Universiteit van Amsterdam

Besturingssystemen

Myalloc

25 april 2016

Studenten:
Shakirullah Hamza
11061413

Sander Hansen
10995080

1 Introductie

Bij deze opdracht voor besturingssystem was het doel om een eigen geheugen allocatie functie te implementeren. In dit technische rapport zijn de doelen te vinden die wij geprobeerd hebben te halen. Vervolgens is er een overzicht van welke datastructuren en algoritmes wij gebruikt hebben.

1.1 Doelen

De doelen die wij geprobeerd hebben te halen zijn;

1. Auteurs bestand en report.pdf
2. Geen errors
3. Geen warnings
4. Werkende mmalloc
5. Werkende mmfree

2 Methode

2.1 Datastructuren

2.1.1 Header

De header structuur is een datastructuur die informatie bevat over het geheugen. De voornaamste reden dat deze datastructuur is bestaat is omdat deze in een *array* bijhoud welke *banks* actief zijn. Ook bevat het een getal dat aangeeft hoeveel *banks* er in totaal beschikbaar zijn in het geheugen. Het bevat ook nog twee getallen die aangeven waar de header start en waar deze eindigd.

```
struct mm_state {
    size_t startBank;
    size_t endBank;
    struct nodeInfo* next;
```

```
    size_t totalBanks;
    char activeBanks[];
};
```

2.1.2 node

Deze datastructuur bevat informatie over een bepaald stuk gealloceerd geheugen. Zo bevat het twee getallen die aangeven bij welke *bank* de *node* start en bij welke deze eindigd. Verder wijst deze *node* naar een eventuele volgende *node* en bevat het een getal die aangeeft of de *node* leeg is of nog gevuld.

```
struct nodeInfo {
    size_t startBank;
    size_t endBank;
    struct nodeInfo* next;
    size_t empty;
};
```

2.2 Algortimes

2.2.1 Initialisatie

De initialisatie wordt eenmalig uitgevoerd om het geheugen te initialiseren. Tijdens de initialisatie zullen een aantal gegevens over het geheugen worden opgeslagen. Vervolgens zal deze datastructuur als *return* waarde worden teruggegeven.

```
hoeveelheidBanks = krijgRamInfo()

banksNodigVoorInit = ((grootteHeader + Bankgrootte - 1) +
        (charGrootte * hoeveelheidBanks)) / bankGrootte)

voor elke banksNodigVoorInit
        activeer bank

header.start = 0
header.volgende = NULL
header.einde = banksNodigVoorInit - 1
header.hoeveelheidBanks = hoeveelheidBanks

voor elke hoeveelheidBanks
    als banksNodigVoorInit
           header.actieveBanks is 1
    anders
       header.actieveBanks is 0

return header
```

2.2.2 Allocatie

De allocatie wordt uitgevoerd als er geheugen gealloceerd moet worden. Deze functie wordt aangeroepen met een *header* en een bepaalde hoeveel *bytes* die moeten worden gealloceerd. Er wordt berekent hoeveel *banks* er nodig zijn en vervolgens wordt er gekeken of en waar deze hoeveelheid *banks* in het geheugen geplaatst kan worden. Hij returned een pointer met het geheugen adres.

```
banksNodigVoorAlloc = ((grootteNode + hoeveelheidBytesNode + bankGrootte -1)
    / bankGrootte)

als header.volgende is NULL
    geheugenAdress = (basisAdress + header.einde + 1) * bankGrootte
    node = geheugenAdress
    node.start = header.einde + 1
    node.einde = node.start + banksNodigVoorAlloc -1
    node.volgende = NULL
    node.leeg = 0
    activeerBanks()

    return geheugenadress

huidige = header.volgende

zolang 1
    als huidige.leeg is 0
        als huidige.volgende is niet 0
            huidige = huidige.volgende
        anders
            als banksNodigVoorAlloc past in het geheugen
                geheugenAdress = (basisAdress + huidige.einde + 1) * bankGrootte
                node = geheugenAdress
                node.start = huidige.einde + 1
                node.einde = node.start + banksNodigVoorAlloc -1
                node.volgende = NULL
                node.leeg = 0
                activeerBanks()

                return geheugenadress
            anders
                return 0
    anders
        als banksNodigVoorAlloc past in de Node
            node.leeg = 1
            als nog ruimte over in Node
                geheugenAdress = (basisAdress + huidige.einde + 1) * bankGrootte
                node = geheugenAdress
                node.start = huidige.einde + 1
                node.einde = node.start + banksNodigVoorAlloc -1
                node.volgende = NULL
                node.leeg = 0
            return geheugenadress;
        anders
            als huidige.einde is totalBanks

            return 0
```

2.2.3 Vrijgeven

Het vrijgeven wordt aangeroepen wanneer een bepaald stuk geheugen niet meer gebruikt wordt. Deze functie wordt aangeroepen met de *header* en een *pointer* naar het adres waar dit geheugen zich bevindt.

```
adres = (pointer - basisAdres - grootteVanNode) / bankGrootte

node = header.volgende

zolang 1
    als node.start is adres
        node.leeg = 1
        return
    anders
        node = node.volgende
```