Lista 7

tags: SO

Zadanie 1

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.

::: info

- **odwzorowanie pliku w pamięć** – część pamięci dedykowana zawartości danego pliku, rzeczywista zawartość pliku jest kopiowana stronami do pamięci w sposób leniwy.
- **odwzorowanie pamięci anonimowej** – strony inicjowane zerami bez odpowiadającego im pliku
- **odwzorowanie prywatne** – wprowadzone do zasobu prywatnego zmiany są widoczne tylko dla modyfikującego procesu (tj. modyfikowana część jest kopiowana i ponownie odwzorowywana)
- **odwzorowanie dzielone** – zasób (i wszystkie jego zmiany) jest wspólny dla dzielących go procesów :::

Czy pamięć obiektów odwzorowanych prywatnie może być współdzielona?

Tak, dopóki nie są wykonywane na niej modyfikacje

Czemu można tworzyć odwzorowania urządzeń blokowych w pamięć a znakowych nie?

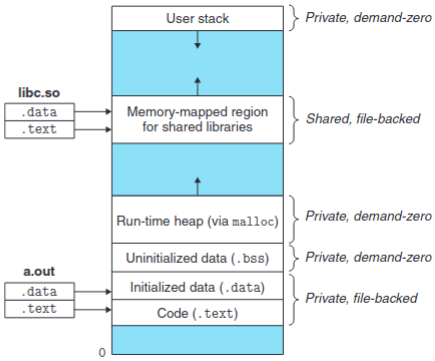
Urządzenia znakowe nie pozwalają na random access do swojej zawartości?

Zadanie 2

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]?

Scenariusze użycia

Figure 9.31
How the loader maps the areas of the user address space.



malloc -- prywatne odwzorowanie pamięci anonimowej **komunikacja za pomocą pliku** -- dzielone odwzorowanie pliku

Table 49-1: Purposes of various types of memory mapping

Visibility of modifications	Mapping
	File
Private	Initializing memory from contents of
Shared	Memory-mapped I/O; sharing memo between processes (IPC)

