1

W systemach uniksowych wszystkie procesy sa zwiazane relacja **rodzic-dziecko**. Uruchom polecenie

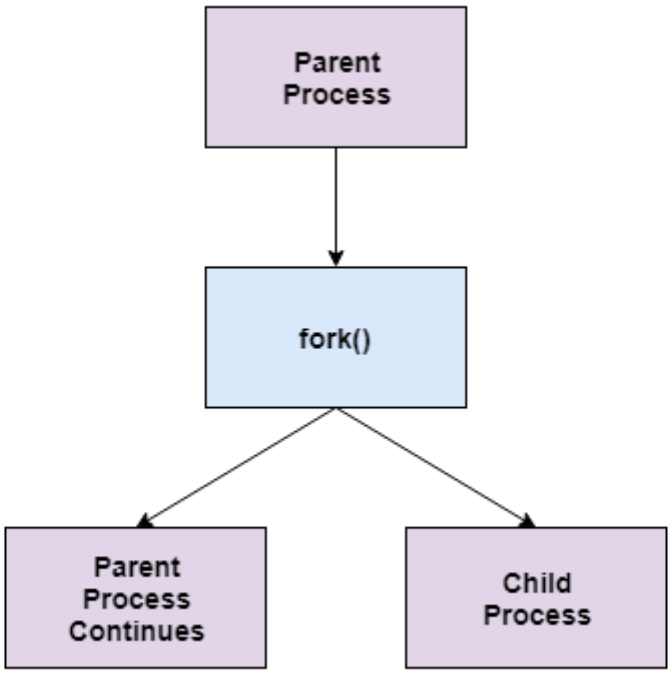
```
ps -eo user,pid,pgid,tid,pri,stat,wchan,cmd
```

Na wydruku zidentyfikuj **identyfikator**, **grupe**, **rodzica** oraz **właściciela** procesu. Kto jest rodzicem procesu init? Wskaż, które z wyświetlonych zadań sa **watkami jadra**. Jakie jest znaczenie poszczególnych znaków w kolumnie STAT? Wyświetl drzewiasta reprezentacje **hierarchii procesów** poleceniem pstree – które z zadań sa watkami?

1.1 Relacja rodzic-dziecko

All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the parent process. In other words, a parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.

A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask. A child process is created as its parent process's copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel. If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.



1.2 Indentyfikator (PID)

is a number used by most operating system kernels such as those of UNIX, macOS and Microsoft Windows to uniquely identify an active process. This number may be used as a parameter in various function calls, allowing processes to be manipulated, such as adjusting the process's priority or killing it altogether.

1.3 Grupa

in a POSIX conformant operating system, a process group denotes a collection of one or more processes. Among other things, a process group is used to control the distribution of a signal; when a signal is directed to a process group, the signal is delivered to each process that is a member of the group.

Process groups are identified by a positive integer, the process group ID, which is the process identifier of the process that is (or was) the process group leader. Process groups need not necessarily have leaders, although they always begin with one.

Process groups allow the system to keep track of which processes are working together and hence should be managed together via job control.

1.4 Właściciel procesu

Użytkownik, który uruchomił dany proces (np 'root' lub 'kowalsky').
Pośrednio określa uprawnienia procesu i do procesu.

1.5 Watek jadra

A kernel thread is a kernel task running only in kernel mode; it usually has not been created by fork() or clone() system calls. An example is kworker or kswapd.

They are same as user space threads in many aspects, but one of the biggest difference is that they exist in the kernel space and execute in a privileged mode and have full access to the kernel data structures. These are basically used to implement background tasks inside the kernel. The task can be handling of asynchronous events or waiting for an event to occur.

1.6 Wskaż, które z wyświetlonych zadań sa watkami jadra

USER: root, PGID: 0 , CMD w kwadratowych nawiasach []

```
USER        PID  PGID   TID PRI STAT WCHAN  CMD
root          1     1     1  19 Ss   -      /sbin/init <- ten nie
root          2     0     2  19 S    -      [kthreadd]
root          3     0     3  39 I<   -      [rcu_gp]
root          4     0     4  39 I<   -      [rcu_par_gp]
root          6     0     6  39 I<   -      [kworker/0:0H-kblockd]
root          8     0     8  39 I<   -      [mm_percpu_wq]
root          9     0     9  19 S    -      [ksoftirqd/0]
```

