Obsługa błędu braku strony.

Magdalena Kowalska(mk181053@zodiac.mimuw.edu.pl)
Bartosz Wilczyński (bw176398@zodiac.mimuw.edu.pl)

16 listopada 2000

1 Krótko o stronicowaniu.

Stronicowanie polega na podziale pamięci na spójne bloki o stałej długości. Taki pojedynczy blok nazywa się stroną, a jego wielkość jest określona przez stałą PAGE_SIZE w pliku nagłówkowym **page.h**. W omawianej wersji systemu, wielkość strony wynosi 4096 bajtow czyli 4 kB.

Stronicowanie umożliwiło zaimplementowanie **pamięci wirtualnej**, która pozwala na pisanie i uruchamianie programów zajmujących dużo więcej miejsca niż pozwala na to rozmiar pamięci fizycznej. Mówiąc krótko, procesy nie są umieszczane w pamięci w całości, ale są dzielone na strony i w pamięci umieszcza sie jedynie te strony, których proces w danym momencie potrzebuje. Pozostałe są umieszczane w pamięci pomocniczej i są przywoływane do pamięci fizycznej w miarę potrzeby. Również strony długo nie używane są odsyłane do pamięci pomocniczej.

2 Konstrukcja naszego referatu.

Referat składa się z dwóch zasadniczych części. W pierwszej przedstawiony jest teoretyczny opis algorytmu będącego przedmiotem opracowania. W drugiej - jego linuksowa implementacja. Wydaje nam się to o tyle sensowne, że sama linuksowa implementacja jest dosyć skomplikowana i trudna do zrozumienia, ze względu na dość dużą ilość odwołań do funkcji pomocniczych i złożonych struktur danych. Umieszczenie teoretycznego opisu, nawet niekoniecznie pokrywającego się z rzeczywistością powinno pomóc w zrozumieniu jak to naprawdę działa.

3 Algorytm obsługi błędu braku strony.

3.1 Teoretyczny opis algorytmu.

3.1.1 Struktury danych wspierające stronicowanie na żądanie.

Z punktu widzenia mechanizmu stronicowania najistotniejsze są następujące struktury danych:

- tablica stron
- deskryptory bloków dyskowych
- tablica ramek stron
- tablica wymiany.

Tablice stron. Służą do uzyskiwania dostępu do pamięci fizycznej i znajdują się w każdym segmencie. Każda pozycja tablicy stron zawiera fizyczny adres strony, bity ochrony oraz bity używane do stronicowania na żądanie, z których istotny dla nas będzie bit ważności, określający, czy zawartość strony jest legalna. Właśnie błąd braku strony powstaje, gdy proces odwołuje się do strony, której bit ważności nie jest ustawiony. A nie jest on ustawiony dla ramek, które znajdują się poza przestrzenią adresową procesu lub dla tych, które leżą w tej przestrzeni, ale nie mają w tej chwili przydzielonej fizycznej strony pamięci. Każda pozycja tablicy stron jest ponadto związana z deskryptorem bloku dyskowego, opisującego kopię dyskową strony wirtualnej. Pozwala to na współdzielenie pozycji tablicy oraz deskryptorów bloków dyskowych stron przez procesy współdzielące segment.

Deskryptory bloków dyskowych. Zawartość strony wirtualnej może się znajdować w określonym bloku pamięci pomocniczej, może stanowić część wykonywalnego pliku lub może jej nie być w pamięci pomocniczej. Deskryptor bloku zawiera numer logicznego urządzenia wymiany a także numer bloku zawierającego tę stronę. Jeżeli strona jest częścią pliku wykonywalnego, to deskryptor zawiera logiczny numer bloku tego pliku, który to numer jądro może szybko odwzorować na adres dyskowy. W deskryptorze są również określone warunki, czy strona ma być "sprowadzana na żądanie" czy "zerowana na żądanie". Ustawiane są one przy wykonywaniu funkcji exec, ale dla podprogramu obsługi błędu braku strony nie jest istotne w jaki sposób.

