Systemy operacyjne

Pracownia 1

Studenci są zachęcani do przeprowadzania dodatkowych eksperymentów związanych z treścią zadań i dzieleniem się obserwacjami z resztą grupy. Proszę najpierw korzystać z podręcznika systemowego (polecenia `man` i `apropos`), a dopiero potem szukać niezbędnych wyjaśnień w Internecie. Głównym podręcznikiem do zajęć praktycznych jest The Linux Programming Interface: A Linux and UNIX System Programming Handbook. Należy zapoznać się z treścią rozdziału 2 w celach poglądowych, a resztę książki czytać w razie potrzeby. Bardziej wnikliwe wyjaśnienia zagadnień można odnaleźć w książce Advanced Programming in the UNIX Environment.

Rozwiązania należy starannie przygotować w domu – mają być w postaci listy poleceń do wykonania w terminalu wraz z komentarzami. Na początku zajęć student deklaruje przygotowane zadania. Prowadzący podchodzi do kolejnych studentów i odpytuje z jednego wybranego zadania. Należy znać definicje pojęć oznaczonych pogrubioną czcionką. W przypadku zbędnego przeciągania czasu odpowiedzi lub niedostatecznego przygotowania student może otrzymać punkty ujemne. Procedura będzie kontynuowana zgodnie z algorytmem *round-robin* aż do wyczerpania czasu przeznaczonego na zajęcia.

Ćwiczenie 1

Sprawdź i szczegółowo opisz działanie następujących poleceń `ps`, `ps -A`, `ps -a`, `ps -unazwa_uzytkownika`, `ps -Cpolecenie`. Jak za pomocą polecenia `ps` znaleźć **identyfikator**, **identyfikator rodzica**, **grupy**, **terminal sterujący**? Wskaż, które z wyświetlonych zadań są wątkami jądra. Wyświetl drzewiastą strukturę procesów, a następnie do listingu dodaj wątki. Przyjrzyj się procesowi `init`, kto jest jego **rodzicem**?

Ćwiczenie 2

Uruchom jakiś **proces** (np. `gedit`), a następnie zakończ go poleceniem `kill`. Przetestuj wygodniejszą metodę kończenia procesów (np. `xkill`, `pkill`). Zapoznaj się z dokumentacją polecenia `pgrep`, służącego do filtrowania procesów zgodnie z podanymi kryteriami. Jaki **sygnał** wysyła domyślnie polecenie `kill`? Niektóre sygnały proces może przechwycić – jak wymusić zakończenie danego procesu?

Ćwiczenie 3

Prześledź listę **przodków** bieżącego procesu. Zaprezentuj co się stanie gdy proces utraci swojego rodzica. W tym celu w **terminalu** uruchom świeżą kopię **powłoki** `bash` i uruchom program `gedit` **w tle** (poleceniem `gedit&`). Następnie odczytaj ***pid*** obydwu uruchomionych wcześniej procesów – kto jest rodzicem procesu `gedit`? Poleceniem `kill` wyślij sygnał `SIGKILL` do uruchomionej wcześniej powłoki. Wreszcie poleceniem `ps` wypisz rodzica procesu `gedit`. Wyjaśnij przebieg i wyniki powyższego eksperymentu. Co się stanie, gdy zamiast `SIGKILL` wyślemy powłoce sygnał `SIGHUP`?

Ćwiczenie 4

Czym jest **grupa procesów**, **grupa procesów pierwszoplanowych**, **sesja**, **terminal**? Pokaż wszystkie zadania należące do **bieżącej sesji**, a potem **przyłączone** do bieżącego terminalu. Czym jest **identyfikator sesji** i jaki ma związek z **terminalem sterującym**? Wyświetl listę wszystkich zadań i zidentyfikuj procesy będące **przywódcami sesji**. Znajdź procesy działające w obrębie jednej sesji i jednej grupy procesów. Czemu wprowadzono to rozróżnienie?

Ćwiczenie 5

Wykonaj jakieś polecenie i zbadaj jego **kod wyjścia** (np. poleceniem `echo $?`). Co się stanie, jeśli zakończysz proces przy pomocy sygnału? Jak interpretować otrzymane wartości?

Ćwiczenie 6

Znajdź identyfikator `pid` jednego ze swoich procesów. Następnie wylistuj katalog `/proc/${pid}`[1]. Wyświetl pliki zawierające argumenty wywołania Twojego procesu oraz jego aktualne **zmienne środowiskowe**. Wyświetl plik `maps` i przeanalizuj jego zawartość. Wskaż gdzie znajduje się sterta, stos, segment `text` / `data` / `bss` programu oraz pozostałości dynamicznego konsolidatora.

Ćwiczenie 7

Uruchom dowolny program (np. `xeyes`), następnie wyślij tej aplikacji sygnał **`SIGSTOP`**. Co się stanie? W jaki sposób przywrócić zatrzymany program do działania? Wyświetl (np. poleceniem `ps` lub przez inspekcję pliku `/proc/$(pid)/status`) **maskę sygnałów** zgłoszonych procesowi (ang. *pending signals*). Jak zmieni się ta maska gdy będziemy wysyłać zatrzymanemu procesowi kolejne sygnały (np.: `SIGUSR, SIGUSR2, SIGHUP`) ?

Ćwiczenie 8

Większość zasobów w systemach uniksowych ma **semantykę pliku**. Zapoznaj się z dokumentacją programu `lsof`. Uruchom program `gedit` i wyświetl wszystkie otwarte pliki należące do tego procesu. Zidentyfikuj, które z zasobów są plikami, katalogami, **urządzeniami**, **gniazdkami**. Przeanalizuj właściwości każdego z plików. Otwórz w edytorze `gedit` jakiś plik tekstowy i zobacz, jak zmieniły się zasoby procesu.

Ćwiczenie 9

Zapoznaj się z dokumentacją poleceń `strace` i `ltrace`. Uruchom jakiś prosty program w trybie śledzenia **wywołań systemowych** i **wywołań bibliotecznych**. Podłącz się do jakiegoś działającego procesu i obserwuj jego działanie. Jak śledzić aplikacje złożone z wielu procesów lub wątków?

Ćwiczenie 10

Przetestuj działanie polecenia `renice`. Wygeneruj sztuczne obciążenie systemu - uruchom polecenie `echo "" | awk '{for(;;) {}}'` w kilku terminalach jednocześnie. Następnie uruchom przeglądarkę plików PDF po czym poleceniem `renice` zmniejsz jej **priorytet**. Jak zmienił się **czas reakcji** procesu? Czy możesz przywrócić priorytet procesu do poprzedniej wartości?

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