Wprowadzenie do systemu Unix
Procesy

Wprowadzenie

Niniejsze zajęcia są próbą przedstawienia podsystemu procesów w środowiskach uniksowych. Z tej instrukcji będziesz mógł dowiedzieć się, w jaki sposób znajdować podstawowe parametry procesów (identyfikator PID, identyfikator rodzica, priorytet oraz wiele innych), jak tworzyć i zarządzać wykonywaniem procesów, planować wykonywanie przyszłych zadań oraz prowadzić komunikację międzyprocesową (na przykładzie sygnałów).

W ćwiczeniu wykorzystamy też parę programów, które samodzielnie skompilujemy – użyjemy do tego kompilatora GNU Compiler Collection (GCC).[1]

[1] Jego strona WWW: https://gcc.gnu.org/.

Poznajemy też tutaj możliwości zarządzania zadaniami (*jobs*), które oferuje Bash. Komenda **jobs** wyświetla nam procesy (zadania) aktualnie uruchomione z danej instancji shella. Do tych procesów możemy odwoływać się nie tylko za pomocą ich identyfikatorów (PID), ale także za pomocą *job id*, które poprzedzamy znakiem procenta (**%**).

Każde z zadań może działać na pierwszym planie lub w tle. Zadania działające na pierwszym planie możemy przenieść do tła za pomocą kombinacji klawiszy **Ctrl**-**z**. Tak naprawdę, kombinacja ta zatrzymuje aktywny proces. Musimy go jeszcze obudzić za pomocą komendy **bg** – budzi ona proces w tle. Alternatywnie, komenda **fg** obudzi proces przenosząc go z powrotem na pierwszy plan.

Aby uruchomić proces w tle już przy jego starcie, musimy komendę zakończyć ampersandem (**&**). Na przykład:

```
firefox &
```

Stan procesu można sprawdzić przy pomocy polecenia **ps** z włączonym wyświetlaniem kolumny **stat**. W wyniku wykonania tej komendy możemy zobaczyć poniższe oznaczenia:

O stanach procesów więcej na wykładzie lub tutaj: https://idea.popcount.org/2012-12-11-linux-process-states/.

- **D** – nieprzerywalne oczekiwanie procesu, np. na zakończenie operacji wejścia/wyjścia (ang. *uniterruptible sleep*),
- **R** – wykonywanie procesu (ang. *running*) lub jego oczekiwanie w kolejce do wykonania (ang. *runnable*),
- **S** – oczekiwanie procesu, które może zostać przerwane (ang. *interruptible sleep*),
- **T** – proces w stanie zatrzymania (ang. *stopped*), np. po wysłaniu do niego sygnału **SIGSTOP**),
- **X** – proces zabity (ang. *dead*), w normalnych warunkach taki stan nie powinien występować,
- **Z** – proces „zombie”, po zakończeniu jego wykonywania się, jeśli proces rodzica nie pobrał jego kodu wyjścia.

Często pojawiają się tam dodatkowe ozanczenia:

- **<** – proces o wysokim priorytecie,
- **N** – proces o niskim priorytecie,

- **L** – strony procesu są zablokowane w pamięci (mechanizm używany w przypadku operacji bezpośredniego dostępu do pamięci, tzw. *DMA*),
- **s** – proces jest liderem sesji (identyfikator sesji SID procesu jest równy jego numerowi PID),
- **l** – proces wielowątkowy,
- **+** – proces wykonywany pierwszoplanowo.

Zadania

1. Zapoznaj się z pomocą dotyczącą komendy **ps**. Ustal, w jaki sposób można:

   - określić numer procesu (PID) i numer procesu rodzica,
   - określić zajmowaną pamięć i czas uruchomienia,
   - określić terminal
   - ustalić, jakim (pełnym) poleceniem został wywołany proces.

   Czym różnią się opcje charakterystyczne dla UNIX i BSD?

   O terminalach w systemach uniksowych więcej tutaj: `https://www.linusakesson.net/programming/tty/`.
   Należy pamiętać o tym, aby unikać mieszania opcji uniksowych i BSD w komendzie **ps**.

2. Zaproponuj postać komendy pozwalającej zwrócić informacje o właścicielu procesu, numerze PID, wartościach priorytetu i parametru NICE oraz z jakiego terminala korzysta proces. W jaki sposób można specyfikować oczekiwane kolumny?