Tablica ramek stron. Opisuje każdą stronę pamięci fizycznej. Jej indeksami są numery stron. Każda pozycja zawiera:

- stan strony (czy jest w pamięci pomocniczej, czy w pliku wykonywalnym, czy jest właśnie odczytywana z pamięci pomocniczej, czy może być ponownie przydzielona).

- Liczba procesów odwołujących się do strony.
- Numer urządzenia logicznego i bloku zawierającego kopię strony(jak w deskryptorze bloku dyskowego).
- Wskaźniki do pozycji tablicy ramek stron znajdujących się na liście wolnych i w kolejce mieszającej.

Lista wolnych, to lista stron, które mogą zostać ponownie przydzielone, a kolejka mieszająca organizuje porządek przed wykonaniem operacji wejścia/wyjścia.

Tablica wymiany. Zawiera pozycje dla wszystkich stron znajdujących się w pamięci pomocniczej. Każda pozycja zawiera licznik odwołań informujący, ile pozycji tablicy stron wskazuje na jedną stronę w pamięci pomocniczej.

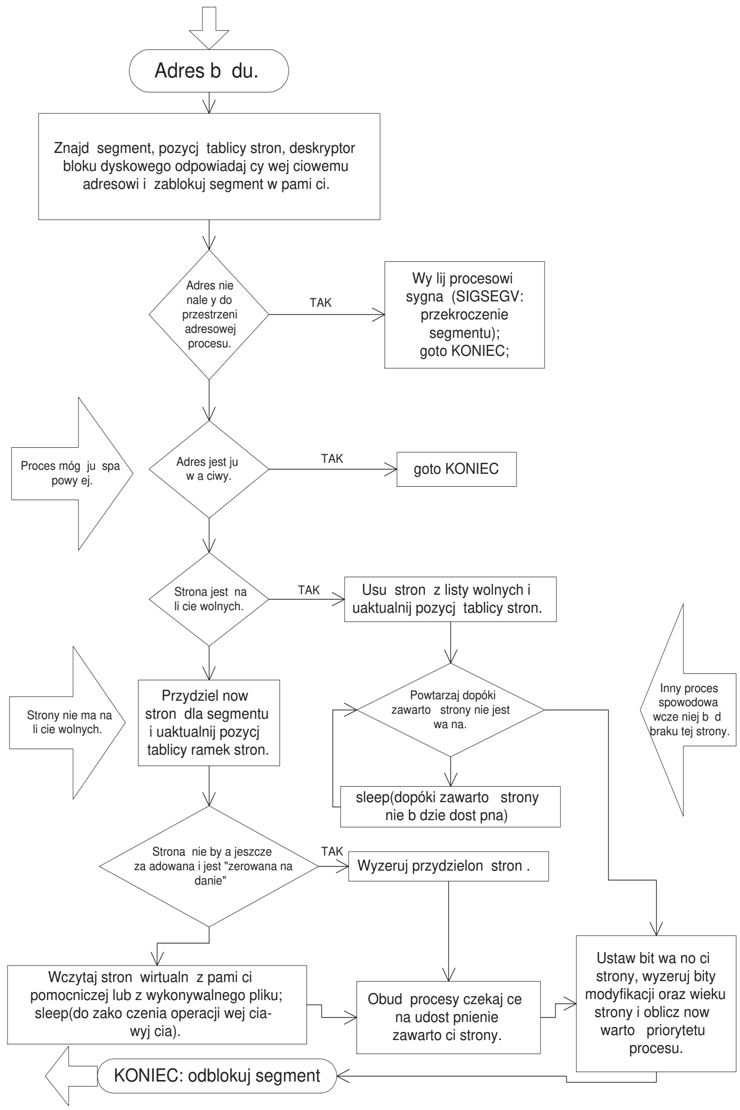
3.1.2 Teoretyczny algorytm obsługi błędu braku strony.

