

The GUI project

Murali Krishnan Ganapathy*

Abstract

This project aims at creating an all purpose CD-ROM for the Intel Platform. It can be used to install OSes, software, troubleshoot machines, backup and restore hard disks and more. In short it aims to be the one thing which system administrators cannot do without. Currently the OSes supported are Windows XP Professional and our custom distributions of Linux.

1 The goals

The GUI CD, can boot into multiple OSes and can help automate the following:

- Install OSes,
- Backup and recovery of data,
- Troubleshoot problems which require an external OS,
- and much much more...

Put simply, it can boot into multiple network aware OSes and run network scripts or drop you into a shell. This together with the multitude of scripts it is armed with, enable it to just about anything. Whats special about this CD is that (almost) all the scripts as well as the OSes to be installed are on the network. So adding functionality to the CD, is as simple as changing the appropriate files on the network. Currently the CD can boot into a Linux, DOS and a Windows XP recovery shell. We currently use this CD to install Windows and Linux on our machines, and for some troubleshooting. This CD has also been used to clone redhat installations as well as prepare machines for resale.

All this is made possible thanks to ISOLINUX (<http://syslinux.zytor.com>) which is a CD-based boot loader. It is used in the distribution of all major flavors of Linux as well as FreeDOS. ISOLINUX allows one to boot into Linux kernels, no-emulation CD "boot sectors", as well as boot off a Floppy or Hard disk image.

The images in the GUI CD include:

- prep** Prepare the hard disk for OS installation and ask configuration questions [e.g. hostname,ip,...]
- winstall** Install Windows using the information gathered in the **prep** stage.
- linstall** Install Linux using the information gathered in the **prep** stage.
- dosresc** Boots into a network aware DOS shell.
- winresc** This boots into a Windows Shell (Recovery Console).
- linresc** Boots into a network aware Linux shell.
- hdwipe** Wipe the hard disk to prepare the machine for reselling.
- backup** Backup the hard disk [Not yet implemented]

*gmurali+guicd(at)cs.uchicago.edu

2 GUICD on the web

The homepage of the GUI-CD project is at <http://people.cs.uchicago.edu/~gmurali/gui>. There you can find the latest information as well as downloads as and when they become available.

3 The Windows Recovery Console image

ISOLINUX has the ability to boot other CD “boot sectors”. So all one has to do is to extract the boot sector from the Windows Installation CD and copy the files from the Windows Install CD which are required for the booting into the recovery console. That is about 8MB worth of files. Once this is done, one has to instruct ISOLINUX to boot off the “boot sector” we extracted.

4 Creating the DOS image

Here we use ISOLINUX’s ability to boot off floppy images. We start by creating a bootable DOS floppy which contains appropriate network drivers and utilities to initialize its network card, and mount a windows share. Then this floppy is dd’ed into a file and ISOLINUX is asked to boot off this floppy image.

Creating a bootable DOS floppy with multiple network drivers can be a bit tricky. MS-DOS network client is setup in such a way that it does not need any support at boot time, i.e. appropriate drivers need not be installed in `CONFIG.SYS`. We use a DOS utility which scans the PCI Cards, and looks up their Device and Vendor ID’s in a specified file and returns the appropriate driver to use. Then we create appropriate configuration files and then start the network client using the just generated configuration files.

5 Creating the linux image

Conceptually, this is the simplest of all three. Just compile a kernel with the right configuration, and create a root file system which has the network drivers and other basic utilities. Compress the root file system and store that and the kernel on the CD. Then ask ISOLINUX to boot the kernel using the compressed root file system as the initial ramdisk. Here we dont have any space constraints as the kernel as well as the initial ramdisk live on the CD.

6 ISOLINUX

ISOLINUX generates a small CD “boot sector” which reads the `CDROM:/isolinux/isolinux.cfg` file at boot time and presents a LILO like prompt, and boots into the image specified on the prompt. Shown below is a portion of an `isolinux.cfg` file.

```
-- snip --
label prep
    kernel gmklinux
    append root=/dev/ram0 initrd=rootfs.img image=fdisk

label wininstall
    kernel memdisk
    append initrd=wininstall.img

label linstall
    kernel gmklinux
    append root=/dev/ram0 initrd=rootfs.img image=linstall

label dosresc
    kernel memdisk
```

```
    append initrd=dosresc.img

label winresc
    kernel w2ksest.bin
--snip --
```

So typing in `linstall` at the boot prompt, will boot the `gmklinux` kernel and pass it the options specified in the `append` line. You will also notice that booting off a floppy image is not done by `ISOLINUX` per se, but by the `memdisk` kernel. The `memdisk` kernel looks at the size of its image, and decides whether to do a floppy emulation or a hard disk emulation. Recent versions of `memdisk` supports compressed images as well.

7 prep image

The `[prep]` image boots into linux, and once all the init scripts are done, it downloads the “`fdisk`” script and runs it. The `fdisk` script inturn downloads the real `fdisk.sh` script as well as all the other files which it depends on, and then runs `fdisk.sh`. Finally, when `fdisk.sh` is run, it asks the user which combination of OSes needs to be installed, and some basic information for each OS, like `hostname`, `static/dhcp`, ip address (if `static`)... In case of linux, it also indentifies the kernel drivers required for the peripherals. Once it acquires all this information, it re-partitions the hard disk and formats all the partitions with the appropriate file systems. Finally, it stores all the acquired information in the appropriate partitions and then reboots.