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int
```

Returns starting address of mapping on s

- **addr** – początek odwzorowania w przestrzeni adresowej procesu, NULL jeśli wybór pozostawiamy jądru
- **length** – rozmiar odwzorowania w bajtach
- **prot**

Value	Description
PROT_NONE	The region may not be accessed
PROT_READ	The contents of the region can be read
PROT_WRITE	The contents of the region can be modifi
PROT_EXEC	The contents of the region can be execut

- **flags**

Value	Description
MAP_ANONYMOUS	Create an anonymous mapping
MAP_FIXED	Interpret *addr* argument exactly (S
MAP_LOCKED	Lock mapped pages into memory
MAP_HUGETLB	Create a mapping that uses huge p
MAP_NORESERVE	Control reservation of swap space
MAP_PRIVATE	Modifications to mapped data are
MAP_POPULATE	Populate the pages of a mapping (
MAP_SHARED	Modifications to mapped data are and propagated to underlying file
MAP_UNINITIALIZED	Don’t clear an anonymous mappi

- **fd** – deskryptor pliku (ma znaczenie tylko, gdy *MAP_ANONYMOUS* nie jest ustawiona)
- **offset** – pozycja w pliku, od której zaczynamy odwzorowywanie (również działa tylko dla odwzorowań pliku i musi być wielokrotnością rozmiaru strony)

MAP_ANONYMOUS and /dev/zero

On Linux, there are two different, equivalent met
mapping with *mmap()*:

- Specify MAP_ANONYMOUS in *flags* and specify *fd* as
  ignored when MAP_ANONYMOUS is specified. Howev
  require *fd* to be −1 when employing MAP_ANON
  should ensure that they do this.)

  > We must define either the _BSD_SOURCE or
  > to get the definition of MAP_ANONYMOUS fro
  > constant MAP_ANON as a synonym for MAP_AN
  > other UNIX implementations using this al

- Open the /dev/zero device file and pass the resu

  > /dev/zero is a virtual device that always re
  > Writes to this device are always discarded
  > populate a file with zeros (e.g., using the *d*

Co się dzieje z odwzorowaniami po wywołaniu forka?

Fork wykonuje płytką kopię całej przestrzeni adresowej razem z odwzorowaniami. Wszystkie odzworowania są oznaczane jako `read-only`, a w segmentach `cow`. Wtedy przy próbie zapisu dostajemy `page fault`, którego naprawą zajmuje się jądro.

Czy wywołanie execve tworzy odwzorowania prywatne czy dzielone?

Prywatne, np. segment `data` nie miałby sensu będąc dzielonym

W jaki sposób jądro systemu automatycznie zwiększa rozmiar stosu do ustalonego limitu?

Odwołanie się do adresu pod stosem wywołuje `page fault`, który jest obsługiwany poprzez rozszerzenie stosu. (Jest też `MAP_GROWSDOWN` dla customowych stosów <3)

Kiedy jądro wyśle sygnał SIGBUS do procesu posiadającego odwzorowanie pliku w pamięć?

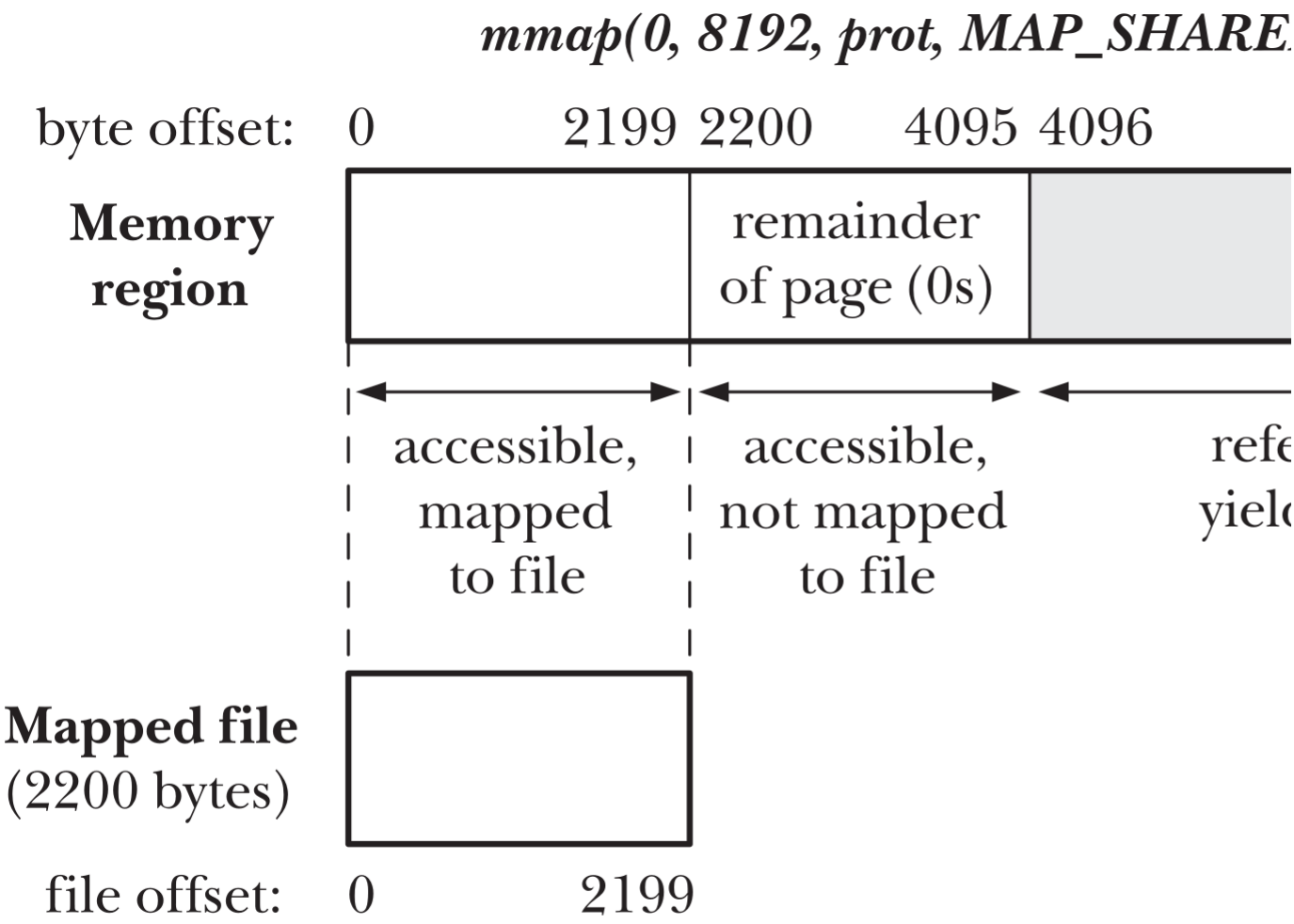


Figure 49-4: Memory mapping extending beyond end of

If the mapping includes pages beyond the rou
and beyond in Figure 49-4), then attempts to acces
in the generation of a SIGBUS signal, which warns th
of the file corresponding to these addresses.
addresses beyond the end of the mapping result in th

Zadanie 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`»?

