initramfs

The initramfs contains all the necessary programs and config files needed to bring up the machine, mount the root file system and hand off the rest of the boot process to the installed system. It can be further customized with additional modules, binaries, files and hooks for special use cases and hardware.

Usage

Automated image generation

Every kernel in Arch Linux comes with its own *.preset* file stored in `/etc/mkinitcpio.d/` with configuration presets for `mkinitcpio`. Pacman hooks build a new image after every kernel upgrade or installation of a new kernel.

Manual image generation

To manually generate a Linux kernel image issue the following command:

```
mkinitcpio -p linux
```

This will generate a new kernel image with the settings of the preset file `/etc/mkinitcpio.d/linux.preset`.

To generate kernel images with every preset available, pass the `-P` argument:

```
mkinitcpio -P
```

Configuration

To customize your initramfs, place drop-in configuration files into `/etc/mkinitcpio.conf.d/`. They will override the settings in the main configuration file at `/etc/mkinitcpio.conf`.

An overview of the settings you can customize:

Setting	Type	Description
`MODULES`	Array	Kernel modules to be loaded before any boot hooks are run.
`BINARIES`	Array	Additional binaries you want included in the initramfs image.
`FILES`	Array	Additional files you want included in the initramfs image.
`HOOKS`	Array	Hooks are scripts that execute in the initial ramdisk.
`COMPRESSION`	String	Which tool to use for compressing the image.
`COMPRESSION_OPTIONS`	Array	Extra arguments to pass to the `COMPRESSION` tool.

> **WARNING:** Do not use the `COMPRESSION_OPTIONS` setting, unless you know exactly what you are doing. Misuse can produce unbootable images!

MODULES

The `MODULES` array is used to specify modules to load before anything else is done.

Here you can specify additional kernel modules needed in early userspace, e.g. file system modules (`ext2`, `reiser4`, `btrfs`), keyboard drivers (`usbhid`, `hid_apple`, etc.), USB 3 hubs (`xhci_hcd`) or "out-of-tree" modules which are not part of the Linux kernel (mainly NVIDIA GPU drivers). It is also needed to add modules for hardware devices that are not always connected but you would like to be operational from the very start if they are connected during boot.

> **HINT:** If you don't know the name of the driver of a device, `lshw` can tell you what hardware uses which driver, e.g.:

```
*-usb:2
     description: USB controller
     product: Tiger Lake-LP USB 3.2 Gen 2x1 xHCI Host Controller
     vendor: Intel Corporation
     physical id: 14
     bus info: pci@0000:00:14.0
     version: 20
     width: 64 bits
     clock: 33MHz
```

```
        capabilities: xhci bus_master cap_list
->      configuration: driver=xhci_hcd latency=0
        resources: iomemory:600-5ff irq:163 memory:603f260000-603f26ffff
```

The second to last line starting with `configuration` shows the driver being used.

Example of a `MODULES` array that adds two modules to the generated image needed for keyboard input, if the keyboard is connected to a USB 3 hub, e.g. a docking station:

```
MODULES=(xhci_hcd usbhid)
```

CAUTION: Keep in mind that adding to the initramfs increases the size of the resulting image on disk. Unless you have created your boot partition (more specifically the EFI System partition at either `/efi`, `/boot` or `/boot/efi`) with generous space, you should limit yourself to modules strictly needed for your system. The `autodetect` hook tries to detect all currently loaded modules of the running system to determine the needed modules to include by default. Only include additional modules if something doesn't work as expected.

ATTENTION: If you use an NVIDIA graphics card, the following modules are **required** in the `MODULES` array for early KMS:

```
MODULES=(nvidia nvidia_modeset nvidia_uvm nvidia_drm)
```

BINARIES

The `BINARIES` array holds the name of extra executables needed to boot the system. It can also be used to replace binaries provided by `HOOKS`. The executable names are sourced from the `PATH` evironment variable, associated libraries are added as well.

Example of a `BINARIES` array that adds the `kexec` binary:

```
BINARIES=(kexec)
```

This option usually only needs to be set for special use cases, e.g. when there's a binary you need included that is not already part of a member in the `HOOKS` array.

FILES

The `FILES` array holds the full path to arbitrary files for inclusion in the image.

Example of a module configuration file to be included in the image, containting the names of modules to auto-load and optional module parameters:

```
FILES=(/etc/modprobe.d/modprobe.conf)
```

This option usually only needs to be set for special use cases.

HOOKS

The `HOOKS` array is the most important setting in the file. Hooks are small scripts which describe what will be added to the image. Hooks are referred to by their name, and executed in the order they are listed in the `HOOKS` array.

> **HINT:** For a full list of availble hooks run:
>
> ```
> mkinitcpio -L
> ```
>
> See the help text for a hook with:
>
> ```
> mkinitcpio -H hook_name
> ```

The default `HOOKS` line in `/etc/mkinitcpio.conf` is as follows:

```
HOOKS=(base udev autodetect microcode modconf kms keyboard keymap consolefont block filesystems fsck)
```

This creates a basic image suitable for most single disk systems.

A quick overview of the hooks and their meaning:

Hook	Description
`base`	Sets up all initial directories and installs base utilities and libraries.
`udev`	Adds the udev device manager to scan and set up devices. Recommended for simple boot process.
`autodetect`	Trims hooks after that come after to only include modules that are needed for the current system. Keeps image slim.
`microcode`	Includes CPU microcode updates in the image.
`modconf`	Includes module configuration files from `/etc/modprobe.d/` and `/usr/lib/modprobe.d/`.
`kms`	Adds modules to bring up graphics cards as early as possible in the boot process.

