Table of Contents

OCLUG live distro project 3
Planning outline for OCLUG live USB distro 3
Goals: 3
Base ideas: 3
Tasks: 3
Investigation of EFI boot (S Murphy) 4
Arch Installation Notes 4
Getting started 4
Get a copy of Arch 5
Create your VMWare VMX file 5
Forcing UEFI Boot 5
The actual installation 6
Intel based System 8
The /boot Partition 8
Enabling the network 9
Final Steps 10
TODO 10
References 10

OCLUG live distro project

This was suggested and discussed in November 2016. See http://wiki.oclug.on.ca/lib/exe/fetch.php?media=wiki:presentations:2016:nash-pandoc-160303.pdf.

There was follow-up discussion to this. See 20161103FollowupDiscussion

Collected comments are on page(s) 2017comments in reverse date order.

Planning outline for OCLUG live USB distro

Date: 17-3-4 John C. Nash

Goals:

- Unordered List Item"boot anywhere" – legacy/UEFI, 32/64, non-pae
- customizable
- remasterable
- installable
- documented, both for "product" and how it has been built

Base ideas:

We have a preliminary liveUSB/iso that is

- Arch based
- boots UEFI or legacy
- 32 or 64 bit, and boots on non-pae processor
- This also boots in VirtualBox VM
  - can run startx to come up in openbox (manual start)
  - very minimal

Tasks:

A goal is to have these "bite sized" if possible

* Graphical interface

- openbox configuration
- openbox add-ons such as
  - tint2 - system panel and task bar
  - conky - system monitor and information center

    - nitrogen - for backgrounds
- some documentation on install and configuration for above

* Customizing the system

- pacman installs
- recommended and necessary tools
- how to remove/reduce to keep things tight
- checking and verifying system
- documentation of the above

* Remastering

- getting a new live image from running virtual system
- Are there choices we need to address?
- Testing image in virtual environment
- packaging image onto USB or other media
- appropriate boot mechanisms in place

- Installer script
    - optional inclusion on the image
    - how to adjust / test

Investigation of EFI boot (S Murphy)

Arch Installation Notes

Scott Murphy scott.murphy@arrow-eye.com

- version: 1.0
- keywords: VMWare, Arch Linux, macOS, UEFI
- 2017-03-21

Getting started

First, the Arch documentation is very good. You can get all of this directly from it, however it makes a few assumptions that might trip you up if you are new to arch, UEFI and running under VMWare.

I'm using VMWare for my Arch install, so this is specific to installing under VMWare Fusion 8 on macOS. This probably translates pretty much directly to other installs if you leave out the VMWare items. The other thing is that my install is the 64-bit install. I don't have a 32-bit system and was not interested in trying it out in VMWare at this time. Maybe in a update.

Get a copy of Arch

You can find the Arch distro links on the [https://www.archlinux.org|Arch Linux website], in the https://www.archlinux.org/download/[Download] section. Select a mirror or bittorrent link and get started. I selected the https://mirror.csclub.uwaterloo.ca/archlinux/iso/2017.03.01/[CS Club mirror] at the university of Waterloo. I picked the https://mirror.csclub.uwaterloo.ca/archlinux/iso/2017.03.01/archlinux-2017.03.01-dual.iso[2017-03-01] image. Once it is downloaded, You can use the iso in whatever way you want. I'm using it as is, in a virtual DVD drive.

You can burn it to a DVD if you have something sufficiently old, however if your device supports booting for USB, I'd go that route.

NOTE: Add a link to the create a USB version instructions

Create your VMWare VMX file

Create a new VMWare machine by selecting `New → Create a custom virtual machine` in the VMWare Library. Go through the full creation up to the point where you start the VM to get the install going. My VM has

- A 20GB disk
- 1024MB RAM
- A single processor

Forcing UEFI Boot

In order to use UEFI, you need to edit the VMX file. Assuming you called the machine `New Arch VM`, you would find the vmx file in the `~/Documents/Virtual Machines.localized/New Arch WM.vmwarevm` directory. In this case, the file will be called `New Arch VM.vmx`. If there is an additional file with the same name and a .lck extansion, the library has the file open. Make sure you have the startup window closed. If there is no `.lck` file, then all you need to do is edit the file and add the following line to it.

```
// Interesting, when I reformat a long line to 80 columns, vi crashes on my
mac
// I've noticed it before, but it seems to be an issue with lines that wrap
// longer than four lines.
```

```
  firmware = "efi"
```

That was pretty straightforward. This will tell VMWare to use UEFI as the boot method instead of legacy BIOS.

