Lista 1

tags: SO

```
xrandr --output eDP-1 --mode 1280x800
xrandr --output [HDMI] --mode 1280x800
xrandr --output [HDMI] --same-as edP-1
```

Zadanie 1

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

::: info

- **relacja rodzic-dziecko** – procesy powstają poprzez pączkowanie przy użyciu forków. Forkujący proces staje się rodzicem swojej kopii, którą nazywamy dzieckiem.
- **właściciel procesu (USER)** – użytkownik, który obecnie kontroluje proces (jest jeszcze owner procesu, czyli użytkownik, który utworzył proces)
- **identyfikator procesu (PID)** – numer po którym możemy unikalnie zidentyfikować proces.
- **identyfikator rodzica (PPID)** – PID rodzica procesu.
- **grupa procesów** – zbiór procesów o tym samym **identyfikatorze grupy (PGID)**.
- **priorytet procesu (PRI)** – decyduje ile czasu procesora otrzyma proces (im niższa wartość tym większy priorytet). Dziecko przy powstaniu dziedziczy PGID rodzica. Procesy mogą tworzyć i zmieniać grupy procedurą *setpgid*.
- **wątki jądra** – wątki zarządzane przez system operacyjny. Użytkownik nie ma do nich dostępu, `ps` drukuje je w nawiasach kwadratowych. Ich rodzicem jest *kthread* (PID 2).

  In the sample `ps` output, kernel daemons appear with their names in square brackets. This version of Linux uses a special kernel process, `kthreadd`, to create other kernel processes, so `kthreadd` appears as the parent of the other kernel daemons. Each kernel component that needs to perform work in a process context, but that isn't invoked from the context of a user-level process, will usually have its own kernel daemon. For example, on Linux

- **stan procesu (STAT)** – jeden z poniższych:
  - D: nieprzerywalny sen (I/O)
  - I: bezczynny wątek jądra
  - R: wykonywany lub oczekujący na wykonanie
  - S: nieprzerywalny sen (oczekiwanie na zdarzenie)
  - T: zatrzymany przez sygnał kontroli zadań
  - t: zatrzymany przez debbuger podczas tracingu
  - N: niski priorytet
  - <: wysoki priorytet
  - s: jest liderem sesji
  - l: jest wielowątkowy
  - X: martwy
  - Z: zombie

First letter:

S: sleeping

T: stopped

R: running

Second letter:

s: session leader

+: foreground proc group

- **TID** – thread ID
- **WCHAN** – nazwa funkcji jądra, w której proces jest uśpiony ("-" jeśli nie jest, "*" jeśli jest wielowątkowy).
- **CMD** – komenda rezultująca danym procesem. :::

Które zadania to wątki?

`ps` otacza je klamrami. Możemy użyć `--hide-threads` by to potwierdzić.

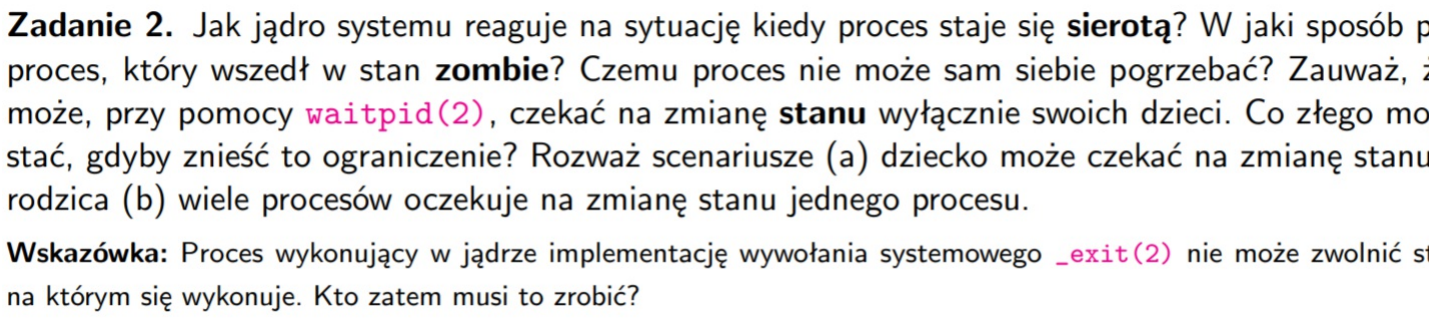
Kto jest rodzicem init?

