

Wstęp do systemu operacyjnego UNIX

Laboratorium 6:

Procesy w systemie UNIX

Proces jest drugą, obok pliku, podstawową jednostką zarządzaną przez system operacyjny. Zasadniczo jest on zbiorowością bajtów, w których system operacyjny wyróżnia obszar (segment) stosu, obszar (segment) instrukcji oraz obszar (segment) danych. W systemach operacyjnych rodziny UNIX istnieje jeden mechanizm uruchamiania procesu. Nowy proces może zostać utworzony jedynie przez proces już istniejący. Pierwszy proces (`init`) jest uruchamiany „ręcznie” podczas inicjalizacji systemu operacyjnego, zawsze posiada numer 1. Proces uruchamiania każdego innego procesu składa się zawsze z dwóch kroków. W pierwszym kroku proces, który chce uruchomić nowy proces (ten pierwszy jest nazywany procesem macierzystym lub rodzicem, zaś drugi potomnym) wywołuje funkcję systemową `fork(2)`¹, która tworzy identyczny proces. Nowo utworzony proces dziedziczy od rodzica całe środowisko (zmienne środowiskowe i ich wartości, deskryptory otwartych plików itd.). Po rozwidleniu procesu, proces rodzica wywołuje funkcję `exec(2)`. Funkcja `exec(2)` dokonuje załadowania do segmentu instrukcji oraz danych procesu potomnego kodu z pliku, do którego ścieżka dostępu jest jednym z argumentów wywołania funkcji. Nowy proces został utworzony i jeśli warunki występujące w systemie na to pozwalają rozpoczyna się jego wykonanie. Proces rodzica może oczekiwać na zakończenie procesu potomnego wykonując funkcję systemową `wait(2)`.

Procesy mogą komunikować się ze sobą na kilka sposobów. Najpopularniejszym z nich jest przesyłanie sygnałów. Sygnał ma charakter asynchroniczny - nadawca nie interesuje się dalszym losem procesu odbiorcy. Sygnał powoduje pewne ustalone zachowanie procesu, który sygnał odebrał. Zachowanie po otrzymaniu sygnału jest zdefiniowane w systemie, lecz są również sygnały, które pozostawiono do zdefiniowania dla użytkownika. Użytkownik może wysłać sygnał do procesu przy pomocy funkcji `kill(2)` lub komendy `kill`, która posługuje się tą funkcją.

Procesy są rozpoznawane w systemie po unikalnym numerze identyfikacyjnym (PID - Process Identification). Zarządzanie procesami odbywa się najczęściej z wykorzystaniem ich numerów identyfikacyjnych, choć ostatnio pojawiły się również pakiety komend umożliwiające zarządzanie procesami z wykorzystaniem ich nazwy (pakiet `procps`). Efektywnie procesem może zarządzać jego właściciel. Najczęściej jest to użytkownik, który proces uruchomił. Użytkownik `root` może zarządzać wszystkimi procesami w systemie.

0. Podłącz się do systemu jako użytkownik root.

1. Sprawdź, czy użytkownik `test` jest zdefiniowany w systemie. Jeśli tak, to zmień jego hasło. Jeśli nie, to dodaj go.

2. Podłącz się z kolejnej konsoli do systemu jako użytkownik `test`.

Podstawową komendą służącą zarządzaniu procesami w systemie UNIX jest komenda `ps` (Process Status). Niestety podsystem zarządzania zadaniami jest tym, który różni się najbardziej między systemami Unix wzorowanymi na BSD i SystemV. I choć podstawowe komendy nazywają się tak samo, to różnią się opcjami oraz dostarczaną informacją. Komenda `ps` w systemie Linux akceptuje opcje systemu BSD i SystemV, przy czym tych pierwszych nie poprzedzamy znakiem „-”. W dalszej części używać będziemy opcji z systemu BSD. Wynik (fragment) użycia komendy `ps aux` zamieszczono poniżej.

	USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
1	root	1	0.0	0.1	1372	480	?	S	04:27	0:04	init [3]
2	root	180	0.0	0.0	0	0	?	SW	04:27	0:00	[kjournald]
3	root	528	0.0	0.2	1428	560	?	S	04:27	0:00	syslogd -m 0
4	root	1139	0.0	0.1	1344	400	tty3	S	04:28	0:00	/sbin/mingetty tty3
5	root	1140	0.0	0.1	1344	400	tty4	S	04:28	0:00	/sbin/mingetty tty4
6	root	1141	0.0	0.1	1344	400	tty5	S	04:28	0:00	/sbin/mingetty tty5
7	root	1142	0.0	0.1	1344	400	tty6	S	04:28	0:00	/sbin/mingetty tty6
8	bory	1145	0.0	0.5	2468	1288	tty1	S	04:28	0:00	-bash
9										
10											

¹ We wszystkich przypadkach kiedy opisujemy funkcje systemowe lub biblioteczne oznaczamy je nawiasami okrągłymi. Wewnątrz nawiasu dodatkowo podajemy rodzaj dokumentacji wbudowanej w którym można znaleźć opis funkcji.

3. W przedstawionym powyżej listingu brakuje informacji o priorytecie i wartości parametru NICE. Znajdź w manualu, przy pomocy jakich opcji można uzyskać tę informację. Jakie są wartości priorytetu i NICE procesu `-bash`.

W dalszej części wykorzystamy trywialny program, którego głównym zadaniem będzie obciążanie procesora:

```
1 int main(int a, char *b[]){
2     double a, b;
3     while (1) {
4         a = b*b;
5         b = 12.7;
6     }
7     return 0;
8 }
```

4. Zapisz powyższy program w pliku o nazwie `prog.c`. Skompiluj go zapisując kod wykonywalny do pliku o nazwie `prog`.

Jak wspomniano we wprowadzeniu proces macierzysty może oczekiwać na zakończenie procesu potomnego lub nie. W przypadku gdy do uruchamiania zadań dysponujemy konsolą z pojedynczą linią komend pozornie znika gdzieś najistotniejsza zaleta systemu Unix, a mianowicie wielozadaniowość. Jest to jedynie pozorne, gdyż zadanie jesteśmy w stanie uruchomić tak, aby linia komend była dostępna, a zadanie wykonywało się w tle. W ten sposób będzie wykonywało się każde zadanie, które uruchomiono podając na końcu linii komend znak `&`.

```
1 [bory@thorin sph]$ ./prog &
2 [1] 1608
```

Numer w nawiasie kwadratowym jest wykorzystywany przez interpreter poleceń do zarządzania uruchomionymi przezeń procesami. Druga liczba to numer identyfikacyjny procesu (PID).

5. Uruchom program `prog` w tle. Przy pomocy komendy `ps` określ jego numer identyfikacyjny PID. Jaki jest numer identyfikacyjny jego rodzica (PPID - Parent Process Identification)? Jaki proces jest procesem macierzystym dla procesu uruchomionego z linii komend?

Zarządzanie procesami zazwyczaj prowadzi się wysyłając do nich odpowiednie sygnały. Z linii komend robimy to poleceniem `kill`. Wszystkie sygnały zdefiniowane w systemie można zobaczyć uruchamiając komendę `kill` z opcją `-l`. Komenda `kill` wymaga podania numeru procesu (PID) do którego sygnał ma zostać wysłany. Numer sygnału lub jego nazwę symboliczną podajemy jako opcje wywołania komendy. Pominięcie opcji spowoduje, że wysłany zostanie sygnał 15 (TERM). Zatem komendy:

```
1 [bory@thorin bory]$ kill 1211
2 [bory@thorin bory]$ kill -15 1211
3 [bory@thorin bory]$ kill -TERM 1211
```

są równoważne.

6. Do uruchomionego programu `prog` wyślij sygnał `STOP`. Jaki jest stan procesu, do którego został wysłany sygnał? Co stan ten w praktyce oznacza?

