Kurs administrowania systemem Linux

Zajęcia nr 9: Initramfs

Instytut Informatyki Uniwersytetu Wrocławskiego

29 kwietnia 2024

Rozruch komputera

- Najpierw *firmware* płyty głównej (UEFI, BIOS, Coreboot itp.) inicjalizuje urządzenia i uruchamia *bootloader* (Grub2, Syslinux, rEFInd, SystemD-Boot itp. — będzie potem).

Bootloader umieszcza w pamięci:

- plik z jądrem systemu operacyjnego `vmlinuz` (skompresowany),
- (opcjonalnie) plik `initrd.img` z *initramfs* (skompresowany),
- strukturę `boot_params` (tzw. *zero page*).

i przekazuje jądru sterowanie.

- Plik z jądrem ma format zImage (*zlib-compressed image*) lub bzImage (*big zImage*, $\geq$ 512KiB). W nagłówku zawiera kod wykonywalny, którego zadaniem jest przejęcie maszyny i rozpakowanie kodu jądra.
- Jądro może być uruchomione w trybie rzeczywistym, chronionym lub długim. Stronicowanie nie jest jeszcze włączone.
- Jądro, pracujące w trybie nadzorcy, uruchamia zarządzanie pamięcią wirtualną (inicjalizuje tablicę stron) i zarządzanie procesami (scheduler). Jest gotowe do uruchamiania procesów w trybie chronionym (w przestrzeni użytkownika).

Jądro zaczyna pracę

- `start()` z modułu `head.S` — przejmuje sterowanie, ew. włącza tryb chroniony lub długi, stronicowanie wyłączone;
- `decompress_kernel()` z modułu `misc.c` — wypakowuje „prawdziwe” jądro `vmlinux`;
- `startup_32()` z modułu `head.S` — inicjalizuje tablice stron i włącza stronicowanie;
- `start_kernel()` z modułu `head.S` — włącza: *scheduler*, `cpu_idle()` (PID 0) i wątki jądra (`kthreadd`, PID 2), po czym uruchamia pierwszy proces (`init`, PID 1) z przestrzeni użytkownika.

Kernel boot time parameters

- Bootloader przekazuje jądru zestaw parametrów zawierających informacje, które nie mogą być wkompilowane w jądro lub które zastępują domyślne wartości zapisane w jądrze.
- Przypominają argumenty przekazywane do programu przez `execve(2)`.
- Składnia parametru: $name[=value_1][,value_2]\ldots[,value_{10}]$
- Po uruchomieniu systemu parametry są dostępne w pliku `/proc/cmdline`.
- Parsowania argumentów dokonuje kod z pliku `init/main.c` jądra.
- Jądro rozumie parametry specjalne, takie jak `root=`, `ro`, `rw`, `debug`, `init` itp. Zbiór dostępnych parametrów zależy od wersji jądra.
- Parametry posiadające pojedynczą wartość, których jądro nie rozumie, są zamieniane na zmienne środowiskowe dla programu `init`. Pozostałe nieznane parametry są przekazywane jako argumenty do programu `init`.
- Zob. `bootparam(7)`.

Parametry specjalne jądra

- `root=/dev/`*`device`* — urządzenie zawierające *rootfs*. Prefiks `/dev/` jest jedynie konwencjonalny. Nowe jądra rozumieją także `root=UUID=`*`disk-uuid`*.
- `resume=/dev/`*`device`* — urządzenie zawierające zrzut pamięci podczas hibernacji (zwykle partycja wymiany).
- `ro`, `rw` — sposób zamontowania *rootfs*. Zwykle `ro`, żeby można było uruchomić `fsck(8)`. Potem zarządca serwisów robi `mount -no remount,rw /`
- `debug`, `quiet` — sterują zapisywaniem logów startowych do bufora logów jądra i na konsolę.
- `init=`*`path`* — ścieżka dostępu w *rootfs* do programu `init`. W razie braku jądro próbuje `/sbin/init`, `/etc/init`, `/bin/init`, `/bin/sh` i *kernel panic* (w podanej kolejności).

