Nowa implementacja pamięci wirtualnej w systemach rodziny BSD

Cezary Kaliszyk ck189400

16 grudnia 2002

Spis treści

1 Wstęp

1.1 Historia

Nowa implementacja pamięci wirtualnej UVM została napisana w 98 roku i wtedy została włączona do dystrybucji NetBSD. Po niedługim czasie została włączona do OpenBSD i kilku pomniejszych dystrybucji opartych na nich.

1.2 Problemy z poprzednimi implementacjami

Podstawowym problemem były łańcuchy obiektów w pamięci, na które wskaźniki mogły być gubione. Strony na które nie było wskaźników były przenoszone na dysk i po pewnym czasie przepełniały obszar wymiany.

Dodatkowo w przypadku operacji wejścia wyjścia miało miejsce wiele zbędnego kopiowania.

1.3 Nowe cechy

Udostępnianie stron innym procesom pozwala procesowi na udostępnienie w trybie tylko do odczytu pewnej swojej strony konkretnemu innemu procesowi lub jądru systemu. Pozwala to na uniknięcie kopiowania w przypadku przekazywania danych.

Przekazywanie stron pozwala jądru systemu na przekazanie strony pamięci do procesu. Pozwala to na uniknięcie kopiowania w przypadku przekazywania danych od jądra.

Przekazywanie i zamiana odwzorowań pamięci . Proces może posiadać kilka odwzorowań pamięci i dynamicznie przełączać się między nimi. Odwzorowania pamięci mogą być także dzielone między procesy. (Jest to naturalne rozszerzenie operacji przekazania deskryptora).

Częściowa dealokacja obszarów nienazwanych.

1.4 Ulepszone cechy

Ulepszone kopiowanie przy zapisie - Dwa poziomy referencji do obiektów zastępują łańcuchy długości od jeden do nieokreślonej. Pozwala to na uniknięcie gubienia wskaźników w przypadku kończenia się procesów.

Spójny obszar pamięci dla wejścia - wyjścia pozwala na łączenie wielu małych operacji w większe. (Szczególnie użyteczne w przypadku operacji na swapie).

Dynamiczny wybór sposobu iterowania przyspiesza wykonywanie operacji wymagających wyszukania konkretnej strony.