PID 0, czyli scheduler.

Hierarchia procesów

```
pstree shows running processes as a tree.  The tree is rooted at either pid or init if pid is omitted.  If a user name is specified, all process trees rooted at processes owned by that user are shown.

       pstree visually merges identical branches by putting them in square brackets and prefixing them with the repetition count, e.g.

           init-+-getty
                |-getty
                |-getty
                `-getty

       becomes

           init---4*[getty]

       Child threads of a process are found under the parent process and are shown with the process name in curly braces, e.g.

           icecast2---13*[{icecast2}]

       If pstree is called as pstree.x11 then it will prompt the user at the end of the line to press return and will not return until that has happened.  This is useful for when pstree is run in a xterminal.
```

Zadanie 2



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

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

::: info

- **sierota** – proces, którego rodzic się zakończył lub został zabity.
- **zombie** – zabity proces, który wciąż zajmuje jakieś zasoby komputera i musi być "pogrzebany" przez rodzica.
- **stan** – proces może znajdować się w jednym z trzech stanów:
  - wykonywany lub oczekujący na wykonanie
  - zatrzymany
  - zakończony :::

Jak jądro systemu reaguje na sytuację kiedy proces staje się sierotą?

Sierota zostaje adoptowana przez proces *init* (PID 1).

W jaki sposób pogrzebać proces, który wszedł w stan zombie?

Rodzic takiego procesu musi na niego zaczekać procedurą *wait* lub *waitpid*.

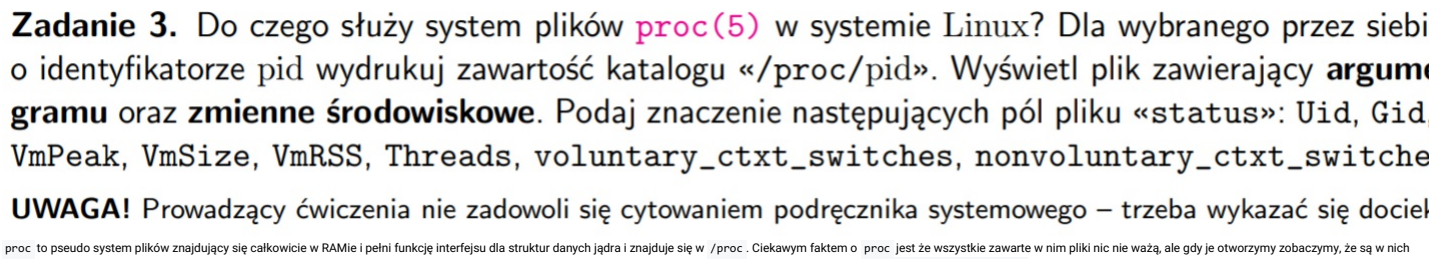
Czemu proces nie może sam siebie pogrzebać?

::: danger Proces zakończył wykonywanie i wyszedł, zatem nie jest mu już przydzielany czas procesora tj. nie może wykonywać już żadnych instrukcji. :::

Co złego mogłoby się stać, gdyby znieść to ograniczenie?

a) dziecko czeka na zmianę stanu swojego rodzica ::: danger Procedura *waitpid* blokuje proces aż do otrzymania sygnału *SIGCHLD*, który jest wysyłany do rodzica, gdy któreś z jego dzieci się zakończy. Jeśli dziecko czeka na rodzica, a rodzic umrze to *SIGCHLD* nigdy nie trafi do tego dziecka. ::: b) wiele procesów oczekuje na zmianę stanu jednego procesu :(