Rozruch przestrzeni użytkownika

Linux jest *dyskowym* systemem operacyjnym

- Programy uruchamiane przez jądro znajdują się na dysku i do pracy wymagają dostępu do dysku (pliki konfiguracyjne itp.).
- Także pierwszy program przestrzeni użytkownika (`init`) znajduje się na dysku.
- Wszelka komunikacja z jądrem odbywa się przez dysk (pseudosystemy `sysfs`, `proc`, `devtmpfs`, `devpts` i in.).

Wniosek

- Zanim jądro uruchomi przestrzeń użytkownika musi wpierw mieć dostęp do systemu plików *rootfs*.

Jak się montuje systemy plików?

- Każdy system plików jest reprezentowany w jądrze za pomocą struktury `file_system_type`.
- Globalna zmienna w jądrze `file_systems` jest wskaźnikiem (typu `file_system_type*`) na listę wszystkich zamontowanych systemów plików.
- Zmienna `file_systems` jest inicjowana przez funkcję `init_rootfs` podczas rozruchu jądra.
- Po zainicjowaniu lista `file_systems` nigdy nie jest pusta — ostatnim elementem jest specjalny system plików `rootfs`.
- Dawniej oryginalny `rootfs` był „systemem plików" zawierającym jedynie katalog `/`.
- Każdy następny system plików (w tym główny) są montowane zwyczajnie za pomocą syscalla `mount(2)`.
- Syscall `mount` wywołuje funkcję `register_filesystem`, która dodaje podany system plików do listy `file_systems`.

Jądro montuje system plików *root*

Skąd jądro wie, gdzie szukać systemu plików *rootfs*?

- Nazwa partycji dysku w której znajduje się system plików / jest przekazana poprzez *parametr bootowania*, np. `root=/dev/sda1` (zob. `bootparam(7)`).

Problemy: brak wsparcia dla jądra z przestrzeni użytkownika.

- Cały kod niezbędny do zamontowania systemu / musi być zawarty w jądrze — możliwe tylko w przypadku bardzo prostej konfiguracji.
- Jeszcze nie działa zarządzanie urządzeniami (`udevd`).
- Nie można uruchomić programów implementujących np. partycje wirtualne (LVM2), szyfrowanie (`dm-crypt`) lub dysk sieciowy (NFS).
- Nie można wyszukiwać systemów plików na podstawie UUID — niejednoznaczność numeracji dysków przez jądro.

Rozwiązanie: tymczasowy system plików zamontowany w /.

- Plik z systemem może być umieszczony w pamięci przez bootloader.
- Dwa rozwiązania: initrd (starsze) i initramfs (młodsze).

Ramdysk

Idea

- Urządzenie blokowe utworzone i przechowywane w RAM-ie.
- Nazwa: `/dev/ram0` itp.

Ograniczenia RAM dysków

- Z góry ustalony rozmiar.
- Nie może być swapowany.

Tworzenie

- Moduł jądra `brd`.
- Polecenie:
  `modprobe brd rd_size=1040576 rd_nr=1 max_part=0`
  tworzy `/dev/ram0` o rozmiarze 1 GiB.
- Parametr `rd_size` określa rozmiar (w KiB), `rd_nr` — liczbę urządzeń, a `max_part` — liczbę partycji na urządzenie (wolnych minorów).

Ramdysk startowy: initrd

- Zawartość przygotowana w postaci (zwykle skompresowanego) pliku ładowanego do pamięci przez *bootloader* zgodnie z konfiguracją *bootloadera*.
- *Bootloader* przekazuje poprzez strukturę `boot_params` informację o załadowania pliku do pamięci. Format tego pliku nie jest znany *bootloaderowi*.
- Sposób przekazania przez *bootloader* informacji o załadowanym pliku jest opisany przez *Linux/x86 Boot Protocol*.
- Obsługa initrd dodana do protokołu w wersji 2.0 (jądro 1.3.73).
- Plików initrd może by kilka. Jądro musi samo zdecydować, co zawierają.
- Obsługa bardziej skomplikowanych systemów plików, np. `ext4`, może być skompilowana w postaci modułu jądra zapisanego w pliku w ramdysku startowym.

