Zarządzanie pamięcią

Pamięć logiczna

W systemie może jednocześnie działać wiele programów. Muszą one przebywać w pamięci, żeby CPU miał skąd brać kolejne instrukcje. Dane i instrukcje leżą w pamięci pod jakimiś adresami. O pamięci można myśleć jak o tablicy z kolejnymi adresami. Podczas kompilacji programu musielibyśmy znać dokładne adresy (wymagałoby np. kompilacji tuż przed wykonaniem jeśli adres początkowy programu byłby inny).

Wygodniej rozdzielić pamięć dodatkową warstwą abstrakcji. CPU (m. in.) i programista mogą operować na **pamięci logicznej**, która abstrahuje od innych programów i adresów ładowania.

Przy każdym odwołaniu do **adresu logicznego** musi być ono przetłumaczone na **adres fizyczny** (ten, na którym operuje jednostka pamięci). Translacji dokonuje **urządzenie sprzętowe MMU (Memory-Management Unit)**, które do adresu logicznego dodaje wartość z **rejestru przemieszczenia** (rejestr relokacji = fizyczny adres początkowy)

Kompilacja i nie tylko

- program źródłowy =*kompilacja*=> moduł wynikowy
- >=1 modułów wynikowych =*konsolidacja*=> moduł ładowalny
- moduł ładowalny =*ładowanie*=> binarny obraz programu

Wiązanie adresów

Program leży na dysku. Trzeba **wprowadzić go do pamięci** (*bring into memory*) i uczynić częścią procesu, by móc go wykonać. Procesy czekają na wprowadzenie do pamięci w **kolejce wejściowej** (*input queue*).

Wiązanie adresów:

- w **fazie kompilacji** (*compile time*) - jeśli adresy są znane z góry, można wygenerować **kod bezwzględny**
- w **fazie ładowania** (*load time*) - jeśli adres ładowania nie jest znany podczas kompilacji, należy wygenerować **kod przemieszczalny** (*relocatable code*)
- w **fazie wykonywania** (*execution time*) - wiązanie może być odłożone aż do wykonania, jeśli

proces można przenosić w trakcie działania.

Konsolidacja dynamiczna (*dynamic linking*)

- Konsolidacja odkładana do fazy wykonywania.
- Mały fragment kodu - namiastka (*stub*) - jest używany do odnalezienia odpowiedniego podprogramu w bibliotece rezydującej w pamięci.
- Namiastka, używając adresu danego podprogramu, wywołuje go i na tym kończy się jej działanie.
- SO sprawdza czy podprogram należy do przestrzeni adresowej procesu.
- Konsolidacja dynamiczna jest szczególnie przydatna w przypadku bibliotek podprogramów (funkcji, obiektów).

Ładowanie dynamiczne

- Podprogram ładujemy leniwie, kiedy potrzebujemy.
- Do realizacji tego nie potrzeba pomocy SO, wystarczy odpowiednie zaprojektowanie programu.

Nakładki

Jakieś stare rozwiązanie sprzed wirtualizacji pamięci. Polegało na tym, że dzielimy program na takie nakładki, by nie używać wszystkiego na raz (bo mało pamięci). Wykorzystujemy jedną nakładkę i podmieniamy, gdy chcemy inną.

- Przechowujemy w pamięci tylko te instrukcje i dane, które są potrzebne w danym czasie.
- Potrzebne i przydatne, gdy proces jest większy niż przydzielona mu pamięć.
- Nakładki są implementowane przez użytkownika, nie jest tu konieczna żadna specjalna pomoc ze strony systemu. Zaprojektowanie i zaprogramowanie struktury nakładek jest rzeczą skomplikowaną.

Wymiana

Procesy mogą być przenoszone (**wymieniane** - *swapped*) z pamięci operacyjnej do pamięci pomocniczej (**wytaczanie** - *roll out*), po czym znów sprowadzane (**wtaczane** - *roll in*) i kontynuowane.

Wymiana w systemach mobilnych

Raczej nie jest realizowana, gdyż pamięć flash w nich wykorzystywana jest nieduża, ma ograniczoną liczbę cykli zapisu i prędkość przesyłu z pamięci flash do CPU jest dość wolna.

- iOS prosi aplikacje o dobrowolne wyzbycie się pamięci
- Android kończy apki, gdy brakuje pamięci, uprzednio zapisując ich stan w celu szybkiego wznowienia

Przydział pamięci

W najprostszym przypadku pamięc operacyjna jest podzielona na:

- **pamięć dolną** (*low memory*) - gdzie siedzi SO z wektorem przerwań
- **pamięć górną** (*high/upper memory*) - trzymającą procesy użytkowników

Przydział ciągły

Każdy proces dostaje ciągły blok pamięci, opisany przez:

- **rejestr przemieszczenia** (*relocation reg.*) - najmniejszy adres fizyczny
- **rejestr graniczny** (*limit reg.*) - maksymalny adres logiczny

Które to rejestry opisują gdzie się blok zaczyna i jak długi jest. Blok musi być na tyle duży by się proces zmieścił.

