Systemy operacyjne

Warsztaty 2

Studenci są zachęcani do przeprowadzania dodatkowych eksperymentów związanych z treścią zadań i dzieleniem się obserwacjami z resztą grupy. Proszę najpierw korzystać z podręcznika systemowego (polecenia `man` i `apropos`), a dopiero potem szukać niezbędnych wyjaśnień w Internecie. Za każde zadanie można otrzymać co najwyżej `1` punkt.

W wykonaniu pierwszych dwóch zadań może pomóc dokument pt. Linux Performance and Tuning Guidelines oraz dokumentacja wirtualnego systemu plików `proc(5)`.

Ćwiczenie 1

- Wyświetl krótką informację wolnej / zajętej pamięci (`free -hm`) i zinterpretuj ją. W szczególności, co oznacza wiersz drugi wydruku?
- Wyświetl informację o pamięci wirtualnej (`vmstat -s -SM`). Co oznaczają wiersze:
  - `active / inactive memory`,
  - `pages paged in / out`,
  - `pages swapped in / out`?
- Jakie dodatkowe informacje znajdują się w pliku `/proc/meminfo`?

Ćwiczenie 2

Używając polecenia "`sysctl -a |& grep ^vm | sort`" wydrukuj wszystkie zmienne jądra Linux dotyczące zarządzania pamięcią wirtualną. Opis tych zmiennych możesz znaleźć tutaj[1]. Do czego służą zmienne: `swappiness`, `dirty_ratio`, `dirty_background_ratio` i `overcommit_memory`?

Ćwiczenie 3

Użyj polecenia: "`grep ^Vm /proc/$$/status`", aby wydrukować statystyki dot. przestrzeni adresowej bieżącego procesu. Przeprowadź podobny eksperyment na innych należących do Ciebie procesach (`PID` znajdź przy pomocy "`ps -f -u $USER`"). Co opisują pola `VmPeak` i `VmSize`? Jakie jest rzeczywiste zużycie pamięci? Jakiej wielkości są segmenty danych, kodu i stosu? Ile pamięci zużywa Linux na tablicę stron opisujących przestrzeń adresową procesu?

Ćwiczenie 4

Jak było wspomniane na ćwiczeniach, potrzebny jest pewien mechanizm przydziału pamięci na użytek bibliotecznego allokatora `malloc` / `free`. Napisz krótki program w języku C, który pozwoli Ci zaprezentować przy pomocy `strace`, które wywołanie systemowe wykorzystywane są do zwiększenia ilości pamięci na użytek procesu. Wykonaj ten eksperyment osobno dla małych (poniżej `1024B`) i większych (powyżej `1024B`) bloków. Powinniście uzyskać różne wyniki.

Ćwiczenie 5

Narzędzie Valgrind jest niesamowicie pomocne w procesie tworzenia i testowania oprogramowania. W tym zadaniu poznamy *Massif* , jedno z narzędzi tego pakietu. Otwórz następujący dokument: Massif: a heap profiler. Użyj programu z punktu `9.2.1`, aby wykonać samodzielnie punkty od `9.2.2` do `9.2.6`. Jak takie narzędzie może pomóc w testowaniu aplikacji?

---

[1] Plik tekstowy `Documentation/sysctl/vm.txt` w źródłach jądra Linuksa.

Ćwiczenie 6

Zapoznaj się z opisem formatu `ELF`, następnie skompiluj poniższy program:

```
#include <stdio.h>

int big_big_array[10*1024*1024];
char *a_string = "Hello, World!";
int a_var_with_value = 0x100;
const int read_only_var = 0x77;

int main(void) {
  big_big_array[0] = 100;
  printf("%s\n", a_string);
  printf("%d\n", read_only_var);
  a_var_with_value += 20;
}
```

Wylistuj (poleceniem `readelf -a` i `objdump -xs`) wszystkie informacje o pliku wynikowym. Wyjaśnij znaczenie pól w tablicy symboli `.symtab`. W jakich sekcjach znajdą się zadeklarowane w programie zmienne? Co o tych sekcjach mówią nagłówki sekcji? W jaki sposób sekcje te zostaną przypisane do segmentów?

Ćwiczenie 7

Zidentyfikuj poleceniem `file` dwa pliki wykonywalne (zwróć uwagę na ich rozmiar):

A. skonsolidowany statycznie (np. `/bin/busybox`),
B. skonsolidowany dynamicznie (np. `/bin/ls`).

Odczytaj strukturę tych plików poleceniem `readelf` lub `objdump`. Sprawdź narzędziem `ldd(1)` jakich bibliotek wymaga program `B`, aby zostać załadowanym. Zauważ, że `ldd` informuje również pod jakie adresy byłyby ładowane biblioteki. Czemu te adresy te zmieniają się z każdym wywołaniem `ldd`?

Ćwiczenie 8

Podejrzymy działanie dynamicznego programu ładującego `ld.so(8)`. Wyświetl domyślne ścieżki poszukiwania bibliotek "`cat /etc/ld.so.conf.d/*`" – można je jednorazowo zmienić ustawiając zmienną środowiskową `LD_LIBRARY_PATH`. Uruchom następujące polecenie: "`LD_DEBUG=libs /bin/ls -l`" – zauważ, że `ld.so` działa również po przekazaniu sterowania do załadowanego programu. Uruchamiając polecenie "`LD_DEBUG=bindings /bin/echo test`" zaobserwuj kiedy przebiega proces wiązania symboli. Zmień wartość zmiennej środowiskowej `LD_DEBUG` na "`bindings,versions,libs`" uruchom program raz jeszcze i zauważ, że symbole są wersjonowane – jak myślisz czemu to służy?