SOWID – PROCESY, SYGNAŁY, DAEMONY

PAWEŁ JÓZIAK
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH
POLITECHNIKI WARSZAWSKIEJ

- Nigdy nie wykonuj poleceń metodą *copy&paste*. Przepisz je ręcznie!
- Nie redukuj pracy nad zadaniem do przeczytania go. Wykonaj je i upewnij się, że rozumiesz jak i dlaczego wynik jest taki, jak na ekranie. Zajrzyj do prezentacji bądź zapytaj prowadzącego, jeśli coś nie będzie jasne.
- Linia zaczynająca się od $ reprezentuje prompt zwykłego użytkownika (nie przepisuj go!). Komendą jest to, co następuje po nim.
- Ostrożnie ze spacjami: nie dodawaj ich więcej ani nie usuwaj takowych, jeśli są!

1. Procesy

(a) Odpal wpis manuala o zapytaniu systemowym `fork()`[1]. Warto czytać strony POSIX-owe!

(b) Ile procesów będzie utworzonych przy wywołaniu:
`for (i=0;i<4;i++) fork();`
Nie, '4' to nie jest dobra odpowiedź!

(c) Sprawdźmy to eksperymentalnie. Otwórz *geany* i zapisz poniższy kod:

```
#include<unistd.h>
#include<stdio.h>
int main()
{
  for (int i=0; i<4; i++) fork();
  printf ("!");
  return 0;
}
```

Zapisz ten kod jako plik C. Kliknij na ikonę *cegły*, by skompilować, i odpal program ikoną *koła zębatego* (albo z terminala). Czy Twoja odpowiedź była poprawna? Jeśli nie – spróbuj przeanalizować kod raz jeszcze.

(d) Czy istnieje zapytanie systemowe `'exec'`?[2]

(e) Skompiluj poniższy kod:

```
#include<unistd.h>
int main()
{
  execl("/usr/bin/ls", "ls", "/etc", (char*)0);
  return 1;
}
```

Jaki jest jego status wyjścia?[3] Dlaczego?[4]

(f) Wyświetl drzewo procesów z PIDami[5]. Pamiętaj, numery w nawiasach klamrowych to numery wątków, nie procesów!

---

[1] `$ man 3p fork`

[2] `$ man 3p exec` – zatem nie do końca. Ale różnica jest czysto formalna

[3] Można sprawdzić:
`$ ./program && echo success`
Ponieważ wydrukowany jest `success` po `&&`, kod wyjścia jest sukcesem (0).

[4] Bo zapytanie systemowe `'exec()'` nadpisuje **cały** kod procesu

[5] `$ pstree -p`

(g) Wypisz wszystkie procesy bieżącej sesji[6]. Jaki jest PID rodzica bieżącej sesji `bash`a?
(h) Wypisz *wszystkie* procesy[7] Nazwy w nawiasach kwadratowych wątki jądra.
(i) A teraz to samo polecenie w syntaksie BSD[8]. Nie używaj myślnika przed opcjami!

Domyślna powłoka kont studentów to ***mini****. W naszych stacjach roboczych to symlink wskazujący na* ***bash****a. Proces przechowuje tylko nazwę programu, który jest wykonany, w naszym przypadku –* ***mini****.*

(j) Jakie procesy wykonują `'bash'`?[9]
(k) Odpal `'top'`. Posortuj procesy według zużycia pamięci, a potem wg zużycia CPU[10].
(l) Poczytaj podręcznik o *procfs*[11]. Sprawdź pliki jednego z procesów na Twoim komputerze.
(m) Uruchom oraz prześledź proces aplikacji `'gvim'`[12].
Niestety, domyślnie śledzony jest tylko główny proces, nie procesy-dzieci (a w przypadku `gvim`a, proces główny uruchamia jedynie procesy-dzieci i sam szybko się kończy).
Uruchom śledzenie procesu wraz jego procesami-dziećmi[13].
Jeśli nie potrafisz czytać dość szybko, przekieruj output do pliku[14].
Czy widzisz w outpucue numer grupy procesów?
Gdy zamkniesz `'gvim'`, sprawdź ile zapytań systemowych zostało wykonanych[15]. Sprawdź, jakie pliki z `/etc`, i w jakiej kolejności, został odpalone przez `gvim`[16].
Sprawdź, jakie ikony zostały użyte[17]. W końcu, agreguj logi procesów do osobnych plików[18].
Ile różnych procesów zostało odpalonych?[19]
(n) Sprawdź, że jedyne, co robi `'pstree'` to sprawne parsowanie zawartości katalogu `/proc`[20].
Powtórz ćwiczenie dla `top`, `pidof`, `ps`, ...

