ISIM: ASK + SO

Ćwiczenia 9: "Zarządzanie pamięcią"

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- "*Modern Operating Systems*" (czwarta edycja); Tanenbaum; 3.1 – 3.6
- "*Operating Systems – Internals and Design Principles*" (siódma edycja); Stallings; 7, 8 (opcjonalnie)
- Dynamic Storage Allocation: A Survey and Critical Review (wybrane ustępy, pomocniczo)

Należy być przygotowanym do wytłumaczenia **wytłuszczonych** haseł.

Zadanie 1

Systemy operacyjne udostępniają wywołania systemowe do przydziału pamięci na użytek bibliotecznego alokatora pamięci (`malloc` i `free`). Opisz wywołania systemowe **sbrk** oraz **mmap** – wyjaśnij czemu bieżące implementacje alokatorów używają tego drugiego. Wyjaśnij czym jest **fragmentacja zewnętrzna** i **wewnętrzna** w kontekście wyżej wymienionych metod przydziału. Jakie zachowania programów sprzyjają powstawaniu fragmentacji? Czy dla powyższego typu alokatora można dokonywać **kompaktowania** pamięci?

Zadanie 2

Wyjaśnij algorytmy zarządzania pamięcią na ciągłym obszarze: **lista bloków**, **mapa bitowa bloków** i **system bliźniaków** (ang. *buddy system*). Które z tych metod są lepsze do zarządania blokami małymi lub dużymi, stałej lub zmiennej długości? Dla każdego z nich – jakie są oczekiwane złożoności czasowe i pamięciowe (narzut struktur danych). W jakim celu wykorzystuje sie technikę **złączania nieużytków**? Kiedy gorliwe złączanie nieużytków może być niekorzystne dla wydajności programu korzystającego z danego alokatora?

Zadanie 3

Biblioteczny alokator pamięci dysponuje `N` bajtowym obszarem pamięci. Dostarcza funkcji o następujących sygnaturach – `malloc: size → #id`[1], `free: #id → ()`. Implementacja bazuje na dwukierunkowej liście wolnych bloków posortowanych (czemu?) 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 4 [bonus]

Bazując na metodzie **quick–fit** zaproponuj hybrydową implementację alokatora typu `malloc` i `free`. Możesz wykorzystać dodatkowe struktury danych (np. *splay*, *red-black tree*, *b-tree*) oraz złączanie nieużytków. Jak będzie działać `malloc` i `free`? Jak będą wyglądały struktury danych przechowujące wolne bloki? Co zrobić by alokator zbędnie nie zaśmiecał pamięci podręcznej (ang. *cache pollution*)?

Zadanie 5

Rozważmy **procedurę obsługi braku strony**, do której przekazywane jest sterowanie w wyniku niewłaściwego odwołania do pamięci. Jakie dane musi otrzymać ta funkcja by prawidłowo wykonywać swoją pracę? Zaproponuj pseudokod procedury umożliwiającej **stronicowanie na żądanie** z uwzględnieniem uprawnień dostępu do stron. Wykorzystaj następujący przykładowy interfejs:

- `page_t *find_pte(void *virt_addr);`
- `void error(const char *msg);`
- `frame_t *alloc_frame();`
- `void read_block(frame_t *frame, int block_addr);`
- `void resume(void *pc);`

Czy oprócz tablicy stron musisz zarządzać jakąś dodatkową strukturą danych?

[1] Tutaj unikalny identyfikator bloku – w praktyce adres pierwszej komórki bloku. Bloki przydzielane przez `malloc` są ponumerowane kolejnymi liczbami naturalnymi.

Zadanie 6

Pokaż w pseudokodzie jak należy rozszerzyć procedurę obsługi braku stron z poprzedniego zadania celem implementacji **wymiany** i **mapowania plików** na pamięć. Czym różni się **pomniejszy błąd strony** od **głównego błędu strony**? Załóż, że posiadasz wiele plików (każdy z unikalnym identyfikatorem *i-node*) i jedną **partycję wymiany**. Zaproponuj prosty interfejs (zbiór funkcji) podobny do tego z poprzedniego zadania. Jeśli wprowadzono dodatkowe struktury danych – opisz je i uzasadnij ich niezbędność.

Zadanie 7 [bonus]

Wyjaśnij zasadę działania mechanizmu **kopiowania przy zapisie** w systemach ze stronicowaniem. Jakie korzyści przynosi? Do czego używa się go w praktyce? Pokaż w pseudokodzie jak rozszerzyć procedurę obsługi braku stron. Zaproponuj prosty abstrakcyjny interfejs. Jeśli wprowadzono dodatkowe struktury danych lub rozszerzono istniejące – opisz je i uzasadnij ich niezbędność.

Zadanie 8

Rozważmy cztery klasy polityk zarządzania pamięcią wirtualną. Wyjaśnij kiedy mają zastosowanie **polityka ładowania** (ang. *fetch policy*), **przydziału miejsca** (ang. *placement policy*), **wymiany** (ang. *replacement policy*) i **usuwania** (ang. *cleaning policy*)? Jakie są wady i zalety stronicowania na żądanie i **stronicowania wstępnego** (ang. *prepaging*)?

Zadanie 9

Wyjaśnij pojęcia **zbioru roboczego** i **zestawu rezydentnego**. Jak te zbiory zmieniają się w czasie działania procesu i jak to się ma do zasady lokalności? Rozważmy **buforowanie stron** – tj. jeśli strony zostały oznaczone do zastąpienia, to nie są usuwane od razu, ale trzyma się je przez jakiś czas w pamięci operacyjnej. Wymień zalety tego rozwiązania w kontekście lokalności i zarządzania zbiorem roboczym, oraz polityki usuwania stron. Jak z buforowania stron może skorzystać algorytm zarządzający **zbiorem rezydentnym zmiennego rozmiaru** działający w obrębie całego systemu?

Zadanie 10

Rozważmy system ze **stałym przydziałem zbioru rezydentnego**. Pamięć fizyczna składa się z 4 ramek, a pamięć wirtualna z 8 stron. Na maszynie wykonuje się jeden proces i generuje następujący ciąg dostępów do stron: `7 0 1 2 0 3 0 4 2 3 0 3 2`. Pokaż działanie następujących algorytmów zastępowania stron: **OPT**, **FIFO**, **CLOCK**, **LRU**. Jak te algorytmy korzystają z pomocniczych bitów w deskryptorach stron? Który z nich generuje najmniej braków stron? Załóż, że zastąpienie ramki zachodzi dopiero w momencie zapełnienia całej pamięci fizycznej.