Temat zajęć: Procesy w systemie operacyjnym

Czas realizacji zajęć:	*45 min.*
Zakres materiału, jaki zostanie zrealizowany podczas zajęć:	Pojęcie procesu, procesy w systemie, usuwanie procesów, priorytety procesów, zarządzanie procesami, status zakończenia procesu, uruchamianie procesów w tle

I. Pojęcie procesu

Każdy uruchomiony w systemie Unix program nosi nazwę procesu. Na proces składają się następujące elementy:

1. Kod binarny procesu załadowany z pliku.
2. Dane programu: struktury danych zadeklarowane w programie oraz pamięć dynamicznie przydzielona do procesu w trakcie jego działania.
3. Dane systemowe: informacja o procesie utrzymywana przez system.

Dodatkowo, podczas tworzenia procesu system inicjalizuje systemowe struktury danych opisujące proces, które następnie są aktualizowane podczas wykonania tego procesu. Do danych systemowych należą:

1. Identyfikator procesu (PID) – unikalna liczba całkowita jednoznacznie identyfikująca proces.
2. Identyfikator procesu macierzystego (PPID) – wartość PID procesu, który stworzył dany proces.
3. Środowisko procesu – zbiór zmiennych środowiskowych. Każdy proces ma swoje, niezależne od innych procesów środowisko wykonania, które początkowo jest kopiowane z procesu-rodzica a następnie jest modyfikowane niezależnie od innych procesów.
4. Standardowe strumienie danych.

Oprócz procesów, w systemie Unix, jak w większości nowoczesnych systemów, wyróżnia się wątki (ang. *thread*), będące najmniejszymi aktywnymi elementami systemu. Wątek jest rodzajem procesu, który dzieli przestrzeń adresową z innym procesem – każdemu wątkowi jest więc przydzielony niezależny identyfikator.

II. Procesy w systemie

Listę procesów dla aktualnej powłoki otrzymamy wywołując polecenie `ps` (ang. *processes*).

```
%ps
PID   TTY          TIME       CMD
14285      pts/0      0:00       -csh
14286      pts/0      0:00       ps
```

numer procesu terminal czas aktywności nazwa

Pełne informacje o procesach aktualnej powłoki podaje polecenie `ps -f` (ang. *full*)

```
%ps -f
USER      PID      PPID    C     STIME       TTY    TIME      CMD
Piotr     14285    14267   0    14:44:00    pts/0  0:00      -csh
Piotr     14286    14285   7    14:44:27    pts/0  0:00      ps
```

nazwa właściciela; numer procesu; numer procesu nadrzędnego; czas uruchamiania; terminal; czas; nazwa procesu

Pełne informacje o wszystkich procesach uzyskamy łącząc opcję `-f` z `-e` (ang. *every process*). Poniżej podano inne przydatne przełączniki polecenia `ps`:

`-a` (ang. *all*) wyświetla listę wszystkich procesów (także tych należących do innych użytkowników)

`-l` (ang. *long*) pozwala wyświetlić dodatkowe informacje o każdym procesie

`-x` (ang. *long*) dołącza do listy informacje o procesach nie dołączonych do terminali (procesy demony)

`-u` powoduje dodanie nazwy użytkownika na początku listy

`-w` powoduje rozszerzenie wyświetlanej listy.

Użycie przełącznika `-l` pozwala na wyświetlenie większej liczby szczegółów dotyczących procesów. W rezultacie wyświetlane są dodatkowe kolumny zawierające następujące informacje:

`S` – status procesu (`S` – proces jest uśpiony; `R` – proces jest aktualnie wykonywany)

`UID` – identyfikator właściciela procesu

`PPID` – identyfikator procesu macierzystego

`PRI` i `NI` – opisują priorytety procesów

`WCHAN` – pozwala sprawdzić jaką funkcję systemową jądra wywołał proces

Wiele z procesów uruchamianych jest przy starcie systemu, pozostałe są uaktywniane przez użytkowników w momencie zlecenia wywołania programów. Dowolny proces może uruchomić kolejny proces potomny i stać się macierzystym (nadrzędnym) wobec tego procesu potomnego. W momencie zarejestrowania się użytkownika w systemie uruchomiony zostaje jego pierwszy proces, czyli powłoka interpretująca polecenia użytkownika.

Wszystkie procesy pracujące w systemie tworzą hierarchiczną strukturę, na szczycie której stoi proces `init`, będący rodzicem wszystkich procesów. Proces `init` ma zawsze wartość PID równą 1. Hierarchię procesów można obserwować korzystając z programu `pstree`. Istnieje także interaktywna wersja komendy `ps` - program `top`. `Top` oprócz wartości zwracanych przez polecenie `ps` wyświetla też inne informacje: aktualną liczbę użytkowników w systemie (linia pierwsza), liczbę procesów (linia druga), obciążenie procesora (linia trzecia) i informację nt. dostępnej pamięci w systemie (linie czwarta i piąta).

