

Kurs administrowania systemem Linux

Zajęcia nr 8: BSD Init i SystemD

Instytut Informatyki Uniwersytetu Wrocławskiego

22 kwietnia 2024

Demon — proces działający w tle zwykle przez cały czas pracy systemu.

- Nie posiada terminala, jego stdin to zwykle `/dev/null`.
- Jego ojcem jest `/sbin/init` (w SysV Init na skutek osierocenia przez skrypt inicjalizacyjny i *reparentingu*, w SystemD oryginalnie).
- Komunikuje się z innymi procesami poprzez mechanizmy IPC (np. sygnały, gniazda, kolejki wiadomości, nazwane potoki) i magistrale (np. dBus).

Trivia

- Angielska pisownia: „daemon” (dawna wersja współczesnej „demon”).
- Nawiązuje do idei demona Maxwella.
- Wymyślona przez członków MIT Project MAC na początku lat '60-tych.

Serwisy

- Usługi systemu.
- Często uruchamiane podczas startu systemu.
- Często realizowane przez demony.

- PPID=1
- CWD=/
 - Działają w osobnej sesji (wykonują `setsid(2)`).
 - `stdin`, `stdout`, `stderr` są połączone z `/dev/null`.
 - Zwykle komunikują się z otoczeniem poprzez gniazda.
 - Komunikaty diagnostyczne piszą poprzez `syslog`.
 - Zwykle uruchamiane za pomocą serwisów.

Uruchamianie i zatrzymywanie serwisów

Klasycznie w System V init

```
# /etc/init.d/serwis [start | stop]
```

„Nowocześnie” w System V init

```
# service serwis [start | stop]
```

W systemd

```
# systemctl [start | stop] serwis[.service]
```

Proces, który pragnie się zdemonizować w SysV Init (zob. `daemon(7)`), powinien:

- Zamknąć wszystkie deskryptory plików.
- Podłączyć `stdin`, `stdout` i `stderr` do `/dev/null`.
- Usunąć własną obsługę sygnałów i zresetować maskę sygnałów.
- Usunąć zbędne zmienne ze środowiska.
- `chdir(2)` na `/` i `umask(2)` na `0`, aby nie blokować systemu plików.
- Osierocić się i utworzyć własną sesję, w której nie jest liderem (np. `fork()`, `setsid()`, `fork()`, po czym oryginalny proces i pierwszy potomny wykonują `exit()`).

Alternatywy:

- Program `nohup(1)` zapewnia namiastkę demonizacji dla zwykłych programów.
- SystemD nie wymaga od procesów demonizacji, wykonuje ją sam (zob. `systemd.exec(5)`). W SystemD samodzielna demonizacja przeszkadza w monitoringu demonów.

Uruchamianie demonów:

- Przed zdemonizowaniem proces powinien sprawdzić, czy demon nie jest już uruchomiony. Robi to zwykle zapisując swój PID do pliku `/run/program.pid`.
- W Debianie specjalny skrypt `start-stop-daemon(8)`.

- Zbiory procesów tworzone na potrzeby zarządzania zadaniami (*jobs*).
- Zarządzanie polega głównie na zarządzaniu dostępem do terminala sterującego i wysyłaniu *sygnałów* do wszystkich członków grupy.
- Grupy są podzbiorami sesji.
- Tylko jedna grupa procesów — grupa *pierwszoplanowa* (*foreground*) — ma `stdin` podłączony do terminala. Pozostałe są zatrzymane lub pracują w tle (*background*).
- Podczas tworzenia przez powłokę rurociągu wszystkie procesy są przypisane do jednej grupy.
- Grupa ma lidera.
- Liderem rurociągu jest jego pierwszy proces.
- PGID (*Process Group ID*) grupy jest PID jej lidera.
- W odróżnieniu o sesji grupy nie zwracają uwagi na śmierć lidera.
- Proces będący członkiem grupy może nie być potomkiem lidera (por. *reparenting*).

- Wysłanie sygnału do wszystkich członków grupy: `killpg(2)`, także `kill(1)` z `PID < -1`.
- `<ctrl>-Z` jest przechwytywane przez jądro i powoduje wysłanie sygnału `SIGTSTP` do wszystkich procesów grupy pierwszoplanowej. Grupą pierwszoplanową staje się grupa, która wywołała wstrzymaną grupę. Powłoka *maskuje* `SIGTSTP`.
- `<ctrl>-C` jest przechwytywane przez jądro i powoduje wysłanie sygnału `SIGINT` do wszystkich procesów grupy pierwszoplanowej.
- Polecenia powłoki: `jobs(1)` ujawnia listę zadań (grup procesów) w bieżącej sesji, `fg(1)` przenosi zadanie na pierwszy plan, `bg(1)` wznowia zadanie zatrzymane (poprzez wysłanie `SIGSTOP` do grupy procesów) w tle (poprzez wysłanie `SIGCONT` do grupy procesów).

Dostępne zasoby dla procesu

- Funkcja `ulimit(3)` i polecenie wbudowane powłoki `ulimit` (liczne opcje: `-HSTabdefilmnpqrstuvx`).

Priorytet procesu

- Liczba z przedziału `-20...19`. Zwykle ujemne priorytety może nadawać tylko root (por. `ulimit -e`).
- Syscall `nice(2)` i polecenia `nice(1)` i `renice(1)`. Zadania wsadowe uruchamiać poleceniem:
`nice -n19 program`

Selektywne nadawanie uprawnień programom

- *Capabilities*, zob. `capabilities(7)`, `setcap(8)`, `getcap(8)` — bezpieczniejsze niż `setuid`.