Jak już to zostało napisane wcześniej, błąd braku strony powstaje, gdy proces odwołuje się do strony, która ma nie ustawiony bit ważności. Dość nietypowe jest to, że podprogram obsługi błędu braku strony wykonuje się w kontekście tego procesu, chociaż tak naprawdę obsługuje przerwanie.

Najpierw sprzętowo jest dostarczany do jądra adres wirtualny, do którego dostęp spowodował wystąpienie błędu braku strony. Jądro wtedy odnajduje odpowiednią pozycję tablicy stron oraz deskryptor bloku dyskowego dla tej strony, a później blokuje w pamięci segment zawierający daną stronę, aby zapobiec używaniu go w tym czasie przez inne procesy (a szczególnie przez proces usuwający strony z pamięci). Jeśli deskryptor bloku dyskowego nie zawiera informacji o poszukiwanej stronie to odwołanie do pamięci jest niepoprawne i wysyłany jest sygnał przekroczenia segmentu. Jeśli odwołanie było poprawne, to jądro przydziela stronę pamięci głównej, do której będzie przesłana brakująca strona (z pamięci pomocniczej, lub z wykonywalnego pliku). Strona do której dostęp spowodował błąd może się znajdować w następujących stanach

- Strona jest w pamięci pomocniczej.
- Strona jest na liście wolnych stron.
- Strona znajduje się w wykonywalnym pliku.
- Strona oznaczana jako "zerowana na żądanie" lub "sprowadzana na żądanie".

Schemat algorytmu obs ugi b du braku strony.



W pierwszym przypadku oznacza to, że strona została usunięta przez proces usuwający strony. Wtedy na podstawie deskryptora bloku dyskowego jądro znajduje numer urządzenia pamięci pomocniczej oraz numer bloku w którym znajduje się strona. Sprawdza także czy strony nie ma na liście stron wolnych. Następnie uaktualnia tablicę pozycji stron, tak aby wskazywała na właściwą stronę fizyczną i umieszcza pozycję tablicy ramek w kolejce mieszającej. Potem wczytuje tą stronę z pamięci pomocniczej. Proces, który spowodował błąd braku strony zasypia na czas operacji wejścia /wyjścia a następnie budzi pozostałe procesy, które czekają na tę stronę.

W drugim przypadku jądro nie musi wykonywać operacji wejścia/wyjścia, bo możliwe, że fizyczna strona nie została, po przesłaniu do pamięci pomocniczej, przydzielona innemu procesowi, albo właśnie jest sprowadzana.

W obu powyższych przypadkach jądro odnajduje właściwą stronę fizyczną, a później zmienia pozycję tablicy stron tak, aby wskazywała znalezioną, zwiększa licznik odwołań do niej, oraz ewentualnie usuwa z listy stron wolnych.

W trzecim przypadku jądro wczytuje stronę z pliku wykonywalnego. Podprogram pobiera na podstawie deskryptora bloku dyskowego logiczny numer bloku oryginalnego pliku i odnajduje jego i-węzeł, związany z pozycją tablicy segmentów. Znając numer bloku dyskowego, jądro wczytuje go do pamięci.

W ostatnim przypadku jądro przydzieli nową stronę pamięci fizycznej i uaktualni odpowiednią pozycję tablicy stron.W przypadku "zerowania"zawartość strony jest dodatkowo zerowana.

Na zakończenie jądro zeruje bity "sprowadzania na żądanie" i "zerowania na żądanie". Strona znajduje się już w pamięci głównej i nie ma jej kopii, ani w pamięci pomocniczej, ani w systemie plików. Podprogram ustawia również bit ważności strony oraz zeruje bit modyfikacji. Na zakończenie oblicza aktualną wartość priorytetu procesu (bo ten mógł długo spać i jego priorytet mógł wtedy nieprzyzwoicie wzrosnąć).

