Mājas darbs #5 - Atmiņas fragmentācija

Rolands Frīdemanis, rf23009
Agris Pudāns, ap08426

komanda: heap_hippies

Saturs

1. Ievads 1
2. Atmiņas rezervēšanas algoritmu realizācijas 2
   - 2.1. First Fit algoritma realizācija 2
   - 2.2. Next Fit algoritma realizācija 2
   - 2.3. Best Fit algoritma realizācija 2
   - 2.4. Worst Fit algoritma realizācija 2
3. Laika mērījumu apraksts 3
4. Fragmentācijas mērījuma funkcija 3
5. Algoritmu realizācijas novērtējums 4
   - 5.1. Ātrdarbības novērtējums 4
   - 5.2. Fragmentācijas novērtējums 5
   - 5.3. Secinājumi par fragmentāciju 5
6. Kopējie secinājumi un labākā algoritma izvēle 5
   - 6.1. Secinājumi algoritmu izvēlei 5
   - 6.2. Labākā algoritma izvēle 6
   - 6.3. Ieguldījums no komandas locekļiem 6

1. Ievads

Projekta komandas dalībnieki ir Agris Pudāns un Rolands Frīdemanis. Katra dalībnieka ieguldījums ir vērtējams 50/100.

Projekts ir veidots, lai testētu un salīdzinātu dažādus atmiņas rezervēšanas algoritmus. Testa kopām ir iz mantota viena no publiskajām - [http://selavo.lv/kursi/lsp/2013/mem-frag-tests/mem-frag-tests-2.zip]. Projekta 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 realizācijas
  - **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. Šis algoritms cenšas minimizēt iekšējo fragmentāciju, izvēloties bloku, kas vistuvāk atbilst nepieciešamajam izmēram.
- **First Fit**: Izmanto pirmo pieejamo bloku, kas ir pietiekami liels. Šis ir vienkāršākais un bieži vien efektīvākais algoritms.
- **Next Fit**: Līdzīgs First Fit, bet turpina meklēšanu no pēdējā izdalītā bloka, lai paātrinātu meklēšanas procesu.
- **Worst Fit**: Meklē lielāko pieejamo bloku atmiņā. Šis algoritms cenšas maksimizēt atlikušos fragmentus, lai tie būtu pietiekami lieli nākamajiem pieprasījumiem, bet var radīt lielāku iekšējo fragmentāciju.

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:

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

2. Atmiņas rezervēšanas algoritmu realizācijas

2.1. First Fit algoritma realizācija

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 pieprasījumas ir jāmeklē no sākuma. Tomēr tas bieži vien ir ātrāks nekā Best Fit, jo tam nav jāmeklē visoptimālākais bloks.

Algoritma realizācija ir šāda:

```
1          Node* find_first_fit(int size, Node* memory) {
2              Node* curr = memory;
3
4              while (curr != NULL) {
5                  if (curr->is_free && curr->value >= size) {
6                      return curr;
7                  }
8                  curr = curr->next;
9              }
10
11             return NULL;
12         }
```

2.2. Next Fit algoritma realizācija

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. Tas gan var būt atkarīgs no pieprasījumu un brīvo bloku izvietojuma.

2.3. Best Fit algoritma realizācija

Best Fit algoritma laika sarežģītība:

- Vidējais gadījums: O(n), kur n ir bloku skaits
- Sliktākais gadījums: O(n)
- Labākais gadījums: O(1)

Best Fit algoritms pārskata visus pieejamos blokus un izvēlas to, kurš ir vistuvāk nepieciešamajam izmēram, bet joprojām ir pietiekami liels. Šādi samazinās iekšējā fragmentācija. Bet tas ir lēnāks par First Fit vai Next fit, jo ir nepieciešams pārskatīt visus blokus.

2.4. Worst Fit algoritma realizācija

Worst Fit algoritma laika sarežģītība:

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

Worst Fit algoritms ir pretējs Best Fit - tas meklē lielāko pieejamo bloku, lai tajā ievietotu pieprasīto atmiņu. Algoritma mērķis ir atstāt pēc iespējas lielākus brīvos fragmentus, lai tie būs lietderīgāki nākotnes pieprasījumiem. Šis algoritms var radīt lielāku iekšējo fragmentāciju un sliktāku atmiņas izmantošanu.

