SOV Tekst Zadatka za Pripremu

Zadaci

Dobili ste dva fajla: `main.cpp` je osnova zadatka čijim editovanjem ga rešavate. Ovaj fajl je baziran skoro sasvim na programu koji ste imali kao primer, tako da bi trebalo da ste dobro upoznati sa tim kako radi. `viz.cpp` je vizuelizator, i ne morate da ga editujete. On je tu da vam dramatično olakša implementaciju zadatka, i sledeća sekcija priča o tome kako se koristi.

Vaši zadaci su da modifikujete `main.cpp` tako da omogućite sledeće nove funkcionalnosti: 1. Mora da je moguće da se alokacija nove memorije pauzira i nastavi. Ovo mora da može da se kontroliše preko vizuelizatora (videti treću sekciju oko detalja). 2. Mora da je moguće da se promeni režim alokacije nove memorije. Trenutno je implementiran 'first-fit' algoritam, ali smo mi na predavanjima diskutovali o alternativama. Vaš posao je da te alternative implementirate i to: last-fit, best-fit, worst-fit. Ovaj režim alokacije mora da može da se promeni iz vizuelizatora (videti treću sekciju). 3. Mora da je moguće da se pokrene kompakcija postojeće memorije. Ovaj proces kompakcie se izvršava u beskonačnoj petlji dok se ne pauzira. Kompakcija radi kao što smo pričali na predavanjima: morate pomerati zauzete segmente po memoriji tako da je memorija 'bolje složena' tj. da između zauzetih sekcija postoje što manji razmaci. Od vas se očekuje da osmislite odgovarajući algoritam. Ovaj algoritam, za potrebe ocenjivanja, ne mora biti optimizovan niti mora raditi savršen posao. Dakle, izazov nije da napravite savršen ili čak dobar algoritam za kompakciju, dovoljan je *nekakav* algoritam za kompakciju. Znate da algoritam radi tako što vam vizuelizator pokazuje sabijanje memorije u realnom vremenu. I kompakciju treba da možete da pokrenete iz vizuelizatora. Process kompakcije se ne izvršava u paraleli (tj. ima samo jedna nit kompaktor) i traje 60ms ako vrši pomeranje segmenta, odnosno 10ms ako nije našla nijedan segment vredan pomeranja (tj. memorija je već onoliko kompaktna koliko je algoritam može učiniti). Process kompakcije, tokom perioda obrade (tj. onih 60ms kopiranja) ne sme da drži zaključanim celu memoriju nego samo one delove koji su neophodni.

Parcijalno odrađeni zadaci *ulaze u obzir za odgovaranje.* Zadatak mora biti funkcionalan za pregledanje: mora se kompajlirati, mora biti adekvatno dokumentovan, mora biti adekvatno upakovan (videti kasnije detalje), ne sme da se ruši, i mora da radi *nešto* ali je savršeno OK da nemate odrađene neke stavke u zadatku, odbrana je ta koja određuje bodove, a kakav vam je zadatak određuje koja je forma te odbrane.

Ono što je apsolutno zabranjeno jeste bilo koja forma prepisivanja ili zajedničkog rada. Ni jedno ni drugo se neće tolerisati.

Prilikom pregledanja zadatak će se testirati na Linux računaru i biće kompajliran sa komandom:

```
g++ -pthread --std=c++14 -o main main.cpp
```

Kako koristiti vizuelizator

Vizuelizator služi da programu koji se izvršava možete slati komande i videti njegovo stanje u bilo kom trenutku.

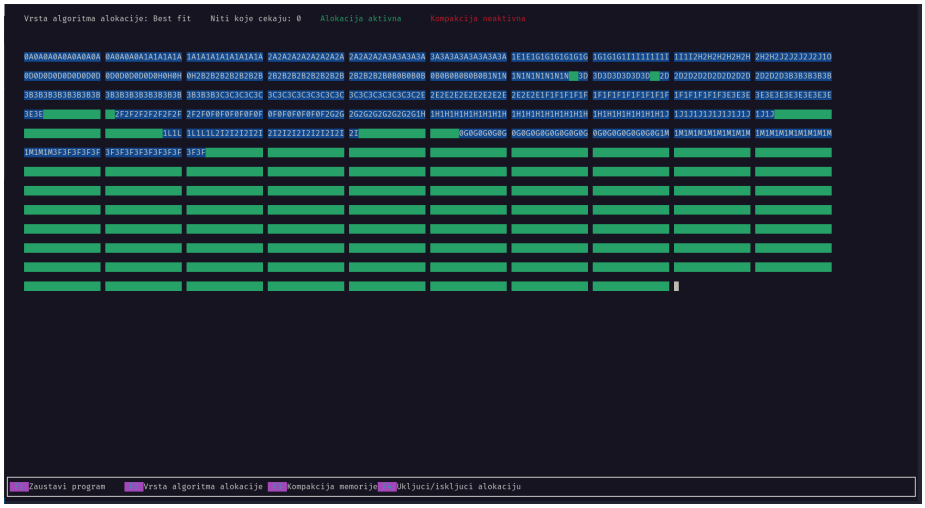

Figure 1: Prikaz ekrana vizuelizatora

Gore vidite tekući status koji uključuje algoritam alokacije memorije koji se koristi, broj niti koje su u stanju čekanja za memoriju plus status o tome da li je alokacija nove memorije uključena ili isključena, i status o tome da li je kompakcija memorije uključena ili ne.

