Systemy operacyjne

Warsztaty 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 – najlepiej gdyby były w postaci pliku tekstowego z listą poleceń do wykonania i komentarzami. Do prezentacji każdego zadania należy utworzyć osobą zakładkę w terminalu. Na początku zajęć student deklaruje przygotowane zadania. Prowadzący podchodzi do kolejnych studentów i odpytuje z jednego wybranego zadania. Należy być przygotowanym do wyjaśnienia pojęć oznaczonych **pogrubioną czcionką**. W przypadku zbędnego przeciągania czasu odpowiedzi lub niedostatecznego przygotowania student może nie otrzymać punktów za zadanie. Procedura będzie kontynuowana aż do wyczerpania czasu przeznaczonego na zajęcia.

Ćwiczenie 1

Sprawdź i szczegółowo opisz działanie polecenia `ps` z opcjami: `-A`, `-a`, `-uuser`, `-Ccmd`. Jak za pomocą polecenia `ps` znaleźć **identyfikator procesu** / **grupy** / **rodzica**, **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`). Który **sygnał** wysyła domyślnie polecenie `kill`? Niektóre sygnały normalnie kończące proces można odebrać i obsłużyć – jak zatem wymusić zakończenie danego procesu? Zapoznaj się z dokumentacją polecenia `pgrep`, służącego do filtrowania procesów zgodnie z podanymi kryteriami – jak zliczyć ilość procesów należących do danego użytkownika?

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 $?`) w zależności od tego czy wykona się poprawnie lub nie. Jaką wartość przyjmie kod wyjścia, jeśli zakończysz proces wysyłając mu sygnał? Jak zinterpretować taki kod?

Ćwiczenie 6

Znajdź identyfikator `pid` jednego ze swoich procesów. Następnie wydrukuj zawartość katalogu `/proc/${pid}`[1]. Wyświetl plik zawierający **argumenty wywołania** badanego procesu oraz jego aktualne **zmienne środowiskowe**. Wyświetl plik `maps` i przeanalizuj jego zawartość (`man proc`). Wskaż gdzie znajduje się sterta, stos, segment `text` / `data` / `bss` oraz procedury dynamicznego konsolidatora.

Ćwiczenie 7

Uruchom dowolny program (np. `xeyes`), następnie wyślij (jak?) tej aplikacji sygnał **`SIGSTOP`**. Co się stanie? W jaki sposób przywrócić zatrzymany program do działania? Wyświetl (np. 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.: `SIGUSR1`, `SIGUSR2`, `SIGHUP`) ? Co opisują (`man proc`) pola `Sig*` z pliku `status`?

Ćwiczenie 8

Większość zasobów w systemach uniksowych ma **semantykę pliku**. Zapoznaj się z dokumentacją programu `lsof`. Uruchom program `firefox` i wyświetl wszystkie pilkopodobne zasoby należące do tego procesu. Zidentyfikuj, które z nich są plikami, katalogami, **urządzeniami**, **gniazdkami**, a następnie przeanalizuj ich właściwości. Otwórz jakąś stronę w nowej zakładce – 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? Jak zliczyć ilość wywołań systemowych, które zrobił program w trakcie swojego wykonania?

Ćwiczenie 10

Zmierz czas wykonania procesu (np. polecenie `find /var`) poleceniem `time`. Czemu czasy `user` i `sys` nie sumują się do `real`? Co zatem oznaczają? Nałóż limity na proces **wbudowanym poleceniem powłoki** ulimit – na **czas wykonania** i osobno na **zużycie pamięci**. Następnie zaprezentuj jak te limity są wymuszane na procesach. Jakim sygnałem system operacyjny kończy procesy po przekroczeniu limitu? Pamiętaj, że po nałożeniu limitów nie można ich cofnąć w obrębie tej samej instacji powłoki.

Ćwiczenie 11

Wyświetl wszystkie uruchomione procesy wraz **priorytetem** oraz wartością **nice**. Przetestuj działanie polecenia `renice`. Uruchom polecenie `echo "" | awk '{for(;;) {}}'` w kilku terminalach jednocześnie celem wygenerowania sztucznego obciążenia systemu. Następnie uruchom przeglądarkę plików PDF (`evince`) po czym poleceniem `renice` zmniejsz jej **priorytet**. Jak zmienił się **czas reakcji** procesu? Czy możesz przywrócić wartość `nice` procesu do poprzedniej wartości?

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