Systemy operacyjne

Lista zadań nr 7

Na zajęcia 24 i 25 listopada 2021

Należy przygotować się do zajęć czytając następujące materiały: [1, rozdział 9.8], [2, rozdział 8.11], [3, rozdział 38 i 49], [4, rozdziały 3.3, 3.5, 10.7].

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. Na podstawie [3, 49.1] wyjaśnij słuchaczom różnicę między **odwzorowaniami plików w pamięć** (ang. *memory-mapped files*) i **odwzorowaniami pamięci anonimowej** (ang. *anonymous mappings*). Jaką zawartością wypełniana jest pamięć wirtualna należąca do tychże odwzorowań? Czym różni się odwzorowanie **prywatne** od **dzielonego**? Czy pamięć obiektów odwzorowanych prywatnie może być współdzielona? Czemu można tworzyć odwzorowania plików urządzeń blokowych w pamięć, a znakowych nie?

Wskazówka: Jądro może udostępniać pamięć karty graficznej albo partycję dysku jako urządzenie blokowe.

Zadanie 2. Na podstawie opisu do [3, tabeli 49–1] podaj scenariusze użycia prywatnych i dzielonych odwzorowań plików w pamięć albo pamięci anonimowej. Pokaż jak je utworzyć z użyciem wywołania `mmap(2)`. Co się dzieje z odwzorowaniami po wywołaniu `fork(2)`? Czy wywołanie `execve(2)` tworzy odwzorowania prywatne czy dzielone? W jaki sposób jądro systemu automatycznie zwiększa rozmiar stosu do ustalonego limitu? Kiedy jądro wyśle sygnał `SIGBUS` do procesu posiadającego odwzorowanie pliku w pamięć [3, §49.4.3]?

Zadanie 3. Przy pomocy polecenia «`cat /proc/$(pgrep Xorg)/status | egrep 'Vm|Rss'`» wyświetl zużycie pamięć procesu wykonującego kod X-serwera. Na podstawie podręcznika `proc(5)` wyjaśnij znaczenie poszczególnych pól. Przypomnij jaka jest różnica między **zbiorem roboczym** i **rezydentnym** procesu. Napisz krótki skrypt (np. w języku `Python` lub `awk(1)`), który wyznaczy sumę «`VmSize`» i osobno sumę «`VmRSS`» wszystkich procesów. Czemu ta druga wartość nie pokrywa się z rozmiarem używanej pamięci raportowanym przez polecenie «`vmstat -s`»?

Zadanie 4. Na podstawie slajdów do wykładu opisz algorytm obsługi błędu stronicowania w systemie Linux. Jakie informacje musi dostarczyć procesor, żeby można było wykonać procedurę obsługi błędu stronicowania? Do czego służą struktury jądra «`mm_struct::pgd`» i «`mm_struct::mmap`» zdefiniowane w pliku `include/linux/mm_types.h`? Kiedy jądro wyśle procesowi sygnał `SIGSEGV` z kodem «`SEGV_MAPERR`» lub «`SEGV_ACCERR`»? W jakiej sytuacji wystąpi **pomniejsza usterka strony** (ang. *minor page fault*) lub **poważna usterka strony** (ang. *major page fault*)? Jaką rolę pełni w systemie **bufor stron** (ang. *page cache*)?

Zadanie 5. Chcemy rozszerzyć algorytm z poprzedniego zadania o obsługę **kopiowania przy zapisie** (ang. *copy on write*). W przestrzeni adresowej procesu utworzono odwzorowania prywatne segmentów pliku wykonywalnego `ELF`. Rozważmy kilkustronicowy segment danych D przechowujący sekcję «`.data`». Wiele procesów wykonuje ten sam program, zatem każdy może zmodyfikować dowolną stronę w swoim segmencie D. Co jądro przechowuje w strukturze «`vm_arena_struct`» opisującej segment D, a w szczególności w polach «`vm_prot`» i «`vm_flags`»? Jak jądro zmodyfikuje «`mm_struct::pgd`» w trakcie pierwszego odczytu ze strony p należącej do D, a jak w trakcie późniejszego pierwszego zapisu do p? Co jądro musi zrobić z tablicą stron procesu, który zawołał `fork(2)`? Czemu jądro nie musi kopiować tablicy stron z rodzica do dziecka?

Wskazówka: Możesz założyć, że jądro pamięta listę stron używanych przez dany segment.

Zadanie 6. Wiemy, że jądro używa **stronicowania na żądanie** (ang. *demand paging*) dla wszystkich odwzorowań. Rozważmy program, który utworzył prywatne odwzorowanie pliku w pamięć. Czy mamy gwarancję, że program nie zobaczy modyfikacji zawartości pliku, które zostaną wprowadzone po utworzeniu tego odwzorowania? Próba otworzenia `open(2)` pliku wykonywalnego do zapisu, kiedy ten plik jest załadowany i wykonywany w jakimś procesie, zawiedzie z błędem «`ETXTBSY`». Podobnie, nie możemy załadować do przestrzeni adresowej `execve(2)` pliku, który jest otwarty do zapisu. Co złego mogłoby się stać, gdyby system operacyjny pozwolił modyfikować plik wykonywalny, który jest uruchomiony?

