Systemy operacyjne

Pracownia 2

Studenci są zachęcani do przeprowadzania dodatkowych eksperymentów związanych z treścią zadań i dzieleniem się obserwacjami z resztą grupy. Dodatkowo student powinien umieć posługiwać się programami `ps`, `kill`, `lsof`, `strace` i `ltrace`.

Treść zadania zawiera w nawiasach nazwy wywołań bibliotecznych, których należy użyć. Proszę najpierw korzystać z podręcznika systemowego (polecenia `man` i `apropos`), a dopiero potem szukać w Internecie. W systemie opartym na pakietach debianowych (np. Ubuntu) należy zainstalować pakiety `manpages-dev` i `manpages-posix-dev`. Głównym podręcznikiem do zajęć praktycznych jest The Linux Programming Interface. Po szczegóły można sięgać do Advanced Programming in the UNIX Environment.

Rozwiązania mają być napisane w języku `C` (a nie `C++`). Kompilować się bez błędów (opcje: `-std=gnu99 -Wall -Wextra`) kompilatorem `gcc` lub `clang` pod systemem `Linux`. Do rozwiązań **musi** być dostarczony plik `Makefile`, tak by po wywołaniu polecenia `make` otrzymać pliki binarne, a polecenie `make clean` powinno zostawić w katalogu tylko pliki źródłowe. Rozwiązania mają być dostarczone w archiwum `tar` (`gz` / `bzip2`), które po dekompresji da jeden katalog w postaci "`${indeks}_${nazwisko}_${imie}_prog${numer_listy}`"[1]. Jeśli wyżej wymienione obostrzenia nie zostaną spełnione, zadania nie będą sprawdzane.

Zadanie 1

Napisz program, który wyświetli numer procesu (`getpid`), rodzica (`getppid`), sesji (`getsid`) oraz grupy procesów do której przynależy (`getpgrp`). W osobnej konsoli wywołaj polecenie `ps` z odpowiednimi argumentami i pokaż, że Twój program drukuje poprawne dane.

Zadanie 2

Napisz program, który tworzy proces (`fork`) zombie. W procesie nadrzędnym (a nie w konsoli!) wykonaj polecenie `ps`, aby pokazać, że istotnie proces potomny stał się zombiakiem. By zapobiec powstawaniu nieumarłych zignoruj sygnał `SIGCHLD` (`sigaction`). Wariant zadania ma być wybieralny poprzez parametr linii poleceń.

Zadanie 3

Zasymuluj program `id`. Wydrukuj uprawnienia, z którymi działa proces - tj. numeryczny i tekstowy identyfikator użytkownika (`getuid` / `getpwuid`), grupy podstawowej (`getgid` / `getgrgid`) oraz grup rozszerzonych (`getgrouplist`). Które z grup to tzw. grupy systemowe?

---

[1] Notacja zmiennych powłoki: `${symbol}` jest zamieniany na wartość zmiennej `symbol`.

Zadanie 4

Napisz proces, który utworzy nową grupę (`setpgid`), a następnie uruchomi sto potomków. Każdy z nich powinien czekać na sygnał (`pause`), a następnie zakończyć działanie (`exit`). Odczekaj w rodzicu pewną ilość czasu, a nastepnie zabij wszystkie procesy (`killpg`) za wyjątkiem rodzica. Taki program zmodyfikuj pokazując jak można przypisywać limit na używane zasoby (`setrlimit`). Zezwól na utworzenie do 50 procesów i uruchom swój program – odpowiednio obsłuż błąd funkcji `fork` (`perror`).

Zadanie 5

Napisz program, który zarejestruje co najmniej jedną funkcję do uruchomienia przy wyjściu (`atexit` lub `on_exit`). Sprawdź czy funkcja ta jest wywoływana, kiedy program zakończy swoje działanie przy pomocy funkcji `abort`, `exit` lub w wyniku otrzymania sygnału. Wariant ustal na podstawie argumentu linii poleceń. Sygnał można dostarczyć wywołując z konsoli polecenie `kill`. Czy proces potomny utworzony wywołaniem `fork` dziedziczy ustaloną funkcję?

Zadanie 6 (2pkt)

Utwórz program symulujący wywołanie `system` z tą różnicą, że ma nie korzystać z pośredniego procesu powłoki. Przeczytaj argumenty linii poleceń (`argv`) i przekieruj je do wywołania `execve` w procesie potomnym. Przyjmij, że ścieżka do programu jest ścieżką absolutną. Poczekaj na zakończenie procesu potomnego (`wait`) i pobierz jego kod wyjścia. Jeśli proces zakończył się w wyniku otrzymania sygnału - wydrukuj jego numer, ew. nazwę.

Zadanie 7

Wydrukuj środowisko programu przy pomocy funkcji `getenv` i `getcwd`. Utwórz proces potomny. Czy jeśli zmienisz w rodzicu środowisko (`setenv`) lub bieżący katalog (`chdir`), to potomek będzie widział te zmiany? Napisz program, który to zaprezentuje.

Zadanie 8

W procesie nadrzędnym otwórz plik do odczytu (`open`). Czy zamknięcie pliku (`close`) w procesie rodzica zamyka także plik w procesie potomnym? Czy odczyt z pliku (`read`) zmienia pozycję kursora (`lseek`) w drugim procesie? Zaprezentuj te zachowania swoim programem.

Zadanie 9

Napisz program, który w podprocesie przechwyci (`sigaction`) sygnał `SIGTERM` oraz zignoruje sygnał `SIGINT`. Obsługa sygnałów powinna drukować odpowiednie komunikaty. Z procesu nadrzędnygo wyślij (`kill`) do dziecka sygnały `SIGTERM`, `SIGINT` i `SIGKILL`. Po jednorazowym otrzymaniu `SIGTERM` przywróć domyślne zachowanie obsługi sygnału.

Zadanie 10

Utwórz kilka wątków (`pthread_create`), które wypiszą własny identyfikator i zaczekają kilka sekund zanim wyjdą. W międzyczasie uruchom program `ps` (`system`), który pokaże, że istotnie w Twoim procesie funkcjonuje więcej niż jeden wątek. Upewnij się, że wątek główny nie zakończy swojego działania, zanim wątki potomne się nie zakończą (`pthread_join`). Czym charakteryzują są wątki odczepione (ang. *detached*)?