


# Kurs administrowania systemem Linux 2024




Lista zadań na pracownię nr 8

Na zajęcia 22 i 24 kwietnia 2024

**Zadanie 1 (1 pkt).** W tym zadaniu będziesz potrzebował:

- dowolnego środowiska okienkowego, które pozwala na zamykanie okna (np. poprzez kliknięcie w ikonę  lub naciśnięcie klawisza `<Alt>-<F4>`),
- aplikacji okienkowej terminala, która podczas zamykania okna wysyła sygnał `SIGHUP` do powłoki uruchomionej w tym oknie (ja użyłem `xterm`-a, ale może być też np. `gnome-terminal`, `konsole` itp.; jeśli użyjesz `gnome-terminal`-a, to pamiętaj, by zamknąć wszystkie jego okna i karty — inaczej osierocone procesy będzie przejmował `gnome-terminal-server`),
- dowolnej aplikacji okienkowej, która kończy działanie po odebraniu sygnałów `SIGHUP` oraz `SIGINT` (ja użyłem `okular`-a, może też być np. `evince`).

Powłoką uruchamianą w aplikacji terminala powinien być `bash`. Dla zwięzłości aplikację terminala będę poniżej nazywał `xterm`, a aplikację okienkową — `okular` (czyli tak, jak w moich eksperymentach). Przeprowadź opisane niżej eksperymenty:

1. Uruchom `xterm`. W uruchomionej w nim powłoce wpisz polecenie `okular&`, a następnie `exit` lub `<Ctrl>-D`. Zauważ, że okno `xterm`-a zniknęło, ale okno `okular`-a nadal działa. Czyim jest dzieckiem? Zamknij je, np. klikając w przycisk .
2. Uruchom `xterm`. W uruchomionej powłoce wpisz polecenie `okular&`. Następnie zamknij okno `xterm`-a (np. klikając w jego przycisk ). Zauważ, że zniknęły oba okna.
3. Uruchom `xterm`. W uruchomionej powłoce wpisz polecenie `okular&`. Za pomocą polecenia `jobs` sprawdź, że `okular` jest zadaniem drugoplanowym powłoki. Wyдай polecenie `disown %1` i powtórnij `jobs`. Następnie zamknij okno `xterm`-a (np. klikając w jego przycisk ). Zauważ, że okno `xterm`-a zniknęło, ale okno `okular`-a nadal działa. Czyim jest dzieckiem?
4. Powtórz poprzedni eksperyment dodając do polecenia `disown` opcję `-h`.
5. Uruchom `xterm`. W uruchomionej powłoce wpisz polecenie `okular&`, potem `shopt -s huponexit`, a na koniec `exit`. Zauważ, że okno `xterm`-a zniknęło, ale okno `okular`-a nadal działa.
6. Powtórz poprzedni eksperyment, tym razem dodając do polecenia `xterm` opcję `-l`. Zauważ, że tym razem okno `okular`-a zniknie. Jeśli wybrana przez Ciebie aplikacja terminala nie umożliwia uruchomienia powłoki w trybie *login shell*, uruchom w niej potomną instancję `bash`-a z opcją `-l`.
7. Uruchom `xterm`. Sprawdź PID powłoki uruchomionej w jego oknie. Wpisz w niej polecenie `okular&`. Następnie poleceniem `kill -HUP` wyślij do niej sygnał `SIGHUP`.

Aby zrozumieć, co tu się stało, zbierz informacje o sygnałach i zarządzaniu zadaniami w `bash`-u. Dobrymi źródłami są:

- strony angielskiej Wikipedii: „`SIGHUP`” i „`Signal (IPC)`”,
- strona podręcznika `bash(1)`, w szczególności podrozdziały `SIGNALS` i `JOB CONTROL` oraz opis poleceń wbudowanych `disown`, `jobs`, `fg` i `bg`.

Przygotuj krótkie omówienie poruszanych wyżej zagadnień (w tym sygnałów `SIGHUP` i `SIGINT` oraz polecenia `disown`). Na koniec dowiedz się, co robi operator `&!` w `zsh`.

**Zadanie 2 (1 pkt).** Przygotuj krótkie omówienie programu `screen(1)`. Szczególnie uwzględnij odłączanie/dołączanie otwartych sesji (loguj się do systemu np. za pomocą polecenia `ssh localhost`), przełączanie pomiędzy kartami, zaznaczanie tekstu, kopiowanie do schowka i przenoszenie tekstu pomiędzy kartami.

**Zadanie 3 (1 pkt).** Wykonaj poprzednie zadanie dla programu `tmux(1)`.

**Zadanie 4 (1 pkt).** Napisz skrypt, który raz na minutę wypisuje komunikaty do logów systemowych (zob. `logger(1)`) oraz na standardowe wyjście. Uruchom aplikację terminala, jak w poprzednim zadaniu, a w niej Twój skrypt w tle, a następnie zakończ pracę aplikacji, np. klikając przycisk  $\otimes$ . Zobacz, że skrypt zostanie zakończony poprzez wysłanie przez powłokę sygnału `SIGHUP` w chwili zamknięcia sesji. Uruchom go ponownie za pomocą programu `nohup(1)` i zobacz, że teraz będzie działał nadal po zamknięciu sesji. Gdzie są zapisywane komunikaty, które skrypt wypisuje na standardowe wyjście? Jak teraz zakończyć jego działanie? Zapoznaj się z poleceniem `trap` powłoki. Zmień skrypt tak, żeby teraz nie korzystał ze standardowego strumienia wyjściowego, ignorował sygnał `SIGHUP`, a po otrzymaniu sygnału `SIGUSR1` wypisywał do logów komunikat o otrzymaniu tego sygnału i kończył działanie.