3. Zapoznaj się z możliwościami zmiany priorytetu wykonywanego procesu.

   - Utwórz w swoim katalogu domowym katalog `c3`. Przejdź do katalogu `$HOME/c3`. Utwórz w nim plik o nazwie `prog.c`, zawierający następujący kod programu w języku C:

   ```c
   int main (int c, char *d[]){
       double a, b;
       while (1) {
           a = 12.345543;
           b = 0.456456;
           a = a*b;
           b = a/b;
       }
       return 0;
   }
   ```

   - Skompiluj kod źródłowy programu poleceniem `gcc -o prog prog.c`. Uruchom otrzymany program w tle.

     Wykorzystaj **&**.

   - Ile pamięci zajmuje ten proces, jakie są wartości priorytetu i parametru NICE tego procesu?
   - Obniż wartość jego parametru NICE, korzystając z komendy **renice**. Porównaj wartości priorytetu i parametru NICE w obu przypadkach.

- Sprawdź procent wykorzystania czasu procesora przez Twoje procesy, wartości priorytetu i parametru NICE w kilku odstępach czasu, gdy pracują one ze standardowym i obniżonym priorytetem.
- Jakie zależności występują między wartością parametru NICE, a priorytetem?
- Wylistuj wszystkie programy pracujące pod kontrolą bieżącej powłoki (polecenie **jobs**).
- Usuń uruchomione w tym zadaniu programy (polecenie **kill**).
- Uruchom program **prog** ponownie, tym razem z obniżoną wartością parametru NICE i w tle. Jakie wartości priorytetu i parametru NICE posiada zadanie teraz? Jak zajmowany jest czas procesora? Po sprawdzeniu tych parametrów, zakończ wykonywanie uruchomionego programu.
- W jakim zakresie można zmieniać parametr NICE?

4. Uruchom program **prog** jeszcze raz, jako pierwszoplanowy. Zobacz, jak wygląda jego wpis w wynikach polecenia **ps**. Wyloguj się i zaloguj ponownie. Czy proces nadal działa?

5. Powtórz powyższą operację, ale uruchom proces w tle. Czy po wylogowaniu i zalogowaniu proces dalej istnieje?

6. Uruchom program **prog** w tle tak, aby wykonywał się on mimo odłączenia się od systemu. Gdzie szukać tego, co program wypisuje na ekran? Usuń uruchomiony proces.

Polecenie **nohup**.

7. Uruchom program **prog** w tle. Wyślij do niego sygnał **SIGSTOP**. Jaki jest stan uruchomionego procesu? Jak z czasem zmienia się stopień wykorzystania procesora przez ten proces? Do zatrzymanego procesu wyślij sygnał **SIGCONT**. Jaki jest teraz stan procesu? Co dzieje się z wartością w kolumnie opisującej stopień wykorzystania procesora? Usuń uruchomiony proces.

8. W katalogu `$HOME/c3` utwórz plik o nazwie `signal.c`, zawierający kod źródłowy programu w języku C zamieszczony poniżej (do pobrania poleceniem: `wget galaxy.agh.edu.pl/~boryczko/signal.c`):

```c
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

void obsluga_sygnalu(int sig){
    printf("Proces: dostalem sygnal %d\n", sig);
}

int main(int argc, char *argv[]){
    int i;
```

```
    for ( i = 1; i < 64; i++ ) {
        /* Podlaczenie funkcji obslugujacej */
        signal(i, obsluga_sygnalu);
    }
    /* Oczekuj nieskonczenie wiele razy po 1 sek */
    while(1) {
        printf ("ciagle dzialam ...\n");
        sleep(1);
    }
    return 0;
}
```

- Skompiluj program analogicznym poleceniem, jak poprzedni.
- Uruchom program. Co stanie się z procesem, jeśli wyślemy do niego sygnał **SIGQUIT**. Jaka jest reakcja programu na ten sygnał?
- Jak proces zareaguje na sygnał **SIGTSTP**. Przejdź do drugiej konsoli. Ustal numer uruchomionego programu.
- Wyślij do niego sygnał **SIGTERM**.
- Jak zakończyć wykonywanie procesu? Funkcje obsługi których sygnałów nie mogą zostać podmienione?

Więcej informacji o sygnałach: https://www.gnu.org/software/libc/manual/html_node/Termination-Signals.html oraz https://www.bogotobogo.com/Linux/linux_process_and_signals.php

9. Podaj PID procesu zastopowanego, który zużywa najwięcej pamięci wirtualnej.
10. Podaj nazwę użytkownika, który ma najwięcej procesów z zerową wartością parametru NICE.
11. Podaj numery PID trzech procesów, które mają najwięcej pamięci wirtualnej.

