Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (13E113OS2, 13S113OS2)
Nastavnik: prof. dr Dragan Milićev
Školska godina: 2022/2023. (Zadatak važi počev od januarskog roka 2023.)

Projekat za domaći rad
- Projektni zadatak –

Verzija dokumenta: 1.0

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 alokaciju memorije. Sistem za alokaciju memorije treba da obezbedi alokaciju i dealokaciju memorije na nivou jezgra operativnog sistema. Na nivou jezgra treba obezbediti da se alokacija i dealokacija podataka samog jezgra radi brzo i kompaktno, korišćenjem sistema ploča (engl. *slab*) i sistema parnjaka (engl. *buddy*).

Traženi zahtevi se implementiraju za školsko jezgro realizovano za RISC V procesor na predmetu Operativni sistemi 1. Implementirano jezgro će se izvršavati u virtuelnom okruženju – emulatoru procesora RISC V. Studenti koji nisu pravili školsko jezgro za RISC V procesor na predmetu Operativni sistemi 1, mogu da implementiraju tražene zahteve za xv6 operativni sistem (xv6 operativni sistem treba da radin na dva procesora).

Opšti zahtevi

Odnos projekta i korisničke aplikacije

Tražene podsisteme treba realizovati na jeziku C++. Korisničku aplikaciju, koja sadrži test primere, treba povezati sa prevedenim kodom projekta u jedinstven konzolni program (.exe). U datoj aplikaciji biće prisutna i funkcija *main*.

Tražene podsisteme treba realizovati na jeziku C++. Korisnička aplikacija sadržaće test primere i biće obezbeđena kao skup izvornih fajlova koje treba prevesti i povezati sa prevedenim kôdom jezgra i datim bibliotekama `app.lib` i `hw.lib` u jedinstven program (`.exe`). Biblioteka `app.lib` sadržaće preveden i povezan kôd korisničkog test programa. Biblioteka `hw.lib` sadržaće module koji se obezbeđuju kao moduli koji pristupaju (zamišljenom, virtuelnom) hardveru, odnosno moduli od kojih će jezgro zavisiti (engl. *stub*).
Glavni program, tj. izvor kontrole toka korisničke aplikacije treba da bude u funkciji:

```
void userMain ();
```

Funkcija `main()` nema argumente niti povratnu vrednost i ostaje u nadležnosti samog jezgra, pa realizacija jezgra može da pod svojom kontrolom ima radnje koje će se izvršiti pri pokretanju programa, a zatim treba da pokrene nit nad funkcijom `userMain()`.

Odnos projekta i operativnog sistema domaćina

Jezgro i korisnička aplikacija treba da se posmatraju kao jedinstven program (`.exe`) dobijen prevođenjem i statičkim povezivanjem kôda na izvornom programskom jeziku na kom su realizovani. Oni će biti pokrenuti unutar edukativnog operativnog sistema xv6 kao *sistema-domaćina*. Ovaj sistem xv6 je za ovu priliku značajno modifikovan tako što su mu izbačene mnoge funkcionalnosti (promena konteksta i raspoređivanje procesa, upravljanje memorijom, fajl podsistem, upravljanje diskom itd.). Sistem-domaćin ima ulogu samo da se sam pokrene i inicijalizuje ciljni hardver, potom napravi samo jedan proces sa virtuelnim adresnim prostorom koji zauzima celu raspoloživu fizičku memoriju, učita kôd programa koji čine realizovano jezgro i sa njim povezana aplikacija i zatim pokrene njegovo izvršavanje u kontekstu tog jedinog procesa. Osim toga, sistem-domaćin obezbeđuje i osnovne usluge hardvera: periodičan prekid od tajmera i pristup konzoli (tastaturi i ekranu). Na PC računarima i njihovim operativnim sistemima cela ova računarska platforma (procesor RISC V, tajmer i konzola, kao i sistem-domaćin xv6) simuliraju se odgovarajućom virtuelnom mašinom (emulatorom).
Sve ovo jednostavno znači da će realizovano jezgro posmatrati svoju platformu kao jednostavan računar sa RISC V procesorom i jedinstvenim adresnim prostorom operativne memorije (samo fizički adresni prostor) u koji je učitan izvršiv program dobijen povezivanjem jezgra i aplikacije, kao što je to slučaj kod ugrađenih sistema.

Jezgro i korisnička aplikacija moraju da se kao program regularno završe, naravno ukoliko u samoj korisničkoj aplikaciji nema neregularnosti. To znači da po završetku svih niti pokrenutih u korisničkoj aplikaciji ceo program treba regularno da se završi. Test primeri

ispitivača biće regularni, pa svaki neregularan završetak programa znači neregularnost u samom jezgru.

