PXE

Vittoria Rezzonico, SB-IT

modules PXE par
Alessandro Crespi, GR-IJ

10 juin 2009

Séminaires passés

Extensions Firefox	Michela	Tout public	3 novembre 2008
IPv6	Laurent	Avancé	28 novembre 2008
Tequila	Claude	Tout public	19 mars 2009

Bientôt ?

- GPG
- Extensions Thunderbird
- Extensions Firefox, II
- PAM
- vi
- Nagios

Contenus

Définition

Preboot eXecution Environment

- environnement pour démarrer un ordinateur à travers le réseau
- indépendamment des périphériques de stockage disponibles
- et des OS y installés
- Introduit en 1999 par Intel

API pour le BIOS

- UNDI (Universal Network Driver Interface)
- Programmes pour bootstrap réseau
  - services Preboot (fonctions génériques)
  - client TFTP
  - UDP (User Datagram Protocol)

Du BIOS au BootLoader

- Le BIOS charge le secteur de boot depuis un périphérique local qu'il est capable de driver (disque dur, disquette, CDROM, clef USB,...)
- traitons le démarrage depuis le disque dur
- les premiers 446 bytes du disque (MBR moins la table des partitions) sont exécutés
- ces 446 bytes vont :
  - soit chercher une partition bootable et exécuter le contenu de son Volume Boot Record
  - soit faire le boulot eux-mêmes (bootloader)

MBR avec table des partitions


```
0000  |.H............|   0090  |...t7f.L...|.D..|   0120  |;D.}<.T.....L...|
0010  |...|........!..|   00a0  |f..D|.....D...f.|   0130  |...l.Z.t...p..1.|
0020  |....8.u.......u|   00b0  |\..D..pf1..D.f.D|   0140  |.....r*....H|‘..|
0030  |.........|...t..|   00c0  |..B..r...p.}....|   0150  |....1.1.....a.&B|
0040  |... ............|   00d0  |s............|.|   0160  ||..}.@.....}.8..|
0050  |u....Y|..1......|   00e0  |D..f1...@f.D.1..|   0170  |...}.0...}.*...G|
0060  |. ..@|<.t...R..}|   00f0  |........@.D.1...|   0180  |RUB .Geom.Hard D|
0070  |.4....tT.A..U..Z|   0100  |...f..f.D|f1.f.4|   0190  |isk.Read. Error.|
0080  |RrI..U.uC.A|..u.|   0110  |.T.f1.f.t..T..D.|   01a0  |........<.u.....|

  01b0  00 00 00 00 00 00 00 00  18 7a 07 00 00 00 00 01  |.........z......|
  01c0  01 00 83 fe 3f 81 3f 00  00 00 c3 dd 1f 00 00 00  |....?.?.........|
  01d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
  *
  01f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
```

- La table des partitions commence à 0x1be (446)

Comment ça marche : le Boot local

- le bootloader est installé sur les premiers 446 bytes de la MBR du disque

LiLo (Linux Loader)

- ne comprend pas les systèmes de fichiers
- le noyau doit donc être trouvable sans la notion de filesystem
- c'est pour cela que la MBR doit être recrite à chaque nouveau noyau – la position sur le disque du noyau y est codée en dur !

GRUB (Grand Unified Bootloader)

- comprend les systèmes de fichiers
- peut avoir accès à la carte réseau et comprendre une implémentation minimale de la stack TCP
- a une interface interactive (ligne de commande)

GRUB (Grand Unified Bootloader)

- la MBR contient le *Stage1*.
- le Stage 1 charge :
  - directement le Stage 2 ou bien
  - le Stage 1.5 qui se trouve dans les premiers 30kB du disque dur juste après la MBR. Ensuite le Stage 1.5 charge le Stage 2.
- Le Stage 2 est un mini-système capable d'interpréter quelques commandes, mais surtout capable de charger un noyau et un initrd depuis le filesystème.