```
cat /proc/$(pgrep Xorg)/status | egrep 'Vm|Rss'
```

```
☁  ~  cat /proc/$(pgrep Xorg)/status | egrep 'Vm|Rss'
VmPeak:  2631124 kB
VmSize:  2022148 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:    263100 kB
VmRSS:    261560 kB
RssAnon:           42288 kB
RssFile:           53232 kB
RssShmem:         166040 kB
VmData:   123100 kB
VmStk:       132 kB
VmExe:      1624 kB
VmLib:     76584 kB
VmPTE:      1444 kB
VmSwap:        0 kB
```

::: info

- **zbiór rezydentny** – dane programu, które są załadowane do pamięci RAM.
- **zbiór roboczy** – podzbiór zbioru rezydentnego, potrzebny do wykonywania procesu. :::

Wyjaśnij znaczenie poszczególnych pól

::: warning

- **Vm** – oznacza pamięć writualną
  - **VmPeak** – szczytowy rozmiar pamięci
  - **VmSize** – obecny rozmiar pamięci
  - **VmLck** – rozmiar pamięci zablokowanej – taka pamięć nie może być usunięta z RAMu
    > **mlock**(), **mlock2**(), and **mlockall**() lock part or all of the calling process space into RAM, preventing that memory from being paged to the swap area.
  - **VmPin** – pamięć przypięta (strony, które nie mogą być przeniesione, ponieważ coś musi bezpośrednio skorzystać z pamięci fizycznej, np. karta graficzna :£)
  - **VmHWM (High water mark)** – szczytowy rozmiar zbioru rezydentnego
  - **VmRSS** – rozmiar setu rezydentego
  - **VmData, VmStk, VmExe** – rozmiar segmentów `data`, `stack`, `text`
  - **VmLib** – rozmiar kodu współdzielonych bibliotek
  - **VmPTE** – rozmiar wpisów tablicy stron
  - **VmSwap** – Swapped-out virtual memory size by anonymous private pages
- **Rss** – rozmiar zbioru rezydentnego
  - **RssAnon** – rozmiar rezydentnej pamięci anonimowej
  - **RssFile** – rozmiar mapowań rezydentnych plików
  - **RssShmem** – rozmiar współdzielonej pamięci rezydentnej :::

Krótki skypt do sumowania

```
from glob import glob

VmSize = VmRSS = 0

files = glob("/proc/**/status")
for f in files:
    for line in open(f, 'r'):
        line = line.split()
        if 'VmSize:' in line: VmSize += int(line[1])
        if 'VmRSS:' in line: VmRSS += int(line[1])

