Adam Kotynia, Łukasz Kowalczyk

DYNAMICZNA ALOKACJA PAMIĘCI

Dynamiczna alokacja pamięci

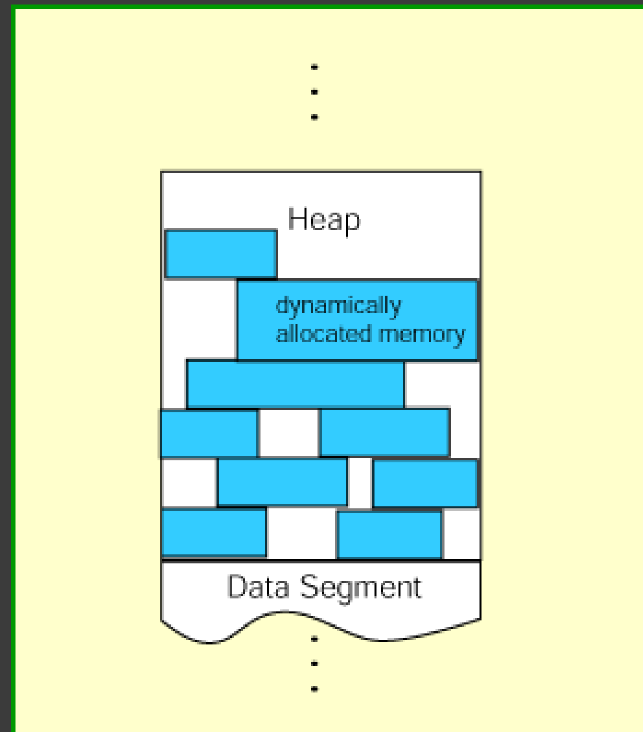
- **Alokacja pamięci oraz dezalokacja pamięci** – jest to odpowiednio przydział i zwolnienie ciągłego obszaru pamięci. Po uruchomieniu, proces (program) otrzymuje od systemu operacyjnego jedną lub więcej pul dostępnej pamięci możliwej do dowolnego wykorzystania. W zależności od przyjętej konstrukcji i zastosowania obszar nazywany jest stertą lub stosem. W trakcie działania program może zażądać od systemu operacyjnego większej ilości pamięci (alokacja) lub też zwolnić niepotrzebny obszar (dezalokacja).
- Wewnętrznie programy samodzielnie zarządzają przydzieloną im pamięcią – niskopoziomowe języki programowania dostarczają interfejs programistyczny od zarządzania stertą, który oferuje wyłącznie możliwość zarezerwowania pewnego obszaru (alokacja) i późniejszego jego zwalniania (dezalokacja). Wiele wysokopoziomowych języków programowania automatycznie przeprowadza procedurę dezalokacji bez udziału programisty.

Dla przykładu w języku C do ręcznej dynamicznej alokacji i dezalokacji pamięci służą funkcje biblioteki standardowej malloc i free. W języku C++ służą do tego specjalne słowa kluczowe (operatory) new oraz delete.

Systemy operacyjne automatycznie zwalniają pamięć przydzieloną procesom, gdy te zakończą działanie bez uprzedniej dezalokacji otrzymanej pamięci. Na poziomie aplikacji pominięcie dezalokacji doprowadza do wycieku pamięci. Co w przypadku aplikacji działającej długi czas (np. serwery) jest uważane za poważny błąd jako, iż z biegiem czasu proces taki będzie potrzebował coraz więcej pamięci, co może doprowadzić nawet do jego zawieszenia, unicestwienia. Znacznego spowolnienia działania całego systemu.

Alokacja nie powiedzie się, gdy nie istnieje wolny ciągły obszar pamięci o wymaganym rozmiarze.

Wykorzystanie kopca może wpływać na fragmentację pamięci. Zbyt duża ilość takich „dziur” powoduje marnowanie pamięci. Dlatego też współczesne systemy same śledzą takie sytuacje i odpowiednio zarządzają tymi sektorami.



- Jeśli chodzi o zastosowanie dynamicznej alokacji pamięci to już w średnio zaawansowanych programach pojawia się potrzeba dynamicznego rezerwowania pamięci. Na przykład, użytkownik podaje nam rozmiar tablicy a my musimy taką tablicę utworzyć i na niej operować (nie znając wcześniej nawet maksymalnego jej rozmiaru). Rozwiązaniem takich problemów jest właśnie dynamiczna alokacja pamięci.

DOS

Alokacja - AH=48h przerwania 21h

BX - liczba paragrafów do zaalokowania (1 paragraf = 16 bajtów).

AX - numer segmentu z zarezerwowaną dla nas pamięcią.

Programy typu .com z założenia zajmują całą dostępną pamięć, więc aby coś zaalokować, należy najpierw trochę pamięci zwolnić.

Zwalnianie pamięci - funkcja 49h

ES - numer segmentu do zwolnienia.

Jak widać, teoria nie jest skomplikowana. Przejdźmy więc może do przykładu.