**Zadanie 5 (1 pkt).** Napisz skrypt, którego zadaniem jest bezcelowe zużywanie mocy obliczeniowej procesora, np. poprzez wykonywanie w pętli jakichś obliczeń. Skrypt powinien uruchamiać podaną podczas wywołania liczbę podprocesów tak, aby dało się obciążyć wszystkie rdzenie procesora. Uruchamiaj go z różnymi wartościami `niceness` (użyj polecenia `nice(1)`) lub na bieżąco zmieniaj `niceness` działającego procesu (użyj polecenia `renice(1)`) i zobacz, jaki ma to wpływ na działanie systemu.

**Zadanie 6 (1 pkt).** Napisz w C prosty program `mystat`, który periodycznie (domyślnie raz na sekundę) odczytuje z pseudopliku `/proc/stat` obciążenie procesora (zob. `proc(5)`) i co jakiś czas (domyślnie co 1 minutę) zapisuje uśrednione/maksymalne/minimalne wartości obciążenia do ustalonego pliku (domyślnie `/var/log/mystat.log`). Podane domyślne wartości powinny być możliwe do zmiany za pomocą opcji `-p` lub `--period`, `-i` lub `--interval` oraz `-f` lub `--logfile` (użyj `getopt_long(3)`).

**Zadanie 7 (1 pkt).** Użyj programu MRTG (<https://oss.oetiker.ch/mrtg/>) aby na podstawie zebranych przez program `mystat` logów generować czytelne wykresy obciążenia systemu.

**Zadanie 8 (1 pkt).** Na podstawie programu `mystat` napisz jego zdemonizowaną wersję `mystatd`, zob. `daemon(7)`. Poza funkcjonalnościami odziedziczonymi z programu `mystat` program powinien także przechwytywać sygnał `SIGHUP` i w razie jego otrzymania zamykać i powtórnie otwierać swój plik z logami (co ułatwi rotowanie logów).

**Zadanie 9 (1 pkt).** Przygotuj jednostkę SystemD `mystat.service` uruchamiającą program `mystat` w systemie zarządzanym za pomocą SystemD. Skonfiguruj system tak, by serwis `mystat` uruchamiał się wraz z `sysinit.target`. Przygotuj także jednostkę `mystat-graph.timer`, która raz na dobę uruchamia skrypt, który tworzy w katalogu `/var/lib/mystat` plik PNG zawierający wykres obciążenia maszyny w ciągu ostatnich 24 godzin. Do wygenerowania wykresu użyj programu MRTG lub np. Gnuplot.

**Zadanie 10 (1 pkt).** Dopisz do `/etc/logrotate.d/` obsługę logów demona `mystat`. System powinien rotować logi raz na dobę i przechowywać logi z ostatniego tygodnia. Pamiętaj o skonfigurowaniu wysyłania sygnału do demona w celu zamknięcia i ponownego otwarcia pliku z logami (zob. `invoke-rc.d(8)`).

**Zadanie 11 (1 pkt).** Zmodularyzuj swój plik `~/.bashrc`. Załóż w tym celu katalog `~/.bashrc.d/`. Różne fragmenty konfiguracji startowej (konfiguracja zmiennej `PS1`, aliasy, itd.) umieść w osobnych plikach o odpowiednich nazwach i wczytuj je z głównego pliku.

**Zadanie 12 (1 pkt).** Zapoznaj się z poleceniem `run-parts(8)`. Przygotuj krótkie omówienie jego działania. Dlaczego nie mogliśmy z niego skorzystać w poprzednim zadaniu? Użyj go, aby sprawdzić, jakie skrypty są wykonywane przez demona `crond` raz na dobę.

**Zadanie 13 (1 pkt).** *Zrób sobie zombie.* Napisz program, który utworzy proces potomny, a następnie nie obsłuży jego zakończenia. Proces potomny powinien się zakończyć. Zobacz, że pozostanie na liście procesów jako *zombie* do czasu, aż zakończy się oryginalny program. Zakończenie procesu potomnego zostanie wówczas obsłużone przez program `init(1)`.

**Zadanie 14 (1 pkt).** Zapoznaj się z dokumentacją polecenia `top(1)` i przygotuj się do zaprezentowania tego programu podczas zajęć. Przedstaw podstawowe skróty klawiszowe. Pokaż, jak można uzyskiwać różne informacje o zbiorze działających procesów przełączając sposób wyświetlania, sortowania itd. Dostosuj kolorystykę wyświetlanych informacji zgodnie z własnymi upodobaniami. Przygotuj odpowiedni plik `.toprc`.

**Zadanie 15 (1 pkt).** Zapoznaj się z programami `ps(1)` i `pstree(1)`. Przygotuj krótką demonstrację ich użycia.