Rozruch bez initrd jest możliwy

Jądro musi mieć statycznie wkompilowaną obsługę:

- magistrali dysku (SATA, PATA, SCSI, itp.),
- urządzeń blokowych (np. sterownik `sd_mod`),
- systemu plików, np. `ext4`.

Własności:

- W większości dystrybucji wymaga to skompilowania własnej wersji jądra.
- Jądro, które obsługiwałoby różne konfiguracje byłoby bardzo wielkie.
- Czas startu systemu skraca się zwykle o 0.5–4 s.
- Wymaga podania sporej liczby parametrów poprzez `bootparams`, np.:

```
rootfstype=ext4 root=/dev/sda2 rd.md=0 rd.lvm=0 rd.dm=0
SYSFONT=True KEYTABLE=us rd.luks=0 LANG=en_US.UTF-8
```

Format initrd

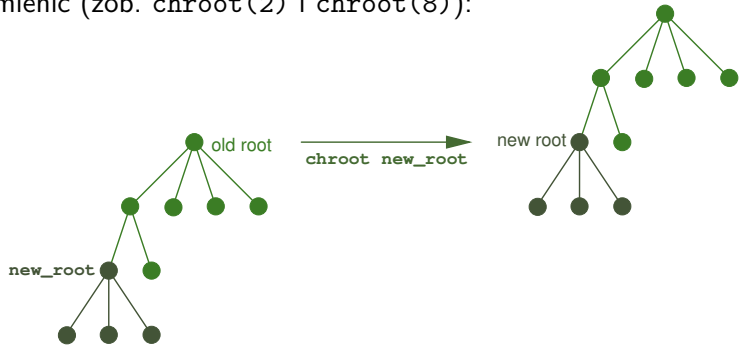
- Obraz systemu plików (np. `ext2`), zwykle skompresowany (zwykle `gzip`).
- Obsługa tego systemu plików oraz dekompresji musi być wkompilowana w jądro.
- *Bootloader* ładuje do pamięci jądro i plik `initrd.img` oraz przekazuje poprzez strukturę `boot_params` informację o załadowania pliku `initrd.img` do pamięci.
- Jądro ma dostęp do treści pliku `initrd.img` poprzez urządzenie blokowe `/dev/initrd` (tylko do odczytu).
- Jeśli jądro skompilowano do obsługi `initrd` a nie chcemy, żeby to robiło, należy podać opcję `noinitrd`.
- W przeciwnym razie jądro rozpakowuje `/dev/initrd` i kopiuje jego zawartość do `/dev/ram0`.
- Jądro montuje `/dev/ram0` jako `/` w trybie `rw`.

Format initrd, cd.

- Jądro uruchamia program `/linuxrc` (może być skryptem).
- Program `/linuxrc` służy głównie do skonfigurowania modułów jądra i sposobu montowania głównego systemu plików.
- Program `/linuxrc` powinien następnie zakończyć działanie.
- Jądro montuje główny system plików zgodnie z opcjami bootowania `root=` i `rw` lub `ro` (domyślnie `ro`), przechodzi do tego katalogu i wykonuje `pivot_root . /initrd`.
- Ramdysk pozostaje dostępny w katalogu `/initrd` (chyba że ten katalog nie istnieje, wtedy ramdysk jest odmontowany i usunięty z pamięci).
- Jądro uruchamia program opisany opcją bootowania `init=` (domyślnie `/sbin/init`).
- Debian 3.1 używał `cramfs` (pozwala na zamontowanie bez rozpakowywania) jako systemu plików initramfs.

