Laboratorium 4 - Procesy, demony i sygnały

1. Teoria

Podczas zajęć omówione zostaną komendy związane z przeglądaniem stanu systemu.

Proces w systemie Linux jest to uruchomiona instancja programu binarnego. Taka uruchomiona instancja zajmuje pewne zasoby w pamięci, system operacyjny zarządza wskaźnikiem instrukcji i wykonaniem kolejnych kroków instrukcji procesu. Procesy posiadają specjalny unikatowy numer w systemie - PID (process ID). Numer ten przyznawany jest z reużytkowalnej puli, ale w danym momencie nie ma dwóch procesów o tym samym PID. Każdy proces uruchomiony jest z poziomu konkretnego użytkownika i dysponuje prawami takimi samymi jak ten użytkownik. Najważniejszym procesem w systemie jest `init` który ma zawsze PID `1`, a jego właścicielem jest użytkownik `root`.

1.1. Procesy a demony

Demony, inaczej usługi, są pewnym rodzajem programów, które są uruchamiane zazwyczaj przy starcie systemu działając najczęściej cały czas w tle. Procesy tych usług posiadają w Linux specjalne określenie: demon (ang. daemon). W praktyce wiele programów umożliwia uruchomienie ich w trybie demona, a typowymi usługami działającymi w ten sposób są: serwery WWW (np. Apache, nginx), DNS (np. bind), mail (pocztowe), X server, baza danych (np. MySQL, PostgreSQL), demon usług sieciowych i DHCP, itp... Procesy demonów będące usługami w każdej chwili mogą przyjąć od innego programu polecenie (żądanie) w celu ich obsłużenia. Demonem jest np. serwer HTTP, który cały czas jest uruchomiony w tle, a w momencie gdy przychodzi żądanie od przeglądarki wysyła odpowiedni plik lub wykonuje jakąś akcję. Demon może również obsługiwać programy działające na tym samym komputerze.

Kolejną cechą charakterystyczną demonów, jest to że nowoczesne dystrybucje Linux najczęściej posiadają wbudowane podsystemy do zarządzania uruchamianiem tych procesów. Np. dbają one o to aby przy każdym starcie systemu procesy demonów były automatycznie uruchomione. Często również do obsługi procesów demonów używa się różnego rodzaju aplikacji monitorujących stan ich procesów po to aby np. gdy demon zakończy się z błędem, był on znowu uruchomiony i dzięki temu było dalej możliwe łączenie się z daną usługą. Do zatrzymania działania demona służą specjalne skrypty, które w popularnych dystrybucjach

(Debian, Ubuntu) znajdują się zazwyczaj w katalogu `/etc/init.d/`. W Ubuntu powinno się używać polecenia `service`, które je wywołuje.

Skrypty te mogą być uruchamiane z następującymi opcjami:

- start - uruchamia demona
- stop - zatrzymuje demona
- restart - zatrzymuje, i ponownie uruchamia demona; przydatne np. w przypadku gdy zmieniona została konfiguracja danego demona

Niektóre mogą przyjmować również inne opcje, jednak te trzy są standardowe.

1.2. Plik `/proc/cpuinfo`

Zawiera informacje na temat procesora.

1.3. Plik `/proc/meminfo`

Zawiera informacje na temat pamięci RAM w systemie.

1.4. `ps`

Wypisuje listę procesów w systemie.

Argument	Opis
brak	wypisuje procesy związane z terminalem
`-l`	dokładniejsze informacje
`-A`	wypisuje wszystkie procesy w systemie
`-u` użytkownik	wypisuje procesy danego użytkownika
`-f`	wypisuje również argumenty poleceń

1.5. `top`

Wyświetla procesy w trybie interaktywnym.

Sortowanie odbywa się domyślnie po kolumnie %CPU. Zmiana sortowania odbywa się przez naciśnięcie `>` i `<`.

Argument	Opis
`-d`	Określa opóźnienie między odświeżeniami ekranu. Można to zmieniać komendą interakcyjną `s`.
`-p`	Monitoruje jedynie procesy o danym PID.
`-S`	Określa tryb kumulacyjny, gdzie każdy proces jest wypisywany z czasem CPU, który spożytkowanym przez niego oraz jego martwe procesy potomne.
`-i`	Ignoruje wszelkie procesy zombie i procesy bezczynne.

Argument	Opis
`-c`	wyświetla linię poleceń zamiast samej nazwy polecenia.

1.6. Sygnały

Procesy komunikują się z jądrem systemu, a także między sobą aby koordynować swoją działalność. Linux wspiera kilka mechanizmów komunikacji zwanych IPC (Inter-Process Communication mechanisms). Jednym z nich są sygnały, zwane inaczej przerwaniami programowymi.

Sygnały mogą być generowane bezpośrednio przez użytkownika (funkcja `kill()`), może wysyłać je jądro oraz procesy między sobą (funkcja systemowa `kill()`) . Dodatkowo pewne znaki z terminala powodują wygenerowanie sygnałów. Na przykład na każdym terminalu istnieje tak zwany znak przerwania (ang. interrupt character) i znak zakończenia (ang. quit character). Znak przerwania (zazwyczaj `Ctrl+C` lub `Delete`) służy do zakończenia bieżącego procesu (wygenerowanie SIGINT). Wygenerowanie znaku zakończenia (zazwyczaj `Ctrl-\`) powoduje wysłanie sygnału SIGQUIT powodującego zakończenie wykonywania bieżącego procesu z zapisaniem obrazu pamięci.
Istnieją oczywiście pewne ograniczenia - proces może wysyłać je tylko do procesów mających tego samego właściciela oraz z tej samej grupy (te same UID i GID). Bez ograniczeń może to czynić jedynie jądro i administrator. Jedynym procesem, który nie odbiera sygnałów jest `init` (PID równy 1).

