Mājas darbs #5 - Atmiņas fragmentācija

Rolands
Agris

1. Ievads

Projekts ir veidots, lai testētu un salīdzinātu dažādus atmiņas rezervēšanas algoritmus. Tā struktūra ir organizēta šādi:

- **Galvenais fails (`main.c`)**: Apstrādā ievadi, izvēlas atbilstošo algoritmu un izpilda to.
- **Bibliotēkas**:
  - **cli**: Komandlīnijas interfeiss parametru apstrādei
  - **algorithms**: Dažādu atmiņas rezervēšanas algoritmu implementācijas
  - **parsers**: Funkcijas failu lasīšanai un skaitļu parsēšanai
  - **utils**: Atbalsta funkcijas, piemēram, fragmentācijas aprēķināšanai

Projekts implementē sekojošus atmiņas rezervēšanas algoritmus:

- **Best Fit**: Meklē mazāko pieejamo atmiņas bloku, kas ir pietiekami liels
- **First Fit**: Izmanto pirmo pieejamo bloku, kas ir pietiekami liels
- **Next Fit**: Līdzīgs First Fit, bet turpina meklēšanu no pēdējā izdalītā bloka
- **Worst Fit**: (Nav pilnībā implementēts)

Projekts izmanto vienvirziena saistīto sarakstu, lai attēlotu atmiņu, kur katrs mezgls satur informāciju par bloka izmēru un to, vai tas ir brīvs vai aizņemts. Saistītā saraksta struktūra ir definēta šādi:

```c
typedef struct Node {
    long int value;     /* Atmiņas bloka izmērs */
    int is_free;        /* Indikators, vai bloks ir brīvs (1) vai aizņemts (0) */
    struct Node* next;  /* Norāde uz nākamo saraksta mezglu */
} Node;
```

2. First Fit un Next Fit algoritmu realizācijas

2.1. First Fit algoritma implementācija

First Fit algoritma laika sarežģītība:

- Vidējais gadījums: O(n), kur n ir bloku skaits
- Sliktākais gadījums: O(n)

First Fit vienmēr sāk meklēšanu no saraksta sākuma, tāpēc tas var būt lēnāks algoritms, jo katra pieprasījuma gadījumā ir jāmeklē no sākuma. Tomēr tas bieži vien ir ātrāks nekā Best Fit, jo tam nav jāmeklē visoptimālākais bloks.

Algoritma implementācija ir šāda:

```c
Node* find_first_fit(int size, Node* memory) {
    Node* curr = memory;

    while (curr != NULL) {
        if (curr->is_free && curr->value >= size) {
            return curr;
        }
        curr = curr->next;
    }

    return NULL;
}
```

2.2. Next Fit algoritma implementācija

Next Fit algoritma laika sarežģītība:

- Vidējais gadījums: O(n), kur n ir bloku skaits
- Sliktākais gadījums: O(n)

Next Fit teorētiski var būt ātrāks nekā First Fit, jo tas turpina meklēšanu no pēdējā izdalītā bloka, nevis sāk no saraksta sākuma. Tomēr realitātē tā efektivitāte ir atkarīga no pieprasījumu un brīvo bloku izvietojuma.

Algoritma implementācija ir šāda:

```
Node* find_next_fit(int size, Node* memory, Node** last_alloc) {
    Node* curr = *last_alloc ? (*last_alloc)->next : memory;

    while (curr != NULL) {
        if (curr->is_free && curr->value >= size) {
            *last_alloc = curr;
            return curr;
        }
        curr = curr->next;
    }

    curr = memory;
    while (curr != *last_alloc) {
        if (curr->is_free && curr->value >= size) {
            *last_alloc = curr;
            return curr;
        }
        curr = curr->next;
    }

    return NULL;
}
```

3. Laika mērījumu apraksts

Gan First Fit, gan Next Fit algoritmos laika mērījums tiek veikts, izmantojot standarta C bibliotēkas funkciju `clock()`. Laika mērīšanas process notiek šādi:

```
start = clock();

// ... algoritma izpilde ...

end = clock();

printf(" - Time taken, seconds: %f\n", ((double) (end - start)) / CLOCKS_PER_SEC);
```

`CLOCKS_PER_SEC` ir konstante, kas norāda pulksteņa tikšķu skaitu sekundē, un tiek izmantota, lai pārveidotu pulksteņa tikšķus sekundēs.