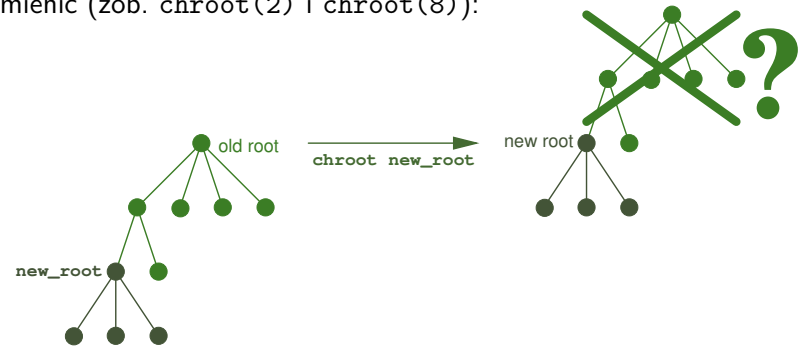
chroot

- Każdy proces posiada *root directory* i *current working directory*.
- *CWD* jest potomkiem *root* w drzewie katalogów.
- *Root directory* większości procesów pokrywa się z korzeniem zamontowanego systemu plików.
- Można to zmienić (zob. `chroot(2)` i `chroot(8)`):



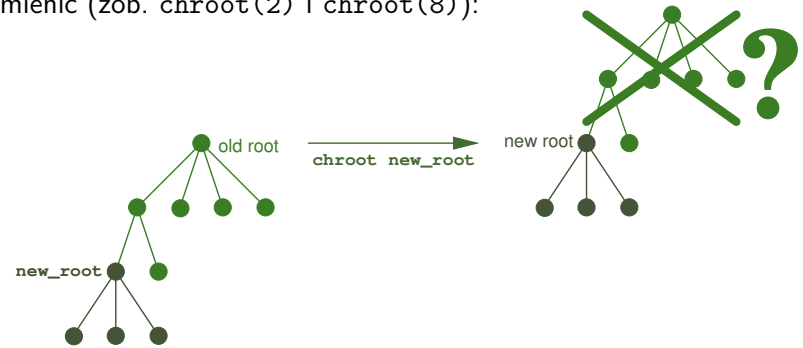
chroot

- Każdy proces posiada *root directory* i *current working directory*.
- *CWD* jest potomkiem *root* w drzewie katalogów.
- *Root directory* większości procesów pokrywa się z korzeniem zamontowanego systemu plików.
- Można to zmienić (zob. `chroot(2)` i `chroot(8)`):



chroot

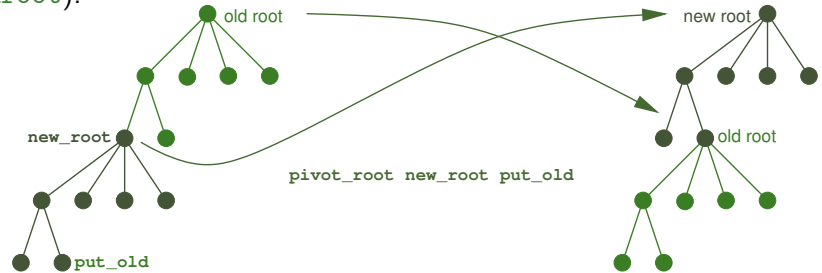
- Każdy proces posiada *root directory* i *current working directory*.
- *CWD* jest potomkiem *root* w drzewie katalogów.
- *Root directory* większości procesów pokrywa się z korzeniem zamontowanego systemu plików.
- Można to zmienić (zob. `chroot(2)` i `chroot(8)`):



- Aby móc usunąć (odmontować) old root, musi on być innym systemem plików niż new root! Można wtedy użyć `pivot_root`.

pivot_root

- `int pivot_root(const char *new_root, const char *put_old);`
  — zob. `pivot_root(2)`
- `# pivot_root new_root put_old` — zob. `pivot_root(8)`.
- `new_root` powinien być katalogiem (zwykle punktem montowania jakiegoś systemu plików)
- `put_old` powinien być podkatalogiem w drzewie `new_root`
- `new_root` powinien być w innym systemie plików niż bieżący *root*
- Proces powinien sam zadbać o aktualizację swojego katalogu głównego i bieżącego (za pomocą `chroot`).



Buforowanie urządzeń blokowych

Urządzenia blokowe są na dole piramidy pamięci

- RAM traktujemy jako *cache* dla urządzeń blokowych.
- Każda strona pamięci RAM jest odwzorowana na pewien blok dysku.
- Flaga *dirty* określa, czy strona wymaga synchronizacji z dyskiem.
- Specjalny wątek jądra zajmuje się synchronizacją.