7. Do procesu `prog` wyślij sygnał `CONT`. Jaki jest teraz stan procesu?

8. Zakończ wykonywanie procesu `prog`.

Często zdarza się tak, że uruchomiony został w linii komend proces, który będzie wykonywał się dość długo, a z pewnych powodów nie chcemy przerywać jego wykonania i uruchomić go ponownie w tle. Zainteresowanie jesteśmy odzyskaniem linii komend bez przerywania zadania, czyli przeniesiem go do tła. Po uruchomieniu programu zatrzymujemy go naciskając kombinację klawiszy `Control-z`, jak poniżej:

```
1 [bory@thorin bory]$ ./prog
2
3 [1]+  Stopped                  ./prog
4 [bory@thorin bory]$
```

Listę zatrzymanych w ten sposób procesów możemy obejrzeć używając komendy `jobs` :

```
1 [bory@thorin bory]$ jobs
2 [1]+  Stopped                  ./prog
```

Proces zatrzymany uruchamiamy komendą `bg` :

```
1 [bory@thorin bory]$ bg
2 [1]+  ./prog &
3 [bory@thorin bory]$ jobs
4 [1]+  Running                  ./prog &
```

W tym momencie proces wykonuje się w tle. Numer procesu w nawiasach kwadratowych jest wykorzystywany wtedy, gdy mamy więcej niż jeden zatrzymany lub pracujący w tle proces. Numer ten podajemy po znaku `%`, co ilustruje przykład:

```
1 [bory@thorin bory]$ jobs
2 [1]   Running                  ./prog &
3 [2]-  Stopped                  ./prog
4 [3]+  Stopped                  ./prog
5 [bory@thorin bory]$ bg %2
6 [2]-  ./prog &
7 [bory@thorin bory]$ jobs
8 [1]   Running                  ./prog &
9 [2]-  Running                  ./prog &
10 [3]+  Stopped                  ./prog
```

Proces pracujący w tle można przenieść na „plan pierwszy” przekazując mu do dyspozycji terminal (linię komend). Służy do tego komenda `fg` :

```
1 [bory@thorin bory]$ fg %2
2 ./prog
```

Przesunięty na plan pierwszy proces można przerwać kombinacją klawiszy `Control-c` co w praktyce oznacza wysłanie do procesu sygnału `INT`. Zadania pracujące w tle można zatrzymać wysyłając do nich odpowiedni sygnał przy pomocy komendy `kill` :

```
1 [bory@thorin bory]$ kill %1 %3
2 [3]+  Stopped                  ./prog
3 [bory@thorin bory]$
4 [1]-  Terminated              ./prog
5 [3]+  Terminated              ./prog
```

9. Uruchom oraz przenieś do tła dwa programy prog (kolejno). Jeden z nich zakończ wysyłając sygnał komendą `kill`. Drugi przenieś na plan pierwszy i zakończ kombinacją klawiszy `Ctrl-c`.

W systemach wielozadaniowych procesor jest przyznawany zadaniu na podstawie wartości priorytetu. Wartość ta jest obliczana dla każdego procesu w systemie cyklicznie w ściśle określonych momentach czasowych. Na wartość priorytetu ma wpływ ogólnie mówiąc historia procesu w systemie oraz wartość współczynnika NICE. W różnych komercyjnych implementacjach systemu Unix różnie wygląda algorytm wyznaczający wartość priorytetu. Użytkownik nie ma bezpośredniego wpływu na priorytet procesu. Może go zmieniać jedynie pośrednio, zmieniając wartość NICE, której wartość jest wykorzystywana przy wyliczaniu priorytetu. Można tego dokonać uruchamiając proces (komenda `nice`) lub zmienić wartość NICE procesu już uruchomionego (komenda `renice`). Należy zwrócić uwagę, że zwykły użytkownik może jedynie zwiększać wartość NICE czyniąc proces mniej ważnym w systemie. Użytkownik `root` może tę wartość zmieniać dowolnie.

10. Przy pomocy komendy `nice` uruchom program `prog` w tle z wartością NICE o 5 większą od wartości domniemanej. Sprawdź komendą `ps` ile wynosi wartość NICE oraz priorytet tego procesu. Usuń proces komendą `kill`.
11. Uruchom program `prog` w tle. Korzystając z komendy `renice` zwiększ wartość NICE o 10 w stosunku do wartości aktualnej. Sprawdź komendą `ps` ile wynosi wartość NICE oraz priorytet tego procesu. Usuń proces komendą `kill`.
12. Uruchom program `prog` w tle. Odłącz się od systemu, a następnie podłącz ponownie. Sprawdź komendą `ps`, czy program `prog` nadal się wykonuje. Jaką opcję należy użyć? Jeśli proces się wykonuje, zakończ go komendą `kill`.