Cgroups i namespaces

- Rozbudowane możliwości grupowania procesów i zarządzania nimi.

To już wiemy:

- Pierwszy proces uruchamiany przez jądro po zamontowaniu głównego systemu plików.
- Zwykle ma PID=1 i PPID=0 i jest przodkiem każdego procesu w przestrzeni użytkownika.
- Odpowiednik w jądrze: `kthreadd` (PID=2 i PPID=0), który jest ojcem każdego wątku w przestrzeni jądra.
- Pracuje przez cały czas życia systemu i umiera jako ostatni podczas zatrzymania.
- Automatycznie adoptuje procesy, które straciły rodziców i zajmuje się nimi (np. usuwa *zombie*).

Teraz zbadamy jego podstawową rolę:

- Nadzoruje uruchomienie i zatrzymanie systemu.

Konfiguracja jądra

- Adres IP komputera, np. 192.168.1.1
- Maska sieci — wycina numer sieci z adresu IP, np. 255.255.255.0. Adresem sieci jest w tym przykładzie 192.168.1.0. Adres IP i maskę podaje się często w notacji CIDR, np.: 192.168.1.1/24.
- Adres bramy domyślnej, np. 192.168.1.254. Musi być w tej samej sieci, co komputer. Zbyteczny w razie komunikacji lokalnej.

Konfiguracja niezbędnych usług w przestrzeni użytkownika

- Adresy serwerów DNS (53/udp)

Dodatkowe usługi

- Adresy serwerów czasu (NTP, 123/udp)
- Adresy serwerów SMB, Netbios itp.
- PXE: protokoły BOOTP i TFTP.

Przykład: konfiguracja stosu protokołów sieciowych jądra

- Komunikacja jądra z przestrzenią użytkownika: interfejs gniazdowy NETLINK.
- Programy takie jak `ip(1)` zapewniają CLI.

Przykład statycznej konfiguracji usług sieciowych

```
ip link set up dev eth0
ip address add 10.13.1.7/16 dev eth0
ip route add default via 10.13.1.1
```

Do tego trzeba skonfigurować usługi warstwy aplikacji (DNS i in.):

```
echo "nameserver 8.8.8.8" | resolvconf -a eth0
```

- Można umieścić w skrypcie powłoki wykonywanym podczas uruchamiania systemu.
- Lepiej: specjalne programy i pliki konfiguracyjne.

Konfigurowanie jądra

Narzędzie CLI do konfigurowania stosu protokołów sieciowych

```
ip [ link | addr | addrlabel | route | rule | neigh | ntable |  
    tunnel | tuntap | maddr | mroute | mrule | monitor | xfrm |  
    netns | l2tp | tcp_metrics ]
```

- Warto znać link, addr i route.
- Inne ważne narzędzia: ethtool i ss.
- *Legacy*: ifconfig, route, netstat, brctl, vconfig.

Przykłady

- `ip link set eth0 up`
- `ip link set eth0 address xx:xx:xx:xx:xx:xx`
- `ip link set eth0 name remote`
- `ip link -d`
- `ip addr add 192.168.1.1/24 dev eth0`
- `ip addr flush dev eth0`
- `ip route add default via 192.168.1.254`

Konfigurowanie DNS

- Biblioteka GNU NSswitch (glibc6), plik `nsswitch.conf(5)`.
- Plik `hosts(5)`.
- Plik `resolv.conf(5)` i opcja `nameserver`.
- Funkcje biblioteczne `getaddrinfo(3)`, `getnameinfo(3)`, `gethostbyname(3)` itp.
- Program GNU `resolvconf(8)` (pakiet `resolvconf`).
- Program `getent(1)`, np. `getent hosts dns-name`.
- Programy `host(1)`, `dig(1)`, `nslookup(1)`.
- Serwery DNS Google'a: 8.8.8.8, 8.8.4.4.
- Lokalne serwery DNS. Wpisy lokalne. Kwestia poufności.

Konfiguracja automatyczna: serwis DHCP

- Automatycznie konfiguruje jądro i usługi przestrzeni użytkownika.
- Wymaga działającego serwera DHCP w sieci.
- Polecenie `dhclient(8)`. Opcje `-v`, `-x`, `-r`.
- Uwaga: polecenie `dhclient(8)` się demonizuje!
Wyłączyć (opcje `-x` lub `-r`) przed ponownym uruchomieniem.

Przykład: konfiguracja sieci w Debianie

- Specjalny program `ifup` (alias `ifdown`, `ifquery`).
- Główny plik konfiguracyjny: `/etc/network/interfaces`.
- Polecenie `ifup -a` uruchamiane podczas startu systemu, a polecenie `ifdown -a` — podczas zatrzymania.
- W katalogach `/etc/network/{if-pre-up,if-up,if-down,if-post-down}.d/` można umieszczać skrypty wykonywane podczas zmiany stanu interfejsów. Programy, które używają sieci (np. demon `sshd`) mogą umieszczać tam swoje *hooki*.
- Współpracuje z pakietami konfiguracyjnymi różne usługi warstwy aplikacji (DNS, NTP, VPN itp.)
- Zob. `ifup(8)`, `interfaces(5)`.

