

Systemy operacyjne

Lista zadań nr 1

Na zajęcia 13 i 14 października 2021

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Arpaci-Dusseau: [Processes](#)¹, [Process API](#)², [Address Spaces](#)³
- Tanenbaum (wydanie czwarte): 2.1, 10.3, 11.4

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wyświetlaczem** czcionką.

Podręcznikiem do zadań praktycznych jest „Advanced Programming in the UNIX Environment” (w skrócie APUE) oraz „The Linux Programming Interface” (w skrócie LPI). Proszę przede wszystkim korzystać z rozdziałów 2 i 3 anglojęzycznego podręcznika systemowego – np. `man 2 read`. Kiedy jego treść nie jest wystarczająco klarowna, to warto spojrzeć na rozdziały opisujące analogiczne tematy w podręczniku FreeBSD – np. `man 2frebsd read`.

Wskazówka: W systemie bazującym na Debianie zainstaluj pakiety `manpages-dev`, `manpages-posix-dev`, `frebsd-manpages`.

Rozwiązania zadań należy prezentować przy pomocy rzutnika. Każde rozwiązanie należy starannie przygotować do prezentacji przed zajęciami – najlepiej w postaci pliku tekstowego z listą poleceń do wykonania i komentarzami. W przypadku zbędnego przeciągania czasu odpowiedzi ze względu na problemy techniczne prowadzący ma prawo skreślić zadanie i postawić jeden punkt ujemny.

UWAGA! Każdy student **musi** przygotować skrypt, który przy pomocy programu `xrandr`⁴ ustawi rozdzielczość ekranu wbudowanego na 1280×800 i sklonuje go na zewnętrzne złącze VGA lub HDMI. Dla programu terminala należy wybrać dużą czcionkę (około 32 wierszy w trybie pełnoekranowym), kontrastowe kolory i jasne tło. Starannie przetestuj swoją konfigurację przed zajęciami!

¹<http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf>

²<http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf>

³<http://pages.cs.wisc.edu/~remzi/OSTEP/vm-intro.pdf>

⁴<https://wiki.archlinux.org/index.php/xrandr>

Zadanie 1. W systemach uniksowych wszystkie procesy są związane relacją **rodzic-dziecko**. Uruchom polecenie «ps -eo user,pid,ppid,pgid,tid,pri,stat,wchan,cmd». Na wydruku zidentyfikuj **identyfikator procesu**, **identyfikator grupy procesów**, **identyfikator rodzica** oraz **właściciela** procesu. Kto jest rodzicem procesu `init`? Wskaż, które z wyświetlonych zadań są **wątkami jądra**. Jakie jest znaczenie poszczególnych znaków w kolumnie STAT? Wyświetl drzewiastą reprezentację **hierarchii procesów** poleceniem `pstree` – które z zadań są wątkami?

Zadanie 2. Jak jądro systemu reaguje na sytuację kiedy proces staje się **sierotą**? W jaki sposób pogrzebać proces, który wszedł w stan **zombie**? Czemu proces nie może sam siebie pogrzebać? Zauważ, że proces może, przy pomocy `waitpid(2)`, czekać na zmianę **stanu** wyłącznie swoich dzieci. Co złego mogłoby się stać, gdyby znieść to ograniczenie? Rozważ scenariusze (a) dziecko może czekać na zmianę stanu swojego rodzica (b) wiele procesów oczekuje na zmianę stanu jednego procesu.

Wskazówka: Proces wykonujący w jądrze implementację wywołania systemowego `_exit(2)` nie może zwolnić stosu jądra, na którym się wykonuje. Kto zatem musi to zrobić?

Zadanie 3. Do czego służy system plików `proc(5)` w systemie Linux? Dla wybranego przez siebie procesu o identyfikatorze `pid` wydrukuj zawartość katalogu «`/proc/pid`». Wyświetl plik zawierający **argumenty programu** oraz **zmienne środowiskowe**. Podaj znaczenie następujących pól pliku «`status`»: `Uid`, `Gid`, `Groups`, `VmPeak`, `VmSize`, `VmRSS`, `Threads`, `voluntary_ctxt_switches`, `nonvoluntary_ctxt_switches`.

