Systemy operacyjne

Ćwiczenia 6

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Stallings: 7.1 - 7.4, 7A, 8.1
- Tanenbaum: 3.1 - 3.3
- Silberschatz: 8

Zadania 7-8 opracuj na podstawie rozdziału 31 książki *"Memory Systems: Cache, DRAM, Disk"*.

Zadanie 1

Opisz pobieżnie mechanizmy **segmentacji** oraz **stronicowania** (jednopoziomowa tablica stron). Dla obu metod przedstaw funkcję **translacji adresów** (w pseudokodzie) i strukturę danych wymaganą do jej działania. Jakie problemy występujące przy **ciągłym przydziale pamięci fizycznej** rozwiązują te mechanizmy? Porównaj segmentację z **ciągłym przydziałem obszarów o zmiennej długości**. Dlaczego i w tym przypadku segmentacja jest lepsza? Do czego służy **kompaktowanie**?

Zadanie 2

Rozważmy zarządzanie pamięcią w przestrzeni użytkownika – tj. funkcje biblioteczne `malloc` i `free` (to nie są wywołania systemowe!). Systemy operacyjne pozwalają procesom na przydział dodatkowej pamięci na użytek bibliotecznego alokatora pamięci. Opisz wywołania systemowe **sbrk** oraz **mmap** – wyjaśnij czemu bieżące implementacje alokatorów używają drugiego. Wyjaśnij czym jest **fragmentacja zewnętrzna** i **wewnętrzna** w kontekście wyżej wymienionych metod przydziału.

Zadanie 3

Wyjaśnij algorytmy zarządzania wolną pamięcią w ciągłym przydziale: **lista bloków**, **mapa bitowa bloków** i **system bliźniaków**. Na czym polega polityka **first-fit**, **next-fit**, **best-fit**, **worst-fit**. Wskaż, które z tych metod są lepsze do zarządania blokami małymi lub dużymi, stałej lub zmiennej długości. Jakie są oczekiwane złożoności czasowe i pamięciowe (narzut struktur danych) dla poszczególnych algorytmów.

Zadanie 4

W jakim celu wykorzystuje sie technikę **złączania nieużytków**? Bazując na metodzie **quick-fit** zaproponuj hybrydową implementację alokatora. Możesz wykorzystać dodatkowe struktury danych (np. `splay`, `red-black`, `b-tree`) oraz **gorliwe** lub **leniwe złączanie** nieużytków. Jak będzie działać `malloc` i `free`? Jak będą wyglądały struktury danych przechowujące wolne bloki?

Zadanie 5

Biblioteczny alokator pamięci dysponuje `N` bajtowym obszarem pamięci. Dostarcza funkcji o sygnaturach `malloc: size -> #id`[1] i `free: #id -> void`. Implementacja bazuje na dwukierunkowej liście wolnych bloków posortowanych względem adresu i zadanej polityce. Alokator gorliwie złącza bloki. Pomijając narzut związany z utrzymywaniem nagłówka bloków, jak będzie wyglądać zarządzany blok pamięci po wykonaniu żądań dla polityki: `best-fit`, rozmiaru obszaru: `50`, ciągu żądań: `5 14 20 4 #2 #1 #3 7`.

Zadanie 6

Wykonaj polecenia z poprzedniego zadania dla polityki: `first-fit`, rozmiar obszaru: `40`, ciągu żądań: `5 4 8 13 #2 8 #4 #1 3 #3`.

Zadanie 7

Porównaj mechanizm translacji adresów wykorzystujący **hierarchiczną tablicę stron** i **odwróconą tablicę stron**. Wytłumacz pobieżnie jak przebiega proces tłumaczenia adresu z wirtualnej przestrzeni adresowej do fizycznej. Jakiej wielkości są struktury opisujące przestrzeń adresową pojedyńczego procesu? Czemu służą **ogromne strony**?

Zadanie 8

Rozważ stronicowanie z hierarchiczną tablicą stron. Czemu służy pamięć podręczna **TLB**? Jakie korzyści wynikają z jej użycia. Biorąc pod uwagę pamięć podręczną **indeksowaną wirtualnie** i **tagowaną fizycznie** oraz TLB wyjaśnij w jaki sposób mogą powstać **homonimy** i **synonimy**? Jakie inne problemy zauważasz przy zmianie przestrzeni adresowej przy przełączaniu procesów?

[1] Unikalny identyfikator bloku, który w praktyce jest adresem pierwszej komórki bloku. Bloki przydzielane przez `malloc` są ponumerowane kolejnymi liczbami naturalnymi.