1.7 Jakie jest znaczenie poszczególnych znaków w kolumnie STAT?

```
PROCESS STATE CODES
   Here are the different values that the s, stat and state output specifiers
   (header "STAT" or "S") will display to describe the state of a process:

           D    uninterruptible sleep (usually IO)
           I    Idle kernel thread
           R    running or runnable (on run queue)
           S    interruptible sleep (waiting for an event to complete)
           T    stopped by job control signal
           t    stopped by debugger during the tracing
           W    paging (not valid since the 2.6.xx kernel)
           X    dead (should never be seen)
           Z    defunct ("zombie") process, terminated but not reaped by its parent

   For BSD formats and when the stat keyword is used, additional characters may
   be displayed:

           <    high-priority (not nice to other users)
           N    low-priority (nice to other users)
           L    has pages locked into memory (for real-time and custom IO)
           s    is a session leader
           l    is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
           +    is in the foreground process group
```

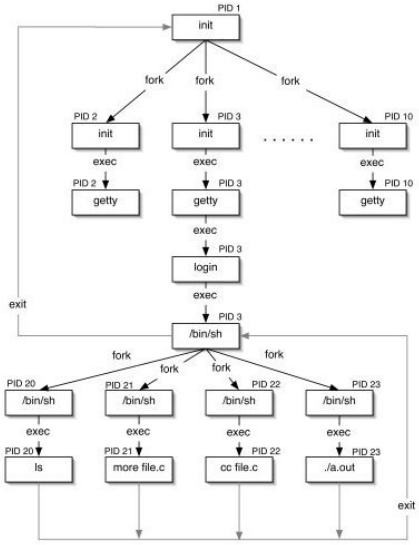
1.8 Kto jest rodzicem procesu init?

t is conventional to say that init has no parent and that, therefore, the PPID value of 0 is a placeholder signaling that it has no parent. Alternatively, one might claim that the kernel is the "parent" of init and that the 0 signifies the kernel.

1.9 Hierarchia procesów

Process Hierarchies

- Parent creates a child process, child processes can create its own process
- Forms a hierarchy
  - UNIX calls this a "process group"
- Windows has no concept of process hierarchy
  - all processes are created equal



1.10 Wyświetl drzewiasta reprezentacje hierarchii procesów poleceniem pstree – które z zadań sa watkami?

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}]

2

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 zawierajacy **argumenty programu oraz zmienne środowiskowe**. Podaj znaczenie nastepujacych pól w pliku

```
stat: State, Groups, VmPeak, VmSize, VmRSS, Threads, voluntary_ctxt_switches,
nonvoluntary_ctxt_switches
```

2.1 Argumenty programu

It is possible to pass some values from the command line to your C programs when they are executed. These values are called command line arguments and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using main() function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.

```
int main( int argc, char **argv ) {
```

2.2 Zmienne środowiskowe

any program that runs on your system has a set of predefined environment variables. Those are global variables defined and passed to the process that runs your program from its parent process (every single process has a parent, except for init, which is the father of all user processes). They define different things that may affect your program's behavior, like the name of the default shell or text editor, the PATH that is searched for binary executables, and so forth.

Actually, the signature more accurately looks like this:

```
int main( int argc, char **argv, char **environ ) {
```

The third parameter is an array of KEY=VALUE strings that is NULL-terminated, i.e. every string consists of a key and a value, like so:

```
LANG=en_US.utf8
```

2.3 proc

Proc file system (procfs) is virtual file system created on fly when system boots and is dissolved at time of system shut down.

It contains the useful information about the processes that are currently running, it is regarded as control and information centre for kernel.

The proc file system also provides communication medium between kernel space and user space.