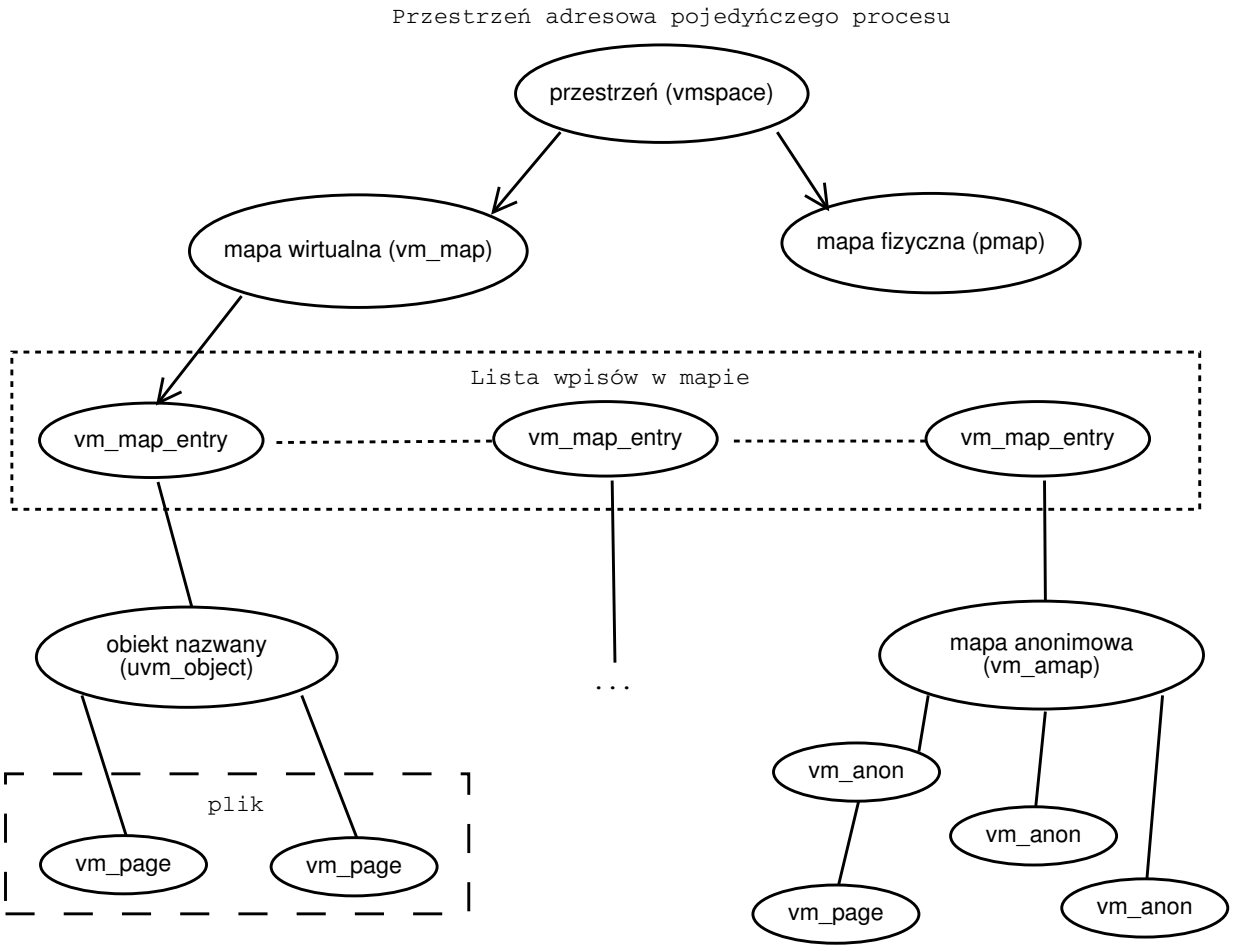
2 Implementacja

Implementacja pamięci wirtualnej w UVM jest podzielona na dwie części:

- Część zależna od sprzętu. Ta część jest identyczna jak w poprzednich implementacjach pamięci wirtualnej.
- Część niezależna od sprzętu

Dodatkowo funkcje i struktury które niewiele się zmieniły od implementacji z BSD vm nazywają się tak jak poprzednio to znaczy z prefiksem vm_, natomiast nowe nazywają się uvm_.

2.1 Struktury danych



Rysunek 1: Zarys struktur danych

2.1.1 uvm_map

Opis wirtualnej przestrzeni adresowej procesu, w pliku `uvm_map.h`

```
struct vm_map {
  struct pmap *           pmap;           /* Physical map */
  struct lock             lock;           /* Lock for map data */
  struct vm_map_entry     header;         /* List of entries */
  int                     nentries;       /* Number of entries */
  vsize_t                 size;           /* virtual size */
  int                     ref_count;      /* Reference count */
  struct simplelock       ref_lock;       /* Lock for ref_count field */
  ...
}
```

Pole `pmap` opisuje obszary pamięci fizycznej zmapowane bezpośrednio poprzez MMU.

W systemach BSD typ `lock` opisuje zamek który można zamykać na dwa sposoby (tylko do odczytu oraz do zapisu) oraz dodatkowo dostępna jest operacja przełączenia zamka między tymi trybami bez jego odmykania. Natomiast typ `simplelock` opisuje zwykły zamek.

2.1.2 vm_map_entry

Opisuje dostępny zmapowany obszar i jego atrybuty, w szczególności miejsce w pamięci fizycznej i obiekt

```
struct vm_map_entry {
  struct vm_map_entry     *prev;          /* previous entry */
  struct vm_map_entry     *next;          /* next entry */
  vaddr_t                 start;          /* start address */
  vaddr_t                 end;            /* end address */
  union {
    struct uvm_object  *uvm_obj;          /* uvm object */
    struct vm_map      *sub_map;          /* belongs to another map */
  } object;                               /* object I point to */
  ...
  struct vm_aref          aref;           /* anonymous overlay */
  ...
};
```

Lista jest utrzymywana w postaci posortowanej. Można zwrócić uwagę na brak stałego typu lista. W BSD każdy fragment kodu implementuje własną listę.

2.1.3 uvm_object

Obiekt jest niższym z poziomów odwzorowania. Może opisywać na przykład zmapowany plik albo urządzenie.

```
struct uvm_object {
  struct simplelock       vmobjlock;      /* lock on memq */
  ...
  struct pglist           memq;           /* pages in this object */
  int                     uo_npages;      /* # of pages in memq */
  int                     uo_refs;        /* reference count */
};
```

W UVM struktura obiekt nie jest główną strukturą służącą do posługiwania się obszarem pamięci wirtualnej, tylko jest trzymana przez większą mapę i operacje są wykonywane na całej mapie.

2.1.4 Pozostałe struktury

vm_aref - wskazuje na odpowiedni vm_amap i pozycję w nim.

vm_amap - opisuje odwzorowanie na anonimowe obszary pamięci (vm_anon).

vm_page - opisuje pojedynczą stronę fizycznej pamięci. Dla każdej strony istnieje taka struktura.

2.2 Obsługa wyjątku braku strony

Funkcja `uvm_fault` jest wywoływana przez część zależną od architektury. Dzieje się tak kiedy MMU został zapytany o adres który nie jest zmapowany w pamięci fizycznej.

Funkcja `uvm_fault` dostaje adres, mapę pamięci, typ błędu (nie zmapowane czy innego procesu) oraz czy zapis.

