GUAN LIANG
JEROME SKODA

Rapport de TP3

A.Générer un noyau Linux

1.Configurer le système

a). Chemin de configuration :

1). Le suffixe qui apparaîtra quand on affichera la version du système :
General setup => Local version -append to kernel release

2). Activer l'option qui permet d'utiliser un «initram» disque :
General setup => [*]Initial RAM filesystem and RAM disk(initramfs/initrd) support

3). Activer le support pour « printk » :
General setup => Configures tandard kernel features(expert users)
=> [*]Enable support for p rintk

4). Activer le support par le noyau Linux des binaires exécutables ELF et des shell scripts :
Executable file format/Emulation => [*]kernel support for ELF binaires
[*]kernel support for scripts starting with#!

5). Activerez le support de 8250/16550
Device Drivers => Character Devices
=> [*]Enable TTY
=> Serial drivers
=> [*]8250/16550 and compatible serial support
[*]Console on 8250/16550 and compatible serial port

6). Activerez le support des pseudo-file systems proc et sysfs :
File systems => Pseudo filesystems
=> [*]/proc file system support
[*]system file system support

b). la documentation est trouvée dans la réporatoire :
TP3/linux4.7.7/Documentation

2.Compilez

a). Estimation de temps :
Je travaille sur ma propre machine, sur le système ubuntu 14.04 LTS.

```
real    1m10.790s
user    3m1.800s
sys     0m12.676s
```

b). Le fichier généré :
Le fichier généré est trouvé dans TP3/linux-477-x86/arch/x86/boot/bzimage.
La taille de ce fichier est : 830128 octs
Le noyau linux est généré sous différentes formes :
=> bzImage : un fichier qui contient une image du noyau compressée.
=> vmlinux : ce fichier est un fichier exécutable sous format ELF.

c). Les commandes de redémarrage de la machine avec le noyau installé :
=>cp TP3/linux-477-x86/arch/x86/boot/bzimage vmlinuz
=>cp System.map /boot
=>reboot

B.QEMU

a). La commande à completer est :

qemu-system-i386 -nographic -append "console=ttyS0" -kernel TP3/linux-477-x86/arch/x86/boot/bzimage

L'affichage :

```
Kernel panic - not syncing: No working init found.  Try passing init= option to
kernel. See Linux Documentation/init.txt for guidance.
Kernel Offset: disabled
---[ end Kernel panic - not syncing: No working init found.  Try passing init= c
ption to kernel. See Linux Documentation/init.txt for guidance.
random: nonblocking pool is initialized
```

Parce que la machine est 64bits mais on a généré un noyau Linux x86.

C.Bonjour le monde!

1. Hello World

a). La différence entre qemu-i386 et qemu-system-i386 :

=> qemu-i386 : permet de faire une émulation pour utilisateur, qui prend en charge un fichier binaire exécutable

=>qemu-system-i386 : permet de faire une émulation complète du système.

On utilise qemu-i386 pour « hellos » parce qu'il est une émulation d'un fichier binaire exécutable, par contre on utilise qemu-system-i386 pour le noyau linux en vue de faire une émulation complète du système.

b). l'arborescence obtenue :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3$ tree -l root
root
└── sbin
    └── init

1 directory, 1 file
```

c). La commande pour généré mon programme 'init' :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3/init$ ls
hello  hello32.c  hellos  init.c  init.h  init.o  libinit.a
```

=> gcc -c -m32 -o init.o init.c

Obtenir le fichier init.o 32bit.

=> ar -rv libinit.a init.o

Une création de libraire statique libinit.a

=> gcc -static -m32 hello32.c -L. -linit -o hellos

Une création de fichier exécutable avec un lien statique.

=> cp hellos /root/sbin/init

Une copie de fichier exécutable hellos vers /root/sbin/init pour générer le programme init.

d). Pourquoir faut-il faire une édition statique ?

Parce que on copie juste le fichier exécutable dans init. Et puis, on va tester init par le noyau installé, si on ne fait pas une édition statique , la libraire ne va pas être trouvée.

e). Le résultat de la commande file :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3$ file init/hellos
init/hellos: ELF 32-bit LSB  executable, Intel 80386, version 1 (GNU/Linux), sta
tically linked, for GNU/Linux 2.6.24, BuildID[sha1]=6278e679e457123e5dd94730af17
7c59de1a38f0, not stripped
```

f). La taille de la commade init/hellos :

733513 octs

ls -l init/hellos

g). =>Taille en memoire :

La commande : size init/hellos

La taille de code : 656803 octs

La taille de données : 4096 octs et 5020 octs non installé

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3$ size init/hellos
   text    data     bss     dec     hex filename
 656803    4096    5020  665919   a293f init/hellos
```

=>Taille sur disque :

La commande : ls -s init/hellos

720 octs sur disque.

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ ls -s init/hellos
720 init/hellos
```

h). Par la commande readelf -l init/hellos

```
En-têtes de programme:
  Type           Décalage Adr. vir.  Adr.phys.  T.Fich. T.Mém.  Fan Alignement
  LOAD           0x000000 0x08048000 0x08048000 0xa06bf 0xa06bf R E 0x1000
  LOAD           0x0a0f48 0x080e9f48 0x080e9f48 0x01038 0x023bc RW  0x1000
```

adresse virtuelle pour le code : 0x08048000

adresse virtuelle pour la donnée : 0x080e9f48

2. Une fois le « disque » fabriqué

a). Une fois , il m'a affiché un message suivant :

```
---[ end Kernel panic - not syncing: No working init found.  Try passing init= o
ption to kernel. See Linux Documentation/init.txt for guidance.
```

Parce que j'ai pas donné une libraire statique pour le progamme init. Donc pour la création de disque, le nombre de bloc n'était pas correct.

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1/root$ find . -print0| cpio --null -ov --form
at=newc | gzip -9 > ../initramfs.cpio.gz
.
./sbin
./sbin/init
16 blocs
```

Après avoir donné une libraire statique à init, il n'y avait plus d'erreurs.

