Systemy operacyjne

Pracownia programistyczna 1

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`.

Dla ambitnych proponuje się przeczytanie rozdziałów od 7 do 12 z książki “Advanced Programming in the UNIX Environment” wydanie drugie.

Zadanie 1

Napisz program, który tworzy proces zombie. W procesie nadrzędnym wykonaj polecenie `ps`, aby pokazać, że istotnie proces potomny stał się zombie.

Zadanie 2

Napisz program, który zarejestruje co najmniej jedną funkcję przy pomocy wywołania `atexit`. Sprawdź czy funkcja ta jest wywoływana kiedy program zakończy swoje działanie przy pomocy funkcji `abort`, `exit` lub w wyniku otrzymania sygnału. Czy proces potomny utworzony wywołaniem `fork` dziedziczy tą funkcję?

Zadanie 3

Utwórz program symulujący wywołanie `system`, tj. argumenty linii poleceń ma przekierować do wywołania `execve` w procesie potomnym. Poczekaj na zakończenie procesu potomnego i pobierz jego kod wyjścia. Jeśli proces zakończył się w wyniku otrzymania sygnału - wydrukuj jego numer.

Zadanie 4

Wydrukuj środowisko programu przy pomocy funkcji `getenv` i `getcwd`. Utwórz proces potomny. Czy jeśli zmienisz w rodzicu środowisko i bieżący katalog, to potomek będzie widział te zmiany? W procesie nadrzędnym otwórz plik przy pomocy wywołania `open`. Czy działania na pliku (`read` / `close`), zmieniają stan deskryptora pliku w obu procesach?

Zadanie 5

Napisz program, który przechwyci sygnał `SIGTERM` (zakończenie programu) oraz zignoruje sygnał `SIGINT` (przerwanie programu z klawiatury). Obsługa sygnałów powinna drukować odpowiednie komunikaty. Uruchom taki program w podprocesie (wywołanie `fork`), a następnie w procesie nadrzędnym wyślij do niego kolejno sygnały `SIGTERM`, `SIGINT` i `SIGKILL`. Czy można przechwycić sygnał `SIGKILL`? Czy po jednorazowym otrzymaniu `SIGINT` można przywrócić stare zachowanie programu?

Zadanie 5+

Napisz program, który przechwyci sygnał `SIGSEGV` (naruszenie ochrony pamięci) przy pomocy wywołania sigaction, zinterpretuje dane związane z tym sygnałem (struktura `siginfo_t`), wypisze jakiś sensowny komunikat na `stderr`. Po ustaleniu procedury sygnału w oczywisty sposób odwołaj się nieprawidłowo do pamięci (np. do wskaźnika ustawionego na `NULL`).

Zadanie 6

Utwórz kilka wątków przy pomocy funkcji `pthread_create`, które wypiszą własny identyfikator i zaczekają kilka sekund zanim wyjdą. Następnie uruchom funkcją `system` program `ps`, 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 (funkcja `pthread_wait`). Czym charakteryzują są wątki odczepione (ang. detached)?

Zadanie 6+

Zmodyfikuj poprzednie zadanie, tak by jeden z wątków wywołał funkcję `fork`. Co się dzieje z wątkami w trakcie klonowania procesu? Czy jesteś w stanie uzasadnić taką decyzję projektantów wątków POSIX?