Osim biblioteka eksplicitno navedenih ovde koje će biti posebno pripremljene, u realizaciji jezgra ne treba koristiti nikakve druge, pa ni standardne C/C++ biblioteke, statičke ili dinamičke, jer one po pravilu sadrže sistemske pozive operativnog sistema-domaćina.

Zadatak: Alokacija memorije u jezgru operativnog sistema

Uvod

Za alokaciju podataka jezgro operativnog sistema koristi alokator pod nazivom sistem ploča (engl. *slab allocator*) koji će biti implementiran u okviru ovog projekta. Na početku rada alokatoru će biti dodeljen kontinualan memorijski prostor koji će alokator kontrolisati. Alokator ima sledeće ciljeve:

- alokacija malih memorijskih bafera sa ciljem da se smanji interna fragmentacija;
- keširanje često korišćenih objekata sa ciljem da se izbegne alokacija, inicijalizacija i uništavanje objekata;

Prvi cilj treba da se postigne održavanjem jednog skupa keševa za alociranje malih memorijskih bafera čija je moguća veličina između 2^5 i 2^{17} bajtova. Ovi keševi se nazivaju *size-N*, gde je *N* veličina potrebnog prostora za jedan bafer. Veličina malog memorijskog bafera (u bajtovima) može biti jednaka samo stepenu dvojke.

Drugi cilj se postiže održavanjem keševa često korišćenih objekata. Kada se nova ploča kreira, određeni broj objekata jezgra se alocira u njemu. Objekti se inicijalizuju ako postoji konstruktor za njih. Kada se objekat oslobodi, ostavlja se u inicijalizovanom stanju da može da bude ponovo upotrebljen.

Alokator treba da obezbedi interfejs za kreiranje, uništavanje i smanjenje keševa, kao i interfejs za alokaciju/dealokaciju objekata i malih memorijskih bafera. Za potrebe testiranja alokator treba da obezbedi i interfejs za štampanje informacija o keševima. Alokator treba automatski da proširuje veličinu keša kada je to potrebno.

Prilikom startovanja sistema alokatoru se dodeljuje prostor koji treba da kontroliše. Taj prostor treba da bude 12,5% početnog prostora za podatke. Ploče treba da budu veličine 2^n kontinualnih blokova. Za vođenje evidencije o slobodnim i zauzetim blokovima koristiti sistem parnjaka (engl. *buddy allocator*). Alokatori smeju da koriste samo dodeljenu memoriju za čuvanje svojih podataka, drugim rečima ne sme da dinamčki alociraju memoriju korišćenjem usluga samog jezika i operativnog sistema domaćina (*malloc*, *new* itd.).

Sistem ploča

Opis keša

Za potrebe alociranja se koriste keševi za objekte svih vrsta i male memorijske bafere. Keševi treba da budu ulančani u listu. Jedan keš sadrži informacije o pločama za jednu vrstu objekata ili za male memorijske bafere. Keš treba da čuva tri liste ploča, jednu za slobodne ploče, jednu za pune ploče i jednu za sve ploče koje imaju mesta a nisu potpuno prazne. Proširenje keša se radi automatski kad su sve ploče popunjene. Smanjivanje jednog keša se radi tako što se prazne ploče uklone, pod uslovom da od prethodnog smanjivanja nije bilo potrebe da se broj ploča poveća. Alokacija jednog malog memorijskog bafera implicitno kreira keš za tu vrstu malih memorijskih bafera.

Informacije koje se štampaju za jedan keš su ime, veličina jednog podatka izražena u bajtovima, veličina celog keša izraženog u broju blokova, broj ploča, broj objekata u jednoj ploči i procentualna popunjenost keša.

Operacije koje za keš treba da se obezbede su:

- inicijalizacija alokatora; prosleđuje se adresa pocetka memorijsko prostora za koji je alokator zadužen i veličina tog memorijskog prostora izražena u broju blokova;
- kreiranje jednog keša; pri kreiranju se zadaje naziv keša, veličina objekta, i opcioni konstruktor i destruktor objekata; povratna vrednost je pokazivač na ručku keša; NULL vrednost može da se prosledi umesto konstruktora i destruktora;
- smanjivanje jednog keša; povratna vrednost je broj blokova koji je oslobođen;
- brisanje jednog keša;
- alokaciju jednog objekta;
- dealokaciju jednog objekta;
- alokacija jednog malog memorijskog bafera;
- dealokacija jednog malog memorijskog bafera;
- štampanje informacije o kešu;
- štampanje greške prilikom rada sa kešom; povratna vrednost funkcije je 0 ukoliko greške nije bilo, u suprotnom vrednost različita od 0.

