Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (13E113OS2, 13S113OS2)
Nastavnik: prof. dr Dragan Milićev
Školska godina: 2017/2018. (Zadatak važi počev od januarskog roka 2018.)

Projekat za domaći rad
- Projektni zadatak –

Verzija dokumenta: 1.1

Važne napomene: Pre čitanja ovog teksta, **obavezno** pročitati opšta pravila predmeta i pravila vezana za izradu domaćih zadataka! Pročitati potom ovaj tekst **u celini i pažljivo**, pre započinjanja realizacije ili traženja pomoći. Ukoliko u zadatku nešto nije dovoljno precizno definisano ili su postavljeni kontradiktorni zahtevi, student treba da uvede razumne pretpostavke, da ih temeljno obrazloži i da nastavi da izgrađuje preostali deo svog rešenja na temeljima uvedenih pretpostavki. Zahtevi su namerno nedovoljno detaljni, jer se od studenata očekuje kreativnost i profesionalni pristup u rešavanju praktičnih problema!

Uvod

Cilj ovog zadatka jeste implementacija uprošćenog školskog sistema za virtuelnu memoriju. Sistem za virtuelnu memorije treba da obezbedi sledeće funkcionalnosti: alokaciju i dealokaciju logičkih segmenata virtuelnog adresnog prostora sa zadavanjem prava pristupa svim stranicama u okviru tih segmenata, deljenje segmenata (drugi deo zadatka), učitavanje stranica na zahtev i zamenu stranica na prostor za zamenu (engl. *swap space*). U sistemu može da postoji više procesa koji dele fizičku memoriju. Treba obezbediti i mogućnost zamene stranica prebacivanjem stranica na hard disk. Za pristup prostoru za zamenu na hard disku obezbeđena je apstrakcija particije u vidu klase `Partition` koja pruža sve potrebne operacije za pristup podacima na disku.

Zadatak se sastoji od dva dela. Prvi deo je obavezan i za uspešnu odbranu projektnog zadatka student mora da ga uradi. Algoritmi za zamenu stranica, za alokaciju slobodnog prostora, sprečavanje pojave zvane *thrashing* i ostali algoritmi i optimizacije se ostavljaju u nadležnosti samog rešenja i biće razmatrani samo u okviru testiranja performansi sistema.

Opšti zahtevi

Odnos projekta i korisničke aplikacije

Tražene podsisteme treba realizovati na jeziku C++. Korisničku aplikaciju, koja sadrži test primere, i biblioteku part.lib, koja predstavlja apstrakciju particije namenjenu za pristup particijama na hard diskovima, treba povezati sa prevedenim kodom projekta u jedinstven konzolni program (.exe). U datoj aplikaciji biće prisutna i funkcija *main*.

Odnos projekta i operativnog sistema domaćina

Projekat se realizuje pod operativnim sistemom Windows 7 x64 ili novijim, koji je u ovom slučaju operativni sistem domaćin. Izradom projekta se ni na koji način ne sme ugroziti ispravno funkcionisanje operativnog sistema domaćina. Svaki eventualni problem koji se pojavi po pokretanju projekta biće smatran kao greška pri izradi projekta. Po završetku rada, okruženje je neophodno ostaviti u neizmenjenom stanju u odnosu na trenutak pre pokretanja projekta, osim onih delova koji se namerno menjaju samim test primerom koji treba da proveri ispravnost projekta. Svi resursi u sistemu koji će biti korišćeni pri izradi projekta moraju biti korišćeni kroz odgovarajući API operativnog sistema domaćina koji je za to namenjen. Deo koda koji je obezbeđen u okviru postavke projekta je pažljivo napisan, i ukoliko se koristi u skladu sa uputstvom za rad, ne može prouzrokovati nikakve probleme i greške pri izvršavanju.

