

SOV Tekst Zadatka za Pripremu

Školska 2021/2022

Uvod

Pred vama je pojednostavljeni simulator ponašanja menadžmenta memorijom u nekom operativnom sistemu. U ovom specifičnom slučaju, procesi mogu da imaju neki broj segmenata koje koriste. Svaki proces ima svoju tabelu segmenata, a sistem vodi računa i o evidenciji slobodne memorije što se radi u obliku linkovane liste odsečaka, na način koji je rađen na času (regioni memorije koji su slobodni koje opisuje pozicija u memoriji i veličina i koji su u jednostruko linkovanoj listi). Proces može da čita ili piše u svoje segmente slobodno, da dealocira segment koji više ne koristi, ili alocira novi. Operativni sistem se bavi alokacijom i može (u ovom rešenju) da koristi First Fit, Best Fit, Last Fit, i Worst Fit algoritme. Sistem poseduje i algoritam za kompakciju memorije koji se može pokrenuti u bilo kom trenutku i koji razdvaja alociranu i dealociranu memoriju.

Od vas će se očekivati da ovo rešenje razumete i da ga modifikujete tako da ima novu funkcionalnost. Delovi postojećeg rešenja za koje se ne očekuje da ih razumete jesu delovi koji implementiraju vizuelizator ili komunikaciju sa njim.

Pripremni zadatak se *mora* izrađivati samostalno ali je upotreba dokumentacije sasvim dozvoljena.

Zadaci

Dobili ste dva fajla: `main.cpp` je osnova zadatka čijim editovanjem ga rešavate. Ovaj fajl je baziran na prošlogodišnjem zadatku po sadržaju, ali je sam zadatak drugačiji i po težini i po tematici. `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:

Za SOV-A (što vas sprema za parcijalno odgovaranje na samom SOV-u) potrebno je da: 1. Obezbedite da se u vizuelizatoru u gornjoj liniji stalno ispisuje koliko ima ukupno slobodne memorije i to u 4K stranicama i u ili bajtima, kilobajtima, ili megabajtima. Jedinica se bira tako da se uzme najveća jedinica gde je vrednost i dalje barem jedan, tj. u slučaju da ima 984K slobodno, treba da se ispiše 984K, ali ako ima 1025K slobodno, onda treba da se ispiše 1M. Konačno, treba da se ispiše veličina (u stranicama od 4K) najvećeg kontinualnog slobodnog regiona u memoriji. Postoji gotova metoda u klasi 'Dijagnostika' koja se zove `reportFreeSpace` kojoj treba da date relevantne podatke i koja će odraditi svu dalju komunikaciju sa vizuelizatorom.

Za SOV-AB, odnosno potpun SOV pripremni zadatak (što vas sprema za potpuno odgovaranje na samom SOV-u) potrebno je da: 1. Obezbedite da se u vizuelizatoru u gornjoj liniji stalno ispisuje koliko ima ukupno slobodne memorije i to u 4K stranicama i u ili bajtima, kilobajtima, ili megabajtima. Jedinica se bira tako da se uzme najveća jedinica gde je vrednost i dalje barem jedan, tj. u slučaju da ima 984K slobodno, treba da se ispiše 984K, ali ako ima 1025K slobodno, onda treba da se ispiše 1M. Konačno, treba da se ispiše veličina (u stranicama od 4K) najvećeg kontinualnog slobodnog regiona u memoriji. Postoji gotova metoda u klasi ‘Dijagnostika’ koja se zove `reportFreeSpace` kojoj treba da date relevantne podatke i koja će odraditi svu dalju komunikaciju sa vizuelizatorom. 2. Samostalno implementirate mehanizam koji održava globalnu tabelu segmenata svih procesa i u slučaju da je mehanizam LRU-a aktiviran preko vizuelizatora (što će pokazati globalni flag `lruActive`) i da su svi procesi u stanju čekanja, ovaj proces dealocira onaj segment sa najstarijom referencom (tj. iz koga je najdavnije čitano odnosno u koji je najdavnije pisano) u skladu sa LRU algoritmom. Onda obavesti procese koji čekaju na slobodnu memoriju, čeka 30ms, i opet proverava (u beskonačnoj petlji) da li su svi procesi u stanju čekanja. Smete koristiti ili LRU ili NFU u vašoj implementaciji u skladu sa tim šta vam je lakše.

Kada predate zadatak, *morate* reći da li predajete SOV-A ili SOV-AB, i odgovaraćete u skladu sa tim. Zadaci moraju da se kompajliraju i moraju da rade (bolje ili lošije) ono što je napisano gore da bi mogli biti predmet odgovaranja. Zadaci koji se ne kompajliraju ili se neizbežno ruše ili nemaju funkcionalnosti ne mogu da budu predmet odgovaranja. Zadaci koji imaju nekakav defekt u radu, povremeni bag, ili se, recimo, ruše prilikom izlaska iz programa dolaze u obzir. Zadaci koji nisu adekvatno imenovani, upakovani, i predani u skladu sa instrukcijama ne dolaze u obzir.