8 wininstall image

The `[wininstall]` image boots into DOS, mounts a network share and then executes a predefined script. This predefined script, then reads the information acquired in the `[prep]` stage, and creates an `unattended.txt` file. Then it drops a `SETUP2` script on the windows partition. Finally, it calls the windows installer and points it at the `unattended.txt` file to do an automated installation. The `unattended.txt` file instructs windows to execute the `SETUP2` script after the OS installation is over.

Once the windows installation is over (3 reboots later), the `SETUP2` script takes over. It mounts the network share, and then executes the post-installation script. The post-installation script, starts off by installing all the software listed in a predefined file (on the network). It then proceeds to install network printers (consults a `printcap` file for the list of printers to install) and finally secures the system.

As of now, after this is done, the user has to use the windows GUI to join this machine to our windows cluster.

9 linstall image

The `[linstall]` image boots into linux and downloads and executes a `linstall` script, which inturn downloads the real script and all its dependencies, and executes the real script. The real script registers the machine with a database giving the database its current identity (it is now using `DHCP`) as well as the distribution of linux it wants installed. Then it just waits...

Once this is done on many machines, we run the `gui-dist.sh` script on our `rdist` server. This script, downloads the list of machines which needs a new OS as well as the distribution it requires, and starts the `rdist`. Once the `rdist` is over (about 3-4 hours later), it creates an `PIKT` identity for this new machine on the `PIKT` master, and stores this on the client as well. It then initiates the second (local) phase of the installation, on each client.

Once the remote phase of the installation is completed, the local phase is started. The local phase, creates important files like `/etc/hostname`, `/etc/local.conf`, `/etc/local/pikt/pikt.conf` as well as `/etc/X11/XF86Config`, which contain all the runtime information specific to the machine. Then it creates the `/etc/lilo.conf` specific to this machine, taking into account, whether it is a dual boot machine, whether it has `CD-writers` installed... Finally, it runs `lilo` to install an `MBR`, and then reboots.

The first time this machine boots, a runonce init script initializes the locate database, downloads the list of public ssh keys of other linux machines and adds its public key to the list. It then ensures that all the PIKT scripts are in place and installed. Finally, it prevents itself from being executed again.

10 A generic linux based image

All the linux based images use the same kernel and root file system to boot. Boot time arguments influence the behaviour of many init scripts. For example, passing “network=dhcp/static/no” sets up the network interface accordingly (ofcourse in case of static the ipaddr boot time argument determines the ipaddress of this machine). Another boot time argument is the “image” argument. Once all the init scripts are done, the script specified as the “image” argument is downloaded (location determined by the “baseurl” argument), and executed. In addition the other virtual consoles display useful debugging information (generated by the scripts) as well the system log.

10.1 How boot time arguments work

The commandline passed to a linux kernel can be accessed via `/proc/cmdline`. One of the early init.d scripts reads this and creates a `/tmp/config.sh`. In order to reduce the size of the commandline, there are defaults for all the options as well. For example, if the command line has `network=static ipaddr=192.168.1.0 mountcd=yes`, the `config.sh` script will contain

```
# Default Configuration
export network=dhcp
export ipaddr=0.0.0.0
export net_mod=all
export mountcd=no

# User specified overrides
export network=static
export ipaddr=192.168.1.0
export mountcd=yes
```

Now every init.d script sources this file and acts accordingly.

11 Automatic Image Selection

In the current setting, one needs to supervise the CD, now and then, telling it which image to boot from, everytime the machine reboots. SYSLINUX allows arbitrary 32-bit and 16-bit code (in a specific binary format) to run and implements some API. This code can take advantage of the COMBOOT API and instruct SYSLINUX to boot a particular image.

Automatic images selection happens in two parts. During the [prep] stage one can determine the exact order in which the different images should be booted (it inturn depends on which combination of OSes are being installed). In the first part we write this boot order into an unused portion of the hard disk (track 0, sector 6).

Then setup ISOLINUX to boot our custom COM32 code, which reads the first entry of the boot order, updates the boot order (by removing the first entry) and asks SYSLINUX to boot into the image we just read. This solves the automatic image selection problem.

12 Unified DOS image

Right now, we have two DOS images on the CD. One to mount a network share and execute a predefined script, and another to mount a network share and drop to a shell. Unlike linux, there is no way to pass boot time information to the OS. So, even though the two images are nearly the same, we still need two images. Secondly a 2.88MB floppy image does not allow us much room for improvement. Ofcourse one can boot of hard disk images, which can be quite large, but that leads to its own problems (Windows

installation with call your real hard disk D: and you are stuck with that name for ever). Both problems are solved using MEMDISK.

MEMDISK allows one to boot off large floppy images with non-standard sizes and geometries. So we can have a 9MB floppy image and boot off it. This gives us lot of room for improvement. See the GUI-website for more information.