Prvi deo: Virtuelna memorija (20 poena)

Uvod

Za evidenciju virtuelnog adresnog prostora jezgro operativnog sistema koristi hardversku podršku za straničnu organizaciju virtuelne memorije. Pored vođenja evidencije o virtuelnoj memoriji, potrebno je napraviti i simulaciju posla koji vrši hardver. Simulacija treba realno da odslikava mogućnosti tehnologije za izradu hardvera (složenost operacije treba da bude $O(1)$). Način preslikavanja pomoću tabela je u nadležnosti samog rešenja (veličina tabela, sadržaj tabela, broj tabela itd.). Na početku rada sistemu će biti dodeljen prostor u memoriji za evidenciju virtuelne memorije, prostor u memoriji za korisničke procese i prostor na disku za zamenu stranica. Van zadatog prostora se smeju alocirati samo male strukture podataka za potrebe sistema, ali nikako delovi tabela ili stranica. Pristup sistemu se vrši iz više niti, pa je potrebno obezbediti adekvatnu sinhronizaciju. Veličina virtuelne adrese je 24 bita, dok je veličina stranice 1KB. Veličinu fizičke adrese treba odrediti na osnovu veličine zadatog fizičkog memorijskog prostora.

Particija

Opis datog interfejsa za pristup particiji

Za potrebe zamene stranica studentima je data na raspolaganje klasa koja predstavlja jednu particiju. Operacije koje ova klasa u svom interfejsu obezbeđuje su (opisi su poređani po istom redosledu kao i metode u klasi):

- kreiranje objekta particije; pri kreiranju se zadaje naziv konfiguracionog fajla koji sadrži sve ostale informacije potrebne za formiranje ovog objekta;
- dohvatanje informacije o broju klastera na particiji;
- čitanje jednog (i samo jednog) klastera; zadaje se redni broj klastera (klasteri su numerisani počev od 0); pročitani podaci se smeštaju na zadato mesto u memoriji; operacija vraća: 0 – neuspeh ili 1 – uspeh; na pozivaocu leži odgovornost da u memoriji obezbedi slobodan prostor dovoljne veličine za smeštanje traženih podataka;
- upis jednog (i samo jednog) klastera; zadaje se redni broj klastera za upis (klasteri su numerisani počev od 0); podaci za upis se čitaju sa zadatog mesta u memoriji; vraća: 0 – neuspeh ili 1 – uspeh;

Sve date operacije su *thread-safe*, što znači da se potpuno bezbedno mogu pozivati iz konkurentnih niti. Operacije čitanja i upisa su blokirajuće. Sa ovim klasterom moguće je pristupiti samo onim klasterima koji u potpunosti pripadaju particiji. Radi pojednostavljenja, za veličinu klastera je uzet 1KB i taj parametar nije moguće menjati.

Particija na jeziku C++

Particija je realizovana klasom `Partition` čija se potpuna definicija nalazi u zaglavlju `part.h`. Interfejs ove klase definisan je u sledećem fajlu:

```
// File: part.h

typedef unsigned long ClusterNo;
const unsigned long ClusterSize = 1024;


class Partition {
```

```
public:
      Partition(const char *);

      virtual ClusterNo getNumOfClusters() const;

      virtual int readCluster(ClusterNo, char *buffer);
      virtual int writeCluster(ClusterNo, const char *buffer);

      virtual ~Partition();
private:
      PartitionImpl *myImpl;

};
```

Sistem

Opis zadatka i funkcionalnosti

Potrebno je realizovati klasu `System`, koja dobija na raspolaganje prostor u memoriji za smeštanje tabela preslikavanja virtuelne memorije i stranica procesa, kao i prostor na hard disku za zamenu stranica. Sistem treba da obezbedi pravednu raspodelu prostora između procesa. U slučaju da nema prostora za realizaciju neke operacije, sistem treba da odbije tu operaciju.