2.4 Dla wybranego przez siebie procesu o identyfikatorze pid wydrukuj zawartość katalogu /proc/pid.

```
ls -l /proc/9151
-r--r--r-- 1 usr usr 0 10-21 13:10 arch_status
dr-xr-xr-x 2 usr usr 0 10-21 13:10 attr
-rw-r--r-- 1 usr usr 0 10-21 13:10 autogroup
-r-------- 1 usr usr 0 10-21 13:10 auxv
-r--r--r-- 1 usr usr 0 10-21 13:10 cgroup
--w------- 1 usr usr 0 10-21 13:10 clear_refs
-r--r--r-- 1 usr usr 0 10-21 13:10 cmdline
-rw-r--r-- 1 usr usr 0 10-21 13:10 comm
-rw-r--r-- 1 usr usr 0 10-21 13:10 coredump_filter
-r--r--r-- 1 usr usr 0 10-21 13:10 cpuset
lrwxrwxrwx 1 usr usr 0 10-21 13:10 cwd -> /home/usr
-r-------- 1 usr usr 0 10-21 13:10 environ
lrwxrwxrwx 1 usr usr 0 10-21 13:10 exe -> /usr/bin/python3.7
dr-x------ 2 usr usr 0 10-21 13:10 fd
dr-x------ 2 usr usr 0 10-21 13:10 fdinfo
-rw-r--r-- 1 usr usr 0 10-21 13:10 gid_map
-r-------- 1 usr usr 0 10-21 13:10 io
-r--r--r-- 1 usr usr 0 10-21 13:10 latency
-r--r--r-- 1 usr usr 0 10-21 13:10 limits
-rw-r--r-- 1 usr usr 0 10-21 13:10 loginuid
dr-x------ 2 usr usr 0 10-21 13:10 map_files
-r--r--r-- 1 usr usr 0 10-21 13:10 maps
-rw------- 1 usr usr 0 10-21 13:10 mem
-r--r--r-- 1 usr usr 0 10-21 13:10 mountinfo
-r--r--r-- 1 usr usr 0 10-21 13:10 mounts
-r-------- 1 usr usr 0 10-21 13:10 mountstats
dr-xr-xr-x 5 usr usr 0 10-21 13:10 net
dr-x--x--x 2 usr usr 0 10-21 13:10 ns
-r--r--r-- 1 usr usr 0 10-21 13:10 numa_maps
-rw-r--r-- 1 usr usr 0 10-21 13:10 oom_adj
-r--r--r-- 1 usr usr 0 10-21 13:10 oom_score
-rw-r--r-- 1 usr usr 0 10-21 13:10 oom_score_adj
-r-------- 1 usr usr 0 10-21 13:10 pagemap
-r-------- 1 usr usr 0 10-21 13:10 personality
-rw-r--r-- 1 usr usr 0 10-21 13:10 projid_map
lrwxrwxrwx 1 usr usr 0 10-21 13:10 root -> /
-rw-r--r-- 1 usr usr 0 10-21 13:10 sched
-r--r--r-- 1 usr usr 0 10-21 13:10 schedstat
-r--r--r-- 1 usr usr 0 10-21 13:10 sessionid
-rw-r--r-- 1 usr usr 0 10-21 13:10 setgroups
-r--r--r-- 1 usr usr 0 10-21 13:10 smaps
-r--r--r-- 1 usr usr 0 10-21 13:10 smaps_rollup
-r-------- 1 usr usr 0 10-21 13:10 stack
-r--r--r-- 1 usr usr 0 10-21 13:10 stat
-r--r--r-- 1 usr usr 0 10-21 13:10 statm
-r--r--r-- 1 usr usr 0 10-21 13:10 status
```

```
-r-------- 1 usr usr 0 10-21 13:10 syscall
dr-xr-xr-x 3 usr usr 0 10-21 13:10 task
-r--r--r-- 1 usr usr 0 10-21 13:10 timers
-rw-rw-rw- 1 usr usr 0 10-21 13:10 timerslack_ns
-rw-r--r-- 1 usr usr 0 10-21 13:10 uid_map
-r--r--r-- 1 usr usr 0 10-21 13:10 wchan
```

2.5 Wyświetl plik zawierajacy argumenty programu oraz zmienne środowiskowe.

```
~ $ cat /proc/9151/environ
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/cbef9a18_4583_4492_9210_b628e3e8062d
(...)
```

```
~ $ cat /proc/9151/cmdline
python3-b%
```

2.6 Podaj znaczenie nastepujacych pól w pliku status

State: Current state of the process. One of "R (running)", "S (sleeping)", "D (disk sleep)", "T (stopped)", "T (tracing stop)", "Z (zombie)", or "X (dead)".

Groups: Supplementary group list.

VmPeak: Peak virtual memory size.
VmSize: Virtual memory size.
VmRSS: Resident set size.

Threads: Number of threads in process containing this thread.

voluntary_ctxt_switches, nonvoluntary_ctxt_switches: Number of voluntary and involuntary context switches

3

Znajdź pid procesu X-serwera, a nastepnie używajac polecenia pmap wyświetl zawartość jego przestrzeni adresowej. Zidentyfikuj w niej poszczególne zasoby pamieciowe – tj. stos, sterte, **segmenty programu, pamieć anonimowa, pliki odwzorowane w pamieć**. Należy wyjaśnić znaczenie kolumn wydruku!