2. Sygnały

(a) Odpal stronę manuala dla sygnałów[21].
Jakie są domyślne działania?[22]
Jakie jest najbardziej typowe działanie?[23]
Jakie sygnały zostały wprowadzone przez POSIX.1-1990?[24]
Jaki sygnał jest jedynym o domyślnej akcji *ignore*?[25]

---

[6] `$ ps -f`
[7] `$ ps -ef`
[8] `$ ps aux`
[9] `$ pidof mini`
albo
`$ pgrep mini`
albo
`$ ps -ef | grep mini`
albo raczej
`$ ps -ef | grep mini | grep -v grep`
[10] używając `<` oraz `>` zmieniasz kolumnę sortowania
[11] `$ man 5 proc`
[12] `$ strace gvim`
[13] `$ strace -f gvim`
[14] `$ strave -f -o gvim.log gvim`
[15] `$ wc -l gvim.log`
[16] `$ grep open gvim.log | grep /etc`
[17] `$ grep open gvim.png | grep png`
[18] `$ strace -ff -o gvim.log gvim`
[19] `$ ls gvim.log* | wc -l`
[20] `$ pstrace -f -o pstree.log pstree`
`$ grep open pstree.log`
[21] `$ man 7 signal`
[22] *ignore*, *terminate*, *terminate with core dump*, *stop*, *continue*
[23] *terminate*
[24] `SIGABRT`, `SIGALRM`, `SIGCHLD`, `SIGCONT`, `SIGFPE`, `SIGHUP`, `SIGILL`, `SIGINT`, `SIGKILL`, `SIGPIPE`, `SIGQUIT`, `SIGSEGV`, `SIGSTOP`, `SIGTSTP`, `SIGTERM`, `SIGTTIN`, `SIGTTOU`, `SIGUSR1`, `SIGUSR2`
[25] `SIGCHLD`

Jaki sygnał jest wysyłany przy niewłaściwym zarządzaniu pamięcią?[26]
Co dzieje się w przypadku dzielenia przez zero?[27]
Jaka liczba odpowiada `'SIGKILL'`?[28] Zapamiętaj tę liczbę.
Które sygnały nie mogą być przechwycone, ignorowane ani zablokowane?[29]

(b) Wystartuj `'top'`. Zamknij go przez wysłanie sygnału `SIGINT` sekwencją kontrolną.[30]
(c) Wystartuj `'top'`. Zamknij go przez wysłanie sygnału `SIGINT` zapytaniem systemowym.[31]
(d) Wystartuj `'top'`. Wyślij doń sygnał `SIGFPE` zapytaniem systemowym.[32]. Jaki był efekt dla działania `top`?
(e) Wystartuj `'vim'` w terminalu. Otwórz drugi terminal, by wysłać do `vim`a sygnał `SIGTERM`[33]. Powtórz to samo, emitując tym razem sygnał `SIGKILL`[34].
Niestety, z błędu emulatora terminala, używanie scrolla może generować śmieci w oknie, w którym działał `vim`. Nie ma to nic wspólnego z sygnałami, to błąd.

Na Linuxach systemd (jak na naszych stacjach roboczych) kolejny punkt może doprowadzić do awarii w dbus (cokolwiek to jest), przez co system może być niestabilny. W razie problemów skontaktuj się z prowadzącym.

(f) Jeśli sesja GUI zawiesi się, możesz ją zakończyć przez przejście na inny TTY[35] i wykonanie:
`$ killall -u $(id -un)`
bądź, jeśli to konieczne,
`$ killall -9u $(id -un)`
Pamiętaj, kończenie procesów przez `'SIGKILL'` może prowadzić do utraty danych.
(g) Wywołaj `'cat'` tak, aby czytał ze *stdin* w terminalu:[36] Wpisz jakiś tekst, potwierdź klawiszem <ENTER>. Powtórz ten krok kilkukrotnie. Naciśnj sekwencję <CTRL>+ <D>. To nie sygnał! To tylko znak `EOF`, końca pliku, napisany do *stdin*.
(h) Odpal `'thunderbird'` w terminalu[37]. Zabij go za pomocą sekwencji kontrolnej terminala[38]. Teraz odpal `'thunderbird'` jako proces tła.[39]. Wykonaj tę samą sekwencję kontrolną terminala – nic się nie dzieje. Jak to sprawdzić?[40] Jak wysłać `SIGINT` do tego procesu?[41]
(i) Raz jeszcze odpal `'thunderbird'` w terminalu[42] i zastopuj ten proces za pomocą sekwencji kontrolnej terminala[43]. Przywróć go do działania jako proces tła bieżącej sesji[44]. Wyświetl jego numer grupy procesów[45]. Przesuń go na czoło[46] i zamknij generując sekwencją kontrolną `SIGINT`[47]