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1/root$ find . -print0| cpio --null -ov --form
at=newc | gzip -9 > ../initramfs.cpio.gz
.
./sbin
./sbin/init
1434 blocs
```

b). L'option -append

Qemu dispose d'une option réservée au passage de paramètres au noyau Linux : -append.
Les valeurs que on peut passer à -append doivent être mit dans " ". On a les valeurs ci-après :
=>console
=>root
=>ramdisk_size=%dK
=>nfsroot
=>rw
=>init
=>nousb
=>ip
l'information trouvé par :
=>http://blog.csdn.net/rockwill/article/details/6400033
=>https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-148/Qemu-comment-emuler-une-nouvelle-machine-Cas-de-l-APF27

c). Le message affiché après 10 fois 'hello world !' :

```
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
hello world!!!!!
Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000

Kernel Offset: disabled
---[ end Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
```

Parce que le processus init n'est pas autorisé à être tué .

3.Un peu de dynamisme!

a). Les commandes pour générer le programme init :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3/init1$ ls
hello32.c  hellod  init.c  init.h  init.o  libinit.so
```

=>gcc -O fPIC -m32 -c init.c
Obtenir le fichier init.o 32 bit
=>gcc -shared -m32 -o libinit.so init.o
Obtenir la libraire libinit.so de facon dynamique.
=>gcc hello32.c -L. -linit -o hellod
Créer un fichier exécutable hellod avec un lien dynamique.
=>cp hellod /root/sbin/init
Une copie de fichier exécutable hellod vers /root/sbin/init pour générer le programme init.

b). La commande file hellod :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3/init1$ file hellod
hellod: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically l
inked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=1dac2f1078cfa11805
4bc1c2cf28862b7a7157ab, not stripped
```

c). La taille de la commande init1/hellod :

7292 octs

ls -l init1/hellod

d). =>Taille en memoire

La commande : size init1/hellod

La taille de code : 1374 octs

La taille de données : 292 octs

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3$ size init1/hellod
   text    data     bss     dec     hex filename
   1374     288       4    1666     682 init1/hellod
```

=>Taille sur disque :

La commande : ls -s init1/hellod

8 oct sur disque

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ ls -s init1/hellod
8 init1/hellod
```

e). Le résultat de la commande ldd :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ ldd root/sbin/init
        linux-gate.so.1 =>  (0xf779d000)
        libinit.so => /usr/lib/libinit.so (0xf7771000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf75c0000)
        /lib/ld-linux.so.2 (0x56567000)
```

f). Les bibliothèques que j'ai copier dans la arborescence root :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ tree -l root/
root/
├── lib
│   ├── i386-linux-gnu
│   │   └── libc.so.6
│   └── ld-linux.so.2
├── sbin
│   └── init
└── usr
    └── lib
        └── libinit.so

5 directories, 4 files
```

D.Une petite compilation croisée

a). Avec la commande file ./hell_arm :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ file ./hello_arm
./hello_arm: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (SYSV), statically
 linked, for GNU/Linux 2.6.32, BuildID[sha1]=5413c77ca952f2955dac908c54bf5abfa16
c2cd2, not stripped
```

b). Pourquoir -static :

parce que pour la compilation croisée, la compilation et l'exécution ne sont pas dans le même système. Du coup, il est obligé de faire une édition statique pour les libraires, pour que quand l'autre

système exécute le fichier puis trouver les libraires.

c). La première tentative d'exécution :

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3_1$ ./hello_arm
bash: ./hello_arm: cannot execute binary file: Erreur de format pour exec()
```

d). la différence entre première exécution et la deuxième exécution :

Quand on passe la première exécution, dans la machine, elle ne peut pas exécuter cd fichierde format 32bit sans qemu. Par contre, la deuxième exécution, on a installé qemu, du coup, on est capable de traiter les fichier exécutable de 32 bit, donc hello_arm peut être exécuté.

E.BusyBox

a). Les problèmes que j'ai rencontré :

```
can't open /dev/tty4: No such file or directory
can't open /dev/tty2: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty2: No such file or directory
can't open /dev/tty4: No such file or directory
can't open /dev/tty2: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
```

pour résoudre ce problèmes, j'ai utilisé la commande 'touch' dans la reporatoire /dev.

b). La séquance de commande :

J'ai essayé les commandes suivante:cd, ls, ls -l.

c). Taille de fichier exécutable busybox:

=>Taille en memoire

La commande : size init1/hellod

La taille de code : 1374 octs

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3/BB/busybox-1.24.2$ size busybox
   text    data     bss     dec     hex filename
1867953    4712   17230 1889895  1cd667 busybox
```

La taille de données : 292 octs

=>Taille sur disque :

La commande : ls -s busybox

1836 octs

```
guan@Guan:~/Bureau/M2/INFO_EM/TP3/BB/busybox-1.24.2$ ls -s busybox
1836 busybox
```

d). Réduction de taille :

Une façon pour réduire la taille de busybox, est de faire une édition dynamique pour ce fichier binaire, il faut que je copie tous les libraires de dépendances trouvées par la commande ldd.