Kuchařka na PC systémy

Tuesday, May 30, 2023 8:00 PM

Assembler

Zde vypíši jen instrukce, co se zatím objevili v testech. Zbytek si už nějak odvodíte sami ;)

ld $t,[a] : načte hodnotu proměnné a do registru t

```
ld   $t0,[a]      ; load a
```

St $t,[a] : uloží hodnotu registru t do proměnné a

```
st   $t0,[b]      ; store to b
```

mul $t2=$t0,$t1 : vynásobí t0 s t1 a uloží do t2 (t2=t0+t1)

```
mul  $t2=$t0,$t1  ; multiplication
```

add $t2=$t0,$t1 : sečte t0 s t1 a uloží do t2 (t2=t0+t1)

```
add  $t2=$t1,$t2  ; addition
```

slt $t2=$t0,$t1 : pokud je t0 < t1, pak t2 nastaví na 1 (if (t0 < t1) t2 = 1;)

```
slt  $t2=$t0,$t1  ; set on less than ($t0 < $t1)
```

beq $t,x,label : pokud t = x, pak bude pokračovat ve větvi label (if (t = x) goto label;)

```
     beq  $t2,0,lbl    ; branch on equal ($t2 == 0)
     st   $t0,[b]      ; store to b
 lbl:                  ; label
```

FAT

Mějme souborový systém FAT16 s velikostí bloků 4 KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	14	-1	31	15	47	37	51	25	44	55
20	0	27	41	39	-1	42	49	10	32	0
30	36	-1	59	0	38	48	26	18	17	52
40	50	28	40	45	35	21	0	-1	-1	16
50	-1	19	-1	0	34	58	0	0	11	23

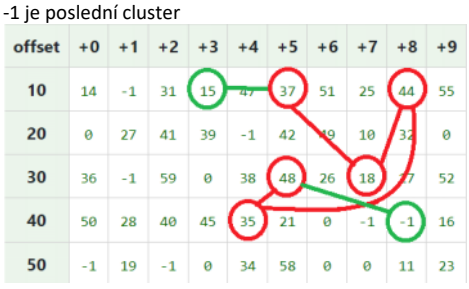
Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `file.dat` je dlouhý 27247 B a začíná na bloku 13. Aplikace soubor otevře, provede operaci **seek** na pozici 2000 a operaci **read** přečte 18000 B. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

První krok : nalezení všech relevantních clusterů
13 je první cluster a číslo v něm značí následující cluster

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	14	-1	31	15	47	37	51	25	44	55
20	0	27	41	39	-1	42	49	10	32	0
30	36	-1	59	0	38	48	26	18	17	52
40	50	28	40	45	35	21	0	-1	-1	16
50	-1	19	-1	0	34	58	0	0	11	23

-1 je poslední cluster



Druhý krok : zjištění, které clustery potřebujeme pro akci
Největší chyták je, že pokud bychom chtěli číst poslední, tak do té posloupnosti nezadávejte ty předchozí!
Další problém může být, že 4 KiB je 4096 bajtů a ne 4000

Odpovedí jsou clustery 13,15,37,18 a 44

Alokování paměti

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

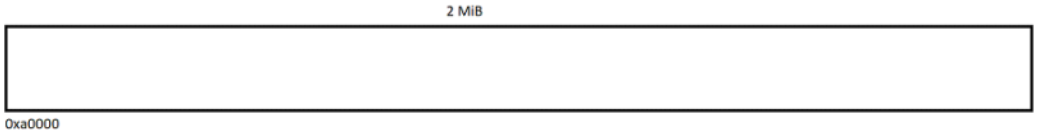
- na začátku máme jeden prázdný blok o velikosti `2` MiB na adrese `0xa0000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0xa0000`)
- heap používá alokaci po blocích velikosti `32` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