Exercice

- Enlevez le Stage 1.5 et regardez ce qui se passe

```
0200  |RV..!.*.^..!f.-.|  02d0  |‘.a.}....<......|  0430  |ot/grub/menu.lst|
0210  |}.......|..t>f..|  02e0  |...!.K.Z.."....!|  0440  |................|
0220  |f1...9E....E.)E.|  02f0  |.?.....!.7..#!.1|  [...]
0230  |f.......D.f.\..D|  0300  |...Loading stage|  1ec0  |.P....... .e.[^_|
0240  |..pPf1..D.f.D..B|  0310  |1.5......Geom.Re|  1ed0  |].Error %u..ext2|
0250  |........p.Vf..f1|  0320  |ad. Error.......|  1ee0  |fs...GRUB loadin|
0260  |.f.4.T.f1.f.t..T|  0330  |.F..<.u.........|  1ef0  |g, please wait..|
0270  |..D.;D.}t..*D.9E|  0340  |................|  1f00  |...internal erro|
0280  |....E.)E.f...T..|  *                      1f10  |r: the second se|
0290  |...L......l.ZR.t|  03f0  |.............. .|  1f20  |ctor of Stage 2 |
02a0  |.P..p..1.....rF.|  0400  |.p".............|  1f30  |is unknown......|
02b0  |..E.X....E.‘....|  0410  |..0.97...../boot|  1f40  |.<..x4...5...5..|
02c0  |..1.1.........!.|  0420  |/grub/stage2 /bo|  1f50  |................|
```

- le client envoye un DHCPDISCOVER sur le port 67 UDP (standard DHCP server port). Cette requête est une requête standard DHCP (RFC2131) plus des options Client PXE
- le serveur envoye un DHCPOFFER (donc configure le réseau du client), en l'étendant avec
  - le nom du serveur de boot
  - le nom du NBP
- le client envoye un DHCPREQUEST
- le serveur de boot envoye un DHCPACK
- le client ouvre une connection TFTP avec le serveur TFTP et charge le Network Bootstrap Program (pxelinux.0)
- le NBP prend contrôle et cherche le fichier correspondant à l'adresse MAC ou IP du client. Ensuite, il charge tj par TFTP
  - un noyau
  - un initrd

```
Source        Destination     Protocol Info
0.0.0.0       255.255.255.255 DHCP     DHCP Discover - Transaction ID 0xba157918
192.168.7.1   255.255.255.255 DHCP     DHCP Offer    - Transaction ID 0xba157918
0.0.0.0       255.255.255.255 DHCP     DHCP Request  - Transaction ID 0xba157918
192.168.7.1   255.255.255.255 DHCP     DHCP ACK      - Transaction ID 0xba157918
192.168.7.79  192.168.7.1     TFTP     Read Request, File: pxelinux.0\000, Transfer type: octet\000, 1456=1
192.168.7.1   192.168.7.79    TFTP     Option Acknowledgement, 1456=1456\000
192.168.7.79  192.168.7.1     TFTP     Acknowledgement, Block: 0
192.168.7.1   192.168.7.79    TFTP     Data Packet, Block: 1
192.168.7.79  192.168.7.1     TFTP     Acknowledgement, Block: 1 [...]
192.168.7.1   192.168.7.79    TFTP     Data Packet, Block: 10 (last)
192.168.7.79  192.168.7.1     TFTP     Acknowledgement, Block: 10
192.168.7.79  192.168.7.1     TFTP     Read Request, File: pxelinux.cfg/01-00-0d-b9-15-79-18\000, Transfer
192.168.7.1   192.168.7.79    TFTP     Error Code, Code: File not found, Message: File not found\000
192.168.7.79  192.168.7.1     TFTP     Read Request, File: pxelinux.cfg/C0A8074F\000, Transfer type: octet\
192.168.7.1   192.168.7.79    TFTP     Option Acknowledgement, 692=692\000, 1440=1440\000
192.168.7.79  192.168.7.1     TFTP     Acknowledgement, Block: 0
192.168.7.1   192.168.7.79    TFTP     Data Packet, Block: 1
192.168.7.79  192.168.7.1     TFTP     Acknowledgement, Block: 1
192.168.7.79  192.168.7.1     TFTP     Read Request, File: vmlinuz-2.6.24-tc\000, Transfer type: octet\000
192.168.7.1   192.168.7.79    TFTP     Option Acknowledgement, 1472088=1472088\000, 1440=1440\000
[...]
192.168.7.79  192.168.7.1     TFTP     Read Request, File: initrd-install.gz\000, Transfer type: octet\000
[...]
0.0.0.0       255.255.255.255 DHCP     DHCP Discover - Transaction ID 0x4b2d2702
192.168.7.1   192.168.7.79    DHCP     DHCP Offer    - Transaction ID 0x4b2d2702
0.0.0.0       255.255.255.255 DHCP     DHCP Request  - Transaction ID 0x4b2d2702
192.168.7.1   192.168.7.79    DHCP     DHCP ACK      - Transaction ID 0x4b2d2702
192.168.7.79  192.168.7.1     TCP      44832 > sunrpc [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=4294893379 TS
192.168.7.1   192.168.7.79    Portmap  V2 GETPORT Reply (Call In 4266) Port:2049
192.168.7.79  192.168.7.1     TCP      950 > nfs [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=4294893380 TSER=0
192.168.7.79  192.168.7.1     NFS      V3 NULL Call (Reply In 4278)
192.168.7.79  192.168.7.1     MOUNT    V3 NULL Call (Reply In 4298)
```