Podczas administrowania systemem często pojawia się konieczność wykonywania pewnych czynności regularnie, w stałych odstępach czasu. W tym przypadku możemy wykorzystać kolejki zadań definiowane w tablicach demona zegarowego `cron`. Niestety nie wszyscy użytkownicy są uprawnieni do korzystania z możliwości demona zegarowego. Decyduje o tym zawartość dwóch plików konfiguracyjnych (tekstowych), które znajdują się w katalogu `/etc` o nazwach `cron.allow` i `cron.deny`. Zawierają one odpowiednio nazwy użytkowników (login name, po jednym w linii) tych, którzy mogą wykorzystywać podsystem i tych dla których jest to zabronione. Tuż po instalacji systemu żaden z tych plików nie istnieje co oznacza, że podsystem może wykorzystywać tylko użytkownik `root`. Warto pamiętać, że jeśli zabronimy użytkownikowi dostępu po zdefiniowaniu przez niego operacji (poprzez tablicę) nie będzie to miało wpływu na zdefiniowane już zadania.

13. Podłącz się do systemu jako użytkownik `root`. Zapoznaj się z zasadami kreowania i działania zbiorów konfiguracyjnych dla korzystania z tablic kolejek demona zegarowego. Utwórz zbiory tak, aby tylko użytkownik `root` oraz `test` mogli z nich korzystać.

Tablicę zadań uruchamianych cyklicznie przez demon zegarowy może posiadać każdy użytkownik. Do zarządzania jej zawartością służy komenda `crontab`. Opcja `-e` uruchamia edytor i pozwala na edycję jej zawartości. Po jej zapisaniu w katalogu `/var/spool/cron` pod nazwą taką jak login użytkownika, demon zegarowy będzie uruchamiał zadania w wyspecyfikowanym czasie. Opisane wcześniej pliki konfiguracyjne `cron.allow` i `cron.deny` wpływają jedynie na polecenie `crontab` czyli w istocie na możliwość edycji lub tworzenia tablicy demona zegarowego.

Opcja `-l` polecenia `crontab` pozwala na listowanie jej zawartości. Usunięcie całej zawartości tablicy umożliwia opcja `-r`. Plik specyfikacji jest plikiem tekstowym, w którym jedna linia oznacza specyfikację jednego zadania. Linia podzielona jest na 6 pól (kolumn) oddzielonych znakiem białym (spacja, tabulacja). Pola oznaczają:

1. minuta , zakres: 0-59
2. godzina , zakres 0-23
3. dzień miesiąca , zakres 1-31
4. miesiąc , zakres 1-12
5. dzień tygodnia , zakres 0-7 – gdzie zarówno 0 jak i 7 oznaczają niedzielę,
6. zadanie do wykonania

Pole specyfikacji czasu uruchomienia zadania może zawierać gwiazdkę (*), która oznacza zakres “pierwszy-ostatni”.

Dozwolone są zakresy liczb. Zakresy są dwiema liczbami, oddzielonymi myślnikiem. Zakres ten jest domknięty. Np, 8-11 dla “godzin” oznacza wywoływanie w godzinach 8, 9, 10, 11.

Dozwolone są też listy. Lista jest zbiorem liczb (lub zasięgów), oddzielonych przecinkami. Przykłady: “1,2,5,9”, “0-4,8-12”.

W ostatniej kolumnie specyfikujemy ścieżkę dostępu (najbezpieczniej bezwzględna) do programu, który ma zostać wykonany.

14. Będąc podłączonym do systemu jako użytkownik `root` zapewnij przeglądanie systemu plików `/home` w parzyste dni każdego miesiąca o godzinie 4:15 w celu znajdowania i usuwania plików o nazwie `core`. Czy popełnienie błędu w tablicy jest przyjmowane przez system bezkrytycznie?.
15. Usuń zawartość tablicy demona zegarowego dla użytkownika `root`.