The actual installation

This will go more like a series of steps. There will only be extra info where it gets somewhat odd.

Now that you have the blank VM created and the VMX file modified, it is time to get started.

- Mount the DVD image and connect it to the VM
- Power on the VM. It should autoboot into a root shell
- Verify that you are in UEFI mode

```
 ls /sys/firmware/efi/efivars
```

- Make sure you can see the internet (The system should have dhcp'd an address)

```
ping archlinux.org
```

- Enable network time protocol

```
timedatectl set-ntp true
timedatectl status
```

- Partition your disk. In this instance, you need to create at least three partitions:
    - Boot (512M, fat32)
    - Swap (I usually use 256M and hope I never start swapping)
    - / (The bulk of the disk)

- you can use `sgdisk` to create your partitions with the following commands, assuming your disk is /dev/sda:

```
sgdisk -p /dev/sda
sgdisk --new=0:0:+512M /dev/sda
sgdisk --new=0:0:+256M /dev/sda
sgdisk --new=0:0:0 /dev/sda
sgdisk --typecode=1:EF00 /dev/sda
sgdisk --typecode=2:8200 /dev/sda
sgdisk --change-name=1:"EFI System" /dev/sda
sgdisk --change-name=2:"Linux swap" /dev/sda
sgdisk --change-name=3:"Linux filesystem" /dev/sda
sgdisk -p /dev/sda
```

That should first show you a disk with no partition table, and then the disk with all three partitions, something like this:

```
Disk /dev/sda: 41943040 sectors, 20.0 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 9E17A0DA-9DED-4E62-BAE7-076E1522C21B
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 41943006
```

```
Partitions will be aligned on 2048-sector boundaries
Total free space is 4061 sectors (2.0 MiB)
```

```
Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048         1048576   511.0 MiB   EF00  EFI System
   2         1050624         1574911   256.0 MiB   8200  Linux swap
   3         1574912        41943006   19.2 GiB    8300  Linux filesystem
```

Now that the disk is properly partitioned, you will format the partitions:

```
mkfs.fat -F32 /dev/sda1
mkswap -L swap /dev/sda2
mkfs.ext4 /dev/sda3
```

NOTE: UEFI partitions are FAT32 formatted and the UEFI BIOS can read FAT32 partitions to locate the correct `.efi` file to boot. This also makes it easy to mount that partition to make fixes.

What are we going todo?

```
. Mount the root partition on /mnt
. Create a new directory to mount the boot partition on and mount it there
. Use the pacstrap program to install the base operating system
. Create a new fstab that reflects the mounted disks
. Chroot to the newly installed OS so initial changes could be made
. Set the timezone
. Set the hardware clock to the running OS
. Enable the en_US lockle and set the system language to en_US UTF8
. Give the system a hostname
. Let the system refer to itself in a zeroconf (bonjour) way
```

In order to acomplish this, enter the following commands:

```
mount /dev/sda3 /mnt
mkdir /mnt/boot
mount /dev/sda1 /mnt/boot
pacstrap /mnt/base
genfstab -U /mnt >> /mnt/etc/fstab
arch-chroot /mnt
ln -sf /usr/share/zoneinfo/America/Toronto /etc/localtime
hwclock --systohc
sed -i 's/#en_US/en_US/' /etc/locale.gen
locale-gen
echo "LANG=en_US.UTF-8" > /etc/locale.conf
echo "myhostname" > /etc/hostname
echo "127.0.0.1     myhostname.localdomain myhostname" >> /etc/hosts
```

Intel based System

If this is an Intel based system, you will need to add the ability to perform cpu microcode updates. I don't know if this even works under VMWare, but if I have to install on a physical system, then I want to remember this part.

You enable the updates that by adding the intel microcode package:

```
pacman -S intel-ucode
```

NOTE: Need to check this out on a test system. I'm not sure you need to perform this step if you are not using grub. I'm not, so this might be useless. I happen to like having grub in place, so this is not wasted documentation, just potentially unused.