Dole je šematski prikaz memorije (4MB). Svaka dva karaktera su jedna 4K stranica (memorija se alocira u inkrementima od 4096 bajtova isključivo da bi mogla lako da se prikaže). Zeleni prazni regioni su slobodni, plavi regioni su zauzeti. Svaki region je označen sa dva slova. Prvo je broj procesa koji je odgovoran, a drugo je broj segmenta označen slovima i simbolima od A pa nadalje. U slučaju velikog broja segmenata može se desiti da počne da korisiti i znakove interpunkcije.

Konačno na dnu prozora je paleta alata. F1 zaustavlja niti u drugom programu, F2 menja algoritme alokacije (to jest, šalje komandu za to, vi ste ti koji morate da implementirate promenu), F3 šalje komandu sa kompakciju memorije i F4 uključuje/isključuje alokacijju.

Vizualizator je kompletan i ne morate da ga editujete (mada ako baš želite, možete. U tom slučaju pošaljite izmenjen viz.cpp fajl i naglasite da ste to uradili u README.md fajlu, više o tome kasnije). Da bi ste ga koristili neophodno je da ga kompajlirate. Za ovo vam možda treba ncurses paket koji omogućava da se iscrtavaju grafičko-tekstuelni interfejsi. Paket je dostupan za svaku Linux distribuciju, i često je instaliran unapred. Sve što vam treba za instalaciju je:

```
sudo apt install libncurses-dev
```

Ovo radi za svaki Ubuntu posle 18.04 LTS. Kada ovo instalirate kompajliranje radi tako što napišete

```
g++ -o viz viz.cpp -lncurses
```

Upotreba vizuelizatora se jako preporučuje zato što čini rad na zadatku znatno lakšim, takođe mi testiramo zadatak koristeći ovu alatku. Ako nikakvim naporima ne možete da pokrenete vizuelizator (što ne bi trebalo da je moguće), program podržava i potpuno teksturalan režim izvršavanja ako ga pokrenete u t-režimu (podrazumevan je 'v' režim) tako što izvršite:

```
./main t
```

Onda sav feedback koji dobijate su tekst poruke ovde (verovatno vam je zgodno da vršite redirekciju output-a u fajl, onda), a komande se izdaju tako što ih otkucate i pristinete enter u konzoli gde je program, ignorišući to što se za vreme toga ispisuje tekst. Komande su `q` za izlazak, `c` za promenu tekućeg aktivnog moda alokacije, `a` za uključivanje/isključivanje alokacije, i `d` za uključivanje/isključivanje kompakcije memorije.

Pokretanje vizuelizatora mora da bude *pre* pokretanja glavng programa. Drugim rečima, pokrenete prvo vizuelizator koji pauzira i kaže vam da pokrenete glavni program. Onda pokrenete glavni program i vizuelizator bi treba da se prikaže u potpunosti. Ovo je neophodno zbog komunikacije preko FIFO pipe-ova.

Integracija vizuelizatora i vašeg programa

Vi ne morate da implementirate gotovo ništa u komunikaciji vizuelizatora i vašeg programa: klasa `Dijagnostics` to radi sasvim za vas i u normalnoj izradi programa ona se ne mora uopšte editovati. Klasa `Diagnostics` štampa na standardni izlaz log svega što se dešava i komunicira sa vizuelizatorom sama. Ono što je na vama, prvo, jeste da pozivate odgovarajuće metode za dijagnostiku kada se dese neki događaji u vašem programu. Već se automatski u postojećem kodu pozivaju metode za alokaciju, dealokaciju, itd. Ono što vi treba da pozovete kada implementirate taj deo vašeg koda jeste:

```
119 void compactionDeallocateMessage(u32 oLoc, u32 oLen, u32 loc, u32 len){
120             unique_lock<mutex> l(m);
121             cout << "The compacter asked to deallocate from " << oLoc << " to " << oLoc + oL
122             cout << "The compacter deallocated from " << loc << " to " << loc + len << endl;
123 
124             if(visual){
125                 int x = loc / 4096;
126                 int ll = len / 4096;
127                 for(int i = 0;i < ll;i++){
128                     outBuffer[7 + (x + i)*3 + 0] = 0;
129                     outBuffer[7 + (x + i)*3 + 1] = 0;
```

```
130                     outBuffer[7 + (x + i)*3 + 2] = 0;
131                 }
132             }
133         }
134 
135         void compactionMessage(int pid, int seg, u32 oBase, u32 len, u32 nBase){
136             unique_lock<mutex> l(m);
137             cout << "Process " << pid << " and segment " << seg << "of length " << len << "
138             if(visual){
139                 int x = nBase / 4096;
140                 int ll = len / 4096;
141                 for(int i = 0;i < ll;i++){
142                     outBuffer[7 + (x + i)*3 + 0] = 1;
143                     outBuffer[7 + (x + i)*3 + 1] = (char)pid;
144                     outBuffer[7 + (x + i)*3 + 2] = (char)seg;
145                 }
146             }
147         }
```