En pratique : configuration DHCP

dhcpd.conf

Ajouter aux options :

- filename "pxelinux.0" ;
- next-server 192.168.2.1 ;
- server-name "192.168.2.1" ;

Après avoir installé et configuré le serveur tftp, il faut déposer dans sa racine :

- le fichier `pxelinux.0` (on le trouve dans le paquetage syslinux)
- un noyau
- éventuellement un initrd
- le répertoire `pxelinux.cfg`

Le repertoire `pxelinux.cfg` contient des fichier texte indiquant correspondants aux clients

Supposons que le client ait comme adresse IP `192.168.7.79` (`C0A8074F` en hexadécimal) et comme adresse MAC `00:0d:b9:15:79:18`. pxelinux.0 ira chercher dans le répertoire les fichiers suivants, dans l'ordre :

- `00-0D-B9-15-79-18`
- `C0A8074F`
- `C0A8074`
- `C0A807`
- `C0A80`
- `C0A8`
- `C0A`
- `C0`
- `C`
- `default`

Exemple de fichier de démarrage

```
default myboot

label myboot
kernel vmlinuz-2.6.24-tc
append initrd=initrd.gz ip=dhcp root=/dev/ram0

label local
      LOCALBOOT 0

label memtest
kernel memtest86
```

Extras pxelinux et syslinux

- Possibilité de menu pseudographique (à la ncurses) avec menu.c32
- ajout de background avec vesaboot.c32
- Possibilité de reconnaître des architectures 32/64 bits avec le module ifcpu64.c32
- créez vos propres modules !

Modules custom

Pourquoi des modules custom ?

- Les modules fournis avec syslinux/pxelinux donnent un bon nombre de possibilités, mais celles-ci ne suffisent pas toujours...
- Par exemple : comportement dépendant de la date/heure, interface interactive plus complexe qu'un menu...

Et pratiquement...

- Il s'agit de modules dans le format .c32 (pratiquement un exécutable 32-bits sans en-tête), qui sont généralement programmés en C.
- Beaucoup de fonctions sont mises à disposition du programmeur, soit sous la forme de fonctions C (une bonne partie des fonctions des bibliothèques C standard peuvent être utilisées), soit sous la forme d'interruptions logicielles.

Un exemple tout simple

Tout d'abord, l'immanquable *Hello world*

```
1  #include <stdio.h>
2  #include <console.h>
3
4  int main() {
5    openconsole(&dev_null_r, &dev_stdcon_w);
6    printf("Hello world!\n");
7    return 0;
8  }
```

Compilation d'un module

- Il faut en principe trois commandes : `gcc` pour compiler le programme, `ld` pour l'édition des liens, et pour finir `objcopy` pour enlever les en-têtes du fichier ELF.