In order to fix the bootloader to use the intel microcode upadtes, we need to install the grub utilities and update the configuration:

```
pacman -S grub
grub-mkconfig -o /boot/grub/grub.cfg
```

or you might need to do it inside the EFI setup. That will ne noted in the /boot secting below.

Now you should set the root password:

```
passwd
```

Congratulations, the system has been installed. Unfortunately, it will not boot just yet.

The /boot Partition

This is where it tends to get a little harder to follow. At this point, you can unmount the disks and reboot - you will not have a bootable system, but you will have a fully functional system if it could only boot. If you mess around enough, it will boot, but you may not know what you did or why it works. I like to compare this to the first time you use grub or PXE boot a system. You are not sure what is happening under the hood, even though you edited those files.

If you rebooted, you will need to boot from your install media again. Don't bother with the selection 'Boot from Installed System' or whatever version it offers, it will not boot.

If you reboot, you need to mount the disks again and chroot again. Once that has happened, you can continue on and make a bootable system.

You are going to need some more tools installed, so fire up `pacman`

```
pacman -S efibootmgr efitools
```

You can check to see if the system is effectively ready for this by running the `efivar` command and seeing the output

```
 efivar -l
```

This should give you a list of “things” that look like a bunch of GUIDs followed by a readable word. If so, everything you need is in place.

This is the secret sauce for making the system bootable:

```
 efibootmgr -d /dev/sda -p 1 -c -L "Arch Linux" -l '\vmlinuz-linux' -u
"root=/dev/sda3 rw initrd=/initramfs-linux.img"
```

NOTE: Intel microcode updates may require the previous line to include two `initrd=` sections, so the previous line would have `-u “root=/dev/sda3 rw initrd=/intel-ucode.img initrd=/initramfs-linux.img”`

Enabling the network

This is for a wired connection, however the procedure is similar for wireless. Look at this first and then check out the wireless section of the networking reference below.

The legacy `net-utils` have been fully deprecated in the current version of Arch, so you will be using the `ip` command

Get the name of the wired interface. It will not be eth0.

```
 ip link
```

The output will be like:

```
 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
  link/ether 00:0c:29:dc:56:64 brd ff:ff:ff:ff:ff:ff
```

The wired interface is **ens33**.

Assuming you will be using dhcp for your networking, this is how you get it working. Enter the following commands:

```
cd /etc/systemd/network
echo > local.network<<EOF
[Match]
Name=ens33
```

```
[Network]
DHCP=ipv4
EOF
systemctl enable systemd-networkd.service
ip a
```

If this is a system you rebooted before you configured the network, you can enable the networking using the command:

```
  systemctl start systemd-networkd.service
```

The final command, `ip a`, should show that you now have an address on your local network. This should come up after a reboot.

Final Steps

At this point, you should be golden. exit from the chrooted system by typing `exit` and then unmount the disk partitions using the command `umount -R /mnt` and then enter `reboot`

TODO

- X11 Setup
- Window Manager installation
- TBD

References

The following pages (mostly from the Arch Wiki) were used to make these notes.

- Installation guide - ArchWiki
- Unified Extensible Firmware Interface - ArchWiki
- EFI System Partition - ArchWiki

* Talk:EFISTUB - ArchWiki * EFISTUB - ArchWiki * Network configuration - ArchWiki * Microcode - ArchWiki * systemd-networkd - ArchWiki * VMware/Installing Arch as a guest - ArchWiki * Arch User Repository - ArchWiki

Also looked at:

* https://blog.uncooperative.org/blog/2014/02/06/the-efi-system-partition/[The EFI System Partition and the Default Boot Behavior - The Uncoöperative Organization] * https://github.com/rasa/vmware-tools-patches[rasa/vmware-tools-patches: Patch and build VMware tools automatically] * http://planetvm.net/blog/?p=2730[Boot via EFI firmware - PlanetVM]

For creating that USB stick, you can use the traditional `dd` method or try out https://etcher.io/[Etcher] if you are command line weary.

Extra Information

Of course if you would prefer to do the install, including the GUI desktop and software selection the easy way, there are two projects of note that can help you out.

They are:

* https://arch-anywhere.org[ArchAnywhere] * https://sourceforge.net/projects/revenge-installer[Revenge Installer]