Odwzorowania stron pamięci

- Pamięć anonimowa — partycja wymiany (*swap*).
- Bufory urządzenia blokowego — urządzenie blokowe.
- Bufory systemu plików — partycja z systemem plików.

A gdyby tak wyłączyć synchronizację brudnych stron?

- Pamięć anonimowa — działa nawet w razie braku przestrzeni wymiany.
- Bufory urządzenia blokowego — ramdysk.
- Bufory systemu plików — ramfs.

Niesynchronizowane bufory

Ramdysk

- Obecnie mało popularne.
- Wyparte przez:
  - urządzenia *loopback* (bufory synchronizowane z plikiem, a nie urządzeniem blokowym),
  - skompresowane ramdyski (`zram`).

Ramfs

- Wady:
  - brak ograniczenia rozmiaru (może zużyć całą pamięć),
  - brak możliwości tymczasowego przeniesienia do przestrzeni wymiany.

Tmpfs

- Ramfs rozszerzony o powyższe możliwości.
- Bardzo prosta implementacja — nawet brak możliwości wyłączenia podczas kompilacji.
- Obecnie oryginalny rootfs to tmpfs.

Tmpfs

- System plików `tmpfs` tworzony przez jądro w pamięci operacyjnej.
- Funkcja `mount(2)` i polecenie `mount(8)` pozwalają zlecić jądru utworzenie tmpfs:

```
mkdir /var/mytmpfs
mount -t tmpfs -o size=100M none /var/mytmpfs
mount -t tmpfs -o remount,size=200M none /var/mytmpfs
umount /var/mytmpfs
```

- Można umieścić odpowiedni wpis w `/etc/fstab`:

```
none /var/mytmpfs tmpfs size=100M 0 0
```

- Można też np. `size=40%` (procent dostępnej pamięci RAM).
- Domyślnie 50% RAM.
- Uwaga: jednostki `Ki`, `Mi`, `Gi` opisane w `mount(8)` nie działają!

Standardowe tmpfs-y w systemie

Pięć systemów tmpfs jest tworzone podczas startu systemu (przed wykonaniem `/etc/fstab`):

- `/run`,
- `/run/lock`,
- `/dev/shm`,
- `/tmp` (opcjonalnie),
- `/sys/fs/cgroup` (zawiera punkty montowania cgroupfs).

W SysV Init:

- Uruchamiane za pomocą skryptu startowego `/etc/init.d/mountall.sh`.
- Konfiguracja w `/etc/default/tmpfs` (dawniej `/etc/default/rcS`).
- `/tmp` domyślnie wyłączony. Trzeba dodać `RAMTMP=yes`.
- Manual (nieaktualny): `tmpfs(5)`.

Przykład: /tmp w SystemD

W SystemD:

- Odpowiednia jednostka punktu montowania `tmp.mount`.
- Domyślnie jej nie ma (do SystemD 220-5 była).
- Przykładowy plik konfiguracyjny znajduje się w `/usr/share/systemd/tmp.mount`.
- Aby włączyć `/tmp` w RAM trzeba skopiować ten plik do `/etc/systemd/` i wykonać:

  ```
  systemctl enable tmp.mount
  ```

- Zmiana konfiguracji: edycja pliku `/etc/systemd/tmp.mount`.

Współczesne rozwiązanie początkowego systemu plików

Idea

- Skoro oryginalny rootfs to tmpfs, czyli:
  - w pełni funkcjonalny system plików,
  - początkowo pusty — zawiera tylko katalog /,

  to zamiast montować inny system plików w katalogu / można
  - skopiować do niego zawartość jakiegoś archiwum plików.