Przykładowy plik /etc/network/interfaces

```
auto lo
interface lo inet loopback

allow-hotplug eth0
interface eth0 inet static
    address 10.13.1.7
    netmask 255.255.0.0
    gateway 10.13.1.1
    # optional
    network 10.13.0.0
    broadcast 10.13.255.255
    # dns-* options are implemented by the resolvconf
    # package, if installed
    dns-nameserver 8.8.8.8
    # openvpn option is implemented by the openvpn package,
    # if installed
    openvpn privnet
```

Inny przykład konfiguracji interfaces(5)

```
iface dom inet static
    address 192.168.1.1
    netmask 255.255.255.0
    gateway 192.168.1.254
    dns-nameservers 8.8.8.8 8.8.4.4
    openvpn praca
iface szkola inet dhcp
iface domwifi inet dhcp
    wpa-ssid domektomek
    wpa-psk 050edd02ad44627b16ce0151668f5f53c01b
```

Użycie:

- ifup eth0=szkola
- ifup wlan0=domwifi
- Pamiętaj: co się ifup-owało trzeba ifdown-ować!

Trwała konfiguracja (przywracana podczas startu systemu)

Klasyczne rozwiązanie w Debianie

- Katalog: `/etc/network/`
- Plik: `interfaces(5)` i podkatalog `interfaces.d`
- Polecenia: `ifup(8)`, `ifdown(8)`
- Podkatalogi: `if-pre-up.d`, `if-up.d`, `if-down.d`, `if-post-down.d`.
- Mechanizm `run-parts(8)` — *plugins* dla różnych usług sieciowych: DNS (`resolvconf(8)`), VLAN-y, WiFi itp.
- Uruchamiane przez serwis `/etc/init.d/networking` (SysV Init) lub `networking.target` (systemd).

Systemd

- Jednostki `*.link` i `*.network`

Rozwiązanie à la Windows: Network Manager

- W Debianie Network Manager ignoruje interfejsy wymienione w `interfaces(5)`.

RC (*run command*)

- `/sbin/init` musi umieć uruchamiać programy takie jak `ifup/ifdown` podczas startu/zatrzymania systemu.
- Klasyka (Research Unix, BSD): skrypty powłoki `/etc/rc` i `/etc/rc.shutdown` uruchamiane (`execve`) przez `/sbin/init`. Dodatkowo pliki konfiguracji domyślnej (`/etc/default/rc.conf`) i lokalnej (`/etc/rc.conf`). Konsole konfigurowane bezpośrednio przez `/sbin/init` zgodnie z plikiem `/etc/ttys`. W późniejszych systemach dodatkowo skrypt lokalny `/etc/rc.local`.
- Modularyzacja (FreeBSD ≥ 5.0 i NetBSD ≥ 1.5): zestaw skryptów w katalogu `/etc/rc.d/` uruchamianych przez `/etc/rc` zgodnie z konfiguracją w `rc.conf`.
- W Unikсах Systemu V i niektórych dystrybucjach Linuksa bardziej rozbudowany SysV Init.
- W nowych dystrybucjach Linuksa — SystemD.
- Wiele innych rozwiązań, np. OpenRC (Gentoo, także Alpine).

- Pojedynczy skrypt lub plik konfiguracyjny — wygodne do edytowania przez administratora, ale *bardzo* kłopotliwe do automatycznej aktualizacji.
- Bardzo utrudniają np. automatyczne instalowanie pakietów, które wymagają zmiany konfiguracji wielu aspektów systemu.
- Zaleta: po instalacji każdego programu administrator przechodzi „szkolenie” z konfigurowania tego programu, inaczej nie zostanie on uruchomiony!
- Anegdota: Cisco Packet Tracer i `/etc/profile`.

Popularne rozwiązanie modularyzacji konfiguracji

- Główny plik konfiguracyjny programu *programe* to */etc/programe.conf* lub */etc/programe*.
- Dodatkowo katalog */etc/programe.d/*.
- W głównym pliku konfiguracyjnym instrukcja *include /etc/programe.d/**
- Przykłady: *apt/sources.list*, *bash_completion*, */etc/network/interfaces*, *ld.so*, *pam*, *profile*, *rsyslog*, *sysctl*, *timezone* itd.
- Instalator programu *otherprog* dodaje osobny plik konfiguracyjny programu *programe*: */etc/programe.d/otherprog.conf*.

run-parts — uniwersalne narzędzie

```
run-parts [ --test  
           --list  
           --verbose  
           --record ] [ --regex=RE ] [ --arg=arg ...] dir
```

- Uruchamia w kolejności alfabetycznej wszystkie pliki wykonywalne w katalogu *dir* których nazwy pasują do wzorca *RE* przekazując im jako parametry argumenty *arg*.
- Dalsze opcje: `--reverse`, `--exit-on-error`, `--new-session`, `--umask=umask`, `--lsbsysinit`.
- Blokowanie skryptów przed wykonaniem przez `run-parts`: odebrać prawa do wykonania.
- Sprawdzanie, które skrypty zostaną wykonane: `run-parts --test`.

ifup/ifdown

- Program ifup/ifdown wykonuje `run-parts /etc/network/if-option.d`, gdzie *option* to `pre-up`, `up`, `down`, `post-down` odpowiednio przed i po uruchomieniu oraz przed i po zatrzymaniu interfejsu.
- Przekazywane zmienne środowiskowe: `IFACE`, `LOGICAL`, `ADDRFAM`, `METHOD`, `MODE`, `PHASE`, `VERBOSITY`, `PATH`, `IF_*`.
- Dowolny program, którego praca zależy od konfiguracji interfejsów może umieścić w tych katalogach swoje skrypty, które wykonają odpowiednie czynności podczas zmiany stanu interfejsów.

cron

- Raz na godzinę/dobę/tydzień/miesiąc demon `cron`d wykonuje `run-parts /etc/cron.{hourly,daily,weekly,monthly}/`
- Dowolny program, który wymaga okresowego wykonywania pewnych czynności, może umieścić w tych katalogach swoje skrypty.

