Śledzenie wykonujących się procesów

Jakub Szczyrk
235477

Poniedzałek 13:15-14:00

8 grudnia 2018

Spis treści

1 Zadanie 1 narzędzie truss na Solaris

Truss to narzędzie wykonujące określone polecenie oraz generujące ślad wywołań systemowych, sygnałów i błędów urządzenia. Każda linia danych wyjściowych śledzenia, zgłasza nazwę błędu lub sygnału lub nazwę wywołania systemowego wraz z argumentami i wartościami zwracanymi. Argumenty wywołania systemowego są wyświetlane symbolicznie, gdy jest to możliwe, za pomocą definicji z odpowiednich nagłówków systemu.

Przełączniki:

- -u śledzenie wywołania funkcji użytkownika, biblitek oraz moliwość ignorowania ich.
- -s śledzenia lub wykluczenia sygnałów.
- -t śledzenia lub wykluczania wywołań systemowych.
- -o zwraca plik z wyjścia śledzenia.
- -w pokazuje zawartość bufora wejścia / wyjścia dla każdego zapisu.
- -r wyświetla pełną zawartość bufora wejśćia / wyjścia dla każdego odczytu.

1.1 Program truss - analiza wykonania prostego programu wypisującego na ekran jakiś tekst.

Program:

```
#include <unistd.h>
#include <stdio.h>

int main()
{
                sleep(3);

                printf("Zadanie 7\n");
return 0;
}
```

Wywołanie programu:

```
truss ./text
```

Analiza wyjścia truss:

```
execve("text", 0xFFBFFD3C, 0xFFBFFD44)  argc = 1
```

execve() uruchamia program wskazany przez filename - pierwszy parametr funkcji. Drugi parametr to argv, tablica łańcuchów przekazywanych jako argumenty nowego programu. Trzeci parametr to envp, to tablica wskaźników do zmiennych środowiskowych. Tablica envp i argv kończy się pustym wskaźnikiem. Tablica argumentów oraz środowisko są dostępne w funkcji main wywoływanego programu, jeżeli jest ona zdefiniowana jako int main(int argc, char *argv[], char *envp[]).

```
sysinfo(SI_MACHINE, "sun4u", 257)                    = 6
```

sysinfo() zwraca informacje statystyczne dotyczące całego systemu

mmap(0x00000000, 32, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1, 0) = 0xFF3E0000

Funkcja mmap() zleca zamapowanie do pamięci, najchętniej pod adres startowy, w tym przypadku 0x00000000, ilość bajtów pliku - drugi parametr funkcji. Piąty parametr deskryptor "fd" zadaje co mależy zapisać. Szósty parametr odowiada za przesunięcie względem początku o zadany "offset". Rzeczywiste miejsce zamapowania obiektu jest zwracane przez mmap i nigdy nie jest to zero. Argument trzeci (prot) opisuje oczekiwany sposów ochrony pamięci:

- PROT_EXEC Strony mogą być wykonywane.
- PROT_READ Strony mogą być odczytywane.
- PROT_WRITE Strony mogą być zapisywane.
- PROT_NONE Strony nie mogą być dostępne.

Patametr czwarty (flags) określa rodzaj mapowanego obiektu:

- MAP_FIXED polecenie nie wybierania innego adresu niż podany.
- MAP_SHARED polecenie współdzielenia mapowania ze wszystkimi innymi procesami, które mapują ten obiekt.
- MAP_PRIVATE polecenie utworzenia prywatnego mapowania. Zapisywane dane nie będzie wpływać na zawartość oryginalnego pliku.
- MAP_NORESERVE polecenie nie rezerwuje przestrzeni wymiany. Gdy przestrzeń wymiany jest nie zarezerwowana, ma się gwarancję, że nie istnieje możliwość modyfikacji otginału.
- MAP_ANONYMOUS/MAP_ANON argumenty fd i offset zostaną zignorowane.

memcntl(0x00010000, 1908, MC_ADVISE, MADV_WILLNEED, 0, 0) = 0

Funkcja memcntl umożliwia procesowi wywołującemu stosowanie różnych operacji kontrolnych nad przestrzenią adresową identyfikowaną przez odwzorowania ustalone dla zakresu adresów - pierwszy parametr to poącztek, a drugi to rozmiar.

resolvepath("/usr/lib/ld.so.1", "/lib/ld.so.1", 1023) = 12

Funkcja resolvepath() umieszcza wynikową nazwę ścieżki w buforze nazwy ścieżki bez dowiązań symbolicznych, który ma rozmiar 1023.