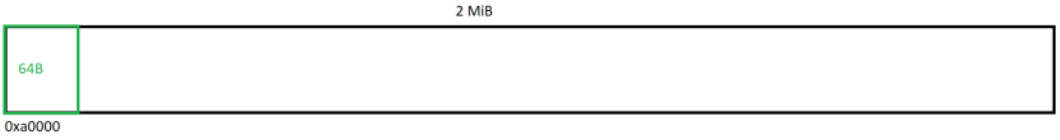
```
A = alloc 33 B
B = alloc 1 KiB
C = alloc 8200 B
D = alloc 33 B
E = alloc 33 B
free D
free C
F = alloc 8240 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

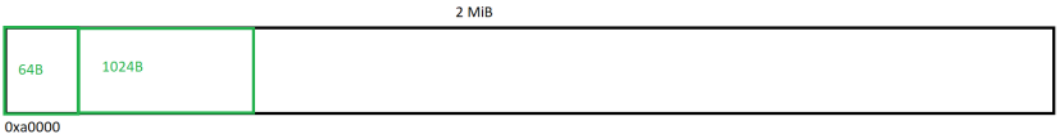
Zatím jsem u všech viděl first fit, ale je možné, že se může objevit i next fit jako algortimus na alokace
First fit (alokace od začátku bloku) X next fit (alokace na nejbližším volném místě za poslední alokací)
Vždy hledá místo od začátku bloku X začne hledat místo od poslední alokace
Alokovat se vždy může jen násobek bloku (v tomto případě jen násobky 32), tudíž pro alokaci 33B musíme použít blok o velikosti 64B
Teď pomocí obrázků nastíním alokace:



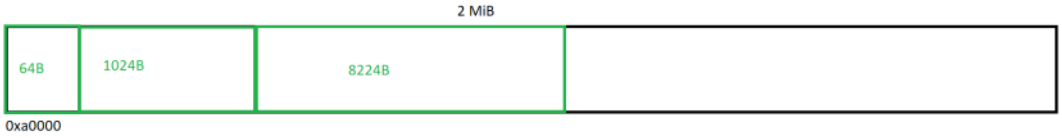
Alokace A



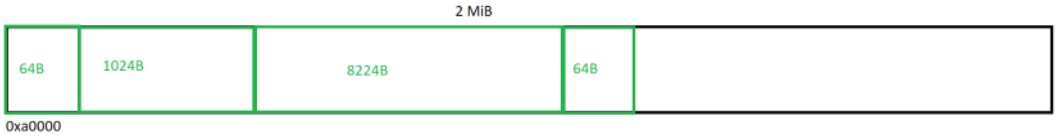
Alokace B



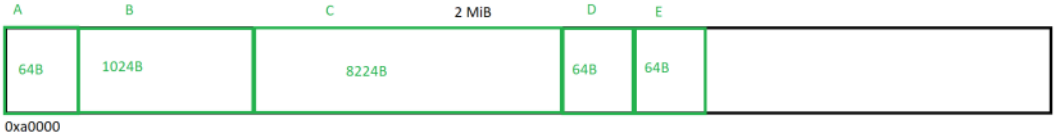
Alokace C



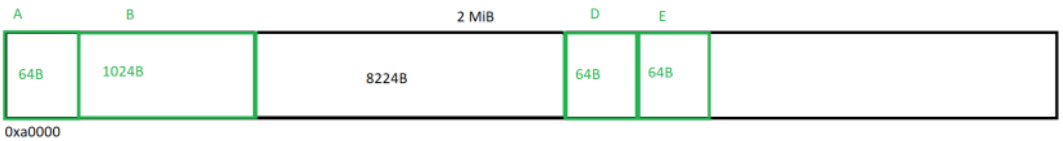
Alokace D



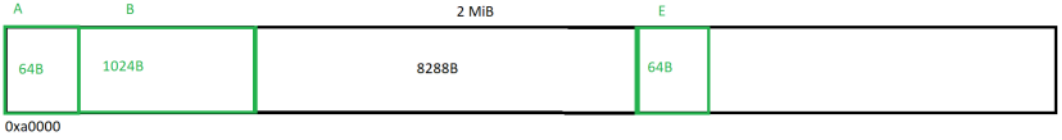
Alokace E



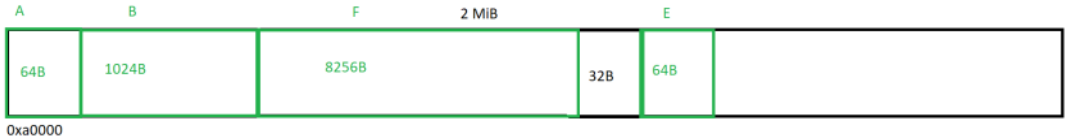
Uvolnění C



Uvolnění D



Alokace F



Offset je tedy 64+1024, tedy 1068 . Výsledná adresa je 0xa000 + 1068 = 0xa000 + 0x042c = 0xa42c

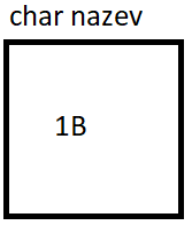
Zarovnání

Pro tento typ příkladu vyberu snad nejhorší možný případ, aby tam bylo všechno, s čím se můžete setkat

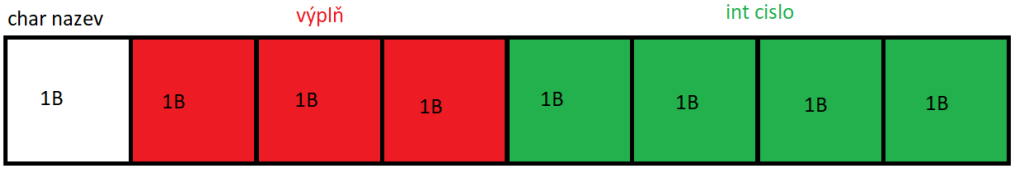
```
struct RandomStruct {
    char nazev;
    int cislo;
    double vetsicislo;
    char nazev2;
};