---

[26] `SIGSEGV`, w standardzie POSIX-2001 też `SIGBUS`
[27] `SIGFPE`
[28] To 9.
[29] `SIGKILL` oraz `SIGSTOP`
[30] <CTRL>+ <C>
[31] `$ kill -SIGINT $(pidof top)`
[32] `$ kill -SIGFPE $(pidof top)`
[33] `$ kill $(pidof vim)`
[34] `$ kill -SIGKILL $(pidof vim)`
albo
`$ kill -9 $(pidof vim)`
[35] <CTRL>+ <ALT>+ <F2> dla `tty2`
[36] `$ cat`
[37] `$ thunderbird`
[38] <CTRL>+ <C>
[39] `$ thunderbird &`
[40] `$ pidof thunderbird`
[41] `$ kill -SIGINT $(pidof thunderbird)`
[42] `$ thunderbird`
[43] <CTRL>+ <Z>
[44] `$ bg`
[45] `$ jobs`
[46] `$ fg`
[47] <CTRL>+ <C>

(j) Otwórz `nano` w terminalu, napisz w nim pojedynczą literę, wyślij do tego procesu `SIGSTOP` sekwencją kontrolującą[48]. Powtórz kilkukrotnie, pisząc za każdym razem nową literę (`'b'`, `'c'`, `'d'`, `'e'`, `'f'`, `'g'`). Wyświetl dostępne procesy w tle[49]. Przywołaj na czoło domyślny[50]. Który proces się przywołał? Zastopuj ten proces ponwnie. Przywołaj ten, w którym napisane zostało `'b'`. Zastopuj go. Raz jeszcze przywołaj na czoło domyślny[51]. Który proces się przywołał? Zastopuj ten proces ponwnie. Przywołaj ten, w którym napisane zostało `'c'`. Zamknij go. Raz jeszcze przywołaj na czoło domyślny[52]. Który proces się przywołał? Eksperymentuj dalej.

3. Daemony i systemd

(a) Wyświetl wszystkie jednostki serwisów[53].
(b) Wyświetl plik serwisu dla daemona sshd[54]. Czy jest on aktywny[55]
(c) Wyświetl listę wszystkich aktywnych jednostek[56]
(d) Wyświetl logi `'journalctl'`[57]. Cóż, `'journald'` pisze wszystkie wpisy do pojedynczego, binarnego, pliku lokalnego. Ogromny plik? Niewydajne parsowanie? Rejestrowanie wrażliwych danych w pliku dostępnym dla każdego użytkownika? Brak backupu i w razie ataku cały log znika? Maksymalna wielkość jest z góry określona, ale brak możliwości zarządzania, co jest częstym szumem i powinny znikać szybciej, a co jest istotne i powinno zostać dłużej? Brak backupu dla istotnych starych wpisów? Potencjał na to, by plik binarny stał się nieparsowalny w wyniku awarii systemu? Konieczność używania dedykowanych narzędzi do czytania? Brak możliwości wyłączenia? Na to wszystko jedna odpowiedź: `systemd`.

4. Cron i syslog

(a) Wyświetl plik *crontab*a[58].
(b) Wyświetl zawartość *crontab*a użytkownika dedykowanym narzędziem[59]. Czy te dwa wyniki się różnią?
(c) Zajrzyj do plików w katalogu *crontab*a[60].
(d) Zajrzyj do dokumentacji `cron`a. W jakich sekcjach podręcznika jej szukać?[61]reprezentujące
(e) Czy istnieją zadania do wykonania codziennie, co godzinę, co tydzień?[62]? Jak często wywoływany jest serwer `ssh`?
(f) Wyślij wiadomość na swój `'syslog'`[63]. Daemon sysloga jest skonfigurowany, by pisać do wszystkie komunikaty logów na `tty12` – jak je obejrzeć?[64]

---

[48] <CTRL>+ <Z>
[49] `$ jobs`
[50] `$ fg`
[51] `$ fg`
[52] `$ fg`
[53] `$ ls /usr/lib/systemd/system`
[54] `$ cat /usr/lib/systemd/system/sshd.service`
[55] `$ systemctl status sshd`
[56] `$ systemctl list-unit-files -all`
[57] `$ journalctl`
[58] `$ cat /etc/crontab`
[59] `$ crontab -l`
[60] `$ cat /etc/cron.d/*`
[61] `$ man cron`
`$ man 5 crontab`
`$ man 1 crontab`
[62] `$ ls /etc/cron.hourly/ /etc/cron.daily/ /etc/cron.weekly/`
[63] `$ logger "moja wiadomość"`
[64] <CTRL>+ <F12>