Secondly, MEMDISK implements a DOS interrupt which allows one to check if we are running in DOS or inside MEMDISK. If we are running inside MEMDISK, one can also get the command line arguments. So even though DOS does not recognize boot time arguments, we can acheive the same effect by running a DOS program which gets the command line arguments and declares environment variables to reflect the same.

13 TUI for GUI

With more and more general purpose OS images, the more command line arguments we need to pass to acheive what we want. For example, if we want to install linux on a machine on a network without a DHCP server, we need to pass arguments to setup static IP. The number of ways in which all these options can be mixed and matched will increase exponentially.

Another current limitation is the fact that the user needs to know which image to boot into (initially atleast). While this is not that big a problem for this application, it might be a problem for linux distros out there. For example, if you need to install Red Hat on a slightly non-standard machine you need to pass arguments to the installer. The difficult part is in figuring out which option to pass to the installer.

TUI for GUI, is a Text User Interface for GUI. Basically, it presents a menu system and allows the user to pick the image (with the right arguments) to be booted. The menu system can be very complicated and supports checkboxes, event handlers (so certain portions of the menu system can be disabled based on other choices). Finally after the user has made all the choices, it exits and instructs ISOLINUX to boot the image specified chosen by the user.

14 Future Improvements

There is a whole slew of improvements which can be made on this CD. Creating a brand new image is as simple as creating a script with the right name and putting it on the webserver.

The first obvious improvement which comes to mind is the as yet unfinished [backup] image. This is not as difficult as one may think, thanks to partimage. This is a linux based utility which allows one to backup harddisks and partitions to a local file, or to a remote location (using the partimaged server). This will allow people to backup their own laptops as often as they want. All we need to do is to give them a CD, which has partimage on it.

There is already a COMBOOT utility which allows you boot off some partition of a hard disk. This can be handy on a machine which has a corrupted MBR. One can conceive integrating this with the TUI for GUI, so that the comboot program inspects the different partitions, identifies the bootable ones, and presents a menu allowing the user to select the partition to boot into.

15 Conclusion

Creating such a CD teaches you a lot about how computers work. There are a lot of problems to overcome in order to create such a CD. Every now and then, you will be up against a seemingly “chicken and egg” type of problem. One utility which is very useful in testing these CD’s without wasting CD’s is VMWare. Using VMWare one can reduce the number of wasted CD’s. Here are some of the problems to watch out for:

- **Boot sectors:** In a dual boot scenario, we need to install windows before linux. The windows installation program does not touch the master boot record, unless it partitioned the hard disk. The windows installation process requires multiple boots. So, we need to setup a DOS MBR during the [prep] stage. Since windows did not format the partition where we are going to install it, it assumes that /dev/hda1 already has a working boot sector and hence only changes the boot code

of the boot sector, so that it can boot into WinXP. The boot code, however, expects to find a data structure describing the geometry of the partition (Extended Bios Parameter Block). Since this is only a logical structure, it can vary with the OS. So we cannot install a boot sector in the [prep] stage. We solve this problem by installing Win98/DOS, and then installing Windows XP on top of it. So this means, we need to remove the underlying DOS from the windows boot loader's configuration file.

- **Initial ramdisk:** Getting a small yet powerful enough initial ramdisk can be tricky. One has to install only essential utilities, ensure that all the libraries required for them are in place and ensure that all the devices required exist and have the right permissions. If ISOLINUX were not there, we would have a 2.88MB disk in which to hold the kernel and the initial ramdisk and all the necessary utilities. In our case, thanks to ISOLINUX we can afford to use a 1MB kernel and 16MB ramdisk.
- **Hardware Detection:** This forms an essential part of the script which runs during the [prep] stage. In case of DOS, we used a utility which scans the PCI bus and finds the vendor and device ID's of the PCI devices. Luckily, there is a linux equivalent which does more. It looks up a database, and finds the kernel module required for the network card, sound card and even the XWindows driver required for the video card. More often than not, the hardware which one has to deal with is too modern to be present in the database. So, one will need to modify the database to include this new hardware as well. Detecting CD-Writers is a little more difficult. We use the ide-scsi drivers for all the optical drives (instead of the ide-cd drivers as is more common). Once we have access to the scsi interface, we use cdrecord to find out the capabilities of each optical drive, and use the output to determine whether the optical drive in question is a CD-Writer or not. The other thing, one has to look out for is the new generation video cards, which require a kernel module to be loaded in addition to the XWindows driver.

16 Acknowledgements

This project was made possible due to (but not limited to):

- ISOLINUX** Without ISOLINUX nothing would have been possible. The author of ISOLINUX and the other members of the SYSLINUX mailing list, who answered all my questions.
- mkisofs** But for all the esoteric options, creating an iso image, which satisfies all the OSes, would have been impossible.
- newsgroups** For helping me through some tricky portions of the script, especially with hardware detection.
- Techstaff** For putting up with me and giving very useful suggestions where we were able to leverage existing infrastructure.
- Microsoft** For the excellent documentation on COM programming (though it was bit difficult to reach) which helped me write all those VBS and Python scripts.