Systemy Operacyjne I: Procesy

Andrzej Stroiński

Politechnika Poznańska

4 kwietnia 2013

Materiały

- Prezentacja oraz inne materiały zostały przygotowane na podstawie:
  - Użytkowanie systemu operacyjnego UNIX - *dr D.Wawrzyniak*
  - Systemy operacyjne - skrypt - *dr C.Sobaniec*
  - Strona przedmiotu
  - Strona *Dr A.Kobusińskiej*
  - Strona *K.Sieka*
  - pomoc systemowa `man`
  - inne...

Procesy

Definicja: Proces [M. Bach WNT 95, T. Borzyszkowski]

Proces jest wykonaniem programu i składa się ze zbiorowości bajtów, które CPU interpretuje jako instrukcje maszynowe (tzw. tekst), dane i stos. Jądro steruje wykonaniem procesu. Proces wykonuje się przechodąc przez ściśle ustalony ciąg instrukcji (stanowiący całość) i nigdy nie wykonuje skoków do instrukcji innego procesu tzn. czyta i zapisuje swoje dane oraz stos, lecz nie może czytać ani zapisywać danych i stosu innych procesów.
Procesy komunikują się z innymi procesami i z resztą świata za pomocą funkcji systemowych.
Procesy rozumiane jako wykonania kodu posiadające własności przedstawione powyżej były podstawową jednostką pracy pierwszych systemów Unixowych. Tylko procesy miały możliwość działania na procesorze.

Procesy

Definicja: Proces [M. Bach WNT 95, T. Borzyszkowski]

Dla każdego programu/procesu/wykonania jądro systemu utrzymuje informacje o:

- Bieżącym stanie wykonania (np. czekanie na powrót z trybu jądra, ...), zwanym kontekstem programu
- Plikach, do których program ma dostęp
- Uprawnieniach dotyczących programu (przeważnie dziedziczone po właścicielu procesu lub grupie procesów)
- Uprawnieniach dotyczących programu (przeważnie dziedziczone po właścicielu procesu lub grupie procesów)
- Bieżącym katalogu programu
- Obszarze pamięci, do którego program ma dostęp, i zawartości tego obszaru

ps

Polecenie ps generuje listę wszystkich aktywnych procesów w systemie, ich stan, rozmiar, nazwę oraz właściciela.
`a` — wyświetla wszystkie procesy nie będące związane z terminalem.
`r` — tylko wykonywane procesy.
`T` — procesy związane z tym terminalem.
`u` — dodaj informacje o użytkowniku procesu.
`h` — pomiń nagłówek.
`w` — wyświetla parametry linii poleceń procesu (do polowy linii)..
`ww` — wyświetla wszystkie parametry linii poleceń procesu niezależnie od długości.
Przykład: `$ ps ar`

ps

W pierwszej linii znajdują się nagłówki informacyjne o znaczeniu kolejnych kolumn wydruku:.
`USER` — właściciel danego procesu.
`PID` — numer identyfikacyjny procesu.
`%CPU` — procentowe wykorzystanie procesora.
`%MEM` — wykorzystanie pamięci przez procesor w procentach.
`VSZ` — pamięć wirtualna przydzielona procesowi.
`RSS` — pamięć wykorzystana przez proces.
`TTY` — terminal kontrolny procesu. Znak ? w tej kolumnie oznacza, ze proces nie jest połączony z żadnym terminalem kontrolnym.

Procesy

ps

`STAT` — stan procesu:

- `S` — proces jest uśpiony.
- `R` — proces jest aktualnie wykonywany.
- `D` — proces jest uśpiony, oczekuje na operacje (zazwyczaj I/O).
- `T` — proces jest debugowany lub został zatrzymany.
- `Z` — zombie.

`START` — czas lub data rozpoczęcia procesu.
`TIME` — przedział czasu wykorzystany przez CPU.
`COMMAND` — nazwa procesu oraz jego parametry.

ps - nowa składnia

- `ps -ef` — wyświetl wszystkie procesy i dla każdego z nich podaj pełną listę
- `ps -f -u user,root` — pełna lista właściwości procesów dla użytkownika `user` oraz root
- `ps -f -p 25009,7258,2426` — listuje procesy z określonymi numerami PID
- `ps -e -o pid,args -forest` — listuje procesy w postaci drzewa

top

Polecenie top jest interaktywna wersja polecenia ps. Zamiast wyświetlać statyczną listę procesów w systemie, polecenie top odświeża ją co 2, 3 sekundy. Istotną wadą tego polecenia jest znaczne obciążenie procesora. Polecenia programu top:

- P — sortuj według zajętości procesora.
- M — sortuj względem zajętej pamięci.
- u — podaj procesy danego użytkownika.
- 1 — informacje o procesorze i/lub rdzeniu.
- q — wyjście.

Przykład:

```
$ top
```

Procesy

`htop`

Nowocześniejsza i bardziej rozbudowana wersja polecenia `top`.

`uptime`

Polecenia `uptime` wyświetla informację o aktualnej godzinie, o tym jak długo system operacyjny jest włączony, ilu użytkowników jest zalogowanych do systemu oraz, tzw. *Load Average*. Jest to średnia ilość procesów, które oczekują na wykonanie w określonej jednostce czasu. W tym poleceniu są trzy przedziały czasowe, pierwsza kolumna oznacza *LA* w ciągu ostatniej minuty, druga w ciągu ostatnich pięciu minut, a trzecia kolumna to ostatnie 15 minut.