Innym programem służącym do obserwowania listy procesów jest program `ksysguard` (dedykowany dla środowiska KDE), lub `gtop` (dla środowiska GNOM).

III. Usuwanie procesów

Dowolny proces może zostać usunięty z systemu przez jego właściciela. Służy do tego polecenie `kill`, wysyłające do procesu o podanym identyfikatorze sygnał przerwania pracy:

`kill` [*-nazwa_lub_numer_sygnału*] *identyfikator_procesu*

Domyślnie, jeśli nie podano numeru sygnału, wysłany zostanie sygnał TERM, powodujący zatrzymanie procesu. Aktualnie uruchomiony proces można również przerwać naciskając kombinację Ctrl-C, co również powoduje wysłanie sygnału TERM. Gdy wysłanie sygnału TERM jest niewystarczające do zatrzymania procesu, należy wtedy wysłać sygnał KILL, który powoduje bezwarunkowe przerwanie procesu.

`kill` -KILL *identyfikator_procesu*

Sygnały mają przypisane numeryczne identyfikatory. Identyfikator sygnału TERM wynosi 2, natomiast sygnału KILL jest równy 9. W poleceniu kill można korzystać także z wartości numerycznych synałów:

`kill` -9 *identyfikator_procesu*

Zatrzymanie wszystkich procesów o danej nazwie powoduje polecenie `killall`. Przykładowo:

`killall` *find*

powoduje zatrzymanie wszystkich programów `find`.

Szczegółową listę sygnałów wraz z ich wartościami numerycznymi zawiera strona pomocy systemowej `signal(7)`.

IV. Priorytety procesów

Każdy proces wykonywany w systemie posiada przypisany mu priorytet, który można odczytać w wyniku wywołania polecenia `ps` z przełącznikiem `-l`.

Kolumna PRI wyświetlana w wyniku tego polecenia zawiera informacje o wartości priorytetu określonego procesu, nadanej mu poprzez system operacyjny. Wartość ta nie może być bezpośrednio zmieniana przez użytkownika. Jednakże użytkownik może wpłynąć na wartość PRI, zmieniając tzw. liczbę *nice*, której aktualna wartość znajduje się w kolumnie NI. Wartość liczby *nice* należy do przedziału: od -20 do 19 i początkowo przyjmuje wartość 0. Im mniejsza wartość liczby *nice* tym wyższy priorytet procesu. Dla działającego procesu liczbę *nice* można zmienić poleceniem:

`renice` *zmiana_priorytetu* [`-p`] *pid* [`-u` *użytkownik*]

Na przykład:

```
renice +10 3442
```

zwiększa liczbe *nice* o 10, co powoduje zmniejszenie priorytetu tego zadania.
Zwykli użytkownicy mogą jedynie zwiększać liczbę *nice*, czyli obniżać priorytet wykonania swoich zadań, natomiast użytkownik `root` jest uprawniony do wykonywania wszelkich zmian na wartości *nice.*

Możliwe jest uruchamianie nowych procesów z ustawionym już nowym priorytetem:

`nice -n` *zmiana_priorytetu* `polecenie`

V. Zarządzanie procesami

Procesy uruchamiane z klawiatury terminala są nazywane pierwszoplanowymi. Powłoka czeka na zakończenie wykonywania procesu i dopiero wtedy jest gotowa na przyjęcie kolejnych poleceń od użytkownika.

Proces można jednak uruchomić w tle. Wówczas powłoka utworzy nowy proces potomny, będący powłoką, której nakaże wykonanie zadanego polecenia, a sama powróci do stanu gotowości na kolejne polecenia. W rezultacie proces, który został uruchomiany w tle zaczyna pracować równolegle z interpreterem poleceń.

Warto zaznaczyć, że praca w tle ma sens jedynie w przypadku programów nieinteraktywnych, czyli takich, które do swojej pracy nie potrzebują interakcji za strony użytkownika. W przypadku uruchamiania programu w tle interpreter poleceń natychmiast przechodzi w stan oczekiwania na następne zlecenie, czyli rozpoczyna czytanie danych z klawiatury. Podobnie, programy działające w tle nie powinny wypisywać informacji na ekranie, bo będą one wypisywane asynchronicznie w stosunku do aktualnie wykonywanych operacji. W tym przypadku, rozwiązaniem tego problemu może być przekierowanie wyników działania takiego programu do pliku i jego późniejsza analiza.