print(f"VmSize: {VmSize} kB\nVmRSS: {VmRSS} kB")
```

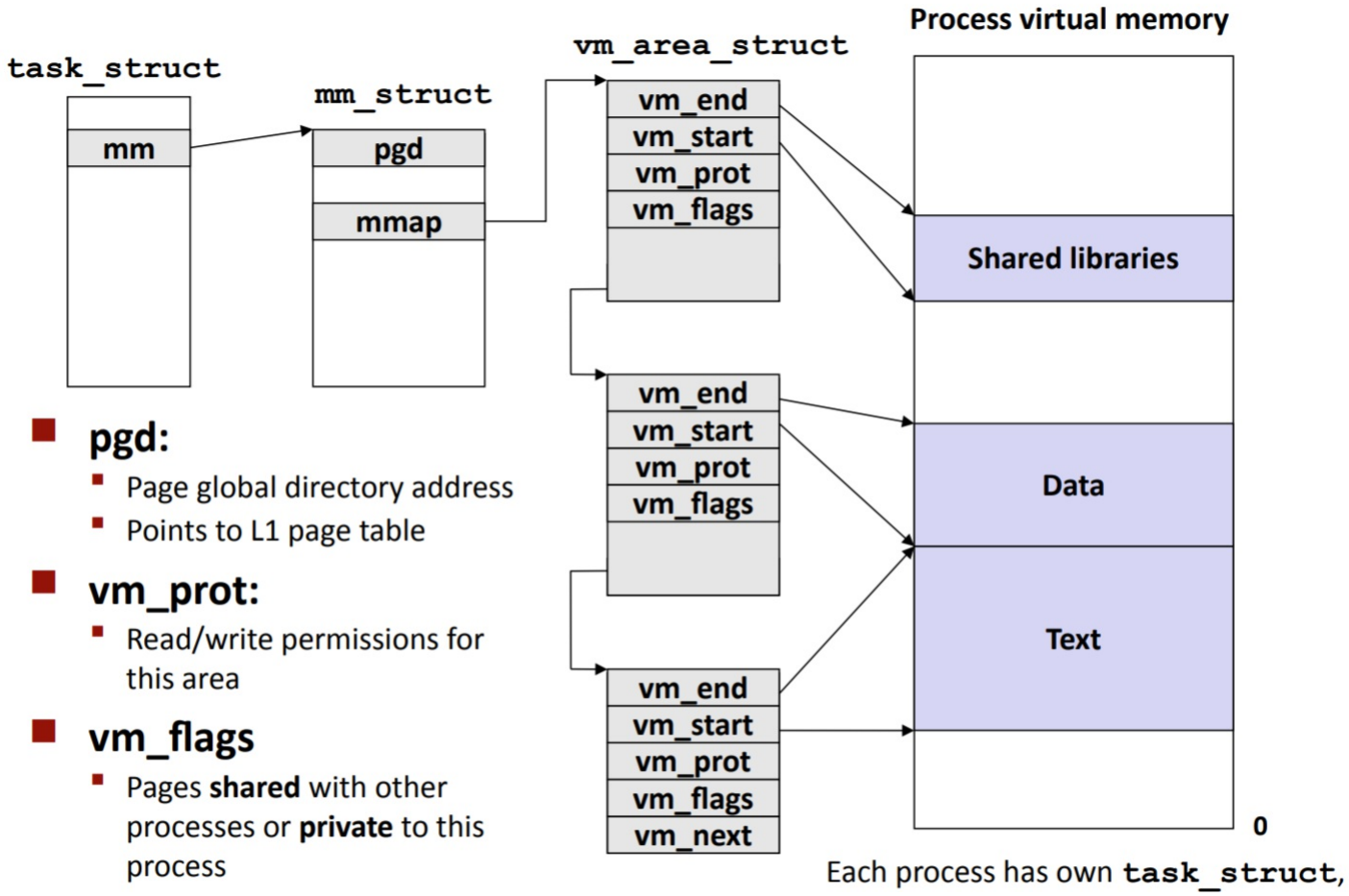
CzEmU WyNiKi dla RSS SiĘ RóŻnIą?!

Hipotetycznie dzieje się tak dlatego, że pamięć współdzielona jest liczona wielokrotnie przy osobnym sumowaniu.

Zadanie 4

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*)?

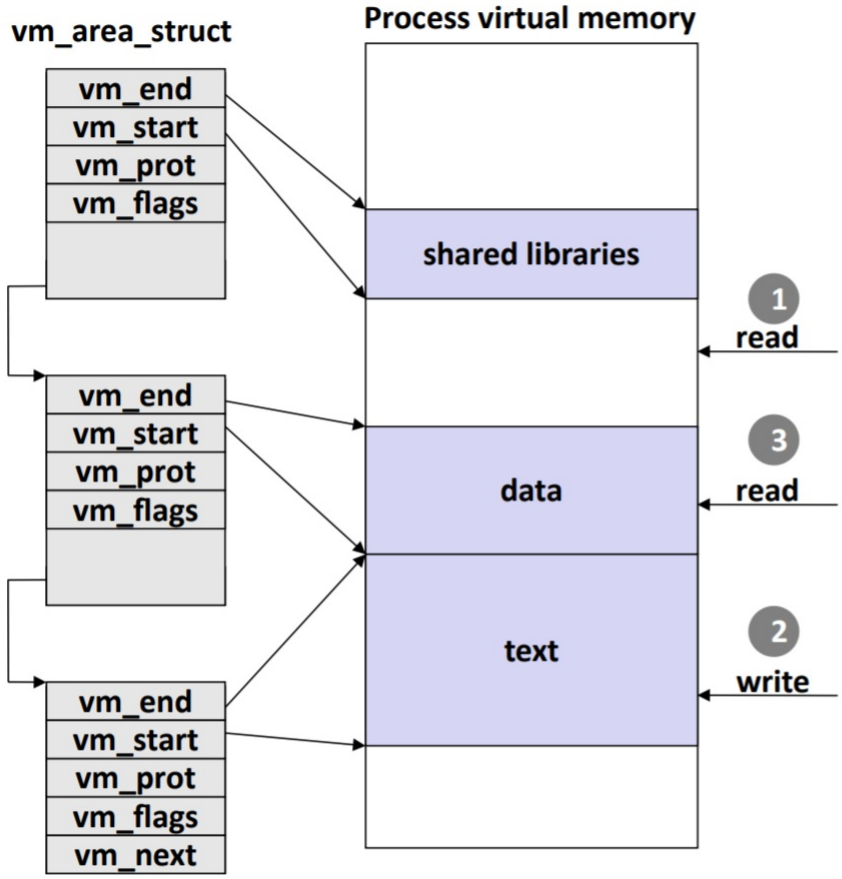
Linux Organizes VM as Collection of "Areas"



- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Each process has own `task_struct`,

Linux Page Fault Handling



Segmentation fault:
accessing a non-existing page

Normal page fault

Protection exception:
e.g., violating permission by writing to a read-only page (Lin reports as Segmentation fault)

Obsługa błędu stronnicowania

Jądro przegląda segmenty procesu i wybiera ten, w którym wystąpił `page fault`. Mamy trzy możliwe przypadki dalszego rozwoju sytuacji:

- **Segmentation fault** – próba dostępu do nieistniejącej w pamięci wirtualnej strony (SIGSEGV z kodem MAPPERR)
