Slax directory structure

All Slax data files are located on the boot media in a single directory. It is no surprise that the name of that directory is 'slax'. All the magic happens inside. Here is an overview of simplified directory structure; directories are red, some interesting files are mentioned as well, using *italic*:

```
slax
├── boot
│   ├── isolinux.bin
│   ├── syslinux.cfg
│   ├── initrfs.img
│   ├── vmlinuz
│   └── ...
├── changes
├── rootcopy
├── modules
│   ├── 01-core.sb
│   ├── 02-xorg.sb
│   ├── 03-desktop.sb
│   ├── 04-chromium.sb
│   └── ...
```

Booting the Linux kernel

When your computer's BIOS boots Slax, it actually just runs SYSLINUX boot loader. The boot loader itself is stored either in file isolinux.bin or ldlinux.sys, depending on your boot media - CD/DVD uses isolinux.bin, USB disk or hard drive uses ldlinux.sys.

As soon as the SYSLINUX boot loader is executed, it learns what to do next from its configuration file (you guessed it) syslinux.cfg. In Slax, this configuration file contains instructions to show some cool boot logo and optionally provide boot menu if the user hits a key before timeout. When the timeout counter reaches zero or the user exited boot menu, SYSLINUX boot loader loads two files into memory: vmlinuz (Linux kernel) and initrfs.img (base root filesystem). The progress is indicated by continuous stream of dots printed on screen. Once the files are loaded, the vmlinuz binary is executed to start the Linux kernel.

Pre-init

Under normal conditions (when a standard Linux distribution is starting from a hard drive), the Linux kernel would mount root filesystem from the hard drive and `/sbin/init` would be executed as the main process which takes care of system startup. In Slax, the situation is different - there is no hard drive to mount the root filesystem from, yet the kernel surely needs some init process to be started. For that purpose, Slax carries a base filesystem in initrfs.img file - it is a compressed CPIO archive with some directories and files inside, including core Linux tools (commands) and the desired init.

So after the Linux kernel has successfully initialized and has a full control of your computer, its last task is to find the mentioned CPIO archive in memory (it was loaded there from file initrfs.img by syslinux boot loader as you surely remember), extract it (into a memory area which acts as a temporary root filesystem, called initramfs) and execute temporary `/init` process from there.

Escaping initramfs

At this moment, we have a fully initialized Linux Kernel running, initramfs area in memory is populated by a temporary root filesystem with just the most basic Linux commands, and temporary init just started.

Having the temporary root filesystem in initramfs is not ideal, since it doesn't support `pivot_root` system call - an important operation which will be used later in the boot up process. We need to switch from initramfs to something else. To do that, the temporary init firstly mounts a tmpfs filesystem over `/m`, moves all files and directories in there including the init script itself, and uses switch_root to make this tmpfs `/m` the new root and to restart the init itself from there too. Blue star denotes the directory which is moved.

initramfs as root: ->

```
(initramfs)
/
├─── bin
│    ├─── sh
│    ├─── mount
│    └─── ...
├─── dev
├─── mnt
├─── m (tmpfs) ★
├─── memory
├─── proc
├─── sys
├─── init
└─── shutdown
```

initramfs after move to tmpfs: ->

```
(initramfs)
/
└─── m (tmpfs) ★
     ├─── bin
     │    ├──sh
     │    ├──mount
     │    └─── ...
     ├─── dev
     ├─── mnt
     ├─── memory
     ├─── proc
     ├─── sys
     ├─── init
     └─── shutdown
```

tmpfs after switch_root:

```
(tmpfs) ★
/
├─── bin
│    ├─── sh
│    ├─── mount
│    └─── ...
├─── dev
├─── mnt
├─── memory
├─── proc
├─── sys
├─── init
└─── shutdown
```

No matter how strange this whole action looks like (we've ended up with the very same directory structure like before, it seems like no improvement at all), the change is significant. Since now, the temporary root filesystem is on tmpfs instead of initramfs, and thus `pivot_root` operation will be available when needed in the future.

You may be wondering why is the current root filesystem still labeled as temporary. It's because we're still at the very beginning of the construction phase and we'll just build the real root filesystem later, as explained below, by combining Slax compressed data images to AUFS union.

Slax data lookup

Before the init process could start to search for Slax data on available devices, it needs to setup the working environment. The proc and sysfs filesystems are mounted over `/proc` and `/sys` respectively. Some important kernel drivers such as aufs, squashfs and loop are loaded using modprobe, and device files are created in `/dev` directory by mdev command.

As soon as storage devices are accessible through device files in `/dev`, blkid command is used to filter out only those which can be mounted (which contain a filesystem known to the running kernel). The devices are examined (mounted read-only over `/memory/data/`) one after another, until a valid Slax data directory is found. Then, all files with .sb extension (Slax Bundles) are processed - for each Slax Bundle, a directory is created in `/memory/bundles/` and the bundle (which in fact is a squashfs compressed image file) is loop-mounted over it. In the diagram below, squashfs content is green.

```
(tmpfs)
/
├─── bin
├─── dev
├─── ...
├─── memory
│    ├─── bundles
│    │    ├─── 01-core.sb (squasfhs mount) < ──┐
│    │    │    ├─── bin                        │
│    │    │    ├─── etc                        │
│    │    │    ├─── sbin                       │
│    │    │    └─── ...                        │
│    │    ├─── 02-xorg.sb ......................│...
│    │    │    ├─── etc                        │  :
│    │    │    ├─── usr                        │  :
```

```
│    │    │    └─── ...                         │  :
│    │    ├─── 03-desktop.sb ...................│..:...
│    │    │    ├─── usr                         │  :  :
│    │    │    └─── ...                         │  :  :
│    │    └─── ...                              │  :  : loop
│    ├─── data (slax device mounted here)       │  :  : mounts
│    │    └─── slax                             │  :  :
│    │         ├─── boot                        │  :  :
│    │         ├─── changes                     │  :  :
│    │         ├─── rootcopy                    │  :  :
│    │         ├─── 01-core.sb ──── > ──── > ───┘  :  :
│    │         ├─── 02-xorg.sb ....................:  :
│    │         ├─── 03-desktop.sb ....................:
│    │         └─── ...
│    ├─── changes (empty yet)
│    └─── union (empty yet)
├─── proc (procfs mounted here)
├─── sys (sysfs mounted here)
└─── init
```