3.1 Segment kodu

(znany również jako text segment albo po prostu text) – obszar pamieci zawierajacy kod maszynowy przeznaczony do wykonania przez procesor komputera. Segment kodu może być umieszczony w pamieci operacyjnej komputera poprzez załadowanie fragmentu (sekcji w przypadku formatu pliku ELF) pliku wykonywalnego zawierajacego instrukcje maszynowe.

3.2 Pamieć anonimowa

Anonymous memory is a memory mapping with no file or device backing it. This is how programs allocate memory from the operating system for use by things like the stack and heap.

3.3 Pliki odwzorowane w pamieć

System Linux umożliwia odwzorowanie zawartości pliku w pamieci wirtualnej procesu. Zawartość pliku nie jest od razu kopiowana do pamieci operacyjnej, a jedynie dołaczana jako nowy obszar do pamieci wirtualnej procesu. Plik może być wtedy traktowany jak ciagły obszar w pamieci procesu, dostepny poprzez bezpośrednie operacje pobrania i podstawienia wartości. Wszystkie modyfikacje dokonane w pamieci sa później zapisywane w pliku na dysku. Unika sie w ten sposób stosowania funkcji systemowych do operacji na plikach i pośredniego etapu kopiowania danych do podrecznej pamieci buforowej.

Mechanizm odwzorowania pamieci znajduje kilka zastosowań:
a. odwzorowanie plików z kodem wykonywanywalnym w pamieci procesów, które ten kod wykonuja (realizacja funkcji exec()),
b. dynamiczne ładowanie modułów programu i bibliotek dynamicznych,
c. odwzorowanie plików w pamieci procesu w celu ułatwienia i przyspieszenia operacji na ich zawartości,
d. współdzielenie przez wiele procesów obszarów pamieci, których zawartość przechowywana jest w odwzorowanych plikach niezależnie od tworzenia i kończenia procesów.

3.4 Znajdź pid procesu X-serwera

```
~ $ ps -eo user,pid,pgid,tid,pri,stat,wchan,cmd | grep "Xorg"
root       693   693   693  19 Ssl+ -      /usr/lib/Xorg
```

3.5 a nastepnie używajac polecenia pmap wyświetl zawartość jego przestrzeni adresowej

```
sudo pmap -x 693 // -X == nawet wincyj informacji, przy prezentacji preferowane(?)
Adres                  KB     RSS  Brudne Tryb  Odwzorowanie
000055b05c323000      180     180       0 r---- Xorg
000055b05c350000     1528    1516       0 r-x-- Xorg
000055b05c4ce000      448     292       0 r---- Xorg
000055b05c53e000       16      16      16 r---- Xorg
000055b05c542000       24      24      24 rw--- Xorg
000055b05c548000      128      56      56 rw---   [ anon ]
000055b05d0be000    10436   10136   10136 rw---   [ anon ]

(...)

00007f00a1917000       12      12      12 r---- libgobject-2.0.so.0.6200.1
00007f00a191a000        4       4       4 rw--- libgobject-2.0.so.0.6200.1
00007f00a191b000       16      16       0 r---- libgudev-1.0.so.0.2.0
00007f00a191f000       16      16       0 r-x-- libgudev-1.0.so.0.2.0
00007f00a1923000        8       0       0 r---- libgudev-1.0.so.0.2.0
00007f00a1925000        4       0       0 ----- libgudev-1.0.so.0.2.0
00007f00a1926000        4       4       4 r---- libgudev-1.0.so.0.2.0
00007f00a1927000        4       4       4 rw--- libgudev-1.0.so.0.2.0
00007f00a1928000       44      44       0 r---- libevdev.so.2.3.0
00007f00a1933000       24      24       0 r-x-- libevdev.so.2.3.0
00007f00a1939000       20      20       0 r---- libevdev.so.2.3.0
00007f00a193e000       24      24      24 r---- libevdev.so.2.3.0
00007f00a1944000        4       4       4 rw--- libevdev.so.2.3.0
00007f00a1945000       20      16       0 r-x-- libmtdev.so.1.0.0
00007f00a194a000     2044       0       0 ----- libmtdev.so.1.0.0
00007f00a1b49000        4       4       4 r---- libmtdev.so.1.0.0
00007f00a1b4a000        4       4       4 rw--- libmtdev.so.1.0.0
00007f00a1b4b000       36      36       0 r---- libinput.so.10.13.0
00007f00a1b54000      172     172       0 r-x-- libinput.so.10.13.0

(...)

00007f00a6e70000        4       0       0 rw-s- i915 (deleted)
00007f00a6e71000        4       4       4 r---- ld-2.30.so
00007f00a6e72000        4       4       4 rw--- ld-2.30.so
00007f00a6e73000        4       4       4 rw---   [ anon ]
00007fff74489000      132      44      44 rw---   [ stos ]
00007fff74590000       12       0       0 r----   [ anon ]
00007fff74593000        4       4       0 r-x--   [ anon ]
ffffffffff600000        4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
razem kB          531808   73864   56952
```