- **Protection exception** – nie zgadzają się uprawnienia (SIGSEGV z kodem SIGV_ACCERR)
- **Normal page fault** – (bit valid ustawiony na zero?) dostęp do pamięci jest poprawny, trzeba tylko naprawić błąd np. sprowadzając stronę z pamięci zewnętrznej / wykonując `cow` itd.

::: info

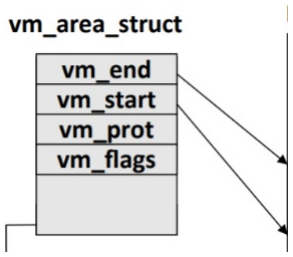
- **Poważna usterka strony** – żądanej strony nie ma w buforze stron w pamięci fizycznej, aby ją naprawić trzeba sięgnąć do pamięci zewnętrznej
- **Pomniejsza usterka strony** – żądana strona jest w buforze stron w pamięci fizycznej (dane należą do innego procesu i naprawienie usterki polega na dzieleniu tej części danych z procesem, który potrzebuje do nich dostępu)
- **Bufor stron** – jądro buforuje przy pierwszym odczycie lub zapisie pliku dane w niewykorzystanym obrzarze RAMu, co przyspiesza odczyt zbuforowanych stron (http://gauss.ececs.uc.edu/Courses/c4029/code/memory/understanding.pdf) :::

Jakie informacje musi dostarczyć procesor, żeby można było wykonać procedurę obsługi stronicowania?