Hook	Description
`keyboard`	Adds modules for keyboards. **Required** for keyboard input in early userspace.
`keymap`	Adds the specified keymap(s) from `/etc/vconsole.conf`.
`consolefont`	Adds the specified console font from `/etc/vconsole.conf`.
`block`	Adds block device modules needed to bring up different kinds of storage devices.
`filesystems`	Adds file system modules. **Required** unless file system modules are specified in `MODULES`.
`fsck`	Adds tools for checking file systems before they are mounted. **Strongly recommended!**

busybox

By default, `mkinitcpio` will generate a busybox-based initramfs. It starts an init script that scans the filesystem of the initramfs for scripts to execute and bring up the system and hand over the remaining boot process to systemd once the root file system is mounted. This is fine for most use-cases.

For special cases some additional hooks may be required for busybox to bring up the machine properly:

Hook	Description
`usr`	Needed for when you have `/usr` on a separate partition
`resume`	Needed for suspend-to-disk (hibernation) support
`btrfs`	Needed for btrfs file systems that span multiple drives, needs the `btrfs-progs` package installed
`net`	Needed for booting from a network drive, needs the `mkinitcpio-nfs-utils` package installed
`dmraid`	Needed for fakeRAID (BIOS RAID) root devices, needs the `dmraid` package installed
`mdadm_udev`	Needed for assembling RAID arrays via udev (software RAID), needs the `mdadm` package installed
`encrypt`	Needed for booting from an encrypted file system, needs the `cryptsetup` package installed
`lvm2`	Needed for booting a system that is on LVM, needs the `lvm2` package installed

One such special case is encryption, which would result in a `HOOKS` array that looks like this:

> **ATTENTION:** The order in which hooks are placed in the array is important!

```
HOOKS=(base udev autodetect microcode modconf kms keyboard keymap consolefont block encrypt filesystems fsck)
```

> **ATTENTION:** In some cases it might be necessary to place the `keyboard` hook before the `autodetect` hook to be able to enter the passphrase to unlock the encrypted file systems, e.g. when using different keyboards requiring a different module from the one in use at the time of building the initramfs.

systemd

If you wish, you can also make systemd bring the whole system up start to finish. In this case bootup will be handled by systemd unit files instead of scripts.

The benefit of this is faster boot times and some additional features not available to a busybox-based intiramfs, e.g. unlocking LUKS encrypted file systems with a TPM or FIDO2 token and automatic detection and mounting of partitions with the appropriate GUID Partition Table (GPT) UUIDs (see: Discoverable Partition Specification).

To instruct `mkinitcpio` to build a systemd-based initramfs:

- replace the `udev` hook with the `systemd` hook
- replace the `keymap` and `consolefont` hooks with the `sd-vconsole` hook

The resulting `HOOKS` array should look something like this:

```
HOOKS=(base systemd autodetect microcode modconf kms keyboard sd-vconsole block filesystems fsck)
```

For special cases some additional hooks may be required for systemd to bring up the machine properly:

Hook	Description
`mdadm_udev`	Needed for assembling RAID arrays via udev (software RAID), needs the `mdadm` package installed
`sd-encrypt`	Needed for booting from an encrypted file system, needs the `cryptsetup` package installed
`lvm2`	Needed for booting a system that is on LVM, needs the `lvm2` package installed

One such special case is encryption, which would result in a `HOOKS` array that looks like this:

> **ATTENTION:** The order in which hooks are placed in the array is important!

```
HOOKS=(base systemd autodetect microcode modconf kms keyboard sd-vconsole block sd-encrypt filesystems fsck)
```

> **ATTENTION:** In some cases it might be necessary to place the `keyboard` hook before the `autodetect` hook to be able to enter the passphrase to unlock the encrypted file systems, e.g. when using different keyboards requiring a different module from the one in use at the time of building the initramfs.

COMPRESSION

The `COMPRESSION` option instructs `mkinitcpio` to compress the resulting images to save on space on the EFI System Partition or `/boot` partition. This can be especially important if you include a lot of modules and hooks and the size of the image grows.

Compressing the initramfs is a tradeoff between:

- time it takes to compress the image
- space saved
- time it takes the kernel to decompress the image during boot

Which one you choose is something you have to decide on the constraints you're working with (slow/fast CPU, available cores, RAM usage, disk space), but generally speaking the default `zstd` compression strikes a good balance.

Algorithm	Description
`cat`	Uncompressed
`zstd`	Best tradeoff between de-/compression time and image size (default)
`gzip`	Balanced between speed and size, acceptable performance
`bzip2`	Rarely used, decent compression, resource conservative
`lzma`	Very small size, slow to compress
`xz`	Smallest size at longer compression time, RAM intensive compression
`lzop`	Slightly better compression than lz4, still fast to decompress
`lz4`	Fast decompression, slow compression, "largest" compressed output

> **NOTE:** See this article for a comprehensive comparison between compression algorithms.

COMPRESSION_OPTIONS

> **WARNING:** Misuse of this option may lead to an **unbootable system** if the kernel is unable to unpack the resultant archive. **Do not set** this option unless you're *absolutely* sure that you have to!

The `COMPRESSION_OPTIONS` setting allows you to pass additional parameters for the compression tool. Available parameters depend on the algorithm chosen for the `COMPRESSION` option. Refer to the tool's manual for available options. If left empty `mkinitcpio` will make sure it always produces a working image.

Revision #8
Created 12 February 2022 00:58:09 by Sebin
Updated 3 February 2025 20:09:05 by Sebin