Compilation d'un module

```
1 gcc -m32 -mregparm=3 -DREGPARM=3 -W -Wall -march=i386 -Os
    -fomit-frame-pointer -I../libutil/include -I../include
     -D__COM32__ -fno-stack-protector -c -o hello.o
    hello.c
2 ld -m elf_i386 -T ../lib/com32.ld -o hello.elf hello.o
    utils.o ../libutil/libutil_com.a ../lib/libcom32.a
    /usr/lib/gcc/i486-linux-gnu/4.2.4/libgcc.a
3 objcopy -O binary hello.elf hello.c32
```

- Mieux vaut utiliser un Makefile... on peut facilement adapter celui des exemples fournis dans le paquetage syslinux.

Un peu de couleur, s'il vous plaît !

- pxelinux peut comprendre les séquences de contrôle de terminal ANSI (comme la console de Linux) : on peut donc effacer l'écran, changer de couleur...

Exemple

```
1  #include <stdio.h>
2  #include <consoles.h>
3
4  int main() {
5    console_ansi_raw();
6    printf("\x1b[1;33;44m"); // jaune sur bleu
7    printf("\x1b[2J\x1b[H\n"); // efface l'ecran
8    printf("Hello, colored world!");
9    return 0;
10 }
```

Quelques fonctions utiles

- Il est possible d'accéder à des fichiers par TFTP, en utilisant les fonctions C standard (`fopen()`, etc.). Il faut cependant noter que certaines fonctions sont absentes (p.ex. `feof()`).
- `getkey()` permet de lire une touche depuis le clavier (y compris les touches spéciales), en spécifiant un timeout.
- `syslinux_idle()` permet d'appeler le *idle loop* de pxelinux, donc la seule fonction actuelle est d'appeler le stack PXE pour répondre à d'éventuelles requêtes ARP. Il faut normalement appeler cette fonction lors de toute attente.
- Des fonctions permettant de calculer des hash SHA1 sont disponibles dans `sha1.h`. Cela peut être utile pour valider des mots de passe, par exemple.

Appel d'autres modules

Un module pxelinux n'est souvent pas suffisant à lui seul : il sera souvent utile de pouvoir en démarrer un autre, ou de charger un noyau Linux. La fonction qui permet de faire cela est plutôt bien cachée... car elle doit être appelée par une interruption logicielle.

Exemple d'appel

```
1 void boot_linux() {
2   command("kernels/vmlinuz-2.6.24-23-generic initrd=
        kernels/initrd.img-2.6.24-23-generic root=/dev/
        sda2 ro");
3 }
```

Appel d'autres modules : le source

Voici donc un *wrapper* en C (largement inspiré de ce qu'on trouve dans les modules fournis avec pxelinux).

```
1  void command(char* cmd) {
2    com32sys_t ireg;
3    const char *p;
4    char *q = __com32.cs_bounce;
5    const char *kernel, *args;
6
7    kernel = q;
8    p = cmd;
9    while (*p && !isspace(*p)) *q++ = *p++;
10   *q++ = '\0';
11   args = q;
12   while (*p && isspace(*p)) p++;
13   strcpy(q, p);
14
15   ireg.eax.w[0] = 0x16;
16   ireg.esi.w[0] = OFFS(kernel);
17   ireg.ds = SEG(kernel);
18   ireg.ebx.w[0] = OFFS(args);
19   ireg.es = SEG(args);
20   ireg.edx.l = 0;
21
22   __intcall(0x22, &ireg, NULL);
23   printf("Call to boot function failed.");
24   for (;;) syslinux_idle();
25 }
```

Appels au BIOS

On ne peut pas tout faire avec ce que pxelinux met à disposition. On pourrait par exemple vouloir supprimer le curseur clignotant à l'écran, ou bien éteindre l'ordinateur... pour cela, on peut assez facilement appeler directement des fonctions mises à disposition par le BIOS.