Klasa `System` treba da realizuje sledeće funkcionalnosti:

- kreiranje objekta klase `System`; tom prilikom se prosleđuje pokazivač na početak prostora u memoriji za smeštanje stranica procesa, kao i veličina tog prostora izraženog u broju stranica, pokazivač na početak prostora u memoriji za smeštanje tabela preslikavanja stranica, kao i veličina tog prostora izraženog u broju stranica, i pokazivač na particiju na disku koja se koristi za zamenu stranica;
- uništavanje objekta klase `System`; prilikom uništavanja treba dealocirati memoriju koju je objekat zauzeo tokom svog izvršavanja, ali ne i ono što mu je prosleđeno prilikom kreiranja;
- kreiranje jednog procesa; stranice procesa na početku ne treba da budu učitane;
- operaciju koja će biti izvršavana povremeno; u toj operaciji može da se radi bilo šta, ali treba voditi računa da poziv ove operacije može biti vremenski zahtevan; povratna vrednost funkcije označava za koliko vremena (izraženom u μs) treba da se pozove ova operacija sledeći put, 0 u slučaju da ona više ne treba da se poziva;
- simulaciju operacije koju treba da izvrši hardver prilikom svakog pristupa nekoj virtuelnoj adresi; parametri operacije su identifikator procesa koji pristupa adresi, kojoj virtuelnoj adresi se pristupa i o kom pristupu se radi (upis, čitanje ili izvršavanje); u slučaju da stranica nije u memoriji ili se pokušava nedozvoljeni pristup, treba vratiti odgovarajući status.

Klasa `System` treba da bude samo interfejs za tražene operacije, dok implementacija traženih operacija treba da bude u klasi `KernelSystem`. Implementacija klase `KernelSystem` se ostavlja u nadležnost samog rešenja.

Sistem na jeziku C++

Za potrebe vođenja evidencije o procesima potrebno je implementirati klasu `System`. Interfejs klase `System` je dat u zaglavlju `System.h`:

```
// File: System.h
#include "vm_declarations.h"

class Partition;
class Process;
class KernelProcess;
class KernelSystem;

class System {
public:
    System(PhysicalAddress processVMSpace, PageNum processVMSpaceSize,
           PhysicalAddress pmtSpace, PageNum pmtSpaceSize,
           Partition* partition);
    ~System();

    Process* createProcess();

    Time periodicJob();

    // Hardware job
    Status access(ProcessId pid, VirtualAddress address, AccessType type);
private:
    KernelSystem *pSystem;
    friend class Process;
    friend class KernelProcess;
};
```

Proces

Opis zadatka i funkcionalnosti

Klasa `Process` treba da realizuje apstrakciju jednog procesa. Potrebno je voditi evidenciju o virtuelnoj memoriji procesa u toku njegovog izvršavanja i preslikavanje virtuelnih adresa u fizičke adrese za potrebe testova. U slučaju da neka operacija ne može da se izvrši, treba vratiti odgovarajući status i sve ostaviti u nepromenjenom stanju. Sve tabele i stranice procesa moraju biti u memoriji koja je dodeljena sistemu, dok sam objekat klase `Process` i drugi mali objekti mogu da budu bilo gde (recimo napravljeni pomoću standardnog operatora `new`). Virtuelni adresni prostori procesa se ne preklapaju.

Rešenje treba da obezbedi podršku za logičke segmente koje proces može da alocira i dealocira, a koji su obavezno poravnati na početak stranice. Sve stranice u jednom logičkom

segmentu imaju ista prava pristupa. Korisnik sve operacije vrši na nivou logičkih segmenata, a ne pojedinačnih stranica.