3. Laika mērījumu apraksts

Visos algoritmos, First Fit, Next Fit, Best Fit un Worst Fit, 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 fragmentāciju aprēķina 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;

        size_t unusable_memory = 0;
        current_free_block = 0;
        curr = memory;
        while (curr != NULL) {
                if (curr->is_free) {
                        current_free_block += curr->value;
                } else {
                        if (current_free_block > 0 && current_free_block <
                        ↪  (size_t)largest_request) {
                                unusable_memory += current_free_block;
                        }
                        current_free_block = 0;
                }
                curr = curr->next;
        }

        if (current_free_block > 0 && current_free_block < (size_t)largest_request) {
                unusable_memory += current_free_block;
        }

        return (double)unusable_memory / total_free;
}
```

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 realizācijas novērtējums

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

1. **Izpildes laiks**: Next Fit algoritms ir visātrākais un gandrīz 6 reizes ātrāks par First Fit, 10 reizes ātrāks par Worst Fit un aptuveni 42 reizes ātrāks par Best Fit. Šie rezultāti atbilst teorētiskajām prognozēm, kur algoritmi, kas meklē optimālo bloku (Best Fit), ir lēnāki nekā tie, kas apstājas pie pirmā pieņemamā bloka (First Fit, Next Fit).
2. **Izdalīto bloku skaits**: First Fit un Next Fit izdalīja vienādu bloku skaitu, kas ir ievērojami vairāk nekā Best Fit vai Worst Fit. Šāda atšķirība liecina par dažādu algoritmu stratēģiju ietekmi uz atmiņas izmantošanas efektivitāti.
3. **Kopējā izdalītā atmiņa**: First Fit un Next Fit izdalīja vienādu, 980 baitu, atmiņas apjomu, Best Fit izdalīja 1024 baitus, bet Worst Fit tikai 1000 baitus. Lai arī Best Fit izdalīja mazāk bloku, tomēr kopā izdalīja vairāk atmiņas. Tas liecina, ka tas efektīvāk aizpildīja pieejamo atmiņas telpu.

Metrika	First Fit	Next Fit	Best Fit	Worst Fit
Izpildes laiks (sekundes)	0.000024	0.000004	0.000168	0.000040
Fragmentācijas koeficients	0.000005	0.000027	0.000093	0.000031
Izdalīto bloku skaits	99	99	56	5
Kopējā izdalītā atmiņa (baiti)	980	980	1024	1000

1. tabula: Visu algoritmu mērījumu rezultāti

5.1. Ātrdarbības novērtējums

Saskaņā ar teorētiskiem apsvērumiem un mērījumiem:

1. **Next Fit** ir visātrākais no visiem testētajiem algoritmiem, kas padara to ideālu lietojumiem, kur svarīgs ir izpildes ātrums.

2. **Best Fit** mēdz būt ātrāks nekā Best Fit, bet ir lēnāks nekā First Fit un Next Fit. Labākajā gadījumā, algoritms apstāsies pie atmiņas bloka, kas ir vienāds ar pieprasīto atmiņas daudzumu - pretējā gadījumā notiek pilnā saraksta pārlase.

3. **Worst Fit** ir vislēnākais algoritms, jo tas vienmēr pārmeklē visu sarakstu, lai atrastu vislielāko atmiņas bloku.

4. Atmiņas izmantošanas efektivitātes ziņā šie algoritmi demonstrē dažādus rezultātus - Next Fit un First Fit izdalīja visvairāk bloku, Best Fit izdalīja mazāk, bet Worst Fit izdalīja vismazāk bloku (5), kas var liecināt par algoritma neefektivitāti situācijās ar ierobežotu atmiņas daudzumu.

5.2. Fragmentācijas novērtējums

Algoritmu izpildes rezultāti fragmentācijas ziņā ir apkopoti 2. tabulā. Analizējot algoritmu ietekmi uz atmiņas fragmentāciju, var novērot šādas likumsakarības:

1. **First Fit** parasti rada mazāku fragmentāciju nekā Next Fit, jo tas vienmēr meklē no sākuma, un tādējādi biežāk izmanto mazākos pieejamos blokus. Mūsu mērījumi to apstiprina, uzrādot viszemāko fragmentācijas koeficientu visā atmiņa, tomēr tas var neatspoguļot situāciju tieši atmiņas sākumposmā.

2. **Next Fit** parasti rada lielāku ārējo fragmentāciju nekā First Fit, jo tas var izlaist mazākus blokus saraksta sākumā. Tā fragmentācijas koeficients ir aptuveni 5.4 reizes augstāks nekā First Fit testa piemērā. Toties nedrīkst aizmirst, ka tas fragmentē atmiņu vienmērīgāk nekā citi algoritmi, jo tas nemēģina katram jaunam atmiņas pieprasījumam atkal meklēt pieejamos atmiņas blokus sākumā, tādejādi tur samazinot fragmentāciju.