Cechy procesu init

- Najważniejszy demon w systemie.
- Jako jedyny proces jest uruchamiany bezpośrednio przez jądro.
- Jest pierwszym procesem uruchomionym w przestrzeni użytkownika.
- Jest przodkiem każdego procesu.
- Kończy działanie jako ostatni.

Zadania procesu init

- Rozruch systemu:
 - skonfigurowanie systemu po uruchomieniu (montowanie dysków, konfiguracja sieci itp.),
 - uruchomienie i nadzorowanie pracy wszystkich serwisów.
- Zamknięcie systemu:
 - dekonfiguracja systemu (np. odmontowanie dysków itp.),
 - zatrzymanie serwisów,
 - wyłączenie maszyny.
- Reparenting osieroconych procesów i sprzątanie po zakończonych procesach.

Skrypty powłoki

- `/etc/rc` — uruchamiany przez `/sbin/init` podczas rozruchu systemu.
- `/etc/rc.shutdown` — uruchamiany przez `/bin/init` podczas zatrzymania systemu.
- `/etc/rc.d/service` — po jednym dla każdego serwisu.
- `/etc/defaults/rc.conf` — konfiguracja domyślna.
- `/etc/rc.conf` — konfiguracja lokalna systemu.

Skrypty `/etc/rc` i `/etc/rc.shutdown`

- Wczytują pliki `/etc/defaults/rc.conf` i `/etc/rc.conf`.
- Używają programu `rcorder(8)` do ustalenia kolejności uruchamiania/zatrzymywania serwisów (jedyne nie-skrypt w systemie!)
- W podanej kolejności wykonują skrypty `/etc/rc.d/service`.

Przykład: ssh

/etc/rc.d/ssh:

```
#!/bin/sh
```

```
# PROVIDE: sshd
# REQUIRE: LOGIN FILESYSTEMS
# KEYWORD: shutdown
```

```
. /etc/rc.subr
```

```
name="sshd"
desc="Secure Shell Daemon"
rcvar="sshd_enable"
command="/usr/sbin/${name}"
keygen_cmd="sshd_keygen"
start_precmd="sshd_precmd"
reload_precmd="sshd_configtest"
restart_precmd="sshd_configtest"
configtest_cmd="sshd_configtest"
pidfile="/var/run/${name}.pid"
extra_commands="configtest keygen reload"
...
load_rc_config $name
run_rc_command "$1"
```

/etc/defaults/rc.conf:

```
...
sshd_enable="NO"
sshd_program="/usr/sbin/sshd"
sshd_flags=""
...
```

/etc/rc.conf:

```
...
sshd_enable="YES"
...
```

Przykład: mój /etc/rc.conf (FreeBSD 13)

```
hostname="host"
keymap="pl.kbd"
ifconfig_re0="DHCP"
kld_list="i915kms.ko"
dumpdev="NO"
local_unbound_enable="NO"
sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
sshd_enable="YES"
ntpd_enable="YES"
powerd_enable="YES"
zfs_enable="YES"
sddm_enable="YES"
dbus_enable="YES"
hald_enable="YES"
clear_tmp_enable="YES"
geli_groups="storage"
geli_storage_flags="-k /etc/crypto.key -p"
geli_storage_devices=\
    "label/storage_disk1 label/storage_disk2 label/storage_disk3 gpt/zstorage_slog"
devfs_system_ruleset="localrules"
```

Zalety

- Demon `/sbin/init` uruchamia pojedynczy skrypt powłoki podczas uruchamiania/zatrzymania systemu i więcej się nie interesuje rozruchem/zatrzymaniem.
- Cały rozruch/zatrzymanie jest wykonywany *wyłącznie* przez zbiór skryptów powłoki.
- Konfigurowanie prostej instalacji ogranicza się do edycji pojedynczego pliku.

Wady

- Sekwencyjny (cały rozruch w pojedynczym wątku procesora).
- Powolny (wykonanie skryptów powłoki).
- Praktycznie brak nadzoru nad uruchomionymi serwisami.

Dygresja: metodologie programowania

Imperatywne

```
silnia = 1;
while (n > 0) {
    silnia *= n--;
}
```

Pułapki

- Niejasne działanie dla $n < 0$.
- Zmienia wartość n .

Deklaratywne

```
silnia n
| n > 0  = n * silnia (n-1)
| n == 0 = 1
| n < 0  = undefined
```

Zalety

- Opis tego, co chcemy osiągnąć, a nie *jak* to zrobić.

Programowania deklaratywne rozwinęło się wraz z językami funkcyjnymi (Lisp, Hope, SML, Miranda, Haskell). Sukcesy w głównym nurcie informatyki:

- Języki skryptowe: Python, Ruby.
- Języki zapytań w bazach danych: SQL.
- Systemy zarządzania konfiguracją: CFEngine, Puppet.
- Init systemy: OpenRC, SystemD.