Zadanie 3



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

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

`proc` to pseudo system plików znajdujący się całkowicie w RAMie i pełni funkcję interfejsu dla struktur danych jądra i znajduje się w `/proc`. Ciekawym faktem o `proc` jest że wszystkie zawarte w nim pliki nic nie ważą, ale gdy je otworzymy zobaczymy, że są w nich jakieś dane. Dzieje się tak dlatego, że kernel sprowadza te dane przy odczycie takiego pliku. (możemy np. zobaczyć plik wykonywalny, który tworzy instancję procesu: `/proc/[pid] && ll exe`)

```
ls /proc
```

```
1      1121   134   1553  1771  19044  1960  2079  2152   2411   278   2928   3226   3504   37181  4      430    45026  46271  472    47788  54   690   833  853  927         crypto         kallsyms       mounts
10     11219  135   156   1774  1906   1971  2082  2154   2431   28    2929   3285   3542   37214  4045   4313   45094  46293  47295  478    55   691   834  854  928         devices        kcore          mtrr
10031  11230  136   16    18    19063  1981  2083  2156   2448   280   2936   33     3559   37390  41     4334   451    46432  473    47810  56   692   835  867  929         diskstats      keys           net
10189  11375  137   160   1800  1911   1987  2086  2160   2475   2865  2937   33284  3587   37403  41605  4355   452    4657   47304  479    57   693   836  87   945         dma            key-users      pagetypeir
1019   11389  138   161   1823  1912   2     2088  2175   2485   2868  294    33301  36     37428  41687  4376   453    46617  47333  48     58   696   837  872  957         driver         kmsg           partitions
10192  11434  14    162   1827  1913   2000  2090  2193   26     2873  2949   33327  3641   37726  41952  4394   45459  468    474    480    59   697   839  879  961         dynamic_debug  kpagecgroup    pressure
1074   12     140   1662  1828  1915   2003  2092  22     265    2874  2963   33346  3657   37730  42     44     458    46859  47493  481    599  698   840  885  acpi        execdomains    kpagecount     schedstat
1075   1224   145   1678  1838  1919   2009  2094  22365  2661   2876  3      33404  36570  38     42051  440    45818  46871  475    482    6    732   841  9    asound      fb             kpageflags     scsi
1078   1225   1465  168   1839  1928   2020  2096  2268   2664   2878  3037   33460  36599  3865   42085  441    45888  46893  476    483    60   7503  842  906  bootconfig  filesystems    latency_stats  self
11     12952  147   1683  1841  1935   2033  21    2271   26821  2884  3098   3353   36612  3882   4219   4423   459    469    47600  484    611  7574  845  907  buddyinfo   fs             loadavg        slabinfo
1107   13     148   1697  1866  1943   2039  2103  2294   27     2885  31     3354   36642  3883   42344  44284  46     47     47681  485    682  810   846  909  bus         i8k            locks          softirqs
11187  131    149   17    1891  1944   2056  2104  23     2729   2886  3109   34     36678  3889   42451  44311  460    470    477    49     683  811   849  913  cgroups     interrupts     mdstat         stat
11191  132    15    1707  1894  1945   2073  2105  2316   275    2889  317    34177  36742  39     4275   4437   461    471    47748  51     687  812   850  914  cmdline     iomem          meminfo        swaps
11192  133    153   1711  1899  1948   2074  2107  24     2765   29    31868  3447   36815  39213  429    4451   46134  47110  47749  52     688  825   851  915  consoles    ioports        misc           sys
11194  1334   1538  1769  19    1953   2077  214   2410   2773   2927  32     3459   37     3934   43     44944  46205  47111  47778  53     689  826   852  926  cpuinfo     irq            modules        sysrq-trig
```

```
ls /proc/1
```

```
arch_status         limits         sched
attr                loginuid       schedstat
autogroup           map_files      sessionid
auxv                maps           setgroups
cgroup              mem            smaps
clear_refs          mountinfo      smaps_rollup
cmdline             mounts         stack
comm                mountstats     stat
coredump_filter     net            statm
cpu_resctrl_groups  ns             status
cpuset              numa_maps      syscall
cwd                 oom_adj        task
environ             oom_score      timens_offsets
exe                 oom_score_adj  timers
fd                  pagemap        timerslack_ns
fdinfo              patch_state    uid_map
gid_map             personality    wchan
io                  projid_map
latency             root
```