3.2 Implementacja algorytmu

3.2.1 Struktury danych.

Struktura mm_struct

Jest strukturą reprezentującą pamięć procesu. Zdefiniowano ją w pliku: /usr/src/linux/include/linux/mm.h. Jej budowa jest następująca:

Pole	Opis
int count	licznik odwołań do struktury
pgd_t * pgd	tablica katalogów stron procesu
unsigned long context	aktualny kontekst procesu
unsigned long start_code	początek segmentu kodu
unsigned long end_code	koniec segmentu kodu
unsigned long start_data	początek segmentu danych
unsigned long end_data	koniec segmentu danych
unsigned long start_brk	początek segmentu mapowania i-węzłów
unsigned long brk	aktualna pozycja w segmencie
unsigned long start_stack	początek stosu
unsigned long start_mmap	adres początku pamięci wirtualnej procesu
unsigned long arg_start	początek segmentu zawierającego param. wywołania
unsigned long arg_end	koniec powyższego segmentu
unsigned long env_start	początek segmentu zmiennych środowiskowych
unsigned long env_end	koniec powyższego segmentu
unsigned long rss	liczba ramek procesu
unsigned long total_vm	liczba stron procesu
unsigned long locked_vm	liczba zablokowanych stron procesu
unsigned long def_flags	różne flagi będące częścią definicji samej strony

Oprócz tego jeszcze są trzy ważne pola:

- struct vm_area_struct * mmap
- struct vm_area_struct * mmap_avl
- struct semaphore mmap_sem

Pierwsze to lista struktur vm_area struct uporządkowana według początkowego adresu. Drugie to korzeń drzewa AVL tych struktur, które jest uporządkowane tak samo. Trzecie zawiera semafor do ochrony dostępu do struktury.

Struktura vm_area_struct Reprezentuje spójny obszar pamięci wirtualnej procesu i jest zdefiniowana w /usr/src/linux/include/linux/mm.h. Posiada następujące pola:

struct mm_struct * vm_mm jest to wskaźnik do struktury mm_struct procesu do którego należy ten vm_area_struct

unsigned long vm_start adres początku obszaru pamięci, który należy do danego vm_area_struct

unsigned long vm_end adres końca tego obszaru pamięci

unsigned short vm_flags flagi opisujące obszar pamięci

struct vm_area_struct *vm_avl_left, *vm_avl_right wskaźniki do lewego i prawego syna w drzewie AVL

struct vm_area *vm_next "fastryga" prefiksowa w drzewie AVL

unsigned long vm_pte pozycja (entry) w katalogu stron

pozostałe pola odpowiadają m.in za funkcje związane z mapowaniem plików. Z punktu widzenia algorytmu bez istotnego znaczenia.

3.2.2 Funkcja do_page_fault

Algorytm braku strony realizowany jest przez funkcję do_page_fault(pt_regs* regs, error_code) znajdującą się w pliku /usr/src/linux/arch/i386/mm/fault.c. MMU rejestruje tę funkcję jako wywoływaną przez procesor w momencie wystąpienia wyjątku z powodu błędu braku strony. Parametr regs zawiera wskaźnik do struktury regs, przedstawiającej stan rejestrów procesora w momencie wystąpienia błędu. Parametr error_code zawiera liczbę całkowitą, której trzy najmniej znaczące bity reprezentacji binarnej oznaczają odpowiednio:

- bit 0 - 0 brak strony | 1 naruszenie praw dostępu
- bit 1 - 0 czytanie | 1 pisanie
- bit 2 - 0 tryb jądra | 1 tryb użytkownika.

Sama funkcja do_page_fault jest napisana niezbyt eleganckim stylem, okraszonym gęsto instrukcjami skoków do etykiet (goto), tłumaczonym przez autorów lepszymi wynikami kompilacji takiego kodu. Wobec tego, że mało czytelny kod utrudnia analizę samego algorytmu, posłużyliśmy się kodem "pseudojęzyka" strukturalnego, zrozumiałym zapewne dla każdego kto programował w językach takich jak C lub Pascal. Korzystając z tej notacji przepisaliśmy ciało funkcji do_page_fault.