- Lennart Poettering, Kay Sievers, Harald Hoyer, Daniel Mack, Tom Gundersen, David Herrmann, 2010.
- Nowe podejście nawiązujące do idei *dependency-based boot*.
- Składniki systemu są opisane za pomocą osobnych plików konfiguracyjnych.
- Brak skryptów rc: wszystkie czynności wykonuje `/sbin/init`.
- Jednostki SystemD opisują zasoby i zależności pomiędzy nimi.
- Katalogi zawierające jednostki: `/etc,run,lib}/systemd/system/`, w podanej kolejności (np. plik w `etc` przesłania `lib`).
- Składnia jednostek: inspirowana przez pliki XDG `*.desktop` i pliki `*.ini` Microsoftu.

SystemD

systemd Utilities

systemctl journalctl notify analyze cglsgcgtop loginctl nspawn

systemd Daemons

systemd
journald networkd
logind user session

systemd Targets

bootmode basic multi-user graphical user-session
shutdown reboot dbus telephony dlog logind user-session display service
tizen service

systemd Core

manager unit login namespace log
systemd service timer mount target multiseat inhibit cgroup dbus
snapshot path socket swap session pam

systemd Libraries

dbus-1 libpam libcap libcryptsetup tcpwrapper libaudit libnotify

Linux Kernel

cgroups autofs kdbus

Składnia jednostki — przykład

[Unit]

Description=OpenBSD Secure Shell server

After=network.target auditd.service

ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]

EnvironmentFile=-/etc/default/ssh

ExecStartPre=/usr/sbin/sshd -t

ExecStart=/usr/sbin/sshd -D \$SSHD_OPTS

ExecReload=/usr/sbin/sshd -t

ExecReload=/bin/kill -HUP \$MAINPID

KillMode=process

Restart=on-failure

[Install]

WantedBy=multi-user.target

Alias=sshd.service

Jednostki systemd w porównaniu z SysV Init

`service` serwis

`socket` gniazdo; usługa podobna do `xinetd`

`device` konfiguracja urządzeń

`mount` punkt montowania; odpowiada pojedynczemu wierszowi `/etc/fstab`

`automount` punkty automontowania

`swap` partycja lub plik wymiany

`target` cel konfiguracji; odpowiada `runlevelowi`

`path` ścieżka w systemie do katalogu lub pliku

`timer` odpowiada usłudze `cron`

`snapshot` zapisany stan SystemD

`slice` hierarchiczna grupa jednostek

`scope` zewnętrznie utworzony proces

- `systemctl`, `journalctl`, `loginctl`, `machinectl`, `busctl`, `timedatectl`, `localectl`, `hostnamectl`
- `systemd-cgls`, `systemd-cgtop`, `systemd-nspawn`, `systemd-analyze`, `systemd-cat`, `systemd-detect-virt`, `systemd-delta`, `systemd-run`, `systemd-path`
- `systemd`, `systemd-notify`, `systemd-tty-ask-password-agent`, `systemd-ask-password`, `systemd-machine-id-setup`, `systemd-tmpfiles`, `systemd-inhibit`, `systemd-escape`, `systemd-stdio-bridge`

Sprawdzanie zależności *targetów*

- `systemctl list-*`
- `systemctl list-dependencies graphical.target`

Zmiana *targetu*

- `systemctl isolate ...`

Sprawdzanie konfiguracji i stanu usług

- `systemctl status|show|cat ...`

Zmiana konfiguracji i stanu usług

- `systemctl enable|disable|start|stop|reload|restart ...`

Konfiguracja SystemD

Stale działające demony `/sbin/init` (link do `/lib/systemd/systemd`):

- systemowy,
- użytkownika (opcja `--user`).

Kolejność wyszukiwania plików jednostek systemowych:

- `/etc/systemd/system/`
- `/run/systemd/system/`
- `/lib/systemd/system/`

Kolejność wyszukiwania plików jednostek użytkownika:

- `$XDG_CONFIG_HOME/systemd/user/`
- `$HOME/.config/systemd/user/`
- `/etc/systemd/user/`
- `/run/systemd/user/`
- `$XDG_DATA_HOME/systemd/user/`
- `$HOME/.local/share/systemd/user/`
- `/usr/lib/systemd/user/`

- Odpowiedniki *runleveli* z SysV Init.
- Tak jak w OpenRC mają swoje nazwy i jest ich wiele (kilkadziesiąt).
- Link symboliczny `/lib/systemd/system/default.target`
- Kompatybilność z SysV Init: `runlevel{0..6}.target` — linki do, odpowiednio, `poweroff.target`, `rescue.target`, `multi-user.target`, `graphical.target` i `reboot.target`.
- Zależności *targetu* *X*: linki symboliczne w katalogu `/etc/systemd/system/X.target.wants/`.
- Dodawanie usuwanie zależności (linków):
`systemctl [enable|disable] unit`
- Nazwa targetu jest podana w treści *unit*-u (`WantedBy`).
- Konfiguracja niestandardowych targetów: ręcznie za pomocą `ln -s`.

- `X.service` opisuje serwis (przeważnie demona). **Zastępuje** `/etc/rc.d/X` (BSD) i `/etc/init.d/X` (SysV).
- `X.socket` opisuje gniazdo (lokalne lub sieciowe). Powinien mu towarzyszyć `X.service`. Serwis jest uruchamiany miarę potrzeb, w razie połączenia klienta z gniazdem. **Zastępuje** `inetd(8)`, `xinetd(8)`.
- `X.timer` opisuje periodyczne uruchamianie usługi. Powinien mu towarzyszyć `X.service` lub inna jednostka uruchamiana periodycznie. **Zastępuje** `cron(8)`, `anacron(8)`.
- `X.device` — opis konfiguracji urządzenia. **Zastępuje** konfigurację `udev` w `/etc/udev/*`.
- `X.link` i `X.network` — konfiguracja interfejsów sieciowych i sieci. **Zastępuje** `interfaces(5)`.
- `X.netdev` — konfiguracja interfejsów wirtualnych. **Zastępuje** `brctl(8)`, `vconf(8)` i in.

