DOROBIĆ: do czego jest algo bliźniaków? -> Zarządzanie wolnymi stronami pamięci fizycznej w jądrze SO. kompaktowanie w szóstym - kompaktowwanie ok bo pamięć wirtualna translacja adresów dorobić z tego wykładu Kolejne strony na adresach podzielnych przez rozmiar Te kubełki to rozmiary stron

i z tymi blizniakami to troche inaczej: nie patrzymy w ogole na drzewo, ono ma nam tylko ewentualnie pomoc mamy listę wolnych blokow , zaczynamy od kubełka, z ktorego wystaje 16 chcemy zaalokowac 8 - szukamy, idać od dołu (1, 2, 4...), gdzie mozna znalezc wolny blok o wielkosci 8 nie mamy wolnego bloku w kubełku 8, idziemy wyzej do wolnego 16, łamiemy go na pół i dorzucamy wolnego bliźniaka do 8

chcemy zwolnic 4 - dodajemy 4 do kubełka z 4 i elo

jesli mamy dwoch blizniakow na liscie wolnych blokow, to sprawdzamy, czy ich adresy roznia sie jedną pozycją - wtedy mozna ich scalic

L7.Z0.

Dla każdej metody przydziału pamięci, która pojawi się w poniższych zadaniach

- 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?

metadane - dane o danych.

nieużytki - powstają we fragmentacji wewnętrznej.

L7.Z1.

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 miałaby leniwa strategia scalania?

Polityka first-fit, gdzie wolne bloki posortowano po rosnącym rozmiarze, to po prostu best-fit.

First-fit: Menedżer pamięci skanuje listę segmentów tak długo, aż znajdzie wolny blok o odpowiedniej wielkości. Ten blok jest następnie dzielony na dwie części - jedna zostaje przeznaczona na proces, natomiast druga na nieużwaną pamięć, chyba że wystąpi mało prawdopodbny przypadek, w którym wolny blok będzie dokładnie odpowiadał rozmiarowi procesu.

W jakim celu wykorzystuje się scalanie? **scalanie** - złączanie dwóch sąsiednich wolnych bloków pamięci. Gdy aplikacja zwalnia pamięć, może powstać luka wolnej

pamięci pomiędzy zajętymi blokami. Scalanie jest używane w celu zmniejszania fragmentacji zewnętrznej.

Jak wybór polityki przydziału wpłynął na algorytm zwalniania bloku?

- sortowanie po adresie - scalanie jest łatwe, bo patrzę na sąsiadów na liście - jak są wolni, to scalam, jak nie, to nie
- sortowanie po rozmiarze - trudniej, bo nie mogę znaleźć fizycznego poprzednika, ale da się to ogarnąć przy scalaniu leniwym

Jaki wpływ na algorytm przydziału miałaby leniwa strategia scalania?

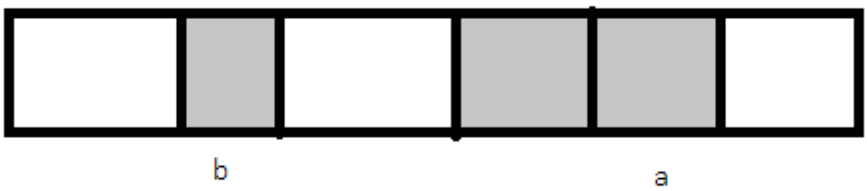
- gorliwe - musimy przejść całą listę i sprawdzić, które bloki są poprzednikiem i następnikiem
- leniwe - (przykład): podczas alokacji (w trakcie chodzenia po liście od najmniejszych bloków do największych) przy okazji sprawdzam, czy następnik fizyczny jest wolny - jak tak, to od razu scalam go w przód

L7.Z2.

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? -> 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.

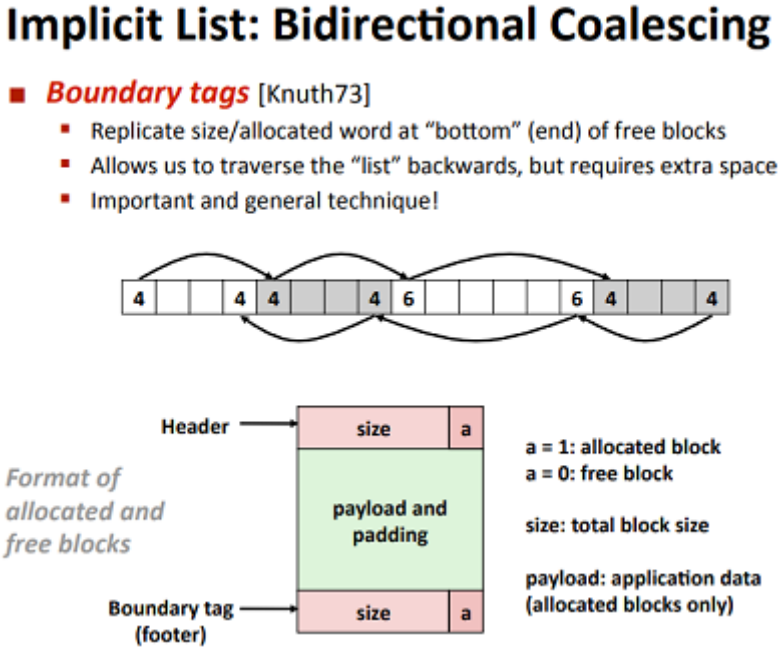
Best-ft: Polega na przeszukaniu całej listy od początku do końca i wybraniu najmniejszego wolnego bloku, który umożliwia zamieszczenie proces.