Exemple d'appel au BIOS

```
1  void setcursor(int s1, int s2) {
2    com32sys_t ireg;
3
4    ireg.eax.w[0] = 0x0100;
5    ireg.ecx.b[0] = (s2 & 0xFF);
6    ireg.ecx.b[1] = (s1 & 0xFF);
7    __intcall(0x10, &ireg, NULL);
8  }
```

Modules custom : conclusion

- Il est possible de réaliser pratiquement toute fonction qui ne l'est pas directement avec les outils fournis par pxelinux. Cela dépend surtout de la fantaisie du programmeur...
- Certaines fonctions sont "difficiles" à trouver ou à implémenter, entre autre car la documentation n'est pas énorme : regarder les fichiers `.h` et le code source des modules fournis avec syslinux peut s'avérer très utile.
- Documentation : `http://syslinux.zytor.com/wiki/index.php/Comboot_API`.

pxegrub

- peut être utilisé comme noyau ou en remplacement à pxelinux.0
- patcher et recompiler GRUB avec l'option `--enable-diskless` et une carte réseau (n'importe laquelle)
- vous pouvez englober un menu dans pxegrub avec l'option `--enable-preset-menu`
- on aura le fichier `lib/grub/i386-pc/pxegrub`. On peut l'appeler depuis le serveur dhcp `filename "pxegrub"` ou bien depuis un des fichiers dans `pxelinux.cfg`. Il faudra alors le renommer `pxegrub.0`.

Après avoir chargé le noyau et le initrd, le processus de boot, que ça soit local ou réseau, continue ainsi :

- le script d'init est chargé (`/linuxrc` ou autre si indiqué dans les paramètres de boot). Dans ce script, obligatoirement :
  - on monte la future racine
  - on exécute un `pivot_root` entre la racine actuelle (notre initrd) et la future racine
  - on exécute un `chroot` dans la nouvelle racine
  - on passe la main à `init`

Après le NBP : le noyau

On va se focaliser maintenant sur le démarrage après le NBP.

- Le client a chargé un noyau (et éventuellement un initrd)
- Comment doit être le noyau pour que le démarrage continue normalement ?
  - doit permettre d'acquérir une adresse IP
  - doit permettre de monter une racine par NFS

```
Network -> networking Options -> [*] IP: kernel level autoconfiguration
                                 [*] IP: DHCP support
File Systems -> Network File Systems -> [*] NFS system support
                                        [*] Root file system on NFS
```

- Root sur NFS : read-only
- ...mais les fichiers de log doivent aller quelque part !
- avec UnionFS on monte le root NFS read only, on crée un tmpfs et on uni les deux

```
mount -t nfs 192.168.7.1:/srv/thinclient /sta
mount -t tmpfs -o size=300M none /dyn
mount -t unionfs -o dirs=/dyn=rw:/sta=ro unionfs /unionroot
```

- SquashFS permet de comprimer des systèmes de fichiers
- limitation : le fs monté en squashfs est read-only
- $\Rightarrow$ idéal pour notre cas

```
mksquashfs /var/arch /var/arch.sqsh
mkdir /mnt/tmp
mount /var/arch.sqsh /mnt/tmp -t squashfs -o loop
```

- Définition : root initial, monté avant que le vrai root est disponibe
- Contenus : les exécutables nécessaires jusqu'à l'appell de init, les librairies dont ils dépendent et des modules noyau
- Format : `cpio` ou `loop`

Construire un initrd

```
mkdir /mnt/loop
dd if=/dev/zero of=initrd BS=1M count=8
mke2fs initrd
mount -o loop initrd /mnt/loop
  cd /mnt/loop/
  mkdir etc dev lib bin proc cdrom sta dyn unionroot
  touch linuxrc etc/mtab etc/fstab
  cd bin/
  for i in cat chroot insmod mkdir mount pivot_root portmap sh tar umount; do \
                    tocopy=‘which $i‘; cp $tocopy . ; done
  cd ../lib
  for i in ‘\ls‘; do ldd $i; done | awk ’{ print $3 }’ | \
                    sort -u | grep so | xargs  -I{}  cp {} .
  cp -H /lib/ld-linux.so.2 .
  vi ../linuxrc
cd
umount /mnt/loop/
cd ..
find ./ | cpio -H newc -o > ~/initrd
cd
gzip ~/initrd
mv initrd.gz initrd
```

```
#!/bin/sh
mount -t proc none /proc
portmap

mount -t nfs 192.168.7.1:/srv/thinclient /sta
mount -t tmpfs -o size=300M none /dyn
mount -t unionfs -o dirs=/dyn=rw:/sta=ro unionfs /unionroot

cd /unionroot
/bin/pivot_root . initrd
mount --move /initrd/dyn /mnt/dyn
mount --move /initrd/sta /mnt/sta
mount -t tmpfs -o size=50M none /tmp

mount -t proc none /proc
cat /proc/mounts > /etc/mtab
umount /proc

exec chroot . /bin/sh <<- EOF >dev/console
umount /initrd
echo "Starting init!"
exec /sbin/init
EOF
```