Format initramfs

- Dostępny od jądra 2.6.13.
- Plik initrd w tej wersji to zwykłe archwum w formacie CPIO (podobny, ale starszy niż TAR), opcjonalnie skompresowane (gzip, bzip2, LZMA, XZ, LZO lub LZ4).
- Jądro rozpakowuje plik (załadowany do pamieci przez bootloader) do rootfs.
- W porównaniu z formatem initrd: jądro musi znać format CPIO i umieć tworzyć tmpfs, ale nie musi obsługiwać żadnego innego systemu plików (np. `ext2`). Tworzenie archiwum CPIO jest wygodniejsze niż kompresowanie obrazu urządzenia blokowego z systemu plików.

Działanie dysku startowego w formacie initramfs

- Jądro uruchamia program `/init` (może być skryptem powłoki).
- Program `/init` przejmuje na stałe przestrzeń użytkownika i nie powinien zwrócić sterowania do jądra.
- Program `/init` powinien samodzielnie zamontować główny system plików.
- Zwykle honoruje opcje jądra `root=`, `init=` itp., ale może je zignorować.
- Po zamontowaniu systemu plików nie może wykonać `pivot_root`, bo oryginalnego `rootfs` nie można odmontować!
- Zamiast tego odpowiednia funkcja biblioteczna, np. `switch_root` w Busybox lub `run_init` w klibc poprzez `exec`.

switch_root

- Zob. `switch_root(8)`.
- `switch_root` *`newroot init [arg...]`*
- Przemontowuje katalogi `/sys`, `/proc` i `/dev` do *`newroot`*.
- Usuwa cały stary tmpfs z pamięci (!!!).
- Montuje (z opcją `--move`) katalog *`newroot`* w miejscu `/`.
- Uruchamia poprzez `exec` podany program *`init`*.

Systemy działające tylko w RAM

- Skoro `/init` w initramfs nie kończy działania, to może wykonywać dowolne czynności.
- Przełączanie głównego systemu plików nie jest konieczne!
- Ramdysk może być *docelowym* głównym systemem plików!
- Archiwum CPIO musi wtedy zawierać docelową konfigurację głównego systemu plików.
- Nie wszystko trzeba ładować do pamięci! Np. katalog `/usr` można montować w osobnym systemie plików z dysku.
- Tak działają Live CD (płyta CD montowana *read only*), w szczególności instalatory systemów.
- Można korzystać z *unionfs*: system plików rw (np. tmpfs lub pendrive) montowany *na* systemie ro (np. CDROM).
- Tiny Core Linux, Puppy Linux i Alpine Linux mogą docelowo działać w ten sposób.
- Wygodne w połączeniu z PXE — pozwala uruchamiać *diskless stations* bez konieczności montowania głównego systemu plików przez NFS.

Eksperyment: przejście na system plików w tmpfs

```
mkdir /new_root
mount -t tmpfs -o size=500M none /new_root
cd /new_root
debootstrap bullseye . http://ftp.pl.debian.org/debian/
mount -t proc none proc
mount -t sysfs none sys
mount -o bind /dev dev
mkdir old_root
unshare -m
pivot_root . old_root
chroot .
```

Przykład: pivot_root do zaszyfrowanego dysku

```
# Stary główny system plików. Są na nim takie narzędzia,
# jak cryptestup. Przygotowujemy i montujemy nowy system
# plików z zaszyfrowanego dysku.
mkdir -p /new_root
cryptsetup open /dev/sda2 new_root
mount /dev/mapper/new_root /new_root
mkdir -p /new_root/old_root
cd /new_root
pivot_root . old_root
# Katalog główny (root) i bieżący (cwd) powłoki odnosi się
# jeszcze do starego systemu plików.
exec chroot . sh <dev/console >dev/console 2>&1
# Teraz powłoka działa w nowym systemie plików.
# Stary system plików jest dostępny w katalogu /old_root
umount /old_root
```

Initrd we współczesnych dystrybucjach

- Prawie wszyscy używają formatu initramfs.
- W Debianie initramfs dobrany do konkretnej konfiguracji. Zmiana konfiguracji wymaga utworzenia nowej wersji initramfs.
- W Ubuntu i Fedorze uniwersalny initramfs, który się dynamicznie konfiguruje (dużo heurystyk i wyszukiwania!).
- Powłoka i podstawowe narzędzia — zwykle BusyBox: mały, nie wymaga glibc — używa klibc, implementuje ponad 200 poleceń, pozwala na współdzielenie kodu bez potrzeby tworzenia bibliotek współdzielonych.

