Kurs administrowania systemem Linux

Zajęcia nr 7: Procesy

Instytut Informatyki Uniwersytetu Wrocławskiego

12 kwietnia 2022

Wielozadaniowy system operacyjny

System operacyjny

- Zapewnia abstrakcję i wirtualizację *hardware'u*:
  - procesora i pamięci: system wielozadaniowy z podziałem czasu;
  - pamięci masowej: system plików.
- Wirtualizacja wymaga wsparcia sprzętowego:
  - wirtualizacja procesora: tryb nadzorcy, wywołania systemowe;
  - wirtualizacja pamięci: ochrona i zarządzanie pamięcią (MMU, *Memory Management Unit*).

Procesy

- Programy działające w zwirtualizowanym środowisku.
- *Przełączanie kontekstów* daje wrażenie wyłącznego i nieprzerwanego dostępu do procesora.
- *Adresy wirtualne* dają wrażenie dostępu do ciągłej przestrzeni adresowej wielkiego rozmiaru (np. 256 TiB).
- *Wywołania systemowe* (*syscall*) dają abstrakcję przerwań.

Procesy i wątki

- Każdy proces ma własny wirtualny procesor i własną wirtualną pamięć.
- Proces może się składać z jednego lub wielu *wątków*.
- Wątki mają własne wirtualne procesory (w tym własne stosy wywołań, liczniki rozkazów itp.), ale w ramach jednego procesu współdzielą jego pamięć wirtualną (w tym zmienne statyczne, stertę itd.).
- Kod jądra jest wykonywany w trybie nadzorcy, zwykłe procesy — w trybie użytkownika.
- Jądro wykonuje wiele wątków jednocześnie. Wszystkie wątki jądra współdzielą pamięć (jądro monolityczne).
- Procesy mogą uruchamiać nowe procesy (`fork(2)`) i wątki (`clone(2)`).
- Każdy proces ma dokładnie jednego ojca (jest jedno *drzewo procesów*).
- Przodkiem każdego procesu w przestrzeni użytkownika jest `init(1)`. Ojcem każdego wątka jądra jest `kthreadd`.

Numery procesów

- Każdy wykonywany kod podlegający podziałowi czasu ma unikatowy numer PID (*Process ID*):
  - każdy wątek jądra,
  - każdy proces,
  - każdy dodatkowy wątek procesu.
- PID jest zwykle typu `int` (liczba całkowita ze znakiem).
- $0 \leq \text{PID} \leq \texttt{PID\_MAX}$, domyślnie $\texttt{PID\_MAX} = 32767$.
- Dawniej konfiguracja kompilacji jądra, w jądrze 2.6 i później sysctl `kernel.pid_max` równy $\texttt{PID\_MAX} + 1$. W IA-32 co najwyżej 32768, w x86-64: 4 Mi (2^{22}).
- Numery przydzielane są po kolei, a po osiągnięciu `PID_MAX` wyszukiwanie wolnych numerów zaczyna się od 300.
- Umownie PID 0 — `idle` (wykonywany, gdy żaden proces bądź wątek nie jest wykonywany). Formalnie jest ojcem `init(1)` (PID 1) oraz `kthreadd` (PID 2).
- PID ojca procesu nazywa się PPID (*Parent PID*).
- W Bashu zmienne środowiskowe `$$` i `$PPID`.

Cykl życia procesów

- `init(1)` (PID 1) żyje przez cały czas pracy systemu.
- Każdy inny proces jest uruchamiany za pomocą `fork(2)`.
- Ojcem procesu jest ten, kto wywołał `fork(2)`.
- PCB (*Process Control Block*) — struktura w jądrze opisująca proces.
- PCB pozostaje po zakończonym procesie i zawiera m. in. *kod powrotu*.
- Ojciec ma obowiązek „pochować” zmarłego syna (`wait(2)`, `waitpid(2)`, `waitid(2)`) lub jawnie zignorować jego śmierć (np. `SIG_IGN` dla `SIGCHLD`, zob. `sigaction(2)`), w przeciwnym razie zmarły syn staje się *zombie* i tkwi w tablicy procesów.
- *Reparenting*: jeśli ojciec umrze wcześniej niż syn, to syn jest automatycznie adoptowany przez `init(1)`.
- `init(1)` periodycznie dokonuje pochówku wszystkich zmarłych synów.
- Niektóre procesy przed śmiercią (nie dotyczy nagłej śmierci poprzez `SIGKILL`) wysyłają np. sygnał `SIGTERM` nie dopuszczając, by dzieci przeżyły ojca.