Sygnały są mechanizmem asynchronicznym - proces nie wie z góry kiedy sygnał może nadejść i głównym ich zadaniem jest informowanie procesu o zaistnieniu w systemie wyjątkowej sytuacji (np. spadek napięcia w sieci). Ponadto są wykorzystywane przez shelle do kontroli pracy swoich procesów potomnych.

Każdy z sygnałów posiada swoje znaczenie (określające w jakiej sytuacji powinien być wysłany) jak również związana jest z nim pewna domyślna akcja jaką system wykonuje przy jego obsłudze. Oto opis znaczenia poszczególnych sygnałów według numeracji w systemie Linux.

Nr	Nazwa	Opis
2	`SIGINT`	`Ctrl+C`, przerwij proces
3	`SIGQUIT`	zamknij proces i zapisz stan pamięci procesu
9	`SIGKILL`	zakończ proces natychmiast (nie może być przechwycony ani zignorowany)
15	`SIGTERM`	podobny do SIGINT
23	`SIGSTOP`	zatrzymuje proces do ponownego wznowienia (nie może być przechwycony ani zignorowany)

Nr	Nazwa	Opis
20	`SIGTSTP`	`Ctrl+Z`, w porównaniu do SIGSTOP może być obsłużony

1.7. `kill`

Polecenie służy do wysyłania sygnałów do procesów.

Użycie: `kill -SYGNAL PID`, gdzie `SYGNAL` to numer albo nazwa sygnału.

1.8. `killall`

Polecenie służy do wysyłania sygnałów do wielu procesów o tej samej nazwie.

2. Praktyka

2.1. Zadanie

`/proc/cpuinfo`

Sprawdź jaki procesor znajduje się w komputerze w laboratorium oraz na serwerze wierzba. Zauważ, że dodatkowe rdzenie procesorów są reprezentowane jako kolejne procesory na liście.

2.2. Zadanie

`/proc/meminfo`

Sprawdź ile pamięci RAM dysponuje komputer w laboratorium i serwer wierzba. Wartości podaj w MB oraz w GB dokonując odpowiednich przeliczeń.

2.3. Zadanie

`ps`

Sprawdź jakie procesy zostały uruchomione w bieżącej sesji shella.

2.4. Zadanie

Sprawdź jakie procesy zostały uruchomione w całym systemie.

2.5. Zadanie

Sprawdź jakie procesy zostały uruchomione przez twojego użytkownika.

2.6. Zadanie

`top`

Uruchom polecenie `top` i posortuj listę procesów po ilości czasu procesora (TIME), po obciążeniu procesora (%CPU), po zajętej pamięci (%MEM). Znajdź proces, który najbardziej obciąża procesor oraz proces który najbardziej obciąża pamięć. W tym celu możesz uruchomić dowolne programy na komputerze w laboratorium.

2.7. Zadanie

Utwórz prosty skrypt, którym będziesz testować działanie programów zarządzających procesami. W tym celu otwórz ulubiony edytor tekstu i przepisz poniższy kod:

```
#!/bin/bash
echo "Zasypiam na $1 sekund."
sleep $1
echo "Pobudka. Koniec."
```

Skrypt zapisz pod nazwą `spioch.sh`. Nadaj uprawnienia temu plikowi w taki sposób aby można było go uruchomić jako zalogowany użytkownik w systemie (dodaj uprawnienia executable).

2.8. Zadanie

Będąc w tym samym katalogu co plik `spioch.sh` spróbuj uruchomić skrypt w następujący sposób: `./spioch.sh 3`
Skrypt powinien zasnąć na 3 sekundy, a następnie powinien pojawić się napis `Pobudka. Koniec.`

2.9. Zadanie

Uruchom skrypt na wiele sekund (np. 300). Postaraj się zakończyć go przed czasem wciskając kombinację `Ctrl+C`. Jaki sygnał został wysłany do procesu? Czy napis `Pobudka. Koniec.` wyświetlił się na ekranie?

2.10. Zadanie

Uruchom ponownie skrypt na wiele sekund. Otwórz nową kartę terminala. Postaraj się odnaleźć go za pomocą komendy `ps` oraz `top`. Odnajdź PID tego procesu.

2.11. Zadanie

`kill`

Znając numer PID procesu z uruchomionym skryptem śpiocha, spróbuj w osobnej karcie terminala wysłać do niego sygnały: SIGINT, SIGTERM, SIGQUIT, SIGKILL. Za każdym razem sprawdź co się stało z procesem śpiocha. Jeśli trzeba uruchom go ponownie.

2.12. Zadanie

Przy użyciu maszyny wirtualnej (VirtualBox) spróbuj uruchomić proces jako jeden użytkownik, a następnie zabić go jako inny użytkownik. Czy to możliwe? Próbę ponów jako użytkownik `root`.