Czynności wykonywane:

1. Sprawdzamy czy dane są anomimowe lub są uvm_object
2. Jeśli nie zgłaszamy błąd
3. Obsługujemy copy-on-write
4. Mapujemy
5. Zwracamy

2.3 Fork

Każde odwzorowanie obszaru pamięci posiada dodatkowo pole `inherit` które mówi w jaki sposób mapa zostanie przekazana dzieciom procesu. Może ono mieć jedną z wartości:

- none - proces potomny nie ma dostępu do danej mapy procesu głównego.
- share - proces potomny współdzieli tą mapę z procesem macierzystym.
- copy - proces potomny ma operować na kopii mapy procesu głównego. Kopia jest wykonywana w modelu copy-on-write.

Przestrzeń adresowa procesu potomnego jest tworzona przez funkcję `uvmspace_fork`. Funkcja ta przechodzi całe drzewo procesu macierzystego i dla każdej mapy w przestrzeni adresowej procesu macierzystego dodaje odpowiednie dowiązania w procesie potomnym.

W przypadku map które zaczynają być współdzielone między procesy jest ustawiana dodatkowa flaga `shared`, która oznacza, że wszystkie poważne zmiany w mapie muszą być wykonywane w mapach wszystkich procesów związanych z tą mapą.

W przypadku kopiowania mapy najpierw dostaje ona dodatkową referencję, (wartość `reference_count` jest zwiększana), a przy modyfikacji wystarczy zmodyfikować kopię mapy.

3 Nowe operacje

3.1 Pożyczanie Stron

Wypożyczona strona jest zmieniana na tylko-do-odczytu i zwiększany jest licznik procesów które wypożyczyły daną stronę. Tworzona jest tablica wskaźników do anonimowych struktur która może być wpisana do vm_amap procesu docelowego.

Operacja ta ma 4 podstawowe przebiegi:

- obiekt jest wypożyczany do jądra systemu
- struktura anonimowa jest wypożyczana do jądra systemu
- wypożyczenie obiektu jako struktury anonimowej
- wypożyczenie struktury anonimowej jako takiej

W dwóch pierwszych przypadkach strony muszą być rezydentne i muszą zostać przywiązane, aby pagedaemon ich nie wyswapował. Jądro nie może próbować się dostać do strony w swapie, gdyż wtedy nie dałoby się obsłużyć wyjątku braku strony.

Wypożyczenie obiektu wymaga specjalnych więzów spójności, aby obiekt (a nie struktura anon) był właścicielem swojej strony. Dodatkowo przy zwolnieniu obiektu to pewna struktura anon ma się stać właścicielem tych stron. Dlatego obiekt może być wypożyczony

tylko raz. Dodatkowo próba dostępu do stron tej struktury anon wymaga zamknięcia muteksa na obiekcie.

Ostatni przypadek jest najprostszy. Wymaga tylko pod zamkniętym muteksem zwiększenia licznika dowiązań do struktury anonimowej i ewentualnie włączenia ochrony przed zapisem dla strony.

3.2 Przekazywanie stron

Operacja przekazania strony bierze strony posiadane przez jądro lub strony anonimowe i umieszcza je w podanym przez docelowy proces witrualnym adresie lub w obszarze pamięci wirtualnej zarezerwowanym specjalnie na cel przekazywania stron. Po zakończeniu operacji pamięć przekazana może być traktowana dokładnie tak samo jak inne strony pamięci anonimowej.

Dobrym przykładem działania mechanizmu przekazywania stron może być odczytywanie pliku z dysku. Jądro odczytuje fragment pliku do własnej pamięci. Następnie stronę przekazuje do mapy procesu który wywołał read. Proces ten może korzystać z przekazanej sobie strony, tak jakby normalnie była dla niego stworzona. Tylko zwolnienie odbywa się inaczej o czym za chwilę.

Przekazywanie anonimowych stron między procesami jest na przykład używane przez IPC. Wysłanie komunikatu może być naturalnie zaimplementowane jako przekazanie strony zawierającej wiadomość.

Istnieje dodatkowo specjalna funkcja `anflush`, która służy do zwalniania stron przekazanych. Działa tak, że daje procesowi który otrzymał anonimową stronę do zwolnienia tej strony, bez zwalniania amapy. Proces używa tego wywołania kiedy będzie chciał dalej otrzymywać strony.

3.3 Współdzielenie map

Jest szybsze niż inne sposoby komunikacji bo nie kopiuje danych, a wygodniejsze niż pamięć współdzielona, bo obszar nie musi być na stałę zaalokowany na początku. Dzięki temu można na przykład współdzielić dynamiczną strukturę danych, na której są wykonywane operacje malloc i free.

Dla każdego obszaru który chcemy udostępnić wołamy funkcję `mexport` która udostępnia ją odpowiednim procesom, z drugiej strony należy wywołać odpowiednią funkcję `mimport`.

Obszar jest wyciągany z mapy (`uvm_map_extract`) i umieszczany w specjalnej eksportowanej mapie. Importowanie polega na rezerwacji obszaru (`uvm_map_reserve`) i umieszczeniu tam mapy.

Współdzielenie map pozwala na przykład na przekazanie innemu procesowi tylko części pliku co jest naturalnym rozszerzeniem operacji przekazania deskryptora.