Ściągnij ze strony przedmiotu archiwum «so21_lista_7.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «TODO». Pamiętaj o użyciu odpowiednich funkcji opakowujących (ang. *wrapper*) z biblioteki «libcsapp».

Zadanie 7. Program «forksort» wypełnia tablicę 2^{26} elementów typu «long» losowymi wartościami. Następnie na tej tablicy uruchamia hybrydowy algorytm sortowania, po czym sprawdza jeden z warunków poprawności wyniku sortowania. Zastosowano algorytm sortowania szybkiego (ang. *quick sort*), który przełącza się na sortowanie przez wstawianie dla tablic o rozmiarze mniejszym niż «INSERTSORT_MAX».

Twoim zadaniem jest taka modyfikacja programu «forksort», żeby oddelegować zadanie sortowania fragmentów tablicy do podprocesów. Przy czym należy tworzyć podprocesy tylko, jeśli rozmiar nieposortowanej części tablicy jest nie mniejszy niż «FORKSORT_MIN». Zauważ, że tablica elementów musi być współdzielona między procesy – użyj wywołania mmap(2) z odpowiednimi argumentami.

Porównaj **zużycie procesora** (ang. *CPU time*) i **czas przebywania w systemie** (ang. *turnaround time*) przed i po wprowadzeniu delegacji zadań do podprocesów. Na podstawie prawa Amdahla[1] wyjaśnij zaobserwowane różnice. Których elementów naszego algorytmu nie da się wykonywać równolegle?

Zadanie 8. (Pomysłodawcą zadania jest Piotr Polesiuk.)

Nasz serwis internetowy stał się celem ataku hakerów, którzy wykradli dane milionów użytkowników. Zostaliśmy zmuszeni do zresetowania haseł naszych klientów. Nie możemy jednak dopuścić do tego, by użytkownicy wybrali nowe hasła z listy, którą posiadają hakerzy. Listę pierwszych 10 milionów skompromitowanych haseł można pobrać poleceniem «make download».

Program «hashdb» został napisany w celu utworzenia bazy danych haseł i jej szybkiego przeszukiwania. Pierwszym argumentem przyjmowanym z linii poleceń jest nazwa pliku bazy danych haseł. Program wczytuje ze standardowego wejścia hasła oddzielone znakami końca linii i działa w dwóch trybach: dodawania haseł do bazy (opcja «-i») i wyszukiwania (opcja «-q»). Żeby utworzyć bazę danych z pliku zawierającego hasła należy wywołać polecenie «./hashdb -i badpw.db < passwords.txt». Program można uruchomić w trybie interaktywnego odpytywania bazy danych: «./hashdb -q badpw.db».

Implementacja wykorzystuje tablicę mieszającą przechowywaną w pamięci, która odwzorowuje plik bazy danych haseł. Używamy adresowania liniowego i funkcji mieszającej Jenkinsa[2] «lookup3.c». Hasło może mieć maksymalnie «ENT_LENGTH» znaków. Baza danych ma miejsce na 2^k wpisów. Jeśli w trakcie wkładania hasła do bazy wykryjemy konflikt kluczy, to wywołujemy procedurę «db_rehash». Tworzy ona na nową bazę o rozmiarze 2^{k+1} wpisów, kopiuje klucze ze starej bazy do nowej i atomowo zastępuje stary plik bazy danych.

Twoim zadaniem jest uzupełnić kod procedur «db_open», «db_rehash» i «doit» zgodnie z poleceniami zawartymi w komentarzach. Przeczytaj podręcznik systemowy do wywołania systemowego madvise(2) i wyjaśnij słuchaczom co ono robi. Należy użyć odpowiednich funkcji z biblioteki «libcsapp» opakowujących wywołania: unlink(2), mmap(2), munmap(2), madvise(2), ftruncate(2), rename(2) i fstat(2).

[1] https://pl.wikipedia.org/wiki/Prawo_Amdahla
[2] https://en.wikipedia.org/wiki/Jenkins_hash_function

Literatura

[1] „*Computer Systems: A Programmer's Perspective*"
Randal E. Bryant, David R. O'Hallaron
Pearson Education Limited; 3rd edition; 2016

[2] „*Advanced Programming in the UNIX Environment*"
W. Richard Stevens, Stephen A. Rago
Addison-Wesley Professional; 3rd edition; 2013

[3] „*The Linux Programming Interface: A Linux and UNIX System Programming Handbook*"
Michael Kerrisk
No Starch Press; 1st edition; 2010

[4] „*Systemy operacyjne*"
Andrew S. Tanenbaum, Herbert Bos
Helion; wydanie czwarte; 2015

[5] „*Operating Systems: Three Easy Pieces*"
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
https://pages.cs.wisc.edu/~remzi/OSTEP/