Uzyskiwanie informacji o procesach

`ps(1)`

- Wiele opcji w trzech wersjach: BSD, standard i GNU.
- Wypisanie wszystkich procesów (`-e`) w kolumnach `UID PID PPID C STIME TTY TIME CMD` (`-f`) bez skracania wierszy (`-w -w`):
  `ps -efww`
- Wypisanie wszystkich procesów z pominięciem wątków jądra:
  `ps -fN --ppid 2 --pid 2`

`pstree(1)`

- Wypisuje ładnie sformatowane drzewo procesów.
- Wypróbujcie `pstree -lpgUSuna | less`

`top(1)`

- Dynamicznie (domyślnie co 3 sekundy) wyświetla tabelę procesów i statystyki.
- Pełnoekranowy z kolorami.

Użytkownicy i grupy

Każdy proces ma przypisanych czterech użytkowników i grupy:

- *real* — kto uruchomił proces,
- *effective* — czyje prawa dostępu ma proces,
- *saved* — effective UID/GID z chwili uruchomienia procesu,
- *filesystem* — czyje prawa dostępu do plików ma proces.

Można zmieniać UID/GID:

- `{s,g}et{,e,re,res,fs}{u,g}id(2)` zmieniają/ujawniają real/effective/real+effective/real+effective+saved/filesystem UID/GID procesu.
- `su(1)`, `sg(1)` uruchamiają proces z podanym real UID/GID.
- `newgrp(1)` zmienia real GID powłoki.
- `id(1)` ujawnia real i effective UID użytkownika, `whoami(1)` — tylko effective.

Sygnały

- Mechanizm asynchronicznego przesyłania komunikatów pomiędzy procesami lub procesami i jądrem.
- Sygnały mają numery (liczby typu `int`) i przyporządkowane im nazwy (`signal(7)`, `bits/signum.h`).
- Każdy proces może rejestrować procedury wywoływane w razie otrzymania sygnału (`sigaction(2)`).
- Każdy proces może wysłać sygnał do innego procesu (`kill(2)`, obwoluta: `kill(1)`).
- Jądro wysyła sygnały do procesu (np. `SIGHUP`, `SIGINT`, `SIGQUIT`, `SIGILL`, `SIGFPE`, `SIGSEGV`, `SIGPIPE` itd.).
- Jądro przechwytuje sygnały kierowane do procesu: `SIGKILL`, `SIGSTOP`, `SIGCONT`.
- Jądro przesyła do `init(1)` tylko te sygnały, dla których `init` zarejestrował *handlery* (zabezpieczenie przed przypadkowym zabiciem `init`).

Sygnały we współczesnym Linuksie (x86)

1 SIGHUP	12 SIGUSR2	23 SIGURG
2 SIGINT ^C	13 SIGPIPE	24 SIGXCPU
3 SIGQUIT ^\	14 SIGALRM	25 SIGXFSZ
4 SIGILL	15 SIGTERM	26 SIGVTALRM
5 SIGTRAP	16 SIGSTKFLT	27 SIGPROF
6 SIGABRT	17 SIGCHLD	28 SIGWINCH
7 SIGBUS	18 SIGCONT	29 SIGIO
8 SIGFPE	19 SIGSTOP	30 SIGPWR
9 SIGKILL	20 SIGTSTP ^Z	31 SIGSYS
10 SIGUSR1	21 SIGTTIN	32-63 real-time signals
11 SIGSEGV	22 SIGTTOU	

Szczegóły: zob. signal(7).