4. Fragmentācijas mērījuma funkcija

Fragmentācija ir svarīgs rādītājs atmiņas pārvaldības algoritmu novērtēšanā. Šajā projektā fragmentācija tiek aprēķināta ar `calculate_fragmentation` funkciju:

```
double calculate_fragmentation(Node* memory, int largest_request) {
        size_t total_free = 0;
        size_t largest_free_block = 0;
        size_t current_free_block = 0;

        Node* curr = memory;
        while (curr != NULL) {
                if (curr->is_free) {
                        total_free += curr->value;
                        current_free_block += curr->value;
                        if (current_free_block > largest_free_block) {
                                largest_free_block = current_free_block;
                        }
                } else {
                        current_free_block = 0;
                }
                curr = curr->next;
        }

        if (total_free == 0) return 0.0;
```

```
21
22          size_t unusable_memory = 0;
23          current_free_block = 0;
24          curr = memory;
25          while (curr != NULL) {
26                  if (curr->is_free) {
27                          current_free_block += curr->value;
28                  } else {
29                          if (current_free_block > 0 && current_free_block < (size_t)largest_request) {
30                                  unusable_memory += current_free_block;
31                          }
32                          current_free_block = 0;
33                  }
34                  curr = curr->next;
35          }
36
37          if (current_free_block > 0 && current_free_block < (size_t)largest_request) {
38                  unusable_memory += current_free_block;
39          }
40
41          return (double)unusable_memory / total_free;
42  }
```

Šī funkcija aprēķina fragmentācijas koeficientu, kas ir attiecība starp neizmantojamo brīvo atmiņu (brīvie bloki, kas ir mazāki par lielāko pieprasījumu) un kopējo brīvo atmiņu. Funkcijas darbību var sadalīt vairākos posmos:

1. **Kopējās brīvās atmiņas un lielākā brīvā bloka noteikšana**:
   - Funkcija vispirms pārstaigā visu saistīto sarakstu, summējot visu brīvo bloku izmērus (`total_free`).
   - Vienlaikus tiek noteikts arī lielākais nepārtrauktais brīvās atmiņas bloks (`largest_free_block`), kas veidojas no secīgiem brīviem blokiem.
   - Katru reizi, kad ir atrasts aizņemts bloks, nepārtrauktā brīvā bloka skaitīšana tiek atiestatīta (`current_free_block = 0`).
2. **Neizmantojamās atmiņas noteikšana**:
   - Otrajā saraksta traversēšanā funkcija nosaka, cik daudz brīvās atmiņas ir „neizmantojama".
   - Atmiņas bloks tiek uzskatīts par neizmantojamu, ja tā izmērs ir mazāks par lielāko pieprasījumu (`largest_request`).
   - Tiek pieņemts, ka bloki, kas ir mazāki par lielāko pieprasījumu, nevar tikt izmantoti efektīvi, jo tie nespēj apmierināt lielāko iespējamo pieprasījumu.
   - Šeit tiek izmantots arī nepārtraukto brīvo bloku koncepts - ja vairāki secīgi brīvi bloki kopā ir mazāki par lielāko pieprasījumu, tie visi tiek uzskatīti par neizmantojamiem.
3. **Fragmentācijas koeficienta aprēķināšana**:
   - Fragmentācijas koeficients tiek aprēķināts kā neizmantojamās brīvās atmiņas attiecība pret kopējo brīvo atmiņu.
   - Rezultātā iegūstam skaitli robežās no 0 līdz 1, kur:
     - 0 nozīmē, ka visa brīvā atmiņa ir izmantojama (nav fragmentācijas).
     - 1 nozīmē, ka visa brīvā atmiņa ir neizmantojama (pilnīga fragmentācija).
   - Jo zemāks koeficients, jo efektīvāk algoritms izmanto atmiņu un mazāk rada fragmentāciju.

