

Systemy operacyjne (zaawansowane)

Lista zadań nr 7

Na zajęcia 7 grudnia 2017

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 3.2, 10.4
- Stallings (wydanie ósme): 7.2

UWAGA! W trakcie prezentacji rozwiązań należy zdefiniować i wyjaśnić pojęcia, które zostały oznaczone **wytłuszczoną czcionką**. Dla każdej metody przydziału pamięci, która pojawi się w poniższych zadaniach, prowadzący zajęcia może wymagać by student wyjaśnił:

- jak wygląda struktura danych przechowująca informację o zajętości bloków?
- jak przebiegają operacje **alloc** i **free**?
- jaka jest oczekiwana złożoność czasowa powyższych operacji?
- jaki jest narzut (ang. *overhead*) pamięciowy **metadanych**?
- jaki jest maksymalny rozmiar **nieużytków**?
- czy w danym przypadku **fragmentacja wewnętrzna** lub **zewnętrzna** jest dużym problemem?

Pamiętaj, że algorytmy zarządzania pamięcią mogą korzystać tylko i wyłącznie z uprzednio przydzielonych ciągłych obszarów pamięci – również do przechowywania dynamicznych struktur danych. Jakiegokolwiek mechanizmy przydziału pamięci poza tymi obszarami są niedozwolone!

Zadanie 1. Rozważmy listowy algorytm zarządzania pamięcią z polityką **first-fit**, gdzie wolne bloki posortowano po rosnącym rozmiarze. W jakim celu wykorzystuje się **scalanie** (ang. *coalescing*) wolnych bloków? Jak, porównując do *first-fit* z sortowaniem po adresie bloku, wybór **polityki przydziału miejsca** (ang. *placement policy*) wpłynął na algorytm zwalniania bloku? Jaki wpływ na algorytm przydziału miałyby **leniwa strategia** scalania?

Zadanie 2. Rozważmy algorytm listowy **best-fit** z wolnymi blokami trzymanymi w kolejce **FIFO** (ang. *first-in first-out*). W trakcie scalania bloku chcemy w $O(1)$ znaleźć następnika oraz poprzednika i jeśli są wolne złączyć z naszym blokiem. Jak zmodyfikować algorytm przydziału i zwalniania? Jakich dodatkowych metadanych potrzebujesz?

Podpowiedź: Technika ta jest znana pod nazwą „boundary tag” i została opisana w rozdziale 2.5 książki „The Art of Computer Programming: Volume 1 / Fundamental Algorithms”, Donald E. Knuth.

Zadanie 3. Rozważmy algorytm listowy **next-fit** z wolnymi blokami posortowanymi po adresie. Chcemy zaprogramować algorytm przydziału pamięci wyłącznie dla ciągów znakowych nie dłuższych niż 100 bajtów. Możemy przyjąć, że **obszary** (ang. *chunks*) mają maksymalnie 16 stron, a wskaźniki zwracane przez **malloc** nie muszą być **wyrównane** (ang. *aligned*). Rozwiąż poniższe problemy:

- Chcemy efektywnie kodować metadane (tj. minimalizować **narzut pamięciowy**), dla algorytmu listowego przyjmując, że będzie on działał na architekturze 64-bitowej.
- Program korzystający z naszego algorytmu będzie modyfikował zawartość przydzielonych bloków pamięci – chcemy skutecznie i szybko wykrywać błędy zapisu poza koniec ciągu znaków.

Podpowiedź: Zauważ, że przy podanych ograniczeniach metadane można dość skutecznie skompresować.

Zadanie 4. Pokaż jak przy pomocy **algorytmu kubełkowego** (ang. *segregated-fit*, *quick-fit*) przyspieszyć działanie operacji przydziału bloku. Zaproponuj modyfikację tego algorytmu używając innych struktur danych niż lista dwukierunkowa. Odpowiedz na pytania z nagłówka listy! Czy **gorliwe złączanie** wolnych bloków jest w tym przypadku dobrym pomysłem?

Podpowiedź: Rozważ drzewa zbalansowane, np. czerwono-czarne (ang. *red-black*) lub rozchylane (ang. *splay*).

Zadanie 5. Rozważmy **algorytm bitmapowy** przydziału pamięci dla rekordów ustalonego rozmiaru. Pokaż jak przyspieszyć wyszukiwanie wolnych bloków z użyciem wbudowanej instrukcji procesora **find-first-set**¹ i drzewiastej struktury mapy bitowej. Odpowiedz na pytania z nagłówka listy! Jak ulepszyć algorytm przydziału biorąc pod uwagę, że niedawno zwolnione bloki prawdopodobnie znajdują się w pamięci podręcznej procesora?

Zadanie 6. Rozważmy **algorytm bliźniaków** poznany na wykładzie. Zarządzamy blokiem 16 stron. Przyjmijmy, że listy bloków stron są posortowane rosnąco po adresie. Możemy przydzielać i zwalniać wyłącznie bloki 2^k stron wyrównanych do adresu podzielonego przez 2^k . Posługując się strukturą danych z wykładu pokaż, krok po kroku, jak przebiega ciąg operacji *alloc*(8), *alloc*(4), *free*(2₆), *free*(2₈), *free*(2₄), *free*(1₁₀). Ile pamięci straciliśmy w wyniku **fragmentacji zewnętrznej** pamięci fizycznej? Czy można użyć w tym przypadku **kompaktowania** by ją zniwelować?

Zadanie 7 (P). Często używaną metodą zarządzania pamięcią w jądrach systemów operacyjnych jest **algorytm płytowy** (ang. *slab allocator*). Na podstawie materiałów do wykładu **Zaawansowane Systemy Operacyjne**² z MIMUW³ odpowiedz na pytania:

- Jakie przewagi ma algorytm płytowy nad pozostałymi metodami przydziału pamięci?
- Jak zorganizowane są struktury algorytmu płytowego?
- Przedstaw zawartość pliku `/proc/slabinfo` na swoim systemie. Wskaż **dedykowane** i **ogólne pamięci podręczne** obiektów, a następnie wyjaśnij różnice między nimi.

Zadanie 8. Rozważmy zarządzanie pamięcią w środowisku wielowątkowym. Czemu algorytmy przydziału pamięci często cierpią na zjawisko **rywalizacji o blokady** (ang. *lock contention*)? Dlaczego błędnie zaprojektowany algorytm przydziału może powodować problemy z wydajnością programów wynikających z **fałszywego współdzielenia** (ang. *false sharing*)? Opisz pobieżnie struktury danych i pomysły wykorzystywane w bibliotece **thread-caching malloc**⁴. Wyjaśnij jak zostały użyte do poradzenia sobie z problemem efektywnego przydziału pamięci w programach wielowątkowych?

¹https://en.wikipedia.org/wiki/Find_first_set

²http://students.mimuw.edu.pl/ZS0/Wyklady/05_pamiec/5_pamiec.html

³Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego

⁴<http://goog-perftools.sourceforge.net/doc/tcmalloc.html>