1 Programy i procesy

Przez *program* rozumiemy ciąg poleceń dla maszyny (komputera). Rozróżniamy *program wykonywalny* – ciąg poleceń w postaci skompilowanej, trudnej do przeczytania i zrozumienia dla człowieka i *kod źródłowy* program napisany w *języku programowania*, który możemy wykonać po kompilacji lub przy pomocy *interpretera*, który jest w stanie wykonać komendy zapisane w programie.

Przez *proces* rozumiemy instancję działającego programu, która powstała w wyniku jego uruchomienia. W danej chwili może istnieć i działać wiele różnych procesów, będących wynikiem uruchomienia tego samego programu.

Każdy proces posiada numer przydzielony przez system operacyjny (PID, ang. process identification number) i jest wykonywany niezależnie od innych procesów działających na tym samym komputerze. Możliwa jest jednak komunikacja pomiędzy procesami za pomocą mechanizmów dostarczanych przez system operacyjny i stąd, *aplikacje* (programy użytkowe) mogą składać się z wielu programów i procesów współpracujących ze sobą.

1.1 Wywołania w tle

Dodanie do komendy znaku & powoduje, że komenda jest wykonana w tle (ang. background), to znaczy nie blokuje konsoli w trybie interaktywnym, a w skrypcie wykonywane są kolejne polecenia bez oczekiwania na zakończenie działania komendy.

Ćwiczenie 1. Wypróbuj: `sleep 5` i `sleep 5 &` .

1.2 Program `ps`

Program `ps` wyświetla informacje o wybranych aktywnych procesach. Wypróbuj **`ps --help`** żeby zobaczyć skróconą wersję opisu programu oraz `man` **`ps`** i `info` **`ps`** .

Ćwiczenie 2. Porównajmy poniższe dwa programy i spróbuj je uruchomić:

```c
#include<unistd.h>
#include<stdio.h>

int main() {
  printf("Hej\n");
  sleep(10);
  printf("Ho\n");
  return 0;
}
```

```
#!/bin/bash

echo Hej
sleep 10
echo Ho
```

Oba robią mniej więcej to samo: po wypisaniu pierwszego komunikatu program jest wstrzymywany, a następnie wypisywany jest drugi komunikat. Przypuśćmy, że pierwszy program po kompilacji nazywa się `hejho`, a drugi skrypt `hejho.sh`. Po uruchomieniu obu programów *w tle* i wywołaniu `ps`:

```
./hejho &
./hejho.sh &
ps
```

można się spodziewać informacji podobnej do poniższej:

```
    PID TTY          TIME CMD
 555115 pts/4    00:00:00 bash
```

```
605697 pts/4    00:00:00 hejho
605771 pts/4    00:00:00 hejho.sh
605772 pts/4    00:00:00 sleep
605797 pts/4    00:00:00 ps
```

Widzimy, że powyższe wywołania stworzyły dodatkowy proces `sleep`.

Ćwiczenie 3. Sprawdź który z powyższych programów spowodował uruchomienie osobnego procesu `sleep`.

Ćwiczenie 4. Sprawdź pomoc dla komendy `sleep`, wypróbuj:

```
man sleep
apropos sleep
info sleep
man -S3 sleep
```

Zwróć uwagę na różnicę pomiędzy komendą `sleep` wywołaną w Bashu i funkcją `sleep` zadeklarowaną w pliku nagłówkowym `unistd.h`.

Ćwiczenie 5. Wypróbuj komendy **ps** i **ps** | wc . Jeśli masz wątpliwości jakie liczby wypisze `wc` w drugim przypadku to masz rację, nie jest to dobrze zdefiniowane. Oba programy są uruchamiane równolegle i nie mamy gwarancji, czy `ps` zdąży zarejestrować działanie `wc`, czy też nie.

1.3 Unicestwienie działającego procesu

Jeśli działający program nie zachowuje się tak, jak planowaliśmy, to możemy go unicestwić komendą `kill` podając numer procesu, np.: **kill** −9 605697 wyśle sygnał o numerze 9 (SIGKILL) do procesu o numerze 605697 powodując jego unicestwienie.

Listę innych zdefiniowanych sygnałów możemy zobaczyć przy pomocy **kill** −l .