> pokazanie obecnego pidu programu (tutaj ls) ls /proc/self

:::info

- **argumenty programu** – argumenty z jakimi został uruchomiony program
- **zmienne środowiskowe** – informacje o środowisku, w którym wykonuje się proces takie jak: HOME(ścieżka do katalogu użytkownika), PATH(lista ścieżek szybkiego dostępu), TEMP itd :::

> argumenty programu `cat /proc/1/cmdline`

> zmienne środowiskowe `sudo cat /proc/1/environ`

> status (czytelna dla człowieka wersja `stat` z której korzysta np. `ps` oraz `statm`) `cat /proc/1/status`

:::warning

- **Uid** – identyfikator użytkownika (`id -u ventus`)
- **Gid** – identyfikator grupy głównej (`id -g ventus`)
- **Groups** – grupy pomocniczne (`id -G ventus`)
- **VmPeak** – maksymalny rozmiar pamięci wirtualnej procesu
- **VmSize** – rozmiar pamięci wirtualnej
- **VmRSS** – rozmiar pamięci rzeczywistej zajmowanej przez proces
- **Threads** – liczba wątków należących do procesu
- **voluntary_ctxt_switches, nonvoluntary_ctxt_switches** - liczba zmian kontekstu (voluntary: wait for I/O, nonvoluntary: clock interrupt) :::

Zadanie 4

Zadanie 4. Znajdź pid procesu X-serwera[5], a następnie używając polecenia «pmap» wyświetl z jego przestrzeni adresowej. Zidentyfikuj w niej poszczególne *zasoby pamięciowe* – tj. stos, stertę, **s programu**, **pamięć anonimową**, **pliki odwzorowane w pamięć**. Należy wyjaśnić znaczenie kolumn

> szukanie PIDu X-serwera `ps -eo user,pid,ppid,cmd | grep Xorg`

> zaglądanie do przestrzeni adresowej procesu (-x dodaje headery, RSS i Dirty) `pmap -x PID`

> żeby pokazać stertę `pmap -X PID`

::: warning

- Address - adres początku * Kbytes - rozmiar * RSS - fizyczny rozmiar używanej pamięci * Dirty - dirty strony w KB * Mode - uprawnienia rwx (czytanie, pisanie, wykonywanie) * Mapping - plik na jaki zmapowana jest pamięć lub [anon]/[stack]

- Address - adres początku
- Kbytes - rozmiar
- RSS - fizyczny rozmiar pamięci
- Dirty - ile pamięci stanowią brudne strony, tj. strony, które uległy zmianie i muszą być zaktualizowane w pamięci zewnętrznej
- Mode - uprawnienia
- Mapping - informuje na co mapowana jest pamięć :::

::: info

- **segment programu** – to dane ładowane z pliku do pamięci, pmap drukuje je pod nazwą pliku, z którego pochodzą (tutaj `Xorg`). Po uprawnieniach możemy się domyśleć jakie sekcje do nich należą (np. segment z uprawnieniami *read* i *execute* to .text).
- **pamięć anonimowa** – przestrzeń nie związana z żadnym plikiem bądź urządzeniem. Jest to pamięć alokowana przez program dla rzeczy takich jak stos i sterta, pmap drukuje je jako `[ anon ]`.
- **pliki odwzorowane w pamięć** – segment pamięci wirtualnej zmapowanej bezpośrednio co do bajtu na plik lub plikopodobny zasób (pliki ten mają rozszerzenie `.so`, czyli *shared object*). Dokonane modyfikacje są potem zapisywane do pamięci zewnętrznej. :::

