Systemy operacyjne

Lista zadań nr 7

Na zajęcia 27–28 listopada 2018

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 należy być gotowym do zdefiniowania pojęć oznaczonych **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?

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

Zadanie 1. Biblioteczny algorytm przydziału pamięci dysponuje ciągłym obszarem 50 słów maszynowych. Dostarcza funkcji o sygnaturach «`malloc: size -> @id`» i «`free: @id -> unit`». Implementacja wykorzystuje dwukierunkową listę wolnych bloków posortowanych względem adresu. Wyszukiwanie wolnych bloków działa zgodnie z polityką **best-fit**. Operacja zwalniania **gorliwie złącza** bloki. Zasymuluj działanie alokatora dla poniższego ciągu żądań:

```
5 14 20 4 @2 @1 @3 7
```

Wskazówka: Przejrzyj poprawione slajdy do wykładu. Rozwiąż zadanie podobnie, ale po jednym kroku na operację.

Zadanie 2. 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? Porównując do *first-fit* z sortowaniem po adresie bloku – jak wybór **polityki przydziału miejsca** (ang. *placement policy*) wpłynął na algorytm zwalniania bloku? Jaki wpływ na algorytm przydziału miałaby **leniwa strategia** scalania?

Zadanie 3. Rozważmy algorytm listowy **best-fit** bez ustalonego porządku listy wolnych bloków (np. zawsze dodajemy na początek). 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 / Fundametal Algorithms", Donald E. Knuth.

Zadanie 4. 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 efektywne 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 5. 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 6. 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. Uwzględnij, że blok pamięci podręcznej ma 64 bajty. 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 7. 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`[2]. Wyjaśnij jak zostały użyte do poradzenia sobie z problemem efektywnego przydziału pamięci w programach wielowątkowych?

[1] `https://en.wikipedia.org/wiki/Find_first_set`
[2] `http://goog-perftools.sourceforge.net/doc/tcmalloc.html`