Jak zmodyfikować algorytm przydziału i zwalniania? Przydział: wrzucać zwalniane bloki na stos i najpierw szukać bloków na stosie, a nie w liście, bo jest duża szansa, że będziemy chcieli zaalokować blok długości takiej jak ostatnio zwolniliśmy.



Zwalnianie: a) ok, bo na początku ma info, jakiej jest długości b) po prawej ok, bo znamy długość, po lewej nie, bo nie wiemy, jakiej długości jest ten po lewej

Rozwiązanie: boundary tag (dodanie info na koniec bloku o jego długości) Boundary tag:



L7.Z3.

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. -> Zauważ, że przy podanych ograniczeniach metadane można dość skutecznie skompresować.

Next-fit: Algorytm działa w taki sam sposób, jak first-fit, poza tym, że zapamiętuje miejsce, w którym został znaleziony wolny blok o odpowiedniej wielkości. Kiedy algorytm zostanie wywołany następnym razem w celu znalezienia wolnego bloku, rozpoczyna wyszukiwanie od miejsca, w którym zakończył szukanie ostatnim razem, a nie zawsze od początku, jak w przypadku first-fit. Wydajność: nieco gorszy od first-fit.

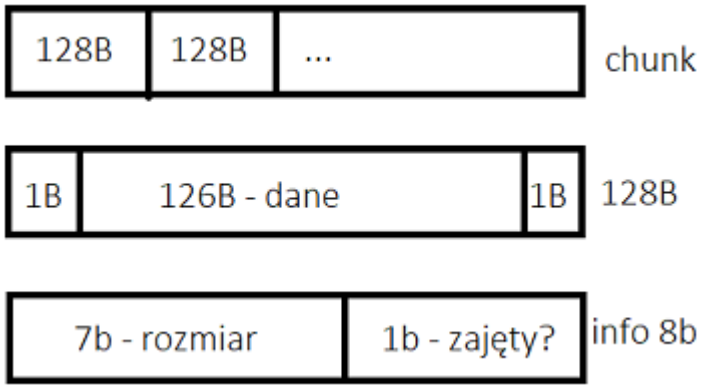
obszar (ang. *chunk*) - spójny obszar stron pamięci wirtualnej. Suppose, that main memory is partitioned into equal fixed-size chunks that are relatively small, and that each process is also divided into small fixed-size chunks of the same size. Then the chunks of a process, known as pages, could be assigned to available chunks of memory, known as frames, or page frames.

wyrównanie (ang. *align*) - malloc zwraca blok, w który można wpisać dowolny typ z języka C - adres jest wyrównany do najdłuższego słowa maszynowego.

Trzymamy gdzieś wskaźnik na przydzieloną pamięć (bo next-fit). Na koniec bloku wstawiamy nagłówek H albo jego negację. Możemy tak łatwo wykryć, czy ktoś nam go

nie nadpisał. Przy wstawianiu do bloku (bloki są na początku 127 bajtowe) jeżeli ktoś chce mało pamięci, to dzielimy blok na dwa.

Mamy chunk. Zawsze będziemy zapisywać maksymalnie 100 bajtów. Dzielimy chunk na bloki po 128 bajtów. Nagłówek i stopka zajmują 1 bajt, zostaje 126 bajtów na dane. Przy złączaniu nie łączymy w kawałki większe niż 128 bajtów. Tylko jeden bajt na zakodowanie metadanych - minimalizowanie narzutu. Nie robimy dzielenia gorliwie, bo dzielenie takiego kawałka będzie długie. Pamiętajmy, gdzie jest granica pamięci, gdzie są bloki.



L7.Z6.

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 podzielnego przez 2^k. Posługując się strukturą danych z wykładu, pokaż, krok po kroku, jak przebiega ciąg operacji alloc(8), alloc(4), free(2_6), free(2_8), free(2_4), free(1_10). Ile pamięci straciliśmy w wyniku fragmentacji zewnętrznej pamięci fizycznej? Czy można użyć w tym przypadku kompaktowania, by ją zniwelować?

Tego drzewa nie ma w pamięci, w pamięci jest PAGES i lista pustych. Węzeł i jego lewe dziecko mają ten sam adres, bo one się dzielą. Żeby można było scalić, liczby muszą się różnić na jednym bicie. Kompaktowanie - przy translacji adresów tak naprawdę kompaktujemy to i wtedy nie ma fragmentacji.

kompaktowanie pamięci - kiedy w wyniku wymiany w pamięci tworzą się duże luki, można je scalić w jeden ciągły blok poprzez przeniesienie wszystkich prcoesów tak daleko w dół, jak się da. Zwykle się tego nie robi, ponieważ operacja ta wymaga dużo czasu procesora.

Drzewo kodujemy w tablicy PAGES. Dany węzeł w drzewie kodujemy w następujący sposób: 0, jeśli idę w lewo i 1, jeśli idę w prawo. To nam daje adres w tablicy PAGES. Przy zwalnianiu wstawiamy bloki na listę pustyc oraz modyfikujemy odpowiednio tablicę stron. Algorytm może stwierdzać, czy dane bloki są bliźniakami - patrzy na listę.