getcwd("/home/jszczyrk/scr/lab7", 1018) = 0

Funkcja getwcd() odczytuje bierzący katalog roboczy

stat64("/home/jszczyrk/scr/lab7/text", 0xFFBFF800) = 0

Funkcaj stat64() pobiera informacje o pliku wskazanym w pierwszym parametrze.

open("/var/ld/ld.config", O_RDONLY) = 3

Funkcaj open() otwiera podany plik w pierwszym parametrze. Drugi parametr odpowiada za ochrone danych np. O_RDONLY tylko do odczytu.

close(3) = 0

Funkcja close() zamyka deskryptor pliku, podany w parametrze.

munmap(0xFF332000, 65536) = 0

Funkcja munmap() usuwa mapowanie z podanego adresu.

getcontext(0xFFBFF670)

Funkcja getcontext() zapoczątkowuje podny kontekst do aktualnego otawrtego kontekstu

getrlimit(RLIMIT_STACK, 0xFFBFF650) = 0

Funkcja getrlimit() pobiera limit zasobów.

getpid() = 17410 [17409]

Funkcja getpid() pobiera identyfikatora procesu.

setustack(0xFF352A88)

Funkcja setustack() zmienia granice stosu wątku na wartość podaną w parametrze.

nanosleep(0xFFBFFC70, 0xFFBFFC68) (sleeping...)

Funkcja nanosleep() opóźnia wykonywanie programu przynajmniej o czas podany w pierwszym argumencie.

ioctl(1, TCGETA, 0xFFBFFAAC) = 0

Zadanie 7

Wyświetlenie printf().

write(1, "␣Z␣a␣d␣a␣n␣i␣e␣␣␣7\n", 10) = 10

Funkcja Write() umożliwia komunikowanie się z innymi użytkownikami poprzez kopiowanie linii z terminala na ich.

_exit(0)

Funkcja _exit() kończy bieżący proces.

1.2 Identyfikacja oraz zlokalizowanie wszystkich plików konfiguracyjnych, które powłoka bash próbuje odczytać przy starcie.

Za pomocą komendy:

```
truss -t open bash
```

dostajemy odpowiedz o wszytkich odczytach i otwarciach plików w "bash". Próbę odczytu pliku konfgiuracyjnego widzimy w tej linice:

open64("/etc/bash.bashrc", O_RDONLY) **Err#2 *ENOENT***

Jednakże widać, że zwrócił nam błąd, który mówi, że plik nie istnieje albo podana ścieżka jest błędna.

1.3 Zliczenie ilość wywołań funkcji printf, które wykonuje program ls.

Za pomocą komendy:

```
truss -c -t write ls
```

Podczas kompilacji funkcja printf() zamienia się w funkcję write(). Jak widać pod rubryką "calls" dostajemy wynik ile było użycia funkcji write() - osiem.

```
-bash-3.2$ truss -c -t write ls
      syscall         seconds    calls   errors
write                    .000        8
                    --------  ------   ----
sys totals:              .000        8        0
```

1.4 Prześledzenie w jaki sposób edytor tekstu emacs postępuje z edytowanym plikiem.

Po otwarciu jakiegoś pliku tekstowego za pomocą programu emacs, rozpoczynamy śledznie go za pomocą funkcji

```
truss -p [PID]
```

Plik ten jest otwarty przez cały czas działania edytora, wykazuje to śledznie go. Wszelekie zmiany tekstu, ale także cursor'a są odrazu wyświetlane.

2 Zadanie 2 narzędzie strace na Linuksie

Narzędzie strace jest niemal identyczny do truss, wywołania na Linuksie są bardzo podobne do tych na Solarisie.

- strace [program] można załóważyć zmiane w uruchamianych funkcjach, np.

  ```
  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
                                          (No such file or directory)
  ```

  Funkcja access() określa czy dostęp do pliku jest możliwy,pomyślnie kończy pracę, jeżeli plik jest dostępny z podanymi prawami.

- strace -e open bash

  ```
  open("/etc/bash.bashrc", O_RDONLY)      = 3
  ```

  Jak widać bash odczytał plik konfiguracyjny i zwrócił nowy deskryptor pliku.

- strace -c -e write ls można załóważyć zmiane w ilość urzycia funkcji printf() - cztery

% time	seconds	usecs/call	calls	errors	syscall
0.00	0.000000	0	4		write
100.00	0.000000		4		total

- Strace -p [pid]

Oprócz tego, zauważalną zmianom ulega parametr wywołania polecenia "-t" na "-e".