Kako ovo radi nije bitno, bitno je da prva služi kada proces kompakcije oslobodi nekakav prostor, a druga kada proces kompakcije premesti segment sa jedne na drugu adresu. Parametri prve su:

- **`oLoc`** Šta smo krenuli da dealociramo, tj. gde je segment koji oslobađamo.
- **`oLen`** Koliko je veliko to što smo dealocirali, tj. koliki je segment koji oslobađamo.
- **`loc`** Gde je ono što u stvari dealociramo kada se potencijalno izvrši spajanje sa prethodnim odsečkom slobodne memorije.
- **`len`** Koliko je ono što u stvari dealociramo kada se potencijalno izvrši svo spajanje sa odsečcima.

Parametri druge su:

- **`pid`** Process ID segmenta kojij se pomera
- **`seg`** ID Segmenta koji se pomera
- **`oBase`** gde je segment bio
- **`len`** Koliki je segment koji se pomera
- **`nBase`** gde je segment pomeren

Vizuelizator/klasa `Diagnostics` je takođe odgovoran za kontrolu vašeg programa i prenosiće komande za vaš kod, i to: 1. Komanda sa izlazak iz programa je već implementirana. 2. Komanda za uključivanje i isključivanje alokacije će promeniti globalnu promenljivu `allocationEnabled` tako da je ili `true` ili `false` i pozvaće funkciju `onAllocationChanged` 3. Komanda za uključivanje i isključivanje kompakcije će promeniti globalnu promenljivu `compactingActive` u `true` ili `false` i pozvaće funkciju `onCompactionChanged` 4. Komanda za promenu tipa automatski podešava promenljivu `type` na odgovarajući sledeći tip algoritma za alokaciju sama i poziva `onTypeChanged`.

```
32 const int EF_FIRSTFIT = 0;
33 const int EF_LASTFIT = 1;
34 const int EF_BESTFIT = 2;
35 const int EF_WORSTFIT = 3;
```

Tipovi algoritma za alokaciju koje treba implementirati.

Šta zadatak treba da ima?

Minimalan zadatak je `main.cpp` datoteka zapakovana u arhivi oblika SOV-RA14-2072.zip za broj indeksa RA 14/2072. Arhiva može takođe sadržati i `viz.cpp` ako je on menjan. Konačno, arhiva može imati i `README.md` fajl koji u Markdown formatu (ako ne znate kako on radi tretirajte ga kao običan tekst fajl) sadrži vaše komentare o zadatku. `README.md` je *obavezan* ako: 1. Ste koristili tekstualni mod, ne vizuelizator. 2. Ste menjali vizuelizator. 3. Niste uradili ceo, nego samo deo zadatka.

U bilo kom od ovih slučajeva, morate to naznačiti u README.md fajlu i, u slučaju rada samo dela zadatka, navesti precizno ali konzizno tačno šta ste radili a šta niste.

Arhiva se mora zvati u ovoj formi i fajlovi u njoj moraju imati ova imena, i arhiva ne sme sadržati *bilo šta drugo.* Ako prekršite pravila pakovanja i dokumentacije može da se desi da ne budete pozvani na odgovaranje. Na odgovaranje takođe nećete biti pozvani ako vam je zadatak nefunkcionalan odnosno ne kompajlira se.

Saveti

Tipovi podataka Vodite računa da je `u32` `unsigned` tip, i da će se operacije između `unsigned` tipova vršiti u `unsigned` režimu. To znači da negativni brojevi postaju (u ovom režimu) jako veliki brojevi. Ako morate raditi sa negativnim brojevima kod računanja razdaljina i sl. vodite računa da vrednosti prevedete u adekvatan `signed` oblik.

Sinhronizacioni problemi Vodite računa da se `mutex` recimo ne može kopirati nikako. To znači da kada implementirate sinhronizaciju za individualne segmente to morate raditi tako što relevantne mutex-e čuvate negde gde signurno neće nikada biti kopirani.

Algoritam kompakcije Funkcionisanje algoritma kompakcije će biti jako vidljivo na vizuelizatoru. To je najbolji način da proverite da li radi. Ako imate problema u debagovanju ne zaboravite da su praktično sve poruke ispisa koncentrisane u klasi `Diagnostics`, te da možete u njoj vrlo lako zakomentarisati one funkcionalnosti koje vam ne trebaju da bi lakše videli vaše debag poruke.