Sygnały

Procesy komunikują się z jadrem systemu, a także między sobą aby koordynować swoją działalność. Jednym z nich są sygnały, zwane inaczej przerwaniami programowymi.
Sygnały mogą być generowane bezpośrednio przez użytkownika, może wysyłać je jądro oraz procesy między sobą. 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 (wygenerowania `SIGINT`). Wygenerowanie znaku zakończenia (zazwyczaj `Ctrl-\`) powoduje wysłanie sygnału `SIGQUIT` powodującego zakończenie wykonywania bieżącego procesu z zapisaniem obrazu pamięci.

Sygnały

Sygnały

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.

Przykładowe sygnały

Nazwa	Nr	Opis	Akcja standardowa
SIGHUP	1	zerwanie łączności z terminalem	zakończenie procesu
SIGINT	2	przerwanie programu	zakończenie procesu
SIGQUIT	3	zakończenie programu	zakończ. proc. i utw. obrazu pam.
SIGILL	4	niedozwolona instrukcja	zakończ. proc. i utw. obrazu pam.
SIGTRAP	5	Śledzenie	zakończ. proc. i utw. obrazu pam.
SIGIOT	6	instrukcja IOT	zakończ. proc. i utw. obrazu pam.
SIGEMT	7	instrukcja emulatora EMT	zakończ. proc. i utw. obrazu pam.
SIGFPE	8	wyjątek zmiennopozycyjny	zakończ. proc. i utw. obrazu pam.
SIGKILL	9	zabicie procesu (unicestwienie)	zakończenie procesu
SIGBUS	10	błąd magistrali	zakończ. proc. i utw. obrazu pam.
SIGSEGV	11	naruszenie segmentacji	zakończ. proc. i utw. obrazu pam.
SIGSYS	12	nieprawidłowy arg. funkcji systemowej	zakończ. proc. i utw. obrazu pam.
SIGPIPE	13	pisanie do łącza nie otwartego do czytania	zakończenie procesu
SIGALRM	14	pobudka	zakończenie procesu
SIGTERM	15	zakończenie programowe	zakończenie procesu
SIGUSR1	16	sygnał definiowany przez użytkownika	zakończenie procesu
SIGUSR2	17	sygnał definiowany przez użytkownika	zakończenie procesu
SIGCLD	18	zakończenie procesu potomnego	odrzuć sygnał
SIGPWR	19	niedobór mocy	zakończenie procesu

Sygnały

`sleep`

Czekaj określoną liczbę jednostek czasu zanim przejdziesz do wykonywania kolejnego polecenia. Jednostkami tymi mogą być: `s` (sekundy), `m` (minuty), `h` (godziny), lub `d` (dni). Domyślnie wybierane są sekundy.
Składnia: `sleep amount[units]`

`kill`

Polecenie wysyłające sygnał do procesu. Użytkownik wysyłający sygnał musi być właścicielem procesu lub być administratorem systemu. Domyślnie (jeśli nie wyspecyfikowany) wysyłany jest sygnał `TERM`.
`-S, -SIGNAL` — Wysyła do procesu Sygnał SIGNAL
Składnia: `kill [params] <pid>`
Przykład: `kill 9 1024`
Przykład: `kill -s 9 1024`

Sygnały

`killall`

jest to polecenie konsoli Uniksa wywołujące polecenie kill dla wszystkich procesów mających w nazwie tekst podany jako argument. **Przykład:** `killall -9 pdflatex`

Procesy

`pgrep`

pgrep [opcje] [wzorzec] — wyszukanie procesów pasujących do wzorca

- `pgrep -u root` — wyszukaj wszystkie procesy użytkownika root
- `pgrep -u root -c` — podaj liczbę wszystkich procesów użytkownika root

`pkill`

pkill [opcje] [wzorzec] — wysłanie synału do procesów pasujących do wzorca

- `pkill sleep` — zabija procesy pasujące do tesktu `sleep`

Priorytet procesów

Sprawdzanie priorytetu

```
$ ps -o pid,user,args,nice
```

nice

Uruchom program ze zmodyfikowanym priorytetem.

Przykład: `nice -n +3 sleep 10`

renice

Zmień priorytet działającemu programowi.

Przykład: `renice +10 4977`

Zadania

- Kontrola zadaniami - możliwość zawiesznia i wznawiania działania procesów. Zadania są pzypisane do konkretnego terminala.
- Zawieszanie procesu z konsoli bash: `<ctrl> + z`
- Kontrola zadania za pomocą sygnałów (mało praktyczne...):
  - Zawieszanie procesu: SIGSTOP (19)
  - Uaktywnienie procesu: SIGCONT (18)

Zadania

`jobs`

Wyświetla tablicę aktywnych zadań.

- `-l` — pełna informacja
- `-p` — tylko wyświetl `PID`.

`fg` *(ang.foreground)*

Uaktywnienie procesu.

- `fg vim` — uaktywnienie procesu vim (tylko w wypadku kiedy działa jeden proces vim)
- `fg 1` — uaktywnienie zadania 1

Zadania

`bg` *(ang. background)*

Uaktywnienie procesu w tle.

- `bg vim` — uaktywnienie procesu vim w tle (tylko w wypadku kiedy działa jeden proces vim)
- `bg 1,2,3` — uaktywnienie zadań 1, 2, 3 w tle

`kill %1`

zabij zadanie nr 1

Zarządzanie uruchamianiem procesów

Uruchamianie procesów w tle

```
$ proces1 &
```

Uruchamianie procesów sekwencyjnie

```
$ proces1 ; proces2 ; proces3
```

Uruchamianie procesów warunkowe

- alternatywa: `$ proces1 || proces2`
- koniunkcja: `$ proces1 && proces2`

Uruchamianie procesów ze zmianą priorytetu kolejności warunków

```
$ echo ''c'' || (echo ''a'' && echo ''b'')
```