```
`fault_addr` -- adres strony, pod którym wystąpił błąd
`fault_pc` -- adres instrukcji, która wywołała błąd,
`fault_size` -- ile bajtów próbowano odczytać,
`_prot` -- maska uprawnień dostępu
```

Do czego służą struktury jądra `pgd` i `mmap`?

- **pgd** – tablica stron procesu
- **mmap** – lista *vm_area_struct*



Zadanie 5

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.

Co jądro przechowuje w strukturze `vm_area_struct`?

https://linux-kernel-labs.github.io/refs/heads/master/labs/memory_mapping.html

A **`struct vm_area_struct`** is created at each **`mmap()`** call issued from user space. A driver that supports the **`mmap()`** operation must comp

- **`vm_start`**, **`vm_end`** - the beginning and the end of the memory area, respectively (these fields also appear in `/proc/<pid>/maps`);
- **`vm_file`** - the pointer to the associated file structure (if any);
- **`vm_pgoff`** - the offset of the area within the file;
- **`vm_flags`** - a set of flags;
- **`vm_ops`** - a set of working functions for this area
- **`vm_next`**, **`vm_prev`** - the areas of the same process are chained by a list structure

https://students.mimuw.edu.pl/SO/Linux/Kod/include/linux/mm.h.html

```
struct vm_area_struct {
        struct mm_struct * vm_mm;       /* VM area parameters */
        unsigned long vm_start;
        unsigned long vm_end;
        pgprot_t vm_page_prot;
        unsigned short vm_flags;
/* AVL tree of VM areas per task, sorted by address */
        short vm_avl_height;
        struct vm_area_struct * vm_avl_left;
        struct vm_area_struct * vm_avl_right;
/* linked list of VM areas per task, sorted by address */
        struct vm_area_struct * vm_next;
/* for areas with inode, the circular list inode->i_mmap */
/* for shm areas, the circular list of attaches */
/* otherwise unused */
        struct vm_area_struct * vm_next_share;
        struct vm_area_struct * vm_prev_share;
/* more */
        struct vm_operations_struct * vm_ops;
        unsigned long vm_offset;
        struct inode * vm_inode;
        unsigned long vm_pte;                   /* shared mem */
};
```

::: warning

- `vm_prot` – uprawnienia dostępu `rwx`
- `vm_flags` – tryb `cow` + `shared` / `private`

:::

Jak jądro zmodyfikuje `pgd` w trakcie pierwszego odczytu ze strony *p*, a jak w trakcie późniejszego pierwszego zapisu do *p*?

Przy pierwszej próbie odczytu strony nie będzie w pamięci i dostaniemy `page fault` i jądro podejmie próbę jego naprawy, tj. wydobycia strony z pamięci zewnętrznej.

Przy pierwszym zapisie również otrzymujemy `page fault`, ponieważ w `pgd` strona jest tylko do odczytu, wtedy jądro musi przebiec się po segmentach, wybrać ten, w którym wystąpił błąd i sprawdzić jego uprawnienia. Jeśli segment jest `read-write` z polityką `cow` to kopiujemy i modyfikujemy stronę, wpp. proces dostaje `segfault`.

Co jądro musi zrobić z tablicą stron procesu, który zawołał fork?

Ustawia wszystkie strony `pgd` na `read-only` oraz flagę `copy-on-write` dla segmentów.

Czemu jądro nie musi kopiować tablicy stron z rodzica do dziecka?

Zgodnie z polityką `copy-on-write` oba procesy mogą współdzielić tablicę stron i "prywatnie" ją modyfikować.

Zadanie 6

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?

::: info **stronicowanie na żądanie** – strony są ładowane do pamięci przy zapisie i odczycie (podejście leniwe) :::

Czy mamy gwarancję, że program nie zobaczy modyfikacji zawartości pliku, które zostaną wprowadzone po utworzeniu tego odwzorowania?

Plik z pamięci zewnętrznej może ulec zmianie w trakcie wykonywania programu korzystającego z odwzorowania tego pliku. W takim razie może się zdarzyć, że polityka stronicowania na żądanie ściągnie nam stronę zmodyfikowaną po utworzeniu odwzorowania.

Co złego mogłoby się stać, gdyby system operacyjny pozwolił modyfikować uruchomiony plik wykonywalny?

Gdyby system pozwolił na modyfikację uruchomionych plików wykonywalnych moglibyśmy zmodyfikować fragment pliku niezaładowany jeszcze do pamięci. W ten sposób wykonujący się program mógłby załadować zmodyfikowany kod (co w oczywisty sposób nie jest zdrowe dla programu).

ETXTBSY – https://lwn.net/Articles/866493/

Zadanie 7

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

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)`.