Staraliśmy się uczynić ten kod na tyle czytelnym, aby czytelnik jako tako obeznany z mechanizmami Linuksa, czy też szeroko pojętą tematyką budowy systemów operacyjnych, po przeczytaniu go, bez żadnych dodatkowych wyjaśnień posiadł podstawową wiedzę o implementacji tego algorytmu w Linuksie i w razie potrzeby mógł łatwo odnaleźć odpowiedzi na ewentualnie nurtujące go bardziej zaawansowane pytania w samym kodzie źródłowym.

W poniższym tekście znajduje się wiele fragmentów opisujących w języku naturalnym czynności wykonywane przez “wstawki” asemblerowe (np. pobierz adres błędnego odwołania do strony) lub przez funkcje zewnętrzne, zwykle wysokiego poziomu (funkcje pomocnicze są opisane w punkcie 4 i każde ich wywołanie w pseudokodzie jest opatrzone stosownym komentarzem). Ogólnie stosowaną notacją komentarzy są nawiasy klamrowe (jak w Pascalu).

```
Pobierz adres błędnego odwołania do strony
Pobierz wskaźniki do wykonywanego procesu i jego mapy pamięci
{te wskaźniki znajdują sie w odpowiednich polach
struktury mm_struct }
if (tryb przerwania) then goto no_context
{jeśli wykonujemy procedurę obsługi przerwania to nie
mamy dostępu do kontekstu procesu}
{mamy “dobry” kontekst procesu}
zablokuj strukturę mapy pamięci {semafor mm->mmap_sem}
znajdź obszar przestrzeni adresowej procesu do którego
należy błędny adres
/* wywołanie funkcji find_vma() */

if (brak obszaru | segment stosu | piszemy a nie mamy
praw | czytamy i nie mamy praw) then
{tutaj tak naprawdę siedzi instrukcja switch oparta na
flagach vma_flags i adresie vm_start }
begin
        odblokuj mapę pamięci {mm->mmap_sem}
        if (tryb użytkownika) {zmienna error_code}
        begin
            Segmentation fault{;-> znamy to, znamy...}
            return
        end
no_context:
przeszukaj tablice wyjątków w poszukiwaniu regs->eip
/*wywołanie funkcji search_exception_table(regs->eip)
*/
if (znaleziono) then
        begin
                ustaw regs->eip
                return
        end

        wypisz komunikaty o błędach
{Komunikat: Unable to handle kernel paging fault}
        sig_kill
```

```
end
else { chyba mamy właściwy obszar pamięci i kontekst procesu}
begin

survive:
        fault := handle_mm_fault()
{wysokopoziomowa procedura obsługi błędu}
        if ( not fault ) then
        begin
                SIGBUS {BUS ERROR}
                if (tryb jądra) then goto no_context
        end
        if (fault < 0 ) then
        begin
                if (pid=1) then
                 {jesteśmy procesem init}
                begin
                        schedule()
{dopóki nie zadziała handle_mm_fault() proces init będzie zasypiał}
                        goto survive

                end
                odblokuj mape pamięci {mm->mmap_sem}
                if (tryb_użytkownika) then
                        if (proces uprzywilejowany)
                        then SIGTERM
                        else SIGKILL
        end {fault<0}
end {survive}

end{do_page_fault}
```

4 Pomocnicze funkcje wykorzystywane przez algorytm

Funkcja do_page_fault korzysta z kilku funkcji pomocniczych, zwykle operujących na wyższym poziomie. Poniżej opisane są te najistotniejsze z punktu widzenia algorytmu (bez wgłębiania się np. w istotę wysłania do procesu sygnału itp.).