Polecenie jest uruchomione w tle, jeśli po ostatnim parametrze następuje znak &:

polecenie &

Aktualnie uruchomiony proces można także zatrzymać wciskając kombinację Ctrl – Z. Spowoduje to wstrzymanie tego procesu. Wstrzymany proces istnieje w systemie, ale nie jest dla niego przydzielany procesor. Zastopowany proces można wprowadzić do wykonania (kontynuacji) w tle poleceniem `bg` (ang. *background*), a nawet przywrócić po dowolnym czasie z powrotem na pierwszy plan poleceniem `fg` (ang. *foreground*), pod warunkiem jednak, że pomiędzy tymi poleceniami nie uruchomimy w tle innego procesu. Listę aktualnie kontrolowanych zadań można wyświetlić poleceniem `jobs`.

Jeśli wstrzymano więcej niż jedno zadanie, niezbędna będzie ich identyfikacja. Interpreter poleceń wewnętrznie przydziela swoje identyfikatory i za pomocą polecenia jobs można wyświetlić ich wartości. Do konkretnego procesu można odwołując się korzystając z identyfikatora poprzedzonego znakiem %.

```
%jobs
[1]-      Stopped   vim praca.html
[2]+      Stopped   find /usr -name signal.h

%fg 1
```

VI. Status zakończenia procesu

Każdy proces w systemie Unix po zakończeniu swojej pracy przekazuje do systemu informację o tym jak zakończyło się przetwarzanie, określaną statusem zakończenia procesu. Status zakończenia jest liczbą jednobajtową, przy czym przyjęto, że wartość 0 oznacza poprawne zakończenie przetwarzania. Wartości różne od 0 oznaczają błąd. Status zakończenia ostatnio wykonywanego programu można uzyskać w następujący sposób:

```
%echo $?
```

Status zakończenia procesu można wykorzystać do warunkowego uruchomiania poleceń. Fakt, że `polecenie_2` można wykonać tylko gdy `polecenie_1` zakończyło się sukcesem zapisujemy następująco:

```
%polecenie_1 && polecenie_2
```

Natomiast gdy `polecenie_2` może być wykonane tylko wtedy gdy polecenie `polecenie_1` zakończyło się niepowodzeniem:

```
%polecenie_1 || polecenie_2
```

Ponadto w systemie UNIX możemy jednym poleceniem uruchomić kilka procesów, oddzielając poszczególne z nich średnikiem:

```
%polecenie_1; polecenie_2; polecenie_3
```

Taką sekwencję można również wprowadzić w tło:

```
%(polecenie_1; polecenie_2; polecenie_3) &
```

VII.Ćwiczenia.

1. Wyświetl listę własnych procesów komenda `ps`. Porównaj wyniki z wynikami poleceń: `ps x` i `ps ax`. Zbadaj działanie przełączników `l` i i `u`. Zaloguj się do systemu kilkukrotnie poprzez wirtualne konsole lub otwierając nowe okno w środowisku graficznym. Sprawdź poleceniem `tty(1)` nazwę terminala, na którym pracujesz.
2. Znajdź proces macierzysty dla procesu `ps`. Odszukaj przodka wszystkich procesów (PID=1). Wyświetl hierarchię procesów poleceniem `pstree(1)`. Obejrzyj listę procesów poleceniem `top(1)` sortując ją wg stopnia zajętości procesora i ilości zajętej pamięci. W środowisku graficznym KDE uruchom program `ksysguard` (dostępny po wciśnięciu `Ctrl-Esc`).

3. Sprawdź identyfikator procesu init. Za pomocą polecenia `pgrep(1)` wyświetl identyfikatory wszystkich swoich interpreterów poleceń.
4. Zapoznaj się z listą sygnałów na stronie pomocy systemowej `signal(7)`. Uruchom sesję edytora `vi` i wysyłaj komendą `kill` kolejne sygnały do tego procesu. Użyj sygnałów: HUP, INT, TERM, QUIT, KILL, STOP, CONT.
5. Zbadaj działanie poleceń `killall(1)` i `pkill(1)`.
6. Uruchom proces `sleep` w tle. Przełącz go do pracy w trybie pierwszoplanowym. Wstrzymaj sesję edytora `vi` kombinacja Ctrl-Z, uruchom nową sesję i wstrzymaj ją również. Wyświetl aktywne sesje komendą `jobs`. Wznów pracę do trybu pierwszoplanowego komendą `fg`, następnie znów przełącz go do pracy w tle komenda `bg`.

VIII.Zadania do samodzielnego wykonania.

1) Sprawdź, jakie polecenia uruchomił dowolny inny użytkownik, pracujący w tej chwili w systemie.
2) Uruchom w tle wyszukiwanie w całym systemie plików o nazwach pasujących do wzorca "`*user*`", błędy przekieruj na urządzenie puste, wyniki do pliku wyniki. Uruchamiając, obniż priorytet tego polecenia do najniższego.

IX. Literatura.

[Sob01] Sobaniec C., *Linux. Podręcznik Użytkownika.*, Wydawnictwo NAKOM, 2001, ISBN 83-86969-53-9.