- `X.path` — monitoruje ustaloną ścieżkę w systemie plików. Powinien mu towarzyszyć `X.service` lub inna jednostka uruchamiana w razie zmiany ścieżki. Wykorzystuje `inotify(7)`. **Zastępuje** `inotify_watch(1)`.
- `X.mount` — opis punktu montażowego. **Zastępuje** jeden wiersz `fstab(5)`.
- `X.swap` — konfiguracja partycji wymiany. **Zastępuje** wpis w `fstab(5)`.
- `X.automount` — wyzwalacz montowania w miarę potrzeb. Powinien mu towarzyszyć opis punktu montażowego `X.mount`. **Zastępuje** `inotify_wait(1)`.
- `X.scope`, `X.slice` — grupowanie procesów z wykorzystaniem `cgroups`. **Zastępują** `libcgroup(3)`.
- `X.snapshot` — zapis stanu systemu.

Przykład: fstrim

```
fstrim.timer
```

```
[Unit]
```

```
Description=Discard unused blocks once a week
```

```
Documentation=man:fstrim
```

```
[Timer]
```

```
OnCalendar=weekly
```

```
AccuracySec=1h
```

```
Persistent=true
```

```
[Install]
```

```
WantedBy=timers.target
```

Przykład: fstrim, cd.

```
fstrim.service
```

```
[Unit]
```

```
Description=Discard unused blocks
```

```
[Service]
```

```
Type=oneshot
```

```
ExecStart=/sbin/fstrim -av
```

Przykład: eth0

```
eth0.link
```

```
[Match]
```

```
MACAddress=00:36:e4:aa:33:76
```

```
[Link]
```

```
MACAddressPolicy=random
```

```
Name=eth0
```

```
99-default.link
```

```
[Link]
```

```
NamePolicy=kernel database onboard slot path mac
```

```
MACAddressPolicy=persistent
```

Przykład: /tmp w ramdysku

tmp.service

[Unit]

Description=Temporary Directory

Documentation=man:hier(7)

Documentation=<http://www.freedesktop.org/wiki/Software/...>

ConditionPathIsSymbolicLink=!/tmp

DefaultDependencies=no

Conflicts=umount.target

Before=local-fs.target umount.target

After=swap.target

[Mount]

What=tmpfs

Where=/tmp

Type=tmpfs

Options=mode=1777,strictatime,nosuid,nodev

[Install]

WantedBy=local-fs.target

Przykład: swap w skompresowanym ramdysku

zramswap.service

```
[Unit]
Description=Configure zram device and use it for swap
After=systemd-modules-load.service

[Service]
RemainAfterExit=yes
ExecStartPre=/bin/sh -c '/sbin/zramctl -f --size 3072M > /run/zramswap'
ExecStart=/bin/sh -c 'test -b $(cat /run/zramswap) || exit 1; \
    /sbin/mkswap $(cat /run/zramswap); \
    /sbin/swapon -o discard $(cat /run/zramswap)'
ExecStop=/bin/sh -c 'test -b $(cat /run/zramswap) || exit 1; \
    /sbin/swapon $(cat /run/zramswap)'
ExecStopPost=/bin/sh -c 'test -b $(cat /run/zramswap) || exit 1; \
    /sbin/zramctl -r $(cat /run/zramswap); cat /dev/null > /run/zramswap'

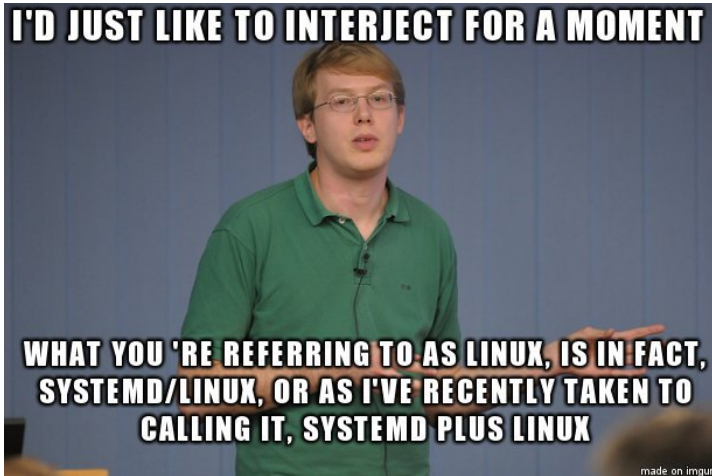
[Install]
WantedBy=sysinit.target
```

- Zbyt skomplikowany i rozbudowany.
- Zamienia Linuksa (filozofia klocków Lego) w monolit złożony z jednego wielkiego programu — nie pozostawia miejsca na wybór komponentów dystrybucji.
- Sprzeczny z filozofią Uniksa.
- Zamiast wolności wyboru fragmentów oprogramowania — dyktatura SystemD.
- Przerost funkcji kosztem modularności i jakości kodu.
- Po wprowadzeniu SystemD jako domyślnego systemu w Debianie powstał fork Devuan.
- W Debianie są spakietowane także stary SysV Init i OpenRC. Jednak nie ma wymogu, aby pakiety zawierały konfigurację dla tych systemów (dla SystemD też nie, ale on ma mechanizmy kompatybilności z SysV Init).