3. **Best Fit** radīja lielāko frgmentāciju, kā jau paredzēts, jo mēģina pēc iespējas vairāk ietaupīt atmiņu, bet tādejādi ignorē izdalīto atmiņas bloku nepārtrauktības principu, kas loģiski palielina fragmentāciju.

4. **Worst Fit** uzrāda zemāku fragmentācijas koeficientu, kas ir tikai nedaudz augstāks nekā Next Fit. Tas skaidrojams ar to, ka algoritms izdala tikai lielākos atmiņas blokus, nevis meklē pēc iespējas sīkāk, lai mazinātu sīko fragmnetu skaitu, kuri procesiem lielākoties nav vajadzīgi.

Algoritms	**Fragmentācijas koeficients**
First Fit	0.000005
Next Fit	0.000027
Best Fit	0.000093
Worst Fit	0.000031

2. tabula: Visu algoritmu fragmentācijas rezultāti

5.3. Secinājumi par fragmentāciju

1. First Fit uzrāda viszemāko fragmentāciju no visiem algoritmiem, kas padara to piemērotu lietojumiem, kur atmiņas efektīva izmantošana ir prioritāte.

2. Next Fit uzrāda salīdzinoši zemu fragmentāciju un samazina to samazina atmiņas sākumā, tāpēc varētu būt praktiski lietderīgs lielākajā daļā situāciju.

3. Best Fit, pretēji teorētiskajiem paredzējumiem, uzrāda, ka teorētiski ōptimālākā"bloka izvēle ne vienmēr veda pie visefektīvākās atmiņas izmantošanas fragmentācijas ziņā.

4. Worst Fit, kaut arī izdalīja vismazāk bloku, neuzrādīja augstāko fragmentācijas koeficientu. Tas norāda, ka fragmentācijas līmenis ne vienmēr korelē ar izdalīto bloku skaitu. Tam arī nav slikta pieeja fragmentācijas kontrolēšanai, bet nav īsti lietderīgs ar ierobežota daudzuma atmiņu.

5. Kopumā visi algoritmi uzrādīja ļoti zemu fragmentācijas līmeni (koeficienti zem 0.0001), kas liecina, ka šajā konkrētajā testā fragmentācija nebija nozīmīga problēma nevienam no algoritmiem.

6. Kopējie secinājumi un labākā algoritma izvēle

6.1. Secinājumi algoritmu izvēlei

1. **Algoritma izvēle atkarīga no lietojuma**:

   - **First Fit** ir labs vispārīgs risinājums, kas nodrošina zemu fragmentāciju un labu izpildes laiku.

- **Next Fit** ir ideāls, kad ātrums ir galvenā prioritāte un fragmentācija ir mazāk svarīga.
- **Best Fit** ir piemērots situācijām, kad vēlamies maksimāli izmantot pieejamo atmiņu, pat ja tas prasa ilgāku izpildes laiku.
- **Worst Fit** šajā testā parādīja sliktākos rezultātus bloku izdalīšanas ziņā, bet var būt noderīgs ļoti specifiskās situācijās, kur vajadzīgs zems fragmentācijas līmenis un ātrs izpildes laiks.

2. **Realizācijas apsvērumi**:
   - Visu algoritmu realizācijas 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 6 reizes ātrāks nekā First Fit, bet rada 5.4 reizes augstāku fragmentāciju, lai gan abi koeficienti ir ārkārtīgi mazi.
   - Best Fit ir aptuveni 7 reizes lēnāks nekā Next Fit, bet izdalīja vairāk kopējās atmiņas (1024 pret 980 baitiem).
   - Worst Fit ir ātrāks nekā Best Fit, bet izdalīja ievērojami mazāk bloku (5 pret 56).
   - Šajā konkrētajā gadījumā kompromiss starp ātrdarbību un atmiņas izmantošanas efektivitāti ir minimāls, jo visi algoritmi demonstrē ļoti zemu fragmentāciju.
4. **Optimizācijas iespējas**:
   - Visi 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.

6.2. Labākā algoritma izvēle

Ņemot vērā visus mērījumu rezultātus, varam secināt, ka labākā algoritma izvēle ir atkarīga no specifisku prasību prioritātēm:

- Ja prioritāte ir **ātrums**, tad **Next Fit** ir nepārprotami labākā izvēle, jo tas demonstrē vislabāko izpildes laiku (0.000004s) un spēj izdalīt lielu bloku skaitu (99).
- Ja prioritāte ir **maksimāla atmiņas izdalīšana**, tad **Best Fit** sniedz labākos rezultātus, izdalot visvairāk atmiņas, kas ir vienāda ar maksimālo izmantojamo (1024 baiti), lai gan ir ar ilgāku izpildes laiku un augstāku fragmentāciju.

Kopumā, balstoties uz šiem mērījumiem un analizējot visus algoritmus, **Next Fit** un **First Fit** ir vispiemērotākie vispārējai lietošanai:

- **Next Fit** ir optimālā izvēle, kad izpildes ātrums ir kritisks - tas ir ievērojami ātrāks par citiem algoritmiem, izdalīja daudz bloku un uzrādīja pieņemamu fragmentācijas līmeni.
- **Best Fit** uzrādīja lēnāku izpildes laiku, bet izdalīja visu atvēlēto atmiņu. Ņemot vērā iespējamās optimizācijas, piemēram, izmantojot BST, šis algoritms ir labākais.

6.3. Ieguldījums no komandas locekļiem

Katrs ieguldīja savus spēkus vienlīdzīgi, tāpēc darba sadalījums ir 50/50.