System utrzymuje informacje o:

- obszarach przydzielonych
- wolnych obszarach - **dziurach**

Dynamiczny przydział pamięci

Polityka wyboru dziur:

- **pierwsze dopasowanie** (*first-fit*) - pierwsza wystarczająca
- **najlepsze dopasowanie** (*best-fit*) - najmniejsza wystarczająca
- **najgorsze dopasowanie** (*worst-fit*) - największa wystarczająca

Fragmentacja

Dzieli się na:

- **zewnętrzną** (*external*) - zamówienie zrealizowane kilkoma dziurami
- **wewnętrzną** (*internal*) - dziura większa niż potrzeba, nieużytki są

Stronicowanie

A co w sytuacji, gdy proces rośnie i przekracza swój rozmiar? Większa dziura musiałaby być znaleziona i pamięć przepisana. Tak nie chcemy. Zamiast tego przestrzeń adresów logicznych może być nieciągła.

Możemy podzielić pamięć na bloki i przydzielać procesom ileś bloków i dosyłać dodatkowe gdy potrzebują czy zwalniać jeśli zbędne. Te bloki nazywamy:

- **ramkami** (*frames*) - gdy mówimy o pamięci fizycznej
- **stronami** (*pages*) - gdy mówimy o pamięci logicznej

Długości bloków i stron są takie same, by odpowiadały sobie nawzajem. Trzymamy informacje o użytych i wolnych blokach. Tworzymy **tablicę stron** (*page table*) do tłumaczenia adresów logicznych na fizyczne.

Tablica stron

Tablica odwzorowująca numer strony na numer ramki.

Translacja adresu logicznego na fizyczny wygląda następująco:

- Adres logiczny bitowo dzieli się na **numer strony** (*page number*) i **odległość od początku strony**(*offset*). Ilość bitów do zapisu jednego i drugiego zależy od wielkości bloków (stron, ramek): `odległość <- [0, długość_bloku - 1]`.
- Numer strony tłumaczony na numer ramki przy użyciu tablicy stron.
- Adres fizyczny otrzymuje się przez wstawienie w miejsce numeru strony numeru ramki. Odległość (offset) zostaje taka sama.

Implementacja:

- Tablica stron siedzi w pamięci operacyjnej
- **Rejestr bazowy tablicy stron** (*page-table base register*) wskazuje tablice stron
- **Rejestr długości tablicy stron** (*page-table length register*)
- Dwa dostępy do pamięci potrzebne:

    i. do przetłumaczenia adresu logicznego do adresu fizycznego X, potrzeba sprawdzić numer ramki
    ii. po wartość pod adresem X
- Ochrona do pamięci realizowana przez trzymanie dla każdego indeksu **bitu poprawności**

Pamięć podręczna dla tablicy stron

Nie chcemy dwóch dostępów do pamięci dla jednej danej. Wykorzystujemy mniejszą, ale szybszą pamięć (blisko CPU), w której zapamiętujemy 'ostatnie' zapytania z tablicy stron (translacja adresów może nie wymagać patrzenia do tablicy stron).

Przykład: jeśli **przeglądamy zwykłą tablicę** (np. intów) i **siedzi ona cała na jednej *stronie*** (w jednej *ramce*) w pamięci to **dla pierwszej zmiennej** wykonamy **dwa dostępy** do RAMu (jeden do tablicy stron, drugi pod adres fizyczny zmiennej), a **dla reszty tylko jeden**, bo ta strona będzie już w keszu (zpl. *kaszu*, eng. *cache*), więc będziemy znali ich adresy fizyczne.

Rodzaje implementacji:

- **TLB** (***translation look-aside buffers***)
- Pamięć asocjacyjna

Struktury tablicy stron

- **Stronicowanie hierarchiczne** (*hierarchical paging*)
    - Jedna wielka tablica stron jest niepraktyczna, będzie bardzo dużo miejsca zajmować, nawet jeśli będziemy korzystali tylko z kilku stron. Zamiast tego lepsza jest kilkupoziomowa tablica stron
    - **Intuicja:** zamiast drzewa z korzeniem i samymi liśćmi (jednopoziomowa tablica stron), mamy drzewo k-arne z być może pustymi wierzchołkami jeśli liście (numery ramek) na końcu ścieżki nie są używane (zaalokowane)
- **Haszowane tablice stron** (*hashed page tables*)
    - tablica stron zaimplementowana jako hash mapa (tablica z hashowaniem)
    - indeksami są hashowane adresy logiczne
    - kolizje nawlekamy na listę
    - wartościami są struktury: (numer strony, numer ramki, wskaźnik na następny)
- **Odwrócone tablice stron** (*inverted page tables*)

Segmentacja

Dzielenie pamięci na logicznie pogrupowane segmenty. Np. procedura, zmienne globalne, stos, tablica symboli ...

Reszta zasad działania podobnie jak przy tablicy stron, tylko tu tablica segmentów.