Putting it together with AUFS

Various parts of the final root filesystem are now mounted read-only under separated folders in `/memory/bundles`. Core Linux system utilities and libraries such as `/bin/bash` or `/lib/ld-linux.so` are located in `/memory/bundles/01-core.sb/`, files related to desktop environment can be found in `/memory/bundles/03-desktop.sb/`, and so on. Combining them into a single root filesystem, which is even writable, is only possible thanks to AUFS - an union-like filesystem developed by Mr. Junjiro Okajima. AUFS is capable of taking several folders (so called branches) and combining them together to a single directory. That is exactly what happens next, the separated parts are combined, together with a directory `/memory/changes/`, to AUFS union, which gets mounted at `/memory/union`.

```
(tmpfs)
/
├─── ...
└─── memory
     ├─── bundles
     │    ├─── 01-core.sb ───────── > ──────┐
     │    ├─── 02-xorg.sb ..................│.......
     │    ├─── 03-desktop.sb ...............│......:........
     │    └─── ...                          │      :       :
     ├─── changes ──────── > ───────┐       │      :       :
     ├─── ...                       v       v      :       :
     └─── union < ═══════ < ═══════ < ───── < ─────┘ < ────┘
          ├─── bin            AUFS
          ├─── etc            mount
          ├─── mnt
          ├─── root
          ├─── sbin
          ├─── usr
          ├─── ...
          └─── var
```

Changes

The empty directory `/memory/changes` is writable, thus the entire AUFS mount in `/memory/union` happens to be writable as well. All new or changed files inside the union are copied-up to this empty directory before the system creates or modifies them. Since the directory for changes resides on tmpfs (that is in RAM), all new and modified files are stored in RAM and thus are lost on reboot.

Yet if Slax is started from a writable media such as USB device or hard disk, it recognizes that and mounts the writable drive over `/memory/changes` before it is joined with the other branches in union, which effectively means that changed and new files will be stored on the boot device rather than in RAM, and reboot won't erase them. This feature is called Persistent Changes and can be turned on or off by a boot menu setting.

Switching to the real root

At this point, fully writable final root filesystem has been built in `/memory/union`. The temporary init's life is coming to its end. It uses `pivot_root` and `chroot` system calls to make `/memory/union` the new root, transferring the temporary tmpfs root to `/run/initramfs/` in the new root. Blue and green stars at the diagram below denote what moves where. Finally, the real `/sbin/init` from aufs root filesystem is executed. The booting of Slax operating system just begins.

tmpfs root before pivot_root syscall:

```
(tmpfs) ★
/
├─── ...
└─── memory
     └─── union (aufs) ★
          ├─── bin
          ├─── etc
          ├─── root
          ├─── run
          │    └─── initramfs (empty)
          ├─── sbin
          │    ├─── ...
          │    └───init
          ├─── usr
          ├─── ...
          └─── var
```

aufs as new root:

```
(aufs) ★
/
├─── bin
├─── etc
├─── root
├─── run
│    └─── initramfs (tmpfs) ★
│         ├─── ...
│         └─── memory
│              └─── union (empty)
├─── sbin
│    ├─── ...
│    └─── init <- gets executed
├─── usr
├─── ...
└─── var
```

Adding modules to Slax on the fly

The root filesystem is writable and we could install new software packages while running Slax just the usual way, by unpacking them. Yet there is another possibility to add new files and directories to Slax on the fly without installing any packages. Thanks to the fact Slax is running with AUFS as root, we can take some other squashfs compressed filesystem, loop-mount it over a directory which resides outside of the aufs tree (for example over `/run/initramfs/memory/bundles/name.sb/` which is on tmpfs), and then issue a remount command which adds the newly mounted directory to aufs as a new branch.

All the files and directories from the new squashfs module will instantly appear as like if they were installed in the system from the beginning, while decompression is done on the fly, only for the files which are actually accessed.

Similarly, we can remove a previously added AUFS branch (mounted squashfs) from the aufs root by another remount command. The files which were part of the branch will instantly disappear from the system, which effectively uninstalls the package.

Slax shutdown

When Slax is shutting down either for reboot or for system power off, it performs all the standard tasks as every other Linux would do, like unmounting all partitions mounted by the user, terminating all processes, and so on. But since the boot device may be still mounted and used for persistent changes at the very end, some more steps need to be done before the real power off is issued, to ensure the boot device is cleanly unmounted.

Instead of turning the system off at the moment when init thinks it should do that, Slax switches back to initramfs (this is handled automatically by systemd) and executes a shutdown script `/run/initramfs/shutdown`.

The shutdown script than takes care of umnounting the remaining mounted disk from which Slax started and the boot device is cleanly ejected. At the end, the computer reboots or shuts down, depending on what the user intended to do.

Quick links

Changelog
Next version
Blog
Support Slax
Privacy policy

Contact

Slax discord
Google group
Facebook