Lennart Poettering

- 15/10/1980, Guatemala
- obywatelstwo niemieckie
- Autor:
 - PulseAudio (2004)
 - Avahi (2005)
 - SystemD (2010)
- Krytykuje filozofię Uniksa.
- Krytykuje przenośność oprogramowania (*portability vs. innovation*), postuluje zerwanie kompatybilności Linuksa z POSIX i SUS.
- Krytykuje linuksowy sposób dystrybucji oprogramowania.

The classic Linux distribution scheme is frequently not what end users want, either. Many users are used to app markets like Android, Windows or iOS/Mac have. Markets are a platform that doesn't package, build or maintain software like distributions do, but simply allows users to quickly find and download the software they need, with the app vendor responsible for keeping the app updated, secured, and all that on the vendor's release cycle.



Dygresja: czego oprogramowanie nie powinno robić

Nieoczekiwane połączenia z Internetem

- systemd-timesyncd.service i serwery NTP. Trzeba wiedzieć, żeby wyłączyć:
`timedatectl set-ntp false`
- Amarok i CDDB
- vim i słownik ortograficzny języka polskiego
- *Aplikacje nieprzeznaczone do łączenia się z Internetem nie powinny tego robić!*

Nieoczekiwane modyfikacje plików

- ebook-viewer modyfikuje przeglądane pliki EPUB
- *Aplikacje nieprzeznaczone do modyfikowania plików nie powinny tego robić!*

Nie inwestuję w instrumenty finansowe, których nie rozumiem.

— Warren Buffet

Nie używam programów, których nie rozumiem.

— Ja

System V Init

- Główny plik konfiguracyjny: `/etc/inittab`.
- Koncepcja *runlevels*.
- Biblioteka skryptów uruchamiających w `/etc/init.d/`.
- Katalog `/etc/rcS.d/` oraz dla każdego runlevelu katalog `/etc/rci.d/`.
- Program `telinit(8)` do zmiany bieżącego runlevelu.

Katalog `/etc/init.d/`

- Zawiera skrypty konfigurujące usługi, po jednym dla każdej usługi.
- Każdy skrypt wymaga jednego parametru:
`start`, `stop`, `reload`, `force-reload`, `restart`, `status`.
- Skrypt można uruchamiać samodzielnie, np.:
`/etc/init.d/usługa stop`.
- Specjalne polecenie `service(8)`:
`service script command`
`service --status-all`

- Opisują stan konfiguracji systemu (które skrypty z `/etc/init.d` należy wykonać z parametrem `start` lub `stop`).
- Ponumerowane liczbami, zwykle 0–6.
- W Debianie było: 0 — *shutdown*, 1 — *single user mode*, 2 — *multiuser terminal*, 3, 4 — *user defined*, 5 — *multiuser graphical*, 6 — *reboot*.
- Dodatkowo: S — *single user mode*.
- Z każdym *runlevelem* *i* jest związany katalog `/etc/rci.d/`. Dodatkowo katalog `/etc/rcS.d/`.
- W każdym katalogu dowiązania symboliczne do skryptów z `/etc/init.d`.

- Nazwa dowiązania *Snnusługa* odpowiada uruchomieniu skryptu */etc/init.d/usługa* z parametrem *start*, a *Knnusługa* — stop.
- Liczby dwucyfrowe *nn* zapewniają odpowiednią kolejność.
- Skrypt */etc/init.d/rc* wywołany z parametrem *i* wywołuje skrypt z */etc/init.d* z parametrem *start* jeśli w nowym *runlevelu* jego dowiązanie zaczyna się na *S*, a w poprzednim *runlevelu* nie było *S* itd.
- Skrypty z */etc/rcS.d/* wykonywane przed (1–5) lub po (0, 6) skryptach */etc/rci.d/*.

Główny plik konfiguracyjny /etc/inittab

Zawiera m. in.:

- Numer domyślnego runlevelu, np. `id:5:initdefault:.`
- Konfigurację terminali, np. `1:2345:respawn:/sbin/agetty tty1`
- Uruchomienie skryptów runlevelu, np. `15:wait:/etc/init.d/rc 5`

Zarządzanie dowiązaniem w katalogach `/etc/rci.d/`

- Dawniej można było ręcznie użyć `ln -s`.
- Problem: dobranie właściwej kolejności skryptów.
- Rozwiązanie: *Dependency-based boot*.
- Nagłówki skryptów z `/etc/init.d`:

```
### BEGIN INIT INFO
# Provides:          boot_facility_1 [ boot_facility_2 ...]
# Required-Start:    boot_facility_1 [ boot_facility_2 ...]
# Required-Stop:     boot_facility_1 [ boot_facility_2 ...]
# Should-Start:      boot_facility_1 [ boot_facility_2 ...]
# Should-Stop:       boot_facility_1 [ boot_facility_2 ...]
# X-Start-Before:    boot_facility_1 [ boot_facility_2 ...]
# X-Stop-After:      boot_facility_1 [ boot_facility_2 ...]
# Default-Start:     run_level_1 [ run_level_2 ...]
# Default-Stop:      run_level_1 [ run_level_2 ...]
# X-Interactive:     true
# Short-Description: single_line_description
# Description:       multiline_description
### END INIT INFO
```

Linux Standard Base script dependency information

- Niskopoziomowe polecenie `insserv(8)` wyliczające właściwą kolejność skryptów na podstawie nagłówków.
- Wylicza pliki `/etc/init.d/.depend.{boot,start,stop}` w składni podobnej do `Makefile`.
- Konfiguracja: `/etc/insserv.conf` i `/etc/insserv.conf.d/*`.
- Konfiguracja katalogów `/etc/rci.d/` — polecenie `update-rc.d(8)`:
`update-rc.d [-n] [-f] usługa remove`
`update-rc.d [-n] usługa defaults`
`update-rc.d [-n] usługa disable|enable [S|2|3|4|5]`
`update-rc.d [-n] usługa start|stop`
- Ostatnia wersja odpowiada poleceniu `service`.