4

Używajac programu lsof wyświetl zasoby plikopodobne podpiete do procesu przegladarki ff. Wyjaśnij znaczenie poszczególnych kolumn wykazu, po czym zidentyfikuj pliki zwykłe, katalogi, urzadzenia, 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 połaczenia sieciowe?

4.1 Zasób plikopodobny

Czyli deskryptor pliku. Czytamy/piszemy do zasobu jakby to był plik ale może to być też inny zasób jak np. stdin/stdout.

4.2 Potoki

Potok (pipe) łaczy stdout jednego procesu z stdin drugiego procesu. Potok może łaczyć ze soba wiele procesów (tzw. pipeline).

Przyklad: otwórz plik 1, wytnij tylko pierwsza linie, połacz pierwsza linie pliku 1 z plikiem 2

```
cat PLIK1 | head -n 1 | cat PLIK2
```

4.3 Używajac programu lsof wyświetl zasoby plikopodobne podpiete do procesu przegladarki ff. Wyjaśnij znaczenie poszczególnych kolumn wykazu, po czym zidentyfikuj pliki zwykłe, katalogi, urzadzenia, gniazda (sieciowe lub domeny uniksowej) i potoki.

By default One file per line is displayed. Most of the columns are self explanatory. We will explain the details about couple of cryptic columns (FD and TYPE).

FD – Represents the file descriptor. Some of the values of FDs are,

cwd – Current Working Directory
txt – Text file
mem – Memory mapped file
mmap – Memory mapped device
NUMBER – Represent the actual file descriptor. The character after the number i.e '1u', represents the mode in which the file is opened. r for read, w for write, u for read and write.

TYPE – Specifies the type of the file. Some of the values of TYPEs are
REG – Regular File
DIR – Directory
FIFO – First In First Out
CHR – Character special file

```
lsof -c ff
COMMAND   PID    USER   FD      TYPE             DEVICE  SIZE/OFF    NODE NAME
ff 12882 usr  cwd       DIR                8,6      4096  262658 /home/usr
ff 12882 usr  rtd       DIR                8,6      4096       2 /
ff 12882 usr  txt       REG                8,6    411968 1188276 /usr/lib/ff/ff
ff 12882 usr  mem       REG                8,6   4729512  530101 /home/usr/.cache/mozilla/
ff 12882 usr  DEL       REG               0,25                75 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                11 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                85 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                73 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                91 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                10 /dev/shm/org.mozilla.ipc.
ff 12882 usr  DEL       REG               0,25                15 /dev/shm/org.chromium.dCb
ff 12882 usr  DEL       REG                0,1            131098 /SYSV00000000
ff 12882 usr  DEL       REG                0,1            131095 /SYSV00000000
ff 12882 usr  mem       REG                8,6    307072  809456

(...)
```