Zaalokowanie 160 bajtów, wyzerowanie i zwolnienie.

```
section .text

..start:
        mov     ah, 49h
        mov     es, [ds:2ch]        ; ES=segment naszych zmiennych środowiskowych
        int     21h                 ; zwalniamy
        mov     ax, seg info
        mov     ds, ax              ; DS = nasz segment danych (w razie czego)
        mov     ah, 48h             ; rezerwuj pamięć
        mov     bx, 10              ; 10 paragrafów
        int     21h
        jc      problem             ; CF=1 oznacza błąd (skok jeśli Carry Flag równe 1)
        mov     es, ax              ; ES = przydzielony segment
        mov     ah, 9
        mov     dx, info
        int     21h                 ; wyświetl pierwszy napis
        mov     cx, 160             ; tyle bajtów wyzerujemy
        xor     di, di              ; poczynając od adresu 0 w nowym segmencie
        xor     al, al              ; AL = 0
        cld                         ; kierunek: do przodu
        rep     stosb               ; zerujemy obszar
        mov     ah, 49h
        int     21h                 ; zwalniamy pamięć
        jc      problem
        mov     ah, 9
        mov     dx, info2
        int     21h
problem:
        mov     ax, 4c00h
        int     21h
koniec:
section .data

info    db      "Udana alokacja pamieci.",10,13,"$"
info2   db      "Udane zwolnienie pamieci.",10,13,"$"

; program typu .exe musi mieć zadeklarowany stos
section stack stack
        resb 400h
```

- Zwalnianie pamięci w programach typu .com polega na zmianie rozmiaru segmentu kodu. Wykonuje się to funkcją AH=4Ah przerwania 21h, w ES podając segment, którego rozmiar chcemy zmienić (nasz segment kodu - CS), a w BX - nowy rozmiar w paragrafach.
  Typowy kod wygląda więc tak:

```
mov     ax, cs
mov     es, ax          ; będziemy zmieniać rozmiar segmentu kodu
mov     bx, koniec      ; BX = rozmiar segmentu kodu
shr     bx, 4           ; BX /= 16 - rozmiar w paragrafach
inc     bx              ; +1, żeby nie obciąć naszego programu
mov     ah, 4ah         ; funkcja zmiany rozmiaru
int     21h
```

Linux

Alokacja - **sys_brk** (ustalającą najwyższy dostępny adres w sekcji danych).

Przyjmuje ona jeden argument:

EBX = 0, jeśli chcemy otrzymać aktualny najwyższy dostępny dla nas adres w sekcji danych. Tę wartość powiększymy potem o żądany rozmiar pamięci.

EBX różny od 0, jeśli chcemy ustawić nowy najwyższy adres w sekcji danych. Adres musi być rozsądny co do wartości i taki, by rezerwowana pamięć nie wchodziła na biblioteki załadowane dynamicznie podczas samego uruchamiania programu.

Jeśli coś się nie udało, sys_brk zwróci -1 (i ustawi odpowiednio zmienną errno) lub też zwróci ujemny kod błędu.

Oczywiście, argument funkcji może być większy (alokacja) lub mniejszy (zwalnianie pamięci) od wartości zwróconej przez sys_brk przy EBX=0.

Jak widać, teoria nie jest skomplikowana. Przejdźmy więc może do przykładu. Ten krótki programik ma za zadanie **zaalokować 16kB pamięci** (specjalnie tak dużo, żeby przekroczyć 4kB - rozmiar jednej strony pamięci - i udowodnić, że pamięć rzeczywiście została przydzielona) i wyzerować ją (normalnie zapisywanie po nieprzydzielonej pamięci skończy się zamknięciem programu przez system).

```
section     .text
global _start
_start:     mov     eax, 45                 ; sys_brk
            xor     ebx, ebx
            int     80h
            add     eax, 16384              ; tyle chcemy zarezerwować
            mov     ebx, eax
            mov     eax, 45                 ; sys_brk
            int     80h
            cmp     eax, 0
            jl      .problem                ; (jump if less) jeśli błąd, to wychodzimy i nic się nie wyświetli

            mov     edi, eax                ; EDI = najwyższy dostępny adres
            sub     edi, 4                  ; EDI wskazuje na ostatni dostępny DWORD
            mov     ecx, 4096               ; tyle DWORDów zaalokowaliśmy
            xor     eax, eax                ; będziemy zapisywać zera
            std                             ; idziemy wspak
            rep     stosd                   ; zapisujemy cały zarezerwowany obszar
            cld                             ; przywracamy flagę DF do normalnego stanu
            mov     eax, 4
            mov     ebx, 1
            mov     ecx, info
            mov     edx, info_dl
            int     80h                     ; wyświetlenie napisu
.problem:
            mov     eax, 1
            xor     ebx, ebx
            int     80h
section .data
            info            db      "Udana alokacja pamieci.", 10
            info_dl         equ     $ - info
```

Koniec…