Sve date operacije se pozivaju isključivo u sistemskom modu rada procesora.

Sve date operacije su *thread-safe*, što znači da se potpuno bezbedno mogu pozivati iz konkurentnih niti. Sinhronizaciju smanjiti na minimum. Implementacija ručke za pristup kešu se ostavlja u nadležnost samog rešenja.

Keš na jeziku C

Interfejs za pristup keševima, zajedno sa potrebnim tipovima i podacima, dat je u fajlu `slab.h`.

```
// File: slab.h
#include <stdlib.h>

typedef struct kmem_cache_s kmem_cache_t;
#define BLOCK_SIZE (4096)

void kmem_init(void *space, int block_num);
kmem_cache_t *kmem_cache_create(const char *name, size_t size,
                                    void (*ctor)(void *),
                                    void (*dtor)(void *); // Allocate cache
int kmem_cache_shrink(kmem_cache_t *cachep); // Shrink cache
void *kmem_cache_alloc(kmem_cache_t *cachep); // Allocate one object from cache
void kmem_cache_free(kmem_cache_t *cachep, void *objp); // Deallocate one object from cache
void *kmalloc(size_t size); // Alloacate one small memory buffer
void kfree(const void *objp); // Deallocate one small memory buffer
void kmem_cache_destroy(kmem_cache_t *cachep); // Deallocate cache
void kmem_cache_info(kmem_cache_t *cachep); // Print cache info
int kmem_cache_error(kmem_cache_t *cachep); // Print error message
```

Sistem parnjaka

Opis zadatka

Sistem parnjaka se koristi samo u priloženom rešenju. Drugim, rečima nije potrebno obezbediti interfejs za korišćenje sistema parnjaka van samog rešenja. Implementacija sistema parnjaka se ostavlja u potpunosti u nadležnost samog rešenja.

Odnos alokatora i ostatka operativnog sistema

Alokator je potrebno koristiti u već postojećim podsistemima operativnog sistema. Za svaki objekat koji pravi kernel potrebno je koristiti keševe objekata. Održavanje keševa objekata (kreiranje, smanjivanje, uništavanje) je potrebno raditi na odgovarajućim mestima u kernelu i to potpada pod nadležnost samog rešenja. Za nizove koje alocira kernel (npr. bafer za prihvatanje podataka sa tastature) potrebno je koristiti alokaciju malih memorijskih bafera. U kernelu nije dozvoljeno da se ništa osim skalarnih podataka (npr. pokazivač, integer, itd.) alocira statički, bez korišćenja implementiranog alokatora.

Testovi

Testiranje može da se radi na dva načina. Prvi je eksplicitno, gde je potrebno prilikom pokretanja funkcije `userMain()` procesor ostaviti u sistemskom modu. Tom prilikom korisnička aplikacija će direktno pozivati funkcije koje se implementiraju. Drugi način je implicitno, gde je potrebno prilikom pokretanja funkcije `userMain()` procesor ostaviti u korisničkom modu. Tom prilikom korisnička aplikacija će pozivati samo sistemske pozive implementirane u okviru predmeta OS1.

Javni testovi

Javni test-program služi da pomogne studentima da istestiraju svoj projekat. Ovi testovi neće obavezno pokriti sve funkcionalnosti koje projekat treba da ima, ali će istestirati 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 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 alokatora), i nisu unapred dostupni studentima.

Testovi performansi

Testovi performansi mere vreme izvršavanja pojedinačnih operacija i proveravaju maksimalne mogućnosti sistema.

Predrok

Za do 10 najboljih radova odbranjenih pre roka, u prvom ispitnom roku posle nastave moguće je osvojiti do 10 dodatnih bodova. Posebno će biti cenjena ona rešenja u kojima se uključi korišćenje virtuelne memorije tako da se onemogući pristup do alociranih podataka u kernelu iz korisničkog moda.

Zaključak

Potrebno je realizovati opisane podsisteme prema datim zahtevima na jeziku C/C++. Testiranje se vrši u laboratorijama katedre na računarima pod operativnim sistemom Windows 10 x64. Virtuelna mašina sa sajta predmeta biće dostupna za vreme odbrane.

Pravila za predaju projekta

Projekat se predaje isključivo kao jedna zip arhiva. U arhivu smestiti samo fajlove sa kodom (.c, .cpp, .h i slične) 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 bilo kakve testove, niti bilo šta što iznad nije opisano, pogotovu ne sme sadržati repozitorijum koda. 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, po pravilu prvi radni dan 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 1.0

Strana	Izmena