4.4 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 połaczenia sieciowe?

```
 diff -u before.txt after.txt | grep "IPv4"
-ff 12882 usr   37u     IPv4             172036       0t0     TCP S410UA:dai-shell->ec2-52
+ff 12882 usr   37u     IPv4              92491       0t0     TCP S410UA:59764->ec2-35-155
-ff 12882 usr   56u     IPv4              89675       0t0     TCP S410UA:41590->server-99-
 ff 12882 usr   58u     IPv4              89674       0t0     TCP S410UA:33048->server-13-
-ff 12882 usr   59u     IPv4             172037       0t0     TCP S410UA:45826->ec2-52-27-
-ff 12882 usr   60u     IPv4             172038       0t0     TCP S410UA:37502->93.184.220
+ff 12882 usr   59u     IPv4              92492       0t0     TCP S410UA:37564->93.184.220
+ff 12882 usr   60u     IPv4              92470       0t0     TCP S410UA:39110->waw02s17-i
-ff 12882 usr   62u     IPv4             172039       0t0     TCP S410UA:37504->93.184.220
+ff 12882 usr   62u     IPv4              90492       0t0     TCP S410UA:45078->lo-in-f155
+ff 12882 usr   98u     IPv4              91428       0t0     TCP S410UA:35918->muc03s08-i
+ff 12882 usr  100u     IPv4              91430       0t0     TCP S410UA:44878->waw02s07-i
+ff 12882 usr  101u     IPv4              91431       0t0     TCP S410UA:44880->waw02s07-i
+ff 12882 usr  116u     IPv4              91432       0t0     TCP S410UA:35926->muc03s08-i
+ff 12882 usr  119u     IPv4              91433       0t0     TCP S410UA:44884->waw02s07-i
+ff 12882 usr  125u     IPv4             171838       0t0     TCP S410UA:57794->waw02s13-i
+ff 12882 usr  127u     IPv4             171839       0t0     TCP S410UA:52286->vip0x00f.m
 ff 12882 usr  128u     IPv4              88389       0t0     TCP S410UA:40358->ec2-52-36-
+ff 12882 usr  137u     IPv4             155407       0t0     TCP S410UA:44834->muc03s08-i
+ff 12882 usr  138u     IPv4              92457       0t0     TCP S410UA:49070->vip0x018.m
+ff 12882 usr  149u     IPv4              92469       0t0     TCP S410UA:44912->waw02s07-i
+ff 12882 usr  155u     IPv4              88804       0t0     TCP S410UA:44914->waw02s07-i
```

5

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

5.1

```
time lsof > /dev/null

real 0m31,532s
user 0m0,533s
sys 0m0,329s
```

5.2 Czemu suma czasów user i sys nie jest równa real

Polecenie time zwraca nam trzy różne czasy: real, user a także czas sys. Aby móc cokolwiek powiedzieć na temat czasu pracy programu, należy potrafić interpretować te wyniki. Otóż czasy te oznaczaja:
Real - czas jaki musieliśmy odczekać od momentu właczenia procesu, do momenty zakończenia go (/zobaczenia jego wyników)
User - czas poświecony na prace procesora "wewnatrz procesu" (tzw. user-mode)
Sys - czas poświecony przez procesor na komunikacje z jadrem systemu (praca "na zewnatrz procesu")

Jak teraz nietrudno sie domyślić, suma czasów user oraz sys, da nam całkowity czas pracy procesora nad danym programem. Dlaczego wiec rzeczywisty czas jaki musieliśmy odczekać różni sie od tego, jaki potrzebował procesor? Odpowiedź jest prosta. Gdyby procesor zajmował sie tylko i wyłacznie tym jednym programem to czas real = user + sys, ale tak nie jest. Procesor musiał obsłużyć wiele innych działajacych procesów, dlatego jego praca była wykolejkowana i czas real uwzglednia również czas oczekiwania w kolejce procesów.

5.3 Czemu suma czasów user i sys może być wieksza od real?

I suppose if the work was done by several processors concurrently, the CPU time would be higher than the elapsed wall clock time.

5.4 Poleceniem ulimit nałóż ograniczenie na czas wykonania procesów potomnych powłoki tak, by limit sie wyczerpał. Uruchom ponownie wybrany program – który sygnał wysłano do procesu?

```
 ~ $ ulimit -a
-t: cpu time (seconds)              unlimited
-f: file size (blocks)              unlimited
-d: data seg size (kbytes)          unlimited
-s: stack size (kbytes)             8192
-c: core file size (blocks)         unlimited
-m: resident set size (kbytes)      unlimited
-u: processes                       31316
-n: file descriptors                1024
-l: locked-in-memory size (kbytes)  64
-v: address space (kbytes)          unlimited
-x: file locks                      unlimited
-i: pending signals                 31316
-q: bytes in POSIX msg queues       819200
-e: max nice                        0
-r: max rt priority                 0
-N 15:                              unlimited

~ $ ulimit -t 1
~ $ ulimit -a
-t: cpu time (seconds)              1


~ $ find /usr > /dev/null
[1]    15877 cpu limit exceeded (core dumped)  find /usr > /dev/null
```

–t Set or display the cpu time limit. The cpu time limit is the maximum amount of CPU time (in seconds) allowed for the process. If the limit is exceeded, a SIGXCPU signal is sent to the process and the process is granted a small CPU time extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a SIGKILL signal. An attempt to set the CPU limit lower than that already used will fail.