Oprócz opisanych poprzednio tablic demona zegarowego związanych z poszczególnymi użytkownikami istnieje w systemie tablica globalna zawierająca zadania „systemowe”. Tablica (`/etc/crontab`) zawiera nazwę użytkownika w kontekście którego zadanie ma zostać uruchomione (jest to 6 kolumna).

16. Zbadaj zawartość globalnej talicy demona zegarowego. Jakie zadanie jest uruchamiane o godzinie 4:22 i w jakie dni tygodnia to następuje.

Oprócz możliwości cyklicznego uruchamiania zadań, demon zegarowy umożliwia również jednokrotne uruchomienie zadania o ściśle określonym czasie. Do umieszczania zadań w kolejce monitorowanej przez demon zegarowy służy komenda `at`. Podobnie jak w przypadku cyklicznego uruchamiania zadań, do korzystania z komendy `at` upoważnieni są użytkownicy zgodnie z konfiguracją zapisaną w plikach `/etc/at.allow` oraz `/etc/at.deny`.

17. Podłącz się do systemu jako użytkownik root. Zkonfiguruj zbiory `/etc/at.allow` oraz `/etc/at.deny` tak, aby wszyscy użytkownicy zdefiniowani w systemie mogli korzystać z komendy `at`.

Składnia komendy `at` jest skomplikowana. W prostym przypadku, jeśli chcemy uruchomić zadanie o nazwie `prog` za 5 minut (minimalna rozdzielczość to 1 minuta) od momentu bieżącego uruchamiamy program `at` specyfikując czas uruchomienia zadania jako argument wywołania komendy. Symbol `at>` to znaczek zachęty programu. Specyfikujemy po jednym zadaniu w linii przechodząc do następnego klawiszem `Enter`. Specyfikowanie kończy się wraz z końcem pliku czyli poprzez kombinację klawiszy `Control-d`:

```
1 [bory@thorin bory]$ at now + 5 min
2 warning: commands will be executed using (in order) a) $SHELL b) login shell c) /bin/sh
3 at> prog
4 at> <EOT>
5 job 1 at 2003-08-02 01:05
```

Zadania oczekujące na uruchomienie możemy podglądać komendą `atq`. Istotny jest dla nas numer, który pojawia się w pierwszej kolumnie:

```
1 [bory@thorin bory]$ atq
2 1          2003-08-02 01:05 a bory
```

Zadanie oczekujące w kolejce zadań do wykonania możemy z niej usunąć komendą `atrm` podając jako argument numer zadania w kolejce:

```
1 [bory@thorin bory]$ atrm 1
```

18. Podłącz się do systemu jako użytkownik test. Zaplanuj uruchomienie programu `prog` za 5 minut od chwili bieżącej. Sprawdź stan kolejki. Sprawdź, czy zadanie zostało uruchomione. Komendą `ps` określ jego numer identyfikacyjny. Zakończ wykonywany program (komendą `kill`).

Zarówno w przypadku podsystemu `cron` jak i `at` uruchamiane zadania przesyłają za pomocą poczty elektronicznej wyniki działania. Wysłany list zawiera zarówno standardowe wyjście jak i standardowe wyjście błędów uruchomionego polecenia (poleceń). Jeśli nie chcemy być atakowani listami musimy zadbać aby uruchamiany program nic nie wypisywał.

19. Usuń utworzone w katalogu `/etc` pliki konfiguracyjne `cron` i `at`.

Poniższy program jest przykładem podłączenia funkcji obsługującej nadchodzący sygnał, w naszym przypadku jest to sygnał `USR1`:

```
1 #include <signal.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <stdio.h>
5
6 void obsluga_sygnalu(int sig){
7     printf("Proces potomny: dostalem sygnal\n");
8 }
9
```

```
10 int main(int argc, char *argv[]){
11     signal(SIGUSR1, obsluga_sygnalu); /* podlaczenie funkcji obslugujacej */
12     while(1) {
13         printf ("ciagle dzialam ...\n");
14         sleep(1); /* oczekuj nieskonczenie wiele razy po 1 sek. */
15     }
16     return 0;
17 }
```

20. Na podstawie powyższego, napisz program, który obsługuje sygnały `USR2` oraz `INT` i informuje o ich otrzymaniu. Jak przerwać działanie takiego programu?