Klasa `Process` treba da realizuje sledeće funkcionalnosti:

- kreiranje objekta klase `Process`; parametar koji se prosleđuje je jedinstveni identifikator procesa;
- uništavanje objekta klase `Process`;
- proširenje virtuelnog adresnog prostora određenim brojem sukcesivnih stranica, koje čine jedan segment (alokacija logičkog segmenta); navodi se virtuelna adresa početka segmenta (greška je ako zadata adresa nije poravnata na početak stranice ili ukoliko se segment preklapa sa već alociranim segmentom); kao parametar se prosleđuju i prava pristupa do stranica tog segmenta;
- inicijalizaciju jednog segmenta u virtuelnom adresnom prostoru; metoda vrši istu funkciju kao i prethodna, samo što na kraju i inicijalizuje stranice sa sadržajem koji je dat kao parametar (smatrati da je sadržaj iste veličine kao i segment koji se učitava);
- uklanjanje jednog segmenta iz virtuelnog adresnog prostora; zadaje se virtuelna adresa segmenta koji se uklanja (greška je ako zadata adresa nije početak segmenta);
- preslikavanje virtuelne adrese u fizičku adresu (vratiti 0 ako preslikavanje nije moguće iz bilo kog razloga);
- obradu stranične greške; parametar je virtuelna adresa koja je izazvala straničnu grešku;

Klasa `Process` treba da bude samo interfejs za tražene operacije, dok implementacija traženih operacija treba da bude u klasi `KernelProcess`. Implementacija klase `KernelProcess` se ostavlja u nadležnosti samog rešenja.

Proces na jeziku C++

Za potrebe vođenja evidencije o adresnom prostoru procesa potrebno je implementirati klasu `Process`. Interfejs klase `Process` je dat u zaglavlju `Process.h`:

```
// File: Process.h
#include "vm_declarations.h"

class KernelProcess;
class System;
class KernelSystem;

class Process {
public:
    Process(ProcessId pid);
```

```
    ~Process();
    ProcessId getProcessId() const;


    Status createSegment(VirtualAddress startAddress, PageNum segmentSize,
                         AccessType flags);
    Status loadSegment(VirtualAddress startAddress, PageNum segmentSize,
                       AccessType flags, void* content);
    Status deleteSegment(VirtualAddress startAddress);

    Status pageFault(VirtualAddress address);
    PhysicalAddress getPhysicalAddress(VirtualAddress address);
private:
    KernelProcess *pProcess;
    friend class System;
    friend class KernelSystem;
};
```

Zajedničke deklaracije tipova i konstanti na jeziku C++

Za sve klase u zaglavlju `vm_declarations.h` date su deklaracije tipova i konstanti:

```
// File: vm_declarations.h

typedef unsigned long PageNum;
typedef unsigned long VirtualAddress;
typedef void* PhysicalAddress;
typedef unsigned long Time;

enum Status {OK, PAGE_FAULT, TRAP};

enum AccessType {READ, WRITE, READ_WRITE, EXECUTE};

typedef unsigned ProcessId;

#define PAGE_SIZE 1024
```

Drugi deo: Deljenje segmenata (10 poena)

Uvod

Cilj drugog dela projekta jeste da se obezbedi deljenje segmenata među procesima i mogućnost za kloniranje jednog procesa. Procesi mogu da dele segmente tako što se odgovarajuće stranice dva segmenta dva različita procesa nalaze u fizičkoj memoriji u istim okvirima. Prilikom kreiranja deljenih segmenata, segmentu se dodeljuje jedinstveno ime u sistemu. Ukoliko se zatraži kreiranje deljenog segmenta sa imenom koje već postoji u sistemu, onda taj postojeći segment postaje deo virtuelnog adresnog prostora procesa na zadatoj virtuelnoj adresi (greška je ako zadata adresa nije poravnata na početak stranice). Moguće je isključiti deljeni segment iz adresnog prostora procesa, kao i ukloniti deljeni segment iz sistema, pri čemu se izbacuje iz adresnog prostora svih procesa koji su ih koristili. Prilikom kloniranja procesa potrebno je prekopirati čitav virtuelni adresni prostor procesa zajedno sa kopiranjem objekta klase `Process`. Za tražene funkcionalnosti potrebno je proširiti interfejse datih klasa sledećim metodama:

```
class Process {
public:
    …
    Process* clone(ProcessId pid);
    Status createSharedSegment(VirtualAddress startAddress,
                PageNum segmentSize, const char* name, AccessType flags);

    Status disconnectSharedSegment(const char* name);

    Status deleteSharedSegment(const char* name);
    …
};


class System {
public:
    …
    Process* cloneProcess(ProcessId pid);
    …
};
```