Wady

- Jest powolny.
- Bardzo trudno zrównoleglić (procesory mają obecnie wiele rdzeni!).
- *Runlevels* są zbyt ograniczone (jest ich mało, nie są hierarchiczne).
- *LSB dependency boot headers* są zbyt ograniczone.
- Poza *respawn* w *inittab* brak możliwości nadzorowania i wznowiania upadłych serwisów.
- Brak jednolitego logowania zdarzeń (*rsyslog* nie wystarcza).
- Brak jednolitego zarządzania zasobami (w tym wykorzystania *cgroups* i *namespaces*).

Lepsze rozwiązania

- SystemD — uniwersalny, rozbudowany.
- OpenRC – prosty, ale z ograniczeniami.

Runlevel powinien opisywać konfigurację docelową, a nie sposób jej osiągnięcia.

Najstarsze implementacje

- SysV Init *LSB script dependency information* — oryginalnie Suse Linux, 2000. Zaadaptowane przez większość dystrybucji, w tym Debiana.
- *Gentoo modular init scripts* (Baselayout), 2001. Niezależny projekt w 2007 pod nazwą OpenRC (główny program przepisany w C). Od 2011 z powrotem jako część projektu Gentoo.

Linux Standard Base script dependency headers

- Usługi: serwisy oraz usługi systemowe, np. `local_fs`, `syslog` (zob. `/etc/insserv.conf`).
- Usługi (*facilities*) są wierzchołkami grafu zależności.
- Zależności opisane za pomocą nagłówków w skryptach:
 - `Provides`
 - `Required-Start`, `Required-Stop` (hard dependency)
 - `Should-Start`, `Should-Stop` (soft dependency)
 - `Default-Start`, `Default-Stop` (runlevel reverse dependency)
 - `X-Start-Before`, `X-Stop-After` (reverse dependency)
 - `X-Interactive` (needs user interaction, don't parallelize)
 - `Short-Description`, `Description`
- `insserv` statycznie wyznacza porządek uruchamiania/zatrzymywania (*Makefile-like* `/etc/init.d/.depend.{boot,start,stop}`).
- `startpar(8)` uruchamiany przez `init(8)` uruchamia równolegle skrypty z `/etc/init.d/` zgodnie z porządkiem zapisanym w plikach `.depend.*`.

- Opis serwisu podzielony na konfigurację `/etc/conf.d/service` i skrypt inicjalizujący `/etc/init.d/service`.
- Oba pliki: POSIX sh, interpretowane przez `/sbin/openrc-run`.
- Biblioteka funkcji pomocniczych: `libeinfo`.
- Na wzór Debiana polecenie `start-stop-daemon(8)`.
- Skrypt serwisu zawiera funkcję `depend` opisującą zależności.
- Argumenty skryptu: `start`, `stop`, `status`.
- Program `/sbin/openrc` uruchamiany przez `init(8)`.
- `openrc(8)` ma za zadanie osiągnąć podany `runlevel`, po czym zapisuje stan systemu i kończy działanie.
- Skompilowana konfiguracja jest zapisana w *cache'u*. `openrc(8)` używa `mtime` do śledzenia zmian w plikach źródłowych.
- Stan systemu można sprawdzać (`rc-status(8)`) i modyfikować (`rc-service(8)`).
- Wywołanie `openrc(8)` bez parametru przywraca konfigurację bieżącego `runlevela`.

Zależności w funkcji `depend()` skryptu serwisu

- `need` (hard dependency)
- `want` (medium dependency)
- `use` (soft dependency)
- `before`, `after` (soft reverse dependency)
- `provide`
- `keyword` (platform-specific override)

Runlevels

- Runlevele mają nazwy, a nie numery. Może ich być dowolnie wiele. Są plikami w katalogu `/etc/runlevels/`.
- Runlevele zależą od serwisów.
- Zależności runleveli można zmieniać za pomocą `rc-update(8)`.
- *Stacked runlevels* — zależności runleveli od innych runleveli.
- Współbieżne uruchamianie jako opcja.

- `openrc(8)` kończy pracę po osiągnięciu podanego *runlevela*.
- Monitorowanie serwisów możliwe poprzez zastąpienie polecenia `start-stop-daemon(8)` wywołaniem demona nadzorującego: `runit` (reimplementacja `daemontools`) albo `s6`.
- Własna implementacja: `supervise-daemon(8)`. Uruchamiany program nie może się *fork-ować*.
- Można korzystać z *cgroups*, np.
`rc-service service cgroup-cleanup`

- OpenRC zachowuje większość terminologii i zwyczajów SysV Init.
- Funkcja `depend` skryptu serwisu odpowiada *dependency boot headers*.
- Wygodniejszy opis runleveli, możliwość ich hierarchizacji.
- Dynamiczne wyznaczanie grafu zależności (w SysV Init statyczne za pomocą `insserv(8)`).
- Możliwość korzystania z *cgroups* w celu np. zatrzymywania grup zależnych serwisów.