4.1 find_vma(mm,addr)

Funkcja find_vma wyszukuje w drzewie AVL spójnych obszarów pamięci wirtualnej pierwszy obszar taki, że jego koniec jest mniejszy niż poszukiwany adres.

(Kod źródłowy funkcji zawarty jest w pliku /usr/src/linux/mm/mmap.c)

4.2 handle_mm_fault (task, vma, adres, write)

Funkcja (znajdująca się w pliku /usr/src/linux/mm/memory.c) obsługuje z założenia przede wszystkim sytuacje: odwołanie do strony, która została przeniesiona do urządzenia wymiany (swapped out), próba zapisu do strony oznaczonej jako write protected - zwykle oznacza to że strona jest współdzielona z innym procesem i nie została jeszcze skopiowana /algorytm Copy On Write/ oraz próba odwołania się do nie istniejącej jak na razie strony. Funkcja ta najpierw oblicza adres logiczny strony, a następnie wywołuje funkcje handle_pte_fault() i zwraca jej wartość.

4.3 handle_pte_fault(task,vma,adres,write,pte)

Funkcja, znajdująca się w tym samym pliku, jest niemalże równie nieskomplikowana, bowiem sprawdza ona (przy pomocy makr: pte_present, pte_none, pte_write badających odpowiednie bity flag strony) czy strona w ogóle istnieje oraz czy jest w pamięci czy też nie. W zależności od wyniku tych sprawdzeń wywołuje jedną z trzech funkcji (wszystkie także zawarte w pliku memory.c):

4.3.1 do_no_page (task,vma,adres,write,pte)

Wywoływana jeśli dana strona nie istnieje. Próbuje stworzyć nowe mapowanie dla strony. Jeśli parametr write wskazuje na próbę zapisu to strona jest tworzona bez współdzielenia (jest to zrobione w ten sposób, że ustawia się bit dirty i bit write nowo utworzonej strony), wpp ma zastosowanie algorytm Copy On Write co oznacza, że jeśli to możliwe, strona będzie dzieliła fizyczną pamięć z jakąś już istniejącą , a skopiowana zostanie dopiero przy próbie zapisu.

4.3.2 do_wp_page (task,vma,adres,write,pte)

Wywoływana w przypadku, gdy proces próbował pisać do strony współdzielonej z innym procesem (COW). Funkcja pobiera nową stronę, sprawdza czy nie "wymieniono" ("wyswapowano") oryginalnej strony w czasie snu procesu. Jeśli tak, to "zrzuca z siebie odpowiedzialność" poprzez zwolnienie nowej strony i zwrócenie jedynki. Jeśli jednak strona jest dostępna, to jest ona kopiowana, po czym modyfikuje się mapę pamięci tak, aby istniało odwołanie do nowo utworzonej strony. (Dodatkowo funkcja ta sprawdza przed kopiowaniem, czy nie jest jedynym użytkownikiem strony, albo też czy nie dzieli strony tylko z pamięcią podręczną swap (swap cache), gdyż wtedy można uniknąć kopiowania).

4.3.3 do_swap_page (task,vma,adres,pte,entry,write)

Wywoływana gdy proces odwołał się do strony aktualnie wymienionej. Pobiera stronę z urządzenia wymiany, poprzez wywołanie funkcji swap_in(..). Wszelkie sprawdzenia dostępności miejsca i strony są wykonywane wewnątrz tej funkcji. Zwracany jest też jej wynik.

4.4 search_exception_table(addr)

Przeszukuje tablice wyjątków jądra i wszystkich jego modułów i zwraca znaleziony adres procedury , albo jeśli parametr addr nie występuje w tablicach wyjątków zwraca zero. (Źródło funkcji w pliku /usr/src/linux/arch/i386/mm/extable.c)

5 Bibliografia.

- Maurice j. Bach “Budowa systemu operacyjnego Unix”
- Abraham Silberschatz “Podstawy systemów operacyjnych”
- Projekt Linux
- Pliki źródłowe jądra 2.2.17.