Šī metrika ir īpaši noderīga, lai salīdzinātu dažādus atmiņas pārvaldības algoritmus, jo tā ņem vērā praktiskos ierobežojumus - atmiņas bloki, kas ir pārāk mazi, faktiski kļūst nelietojami lielu pieprasījumu gadījumā. Tas ir reāls fragmentācijas efekts, ko algoritmi cenšas minimizēt.

5. Algoritmu novērtējums

5.1. Algoritmu mērījumu analīze

No algoritumu izpildes rezultātiem var analizēt algoritmu veiktspēju (sk. 1. tabulu) un var secināt, ka:

1. **Izpildes laiks**: Next Fit algoritms izpildās aptuveni 21.5% ātrāk nekā First Fit, kas atbilst teorētiskajiem paredzējumiem. Tas ir tāpēc, ka Next Fit turpina meklēšanu no pēdējā izdalītā bloka, nevis katru reizi sāk no saraksta sākuma.

2. **Izdalīto bloku skaits**: First Fit algoritms spēja izdalīt vairāk bloku (728) nekā Next Fit (712). Tas liecina, ka First Fit efektīvāk izmanto pieejamo atmiņu.
3. **Kopējā izdalītā atmiņa**: First Fit izdalīja vairāk atmiņas (3548 baitus) nekā Next Fit (3471 baitus). Šī atšķirība ir proporcionāla izdalīto bloku skaitam.

Metrika	First Fit	Next Fit
Izpildes laiks (sekundes)	0.000246	0.000193
Fragmentācijas koeficients	0.362751	0.425682
Izdalīto bloku skaits	728	712
Kopējā izdalītā atmiņa (baiti)	3548	3471

1. tabula: First Fit un Next Fit algoritmu mērījumu rezultāti

5.2. Secinājumi par ātrdarbību

Saskaņā ar teorētiskiem apsvērumiem un praktiskiem mērījumiem:

1. Next Fit ir ātrāks nekā First Fit, kā to pierāda mērījumu rezultāti, kas uzrāda 21.5% ātruma uzlabojumu.
2. First Fit spēj efektīvāk izmantot atmiņu, izdalot vairāk bloku un kopumā vairāk atmiņas.
3. Ātrdarbības uzlabojums Next Fit algoritmā nāk ar kompromisu - mazāku atmiņas izmantošanas efektivitāti.
4. Abiem algoritmiem ir O(n) laika sarežģītība sliktākajā gadījumā, bet praktiskajos rezultātos Next Fit demonstrē labāku vidējo veiktspēju.

Algoritms	Izpildes laiks
First Fit	0.000246
Next Fit	0.000193

2. tabula: First Fit un Next Fit algoritmu fragmentācijas rezultāti

6. Algoritmu fragmentācijas novērtējums

6.1. First Fit fragmentācija

First Fit parasti rada mazāku ārējo fragmentāciju nekā Next Fit, jo tas vienmēr meklē no sākuma un tādējādi biežāk izmanto mazākos pieejamos blokus. Tomēr tas var radīt lielāku iekšējo fragmentāciju, jo tas izvēlas pirmo derīgo bloku, nevis optimālāko.

6.2. Next Fit fragmentācija

Next Fit parasti rada lielāku ārējo fragmentāciju nekā First Fit, jo tas var izlaist mazākus blokus saraksta sākumā. Laika gaitā tas var radīt situāciju, kur saraksta sākumā ir daudz mazu, neizmantotu bloku.

6.3. Fragmentācijas mērījumu analīze

Algoritmu izpildes rezultāti apstiprina teorētiskos paredzējumus:

Algoritms	Fragmentācijas koeficients
First Fit	0.362751
Next Fit	0.425682

3. tabula: First Fit un Next Fit algoritmu fragmentācijas rezultāti

1. First Fit algoritms uzrāda zemāku fragmentācijas koeficientu (0.362751) nekā Next Fit (0.425682).
2. Next Fit fragmentācijas koeficients ir aptuveni 17.3% augstāks nekā First Fit, kas liecina par sliktāku atmiņas izmantošanu.
3. Augstāka fragmentācija Next Fit algoritmā visticamāk ir saistīta ar tā tendenci izlaist mazākus brīvos blokus saraksta sākumā, kas laika gaitā veido neizmantotos atmiņas ķabatas".

