# Systemy operacyjne

Lista zadań nr 2

Na zajęcia 22–23 października 2019

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 **wytłuszczoną** czcionką.

Podręcznikiem do zadań praktycznych jest „Advanced Programming in the UNIX Environment" (w skrócie APUE). Proszę przede wszystkim korzystać z podręcznika systemowego. Jeśli jego treść w *Linuksie* nie jest wystarczająco klarowna, to warto spojrzeć do odpowiednika z BSD – najlepiej z użyciem strony mdoc.su.

Zadania wymagające użycia rzutnika, oznaczenie **(P)**, należy starannie przygotować w domu – najlepiej w postaci pliku tekstowego z listą poleceń do wykonania i komentarzami. Każde zadanie należy mieć właściwie przygotowane do prezentacji przed zajęciami. 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 z użyciem programu `xrandr`[4] ustawi rozdzielczość ekranu wbudowanego na 1024×768 i sklonuje go na zewnętrzne złącze VGA lub HDMI. Dla programu terminala należy wybrać dużą czcionkę (około 32 wierszy w trybie pełnoekranowym) i kontrastowe kolory. Starannie przetestuj swoją konfigurację przed zajęciami!

**Zadanie 1 (P).** W systemach uniksowych wszystkie procesy są związane relacją **rodzic-dziecko**. Uruchom polecenie «`ps -eo user,pid,pgid,tid,pri,stat,wchan,cmd`». Na wydruku zidentyfikuj **identyfikator**, **grupę**, **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 (P).** 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 w pliku «`stat`»: `State`, `Groups`, `VmPeak`, `VmSize`, `VmRSS`, `Threads`, `voluntary_ctxt_switches`, `nonvoluntary_ctxt_switches`.

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

**Zadanie 3 (P).** Znajdź pid procesu X-serwera[5], a następnie używając polecenia «`pmap`» 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 4 (P).** 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 5 (P).** 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?

---

[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`
[5] `https://en.wikipedia.org/wiki/X_Window_System`

Ściągnij ze strony przedmiotu archiwum «so19_lista_2.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

**Zadanie 6 (P).** 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 7 (P).** (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 $0...n-1$.

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

**UWAGA!** Należy wytłumaczyć przy tablicy działanie programu rysując tworzące się drzewa procesów.

[6] https://pl.wikipedia.org/wiki/Problem_ośmiu_hetmanów
[7] https://en.wikipedia.org/wiki/Fork_bomb