Archiwum CPIO

- Rozpakowywanie:
  ```
  zcat /boot/initrd.img | cpio -i
  ```
- Pakowanie:
  ```
  find -print0 | cpio --null -v -o --format=newc | \
     gzip -9 > ../initrd.img
  ```
- Listowanie zawartości:
  ```
  zcat /boot/initrd.img | cpio --extract --quiet --list
  ```
  (por. `lsinitramfs(1)` z pakietu `initramfs-tools`).

Minimalistyczny initramfs (Gentoo wiki)

```
#!/bin/busybox sh

mount -t proc none /proc
mount -t sysfs none /sys

echo "This script just mounts and boots the rootfs"

mount -o ro /dev/sda1 /mnt/root

umount /proc
umount /sys

exec switch_root /mnt/root /sbin/init
```

Automatyzacja tworzenia initramfs: initramfs-tools

- Specjalne polecenia `lsinitramfs`, `mkinitramfs` i `update-initramfs`.
- Konfiguracja w `/etc/initramfs-tools/`.
- `/{usr/share,etc}/intramfs-tools/hooks/` — skrypty wykonywane podczas tworzenia initrmfs (`run-parts(8)`).
- `/{usr/share,etc}/intramfs-tools/scripts/` — skrypty wykonywane podczas pracy intitrmfs.
- Podkatalogi: `init-top`, `init-premount`, `local-top`, `local-block`, `local-premount`, `local-bottom`, `init-bottom` zawierają skrypty wykonywane podczas kolejnych faz rozruchu (zob. `initramfs-tools(8)`).
- Plików initramfs nie należy zmieniać ręcznie, ale dodawać, usuwać i modyfikować za pomocą polecenia `update-initramfs`.
- Tworzenie lokalnego pliku initramfs: `mkinitramfs`.

Przykład: zmiana czcionki w konsoli w initramfs

- Zmiana czcionki w konsoli: `setfont(8)`.
- Konfiguracja w `/etc/console-fonts/` generowana przez `setupcon(8)` zgodnie z `console-setup(5)` (`/etc/default/console-setup`).
- Podczas startu systemu czcionkę konfiguruje `/etc/init.d/console-setup`
- Podczas pracy initramfs czcionka jest standardowa (8×16).
  Na ekranie o rozdzielczości > 140 dpi — nieczytelna.
- Trzeba dodać wywołanie `setfont(8)` do initramfs.
- Hook wykonywany podczas generowania initramfs:
  `/etc/initramfs-tools/hooks/z_cfont`
- Skrypt wykonywany podczas pracy `init` w initramfs:
  `/etc/initramfs-tools/scripts/init-top/z_cfont`

/etc/initramfs-tools/hooks/z_cfont

```
if [ ! -x /bin/setfont ]; then
   echo "setfont is missing."
   echo "Please install the 'kbd' package."
   exit 0
fi

. /usr/share/initramfs-tools/hook-functions

copy_exec /bin/setfont /bin
mkdir -p ${DESTDIR}/etc/console-setup/
cp /etc/console-setup/cached_Uni2-Terminus24x12.psf.gz \
   ${DESTDIR}/etc/console-setup/

exit 0
```

/etc/initramfs-tools/scripts/init-top/z_cfont

```
FONT=/etc/console-setup/cached_Uni2-Terminus24x12.psf.gz

if [ -x /bin/setfont ] && [ -r "$FONT" ]
then
   for N in {1..6}
   do
      setfont -C /dev/tty$N $FONT
   done
fi
```

Inne rozwiązania

Dracut

- Alternatywne narzędzie do generowania initramfs.
- W systemach z SystemD ponownie przełącza rootfs do ramfs podczas zamykania systemu — pozwala odmontować dysk przez zatrzymaniem systemu.

Działanie z RAM lub dysku RO

- Liczne systemy tworzenia LiveCD, np. *Debian Live Systems Project*.