Systemy operacyjne

Lista zadań nr 7

Na zajęcia 26 listopada i 4 grudnia 2019

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 3.3, 3.5, 10.7
- Linux Programming Interface: 38, 49
- APUE (wydanie trzecie): 8.11

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

Zadanie 1. Na podstawie §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**? Dlaczego odwzorowania prywatne wykorzystują technikę **kopiowania przy zapisie**?

Zadanie 2. Na podstawie opisu do tabeli 49–1 podaj scenariusze użycia prywatnych albo dzielonych odwzorowań plików w pamięć albo pamięci anonimowej. Pokaż jak je utworzyć z użyciem wywołania `mmap(2)`. Rozważ co się z nimi dzieje po wywołaniu `fork(2)`. Jakie odwzorowania tworzy wywołanie `execve(2)`? Które z wymienionych odwzorowań mogą wymagać użycia **pamięci wymiany** (ang. *swap space*) i dlaczego?

Zadanie 3. Na podstawie slajdów do wykładu wyjaśnij w jaki sposób system Linux obsługuje błąd stronicowania. 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 **bufor stron** (ang. *page cache*)? Kiedy wystąpi błąd strony przy zapisie, mimo że pole «`vm_prot`» pozwala na zapis do **obiektu wspierającego** (ang. *backing object*)? Kiedy jądro wyśle `SIGBUS` do procesu posiadającego odwzorowanie pliku w pamięć (§49.4.3)?

Zadanie 4. Niech właścicielem programu A oraz B są odpowiednio użytkownik 0 «`root`» i 1000 «`cahir`». Obydwa pliki mają ustawiony bit «`set-uid`». Proces o uruchomiony na uprawnieniach użytkownika 2000 «`student`» używa wywołania `execve(2)` do wykonania procesu A i B. Podaj wartości **rzeczywistego**, **obowiązującego** i **zachowanego** identyfikatora użytkownika po wywołaniu «`execve`». Jakie instrukcje muszą wykonać programy A i B, aby prawidłowo przywrócić tożsamość użytkownika `student`? Należy użyć wywołań `seteuid(2)` i `setreuid(2)`.

Zadanie 5. Co mówi „zasada najbardziej ograniczonych uprawnień” (ang. *principle of least privilege*) w kontekście projektowania oprogramowania? Zreferuj sekcję 4.3 artykułu Capsicum: practical capabilities for UNIX[1] opisującego `capsicum(4)`. Przeanalizuj sposób wydzielenia (ang. *sandboxing*) z programu `gzip(1)` funkcji podatnych na ataki. Zastanów się, które prawa `rights(4)` oraz operacje na deskryptorach plików `cap_rights_limits(2)` powinny zostać nadane podprocesowi przeprowadzającemu dekompresję danych.

Wskazówka: Obejrzyj pierwsze 25 minut prezentacji Capsicum: Practical capabilities for UNIX[2] z FOSDEM 2014.

Zadanie 6 (bonus). Jedną z metod zmniejszania podatności aplikacji na ataki jest zawężanie widoczności systemu plików (ang. *filesystem sandboxing*). Przeczytaj podręcznik do wywołania systemowego `unveil(2)`. Jak użyć tego wywołania do zmniejszenia ryzyka wycieku informacji z programu komunikującego się z Internetem, np. przeglądarki internetowej? Czym różni się ten mechanizm od `chroot(2)`?

Wskazówka: Obejrzyj prezentację Unveil in OpenBSD[3] z BSDCan 2019.

[1] `https://www.usenix.org/legacy/event/sec10/tech/full_papers/Watson.pdf`
[2] `https://www.youtube.com/watch?v=GI9PmtF9jdM`
[3] `https://www.youtube.com/watch?v=gvmGfpMgny4`

Ściągnij ze strony przedmiotu archiwum «so19_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 (P). 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[4] wyjaśnij zaobserwowane różnice. Których elementów naszego algorytmu nie da się wykonywać równolegle?

Zadanie 8 (P). (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[5] «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).

[4] https://pl.wikipedia.org/wiki/Prawo_Amdahla

[5] https://en.wikipedia.org/wiki/Jenkins_hash_function