- Outil d'installation automatique de Debian (hérité donc par Ubuntu)
- Le serveur FAI consiste de :
  - un serveur DHCP
  - un serveur TFTP
  - un *nfsroot*
  - des fichiers de configuration pour la création du nfsroot et pour l'installation des clients
- nfsroot configuré par les fichiers dans `/etc/fai`, ensuite il faut lancer le script `fai-setup`.
- il faut bien ajouter les adresses MAC des clients au serveur DHCP
- pour utiliser PXE, exécuter le script `fai-chboot`
- la configuration des clients se trouve dans `/srv/fai/config`

- on configure DHCP, TFTP
- dans les fichiers `pxelinux.cfg/*` (par exemple, `/tftpboot/pxelinux.cfg/C0A80470`), on indique le fichier kickstart avec l'option `ks`

```
default install
label install
kernel vmlinuz
 append ip=dhcp ksdevice=eth0 load_ramdisk=1 initrd=initrd.img \
      network ks=http://192.168.4.1/install/nodes-ks.cfg \
      ramdisk_size=9216
```

- le fichier pxelinux.cfg/default contient

```
default local
label local
        LOCALBOOT 0
```

- le noyau et le initrd pour le démarrage réseau se trouvent dans le répertoire `images/pxeboot` du medium d'installation.
- il faut une astuce pour faire disparaitre le fichier `tftpboot/pxelinux.cfg/C0A80470`, on va utiliser la partie post du kickstart.

```
%post  --nochroot

IPdec=‘ifconfig | grep "inet addr" | head -1 | \
           awk ’{ print $2 }’ | awk -F: ’{ print $2 }’‘
mkdir /mnt/temp /mnt/tftpserv
cd /
mount -t proc proc proc
mount 192.168.4.1:/data/install /mnt/temp
mount 192.168.4.1:/tftpboot /mnt/tftpserv
cd /mnt/temp/; IPhex=‘./iphex $IPdec‘
cd /mnt/tftpserv/pxelinux.cfg
mv $IPhex $IPhex.disabled
cd; umount /mnt/tftpserv; umount /mnt/temp
rmdir /mnt/tftpserv /mnt/temp
```

- Linux Terminal Server Project
- il s'agit de clients légers
- **mériterait un séminaire**
- le client démarre sur le réseau et se retrouve avec un environnement complet sans nécessiter de disque dur
  - en réalité, le client charge par PXE un mini-système qui lui permet de se connecter par XDMCP au serveur LTSP.
- LTSP nécessite que les services suivant soient fonctionnels :
  - DHCP
  - TFTP
  - NFS
  - XDMCP
- ensemble de scripts pour configurer les services (ltspcfg) et le serveur ltsp (ltspadmin)

Références (liste non exhaustive)

- discussions avec plein de monde (Anne pour KickStart, Laurent pour GRUB et LTSP, Loris pour GRUB, Alessandro pour les modules PXE, Basil pour UnionFS et SquashFS)
- `man hexdump`
- `man tcpdump`
- `man dhcpd.conf`
- `http://www.filesystems.org/project-unionfs.html`
- `http://squashfs.sourceforge.net/`
- `http://www.informatik.uni-koeln.de/fai/fai-guide/ch-impatient.html`
- `http://www.ltsp.org/`

Questions