Savršeno OK da imate neki bag u zadatku ili nekakav sitan defekt, odgovaranje je to što određuje bodove, a kakav vam je zadatak određuje koja je forma tog odgovaranja. Dok god ima nešto što radi dovoljno da se o tome može pričati, može se odgovarati.

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.

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



Figure 1: Prikaz ekrana vizualizatora

li je alokacija nove memorije uključena ili isključena, status o tome da li je kompakcija memorije uključena ili ne, i status o tome da li je LRU mehanizam uključen ili ne. Takođe ispisano je koliko ima slobodne memorije, u skladu sa SOV-A.

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 koristiti 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, promena je implementirana u main.cpp), F3 šalje komandu za kompakciju memorije, F4 uključuje/isključuje alokaciju, a F5 šalje komandu za uključivanje i isključivanje LRU mehanizma. To se u kodu manifestuje tako što se `lruActive` flag promeni i pozove se funkcija `onLRUChanged`. Ostatak implementacije je na vama, ako ste odabrali da radite pun SOV.

Vizualizator je kompletan i ne morate da ga editujete. 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
```

ili ekvivalentna komanda za druge distribucije. Kada ovo instalirate kompajli-

ranje 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, **d** za uključivanje/isključivanje kompakcije memorije, **i** **n** za uključivanje/isključivanje LRU mehanizma.

Pokretanje vizuelizatora mora da bude *pre* pokretanja glavnog 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. Primer su metode koje izveštavaju u kompakciji memorije.

```
145 void compactionDeallocateMessage(u32 oLoc, u32 oLen, u32 loc, u32 len){
146     unique_lock<mutex> l(m);
147     cout << "The compacter asked to deallocate from " << oLoc << " to " << oLoc + oLen << endl;
148     cout << "The compacter deallocated from " << loc << " to " << loc + len << endl;
149
150     if(visual){
151         int x = loc / 4096;
152         int ll = len / 4096;
153         for(int i = 0; i < ll; i++){
154             outBuffer[7 + (x + i)*3 + 0] = 0;
155             outBuffer[7 + (x + i)*3 + 1] = 0;
156             outBuffer[7 + (x + i)*3 + 2] = 0;
157         }
158     }
```

```

159     }
160
161     void compactionMessage(int pid, int seg, u32 oBase, u32 len, u32 nBase){
162         unique_lock<mutex> l(m);
163         cout << "Process " << pid << " and segment " << seg << " of length " << len << "
164         if(visual){
165             int x = nBase / 4096;
166             int ll = len / 4096;
167             for(int i = 0; i < ll; i++){
168                 outBuffer[7 + (x + i)*3 + 0] = 1;
169                 outBuffer[7 + (x + i)*3 + 1] = (char)pid;
170                 outBuffer[7 + (x + i)*3 + 2] = (char)seg;
171             }
172         }
173     }

```

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**. 5. Komanda za uključivanje i isključivanje kompakcije će promeniti globalnu promenljivu **lruActive** u **true** ili **false** i pozvaće funkciju **onLRUChanged**.

Šta zadatak treba da ima?

Minimalan zadatak je `main.cpp` datoteka koja na početku ima komentar oblika

```
1 //RA 123/2072 Petar Petrovic, Termin vezbi: Pon 07:00. Radjeno: SOV-AB. Komentari: ...
```

Komentar je obavezan. Ova datoteka se predaje primarno preko uputstva iz fajla `UputstvoZaUploadNaTeams.pdf` što opisuje kako se predaje fajl preko Teams platforme. Za svaki slučaj, možete `main.cpp` i *ništa drugo* slati na `pveljko@uns.ac.rs`. Mail koji pošaljete na tu adresu kao rezervu *mora* da ima subject koji počinje sa `[SOV]`. Morate predati rad preko Teams platforme, mail služi samo kao rezervna kopija u slučaju tehničkih problema. Ako mail nije dobro formatiran, biće detektovan kao spam i obrisano. Ako, pak, jeste dobro formatiran i nešto pođe naopako u upload-u preko Teams-a, dobro formatirana mail verzija garantuje da i dalje možete da odgovarate.

Morate predati zadatak do 12:00 u ponedjeljak 06.06.2022.

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.

Šta je LRU LRU je objašnjen nekoliko puta na predavanjima i ima ga i u udžbeniku i slajdovima. Kratko rečeno: za segmente čuva vreme kada je zadnji put pozvana `read` odn. `write` operacija nad njima. Kada dođe vreme da se nešto dealocira (ovde možemo da zamišljamo da se smešta na nekakav disk, mada naravno to ne morate da implementirate) bira se ono što je najdavnije korišćeno u skladu sa pravilom lokalnosti.

LRU alternative Bilo šta što oslobađa segmente na osnovu vremena referenciranja dolazi u obzir kao algoritam. Ako ne možete da implementirate pun LRU ili NFU/Aging, implementirajte šta možete.