buildroot/output/build/buildroot-config -C support/kconfig -f Makefile.br conf
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/yasser_jemli/embedded_opera
ting_system/task_5/buildroot/output/build/buildroot-config -DCONFIG_=\"\"   /home/yasser_jemli/embedded_operating_system/task_5/buildroot/output
/build/buildroot-config/conf.o /home/yasser_jemli/embedded_operating_system/task_5/buildroot/output/build/buildroot-config/zconf.tab.o  -o /home
/yasser_jemli/embedded_operating_system/task_5/buildroot/output/build/buildroot-config/conf
>>> host-skeleton  Extracting
>>> host-skeleton  Patching
>>> host-skeleton  Configuring
```

II. Préparation de la Carte SD

préparer une carte SD avec deux partitions : une FAT32 pour le démarrage et une ext pour le système de fichiers racine. Assurons-nous d'activer le drapeau de démarrage sur la partition FAT32.

```
sudo fdisk /dev/sda
sudo mkfs.vfat -n BOOT /dev/sda1
sudo mkfs.ext4 -L rootfs -E nodiscard /dev/sda2
```

III. Installation du Système Linux

Une fois la construction terminée et la carte SD préparée, copions les fichiers nécessaires sur la partition de démarrage :

```
sudo cp MLO u-boot.img zImage am335x-boneblack.dtb /media/yasser_jemli/BOOT/
```

Déballons maintenant notre système de fichiers racine sur la deuxième partition de la carte SD :

```
sudo tar -C /media/yasser_jemli/rootfs/ -xf rootfs.tar
```

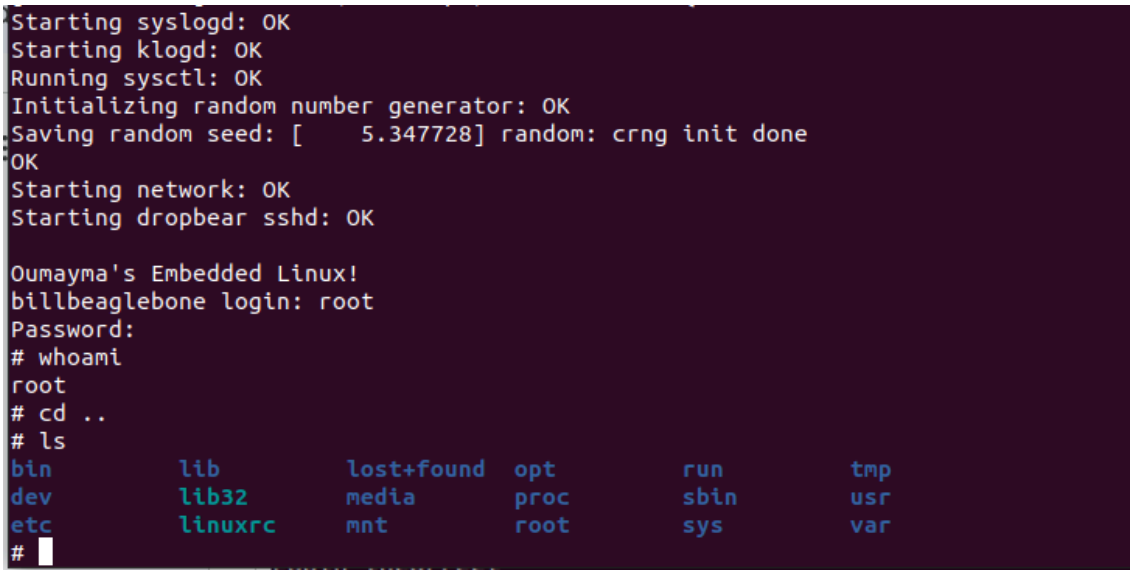
IV. Amorçage du Système

Avec un adaptateur série connecté à votre PC, démarrez un terminal picocom :

`picocom -b 115200 /dev/ttyUSB0` (aprés avoir installer picocom pour assurer la communication)

Insérez la carte SD dans le BeagleBone et démarrez-le en maintenant le bouton USR enfoncé pour forcer le démarrage depuis la carte SD. Vous devriez voir la sortie d'U-Boot et du noyau Linux, suivie de la bannière que vous avez définie dans la configuration Buildroot.

Connectez-vous en tant que root avec le mot de passe que vous avez configuré, et vous avez maintenant installé un système Linux Buildroot sur votre BeagleBone Black !



Conclusion

Ce rapport décrit en détail le processus de création d'un système Linux minimal pour le BeagleBone Black en utilisant Buildroot. En suivant les étapes de configuration, de construction et d'installation sur une carte SD, nous avons illustré comment Buildroot simplifie la création de systèmes Linux embarqués tout en offrant une flexibilité et des options de personnalisation étendues. Ce projet constitue une étape importante dans l'apprentissage du développement Linux embarqué et ouvre la voie à des projets plus complexes et spécifiques dans ce domaine.