6.4. Secinājumi par fragmentāciju

1. First Fit uzrāda mazāku fragmentāciju nekā Next Fit, kas apstiprina teorētiskos pieņēmumus.
2. First Fit fragmentācijas koeficients ir 0.362751, kas ir aptuveni 17.3% zemāks nekā Next Fit (0.425682).
3. Next Fit augstāka fragmentācija ir saistīta ar tā tendenci izlaist mazākus brīvos blokus saraksta sākumā.
4. Fragmentācijas atšķirība starp abiem algoritmiem ir nozīmīga un var būt svarīga atmiņas pārvaldības algoritma izvēlē atkarībā no specifisku prasību konteksta.

7. Kopējie secinājumi algoritmu izveidei

1. **Algoritma izvēle atkarīga no lietojuma**:
   - First Fit ir labs vispārīgs risinājums, kas nodrošina labu līdzsvaru starp ātrdarbību un fragmentāciju.
   - Next Fit ir piemērots gadījumiem, kad ātrums ir svarīgāks par atmiņas efektivitāti.
   - Best Fit (kas arī ir implementēts projektā) ir labs, kad atmiņas efektivitāte ir svarīgāka par ātrdarbību.
2. **Implementācijas apsvērumi**:
   - Visu algoritmu implementācijas izmanto līdzīgu saistītā saraksta struktūru, kas atvieglo to salīdzināšanu.
   - Fragmentācijas aprēķināšana ir svarīga algoritmu novērtēšanai un var būt atkarīga no konkrētā lietojuma.
   - Laika mērījumi ir nepieciešami, lai novērtētu algoritmu praktisko efektivitāti.
3. **Kompromiss starp ātrdarbību un fragmentāciju**:
   - Next Fit ir aptuveni 21.5% ātrāks nekā First Fit, bet rada 17.3% augstāku fragmentāciju.
   - First Fit spēj izdalīt vairāk bloku (728 pret 712) un kopumā vairāk atmiņas (3548 baiti pret 3471 baitu).
   - Šis kompromiss starp ātrdarbību un atmiņas izmantošanas efektivitāti ir jāapsver, izvēloties algoritmu konkrētam lietojumam.
4. **Optimizācijas iespējas**:
   - Algoritmi varētu tikt uzlaboti, izmantojot sarežģītākas datu struktūras, piemēram, balansētus kokus vai meklēšanas tabulas.
   - Konkrētam lietojumam var būt nepieciešams pielāgot algoritmus, piemēram, apvienot blīvi izvietotus brīvos blokus.
   - Hibrīda pieejas, kas apvieno dažādu algoritmu priekšrocības, varētu būt efektīvākas dažādos scenārijos.

8. Secinājumi un labākā algoritma izvēle

Balstoties uz veiktās analīzes rezultātiem, var secināt, ka labākā algoritma izvēle ir atkarīga no specifisku prasību prioritātēm:

- Ja prioritāte ir **ātrums**, tad **Next Fit** ir labākā izvēle, jo tas demonstrē par 21.5% labāku veiktspēju nekā First Fit.
- Ja prioritāte ir **atmiņas efektivitāte** un **minimāla fragmentācija**, tad **First Fit** ir labākā izvēle, jo tas uzrāda par 17.3% zemāku fragmentācijas koeficientu un spēj izdalīt vairāk atmiņas (3548 pret 3471 baitus).

Ņemot vērā abus kritērijus kopumā, **First Fit** var uzskatīt par labāko algoritmu no abiem analizētajiem, jo tas nodrošina labāku atmiņas izmantošanu, un tā ātrdarbības atšķirība pret Next Fit ir samērā neliela.