Zadanie 5

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

::: info

- **zasoby plikopodobne** – https://en.wikipedia.org/wiki/Memory-mapped_file (czyli chyba wszystko co posiada deskryptor ale nie jest plikiem na dysku)
- **pliki zwykłe (REG)** – typowe pliki znajdujące się na dysku
- **katalogi (DIR)** – zbiory plików i innych katalogów
- **urządzenia (CHR)** – np. klawiatura, monitor, słuchawki itd.
- **gniazda (unix, IPv4, sock itd)** – dwukierunkowe połączenie między procesami (tj. gniazdo może zarówno wysyłać jak i przyjmować dane)
- **potoki (FIFO, PIPE)** – zapis lub odczyt z pliku (rozumianego w ogólnym znaczeniu – nie tylko jako plik zwykły np. zapis z wyjścia na wejście standardowe). Inaczej połączenie wyjścia jednego procesu z wejściem innego. :::

::: warning

- **FD** – deskryptor pliku
- **TYPE** – np. REG, DIR itd..
- **DEVICE** – numery urządzeń
- **SIZE/OFF** – rozmiar lub offset pliku
- **NODE** – numer węzła pliku lokalnego lub pliku NFS, protokołu internetowego, np. TCP, STR dla strumienia, itp.
- **NAME** – ścieżka w systemie plików :::

```
pidof firefox
```

```
lsof -p PID1,PID2,PID3,... > before
```

```
lsof -p PID1,PID2,PID3,... > after
```

```
diff -u --color before after | grep IPv4
```

Zadanie 6

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

::: info **czas wykonania** – czas spędzony nad procesem od jego powstania do zakończenia. Można go mierzyć w czasie rzeczywistym(real), czasie użytkownika(user - czas spędzony przez procesor wykonujący dany proces w trybie użytkownika) lub czasie systemu(sys). :::

> mierzenie czasu `bash time find /usr`

Czemu suma user i sys nie jest równa real?

Proces może np. oczekiwać na inny proces lub input użytkownika. Takie oczekiwanie liczy się do czasu rzeczywistego, ale jako, że proces nie jest wtedy wykonywany przez procesor to czas użytkownika i systemu nie płynie.

Czy suma user i sys może być większa od real?

W sytuacji wielowątkowej, gdy mamy wiele procesorów pracujących nad dany procesem wykonamy go szybciej (w kontekście czasu rzeczywistego) niż gdybyśmy dysponowali tylko jednym procesorem, a zatem możliwe jest skrócenie czasu rzeczywistego wykonania procesu, aby stał się on krótszy od czasu CPU (sys + user). :::danger SIGKILL wysyłany jest tylko w bashu, zsh wysyła SIGXCPU :::

Ograniczenie

> dajemy limit 2 sekund czasu procesora dla procesów potomnych `ulimit -t 2`

> wyczerpujemy limit `strace yes`

Do procesu wysłano SIGXCPU

Zadanie 7

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

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

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

Wskazówka: Zagadnienie opisano w APUE rozdział 8.3.

::: info

- **kopiowanie przez referencję** - kopiujemy adres pliku, a nie cały plik, tj. plik jest wspólny dla obu procesów.
- **pozycja kursora** - miejsce w tekście, w którym się znajdujemy :::

Zadanie 8

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 przechowywana jest wartość j znaczy to, że pozycja i-tego hetmana na szachownicy to $(i, j$ niekonfliktujące ustawienie pierwszych $k-1$ hetmanów po kolei startuj n podprocesów z propo ustawieniem k-tego hetmana. Podproces, który wykryje konfliktujące ustawienie hetmanów, ma z swe działanie. W przeciwnym wypadku zachowuje się jak rodzic dla $k+1$ hetmana. Podproces, któ prawidłowe ustawienie n hetmanów, ma wydrukować je na standardowe wyjście. Procedura «nd wraca wielokrotnie z kolejnymi liczbami z zakresu $0...n-1$.

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

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

```
(...)

static int ndselect(int n) {
  /* A loop that spawns processes and waits for them */
  for (int j = 0; j < n; j++) {
    pid_t pid = fork();
    if (pid != 0)
      waitpid(pid, NULL, 0);
    else
      return j;
  }
  exit(0);
}

(...)
```

```
(...)

  /* A loop that initializes recursive algorithm. */
  for (int i = 0; i < size; i++) {
    board[i] = ndselect(size);
    if (!is_valid(board, i))
      exit(0);
  }

(...)
```