UWAGA! Prowadzący ćwiczenia nie zadowolony się cytowaniem podręcznika systemowego – trzeba wykazać się dociekliwością!

Zadanie 4. Znajdź `pid` procesu **X-serwera**⁵, a następnie używając polecenia «`psmap`» wyświetl zawartość jego przestrzeni adresowej. Zidentyfikuj w niej poszczególne **zasoby pamięciowe** – tj. stos, stertę, **segmenty programu**, **pamięć anonimową**, **pliki odwzorowane w pamięć**. Należy wyjaśnić znaczenie kolumn wydruku!

Zadanie 5. Używając programu «`lsof`» wyświetl **zasoby plikopodobne** podpięte do procesu przeglądarki «`firefox`». Wyjaśnij znaczenie poszczególnych kolumn wykazu, po czym zidentyfikuj **pliki zwykłe**, **katalogi**, **urządzenia**, **gniazda** (sieciowe lub domeny uniksowej) i **potoki**. Przekieruj wyjście z programu «`lsof`», przed i po otwarciu wybranej strony, odpowiednio do plików «`before`» i «`after`». Czy poleceniem «`diff -u before after`» jesteś w stanie zidentyfikować nowo utworzone połączenia sieciowe?

Zadanie 6. Wbudowanym poleceniem powłoki «`time`» zmierz **czas wykonania** długo działającego procesu, np. polecenia «`find /usr`». Czemu suma czasów `user` i `sys` (a) nie jest równa `real` (b) może być większa od `real`? Poleceniem «`ulimit`» nałóż **ograniczenie** na **czas wykonania** procesów potomnych powłoki tak, by limit się wyczerpał. Uruchom ponownie wybrany program – który sygnał wysłano do procesu?

⁵https://en.wikipedia.org/wiki/X_Window_System

Ściągnij ze strony przedmiotu archiwum «so21_lista_1.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

Zadanie 7. Napisz program, który będzie prezentował, że pliki procesu są **kopiowane przez referencję** w trakcie wywołania `fork(2)`. W procesie głównym otwórz plik do odczytu `open(2)`. Czy zamknięcie pliku `close(2)` w procesie głównym zamyka plik także w dziecku? Czy odczyt z pliku `read(2)` zmienia **pozycję kursora** `lseek(2)` w drugim procesie? Wyjaśnij zachowanie swojego programu!

Przed każdym komunikatem diagnostycznym wypisz pid procesu. W drugiej części zadania należy wydrukować bieżącą pozycję kursora pliku przed operacją odczytu z pliku. Należy wykorzystać dostarczone funkcje opakowujące uniksowe wywołania systemowe z biblioteki `libcsapp`.

Wskazówka: Zagadnienie opisano w APUE rozdział 8.3.

Zadanie 8. (Pomysłodawcą zadania jest Piotr Polesiuk.)

Rozwiąż **problem n hetmanów**⁶ z użyciem `fork(2)` i `waitpid(2)`. Gdy w i -tym elemencie tablicy «board» przechowywana jest wartość j znaczy to, że pozycja i -tego hetmana na szachownicy to (i, j) . Mając niekonfliktujące ustawienie pierwszych $k - 1$ hetmanów po kolei startuj n podprocesów z proponowanym ustawieniem k -tego hetmana. Podproces, który wykryje konfliktujące ustawienie hetmanów, ma zakończyć swe działanie. W przeciwnym wypadku zachowuje się jak rodzic dla $k + 1$ hetmana. Podproces, który uzyska prawidłowe ustawienie n hetmanów, ma wydrukować je na standardowe wyjście. Procedura «ndselect» wraca wielokrotnie z kolejnymi liczbami z zakresu $0 \dots n - 1$.

Linie wydruków plansz z prawidłowymi ustawieniami hetmanów nie mogą się przeplatać. Uważaj, żeby przez przypadek nie zaprogramować **fork bomby**⁷!

UWAGA! Należy wytłumaczyć działanie programu rysując diagram procesów pokazany na wykładzie.

⁶https://pl.wikipedia.org/wiki/Problem_ośmiu_hetmanów

⁷https://en.wikipedia.org/wiki/Fork_bomb