Zadanie sprawdzające

Poniższe zadanie wykorzystuje kompleksowo wiedzę z tego laboratorium. Postaraj się wykonać je samodzielnie w domu. Jeżeli masz z nim problemy, przestudiuj ponownie materiały źródłowe, rozwiąż wcześniejsze zadania i podejmij kolejną próbę.

Podaj pełną komendę zwracającą konkretną wartość (nie należy np. liczyć wierszy „ręcznie”):

1. Jak wypisać wszystkie dostępne nazwy kolumn dla komendy **ps**?
2. Jak wyświetlić procesy użytkownika root posortowane wg rozmiaru zużytej pamięci (np. *size*), używając tylko **ps**?
3. Ile jest procesów, których ojcem jest proces o PID równym 1?
4. Znajdź użytkownika w systemie, który ma uruchomioną największą liczbę procesów.
5. Znajdź proces, który zużył najwięcej czasu procesora.

6. Ile procesów potomnych ma bieżący terminal? Nie uwzględniaj komendy, która dokonuje obliczeń.

ID bieżącego terminala jest przechowywana w zmiennej systemowej $$

Podsumowanie komend

Na zajęciach przedstawione zostały następujące komendy:

Grupa	Komenda
Wyświetlanie procesów	ps, jobs, top, htop, pgrep
Przenoszenie i odłączanie procesów	fg, bg, nohup
Zarządzanie priorytetem	nice, renice
Zarządzanie procesami	kill, pkill, killall

Przydatne skróty

Sygnały, oprócz polecenia **kill**, mogą być wysyłane do procesów przy pomocy skrótów klawiaturowych (odbiera je wtedy aktualnie wykonywany pierwszoplanowo proces).

Poniżej przedstawione zostały te najbardziej przydatne:

- Ctrl-c – SIGINT
- Ctrl-\ – SIGQUIT
- Ctrl-z – SIGTSTP

*Uwaga: o **SIGHUP** i kończeniu sesji*

Komendy **nohup** używa się do uruchomienia procesu, co do którego nie chcemy, by reagował on na otrzymany sygnał **SIGHUP**. Sygnał ten wysyłany do wszystkich procesów przyłączonych do danego terminala w momencie zakończenia jego wykorzystywania.

To zachowanie, przyjęte w starszych rozwiązaniach, nie jest jednak domyślne w przypadku obecnych systemów. Wtedy, przy odłączaniu użytkownika od systemu, jego działające w tle procesy, zostają „osierocone", ich procesem rodzica staje się proces o PID równym 1 (w zależności od systemu: **init** lub **systemd**), proces traci swój terminal, jednak dalej działa.

W *Bashu* za sterowanie tym zachowaniem odpowiada opcja **huponexit**, domyślnie wyłączona (nie jest wysyłany **SIGHUP**). Zachowanie to można zmienić z użyciem polecenia:

```
shopt -s huponexit
```

Bezpiecznie jest jednak wywoływać każdy proces, co do którego chcemy mieć pewność, że nie zostanie w takiej sytuacji zakończony, z użyciem **nohup** lub z ustawioną odpowiednio obsługą sygnału *SIGHUP*.

*Uwaga: o składni polecenia **ps***

Polecenie **ps** zawiera kilka zestawów przełączników:

1. opcje w stylu UNIX, poprzedzane myślnikiem (można je grupować),

2. opcje BSD, które nie są poprzedzane myślnikiem (także można je grupować),
3. opcje GNU, poprzedzane dwoma myślnikami (tzw. opcje długie).

Opcje różnych typów mogą być ze sobą łączone, jednak istnieje prawdopodobieństwo pojawienia się konfliktów pomiędzy nimi, dlatego należy unikać takich sytuacji.

Należy uważać na wybierane opcje, przykładowo **ps -aux** (**a** - wszystkie procesy, **u** - lista użytkowników, **x** - nazwa użytkownika) jest odmienne od **ps aux** (**a** - wszystkie procesy, **u** - format *user-oriented*, **x** - wyświetla też procesy bez TTY). Jeżeli użytkownik o nazwie **x** nie istnieje, to **ps** może zinterpretować tę komendę jako **ps aux** i wypisać ostrzeżenie, jednak nie należy polegać na tym zachowaniu.