int main()
{
    RandomStruct priklad[20];
    // jaky offset ma priklad[9].cislo ?
    // char má 1B, int má 4B, double 8B
    return 0;
}
```

Nejdůležitější je mít na paměti, že adresa, na které leží daný typ, tak musí být dělitelná velikostí typu a typy se alokují v pořadí, ve kterém jsou zadefinovány ve struktuře. Začneme tedy od začátku: char má 1B, tudíž nemá kritérium a tak ho uložíme na adresu "0"



int má velikost 4B, takže může ležet pouze na adrese dělitelné 4. Z toho důvodu za char přidáme 3B pro zarovnání

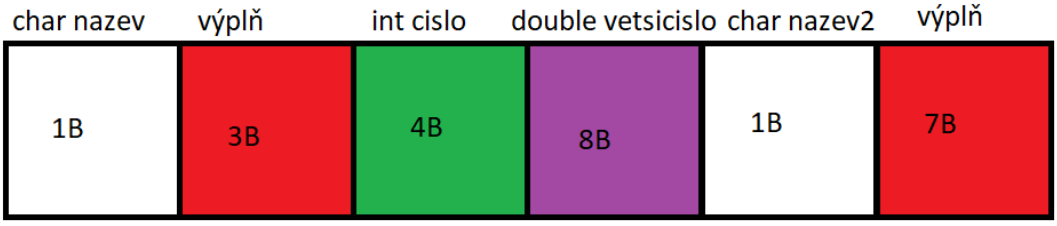


double má velikost 8B, takže musí ležet na adrese dělitelné 8, což máme zajištěno, protože int začíná na adrese dělitelné 4 a má velikost 4B, takže končí na adrese dělitelné 8, kde začne double

char má velikost 1B, takže ho můžeme jednoduše dát hned za double

Poslední krokem je, že velikost struktury musí být dělitelná největším typem ve struktuře. Nyní máme velikost 1 + 3 + 4 + 8 +1 = 17 a to není dělitelné 8, tudíž musíme přidat dalších 7B a výsledná velikost je 24B

Finální výsledek v kompaktní podobě



Pro spočítání offsetu priklad[9].cislo musíme nejprve přičíst 9 krát velikost struktury tedy 9*24 = 216 a poté přičíst offset v samotné struktuře a to je 4. Výsledný offset je tedy 216+4=220