Wysyłanie sygnałów z powłoki

- Program `kill(1)` oraz *shell builtin* `kill` różniące się opcjami.
- `kill -l` wypisuje dostępne sygnały.
- `kill [-`*SIG*`]` *PID* wysyła sygnał *SIG* (domyślnie `TERM`) do procesu *PID* (*PID* $= 0$ oznacza wszystkie procesy z grupy procesu wysyłającego (włączając ten proces), *PID* $= -1$ oznacza wszystkie procesy z wyjątkiem siebie i `init`, *PID* < -1 oznacza wysłanie do grupy $-$*PID*).
- Program `killall(1)` — *kill by name*.
- Programy `pgrep(1)` i `pkill(1)` — fuzja `grep(1)` i `killall(1)`.

Sesje

- Zbiory procesów przypisane do jednego terminala.
- Sesje mogą też dziedziczyć terminal bądź nie być przypisane do terminala.
- *Syscall* `setsid(2)` i program `setsid(1)` tworzą nową sesję.
- Podczas *logowania* (zob. `login(1)`) jest tworzona nowa *sesja*, związana z terminalem sterującym (*controlling terminal*).
- Terminalem może być urządzenie transmisji szeregowej RS-232 (`ttyS{0..}`) albo USB (`ttyUSB{0..}`), terminal wirtualny (`tty{0..}` tworzony przez jądro w trybie tekstowym karty graficznej lub poprzez KMS) bądź pseudoterminal (`pts/{0..}`) tworzony np. przez aplikację terminala w systemie okienkowym bądź poprzez zdalne logowanie (np. `sshd(8)`).
- Domyślnie wszystkie procesy w sesji mają `stdin`, `stdout` i `stderr` związane z terminalem sterującym.

Sesje, cd.

- Sesja ma lidera, którym jest zwykle powłoka systemowa podłączona do tego terminala.
- Jeśli terminal sterujący zostanie rozłączony (*hangup*) lub lider sesji umrze, wszystkie procesy w sesji otrzymują sygnał `SIGHUP`.
- Program `nohup(1)` pozwala uruchomić proces w osobnej sesji nie połączonej z terminalem.
- GNU `screen(1)` i BSD `tmux(1)` pozwalają na uruchomienie sesji, które mogą być wielokrotnie podłączane i odłączane od terminali.

Demony

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 poprzez demony.

Demony

- `PPID=1`
- `CWD=/`
- Działają w osobnej sesji (wykonują `setsid`).
- `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]
```

Demonizacja

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()`, `sesid()`, `fork()`, po czym oryginalny proces i pierwszy potomny wykonują `exit()`).

Demonizacja, cd.

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)`.

Grupy procesów

- 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 (*backgroud*).
- 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*).

Grupy procesów, cd.

- 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)` wznawia zadanie zatrzymane (poprzez wysłanie `SIGSTOP` do grupy procesów) w tle (poprzez wysłanie `SIGCONT` do grupy procesów).

Zarządzanie procesami — dalsze informacje

Dostępne zasoby dla procesu

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

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

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

Cgroups* i *namespaces

- Rozbudowane możliwości grupowania procesów i zarządzania nimi.

/sbin/init

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 wątka 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.

Korzystanie z sieci: potrzebne dane konfiguracyjne

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 kompunikacji 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` 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 defult 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 narzedzia: `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` (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/iterfaces`.
- 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 konfigurującymi 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)` — *pluginy* 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 á 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 (Reasearch 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 $\geq$ 5.0 i NetBSD $\geq$ 1.5): zestaw skryptów w katalogu `/etc/rc.d/` uruchamianych przez `/etc/rc` zgodnie z konfiguracją w `rc.conf`.
- W Uniksach Systemu V i niektórych dystrybucjach Linuksa bardziej rozbudowany SysV Init.
- W nowych dystrybucjach Linuksa — SystemD.
- Wiele innych rozwiązań, np. OpenRC (Gentoo, także Alpine).

Potrzeba modularyzacji

- 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 *progname* to `/etc/`*progname*`.conf` lub `/etc/`*progname*.
- Dodatkowo katalog `/etc/`*progname*`.d/`.
- W głównym pliku konfiguracyjnym instrukcja `include /etc/`*progname*`.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 *progname*: `/etc/`*progname*`.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`.

Przykłady

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 `crond` wykonuje
  `run-parts /etc/cron.{hourly,daily,weekly,monthly}/`
- Dowolny program, który wymaga okresowego wykonywania pewnych czynności, może umieścić w tych katalogach swoje skrypty.