Testovi

Javni testovi

Javni test-program služi da pomogne studentima da elementarno testiraju svoj projekat. Ovi testovi neće obavezno pokriti sve funkcionalnosti koje projekat treba da ima, ali će testirati većinu tih funkcionalnosti. Da bi se projekat uopšte odbranio, neophodno je da projekat sa javnim testom radi u potpunosti ispravno. Studentima se preporučuje da pored javnog testa naprave i svoje iscrpne testove koji će im pomoći da što bolje istestiraju svoj projekat.

Tajni testovi

Tajni testovi detaljnije testiraju sve zahtevane funkcionalnosti u različitim regularnim i neregularnim situacijama (greške u pozivu ili radu), i nisu unapred dostupni studentima.

Testovi performansi

Testovi performansi mere vreme izvršavanja procesa. Ovi testovi nisu obavezni, i mogu, ali ne moraju, doneti dodatne bodove u predroku posle nastave za do 20 najboljih odbranjenih radova. Za potrebe povećanja performansi, sistem može pokrenuti jednu i samo jednu nit u pozadini koja ima samo pravo da pristupa particiji. Ukoliko se implementira algoritam za sprečavanje pojave zvane *thrashing* obezbediti u klasi Process metodu blockIfThrashing() u kojoj treba da se blokira tekuću nit ukoliko se želi blokiranje procesa. Kada se želi nastavak procesa zablokirana nit treba da se odblokira i da se izvrši povratak iz metode.

Zaključak

Potrebno je realizovati opisane podsisteme prema datim zahtevima na jeziku C++. Kao integrisano okruženje za razvoj programa (engl. integrated development environment, IDE) zahteva se Microsoft Visual Studio 2015 radi kompatibilnosti sa datom bibliotekom part.lib. Testiranje se vrši u laboratorijama katedre na računarima pod operativnim sistemom Windows 10 x64.

Pravila za predaju projekta

Projekat se predaje isključivo kao jedna zip arhiva. Sadržaj arhive podeliti u dva foldera: src i h. U prvom folderu (src) treba da budu smešteni svi .cpp fajlovi koji su rezultat izrade projekta, a u drugom folderu (h) treba da budu svi .h fajlovi koji su rezultat izrade projekta. Opisani sadržaj ujedno treba da bude i jedini sadržaj arhive (arhiva ne sme sadržati ni izvršne fajlove, ni biblioteke, ni .cpp i .h fajlove koji predstavljaju bilo kakve testove, niti bilo šta što iznad nije opisano). Projekat je moguće predati više puta, ali do trenutka koji će preko imejl liste biti objavljen za svaki ispitni rok i koji će uvek biti pre ispita. Na serveru uvek ostaje samo poslednja predata verzija i ona će se koristiti na odbrani. Za izlazak na ispit neophodno je predati projekat (prijava ispita i položeni kolokvijumi su takođe preduslovi za izlazak na ispit). Nakon isteka roka za predaju, projektni zadaci se brišu sa servera, pa je u slučaju ponovnog izlaska na ispit potrebno ponovo postaviti ažurnu verziju projektnog zadataka.

Sajt za predaju projekta je https://rti.etf.bg.ac.rs/domaci/index.php?servis=os2_projekat

Zapisnik revizija

Ovaj zapisnik sadrži spisak izmena i dopuna ovog dokumenta po verzijama.

Verzija 1.1

Strana	Izmena
8, 9	Uklonjen tip AccessRight. Sva prava pristupa se izražavaju pomogu tip AccessType.
10	Dodata metoda blockIfThrashing() u klasi Process u delu za testiranje performansi.