PROJEKAT IZ OPERATIVNIH SISTEMA 1 (DOKUMENTACIJA)

2019/20.

Uvod

Ovo je jednostavna, ali funkcionalna implementacija jezgra za 8086 procesor, odrađena u okviru predmeta Operativni Sistemi 1, na Elektrotehničkom fakultetu Univerziteta u Beogradu.

U okviru jezgra realizovan je podsistem za upravljanje nitima, koji obezbeđuje usluge kreiranja i pokretanja niti, koncept semafora i događaja, kao i podršku za deljenje vremena.

U folderima h i src se nalazi sam projekat, dok se u folderu test nalazi javni test. Virtuelna mašina i radno okruženje potrebno za pokretanje projekta nalaze se na sajtu predmeta. U folderu dokumentacija, pored ovog dokumenta, nalazi se kompletna specifikacija projekta za 2019/20. godinu.

Kratak pregled

Klase `Thread`, `Semaphore` i `Event` obezbeđuju sistemske pozive i vidljive su korisniku (javnom testu), dok klase `PCB`, `KernelSem` i `KernelEv` predstavljaju njihove interne implementacije. Većina koda se nalazi u ovim internim klasama, a omotačke klase samo pozivaju njihove funkcije članice. (npr. `Thread` je omotač za `PCB`, `Semaphor` je omotač za `KernelSem` itd.)

Klase `List`, `SemList` u `Queue` su pomoćne strukture podataka. Na primer, `List` se koristi za globalnu listu svih kreiranih `PCB`-ova, a `Queue` se korsiti za skladištenje `PCB`-ova blokiranih na semaforu.

`IdleTh` je zaludna nit koja uposleno čeka i dobija procesor samo ako su sve ostale niti blokirane. Ona se nikako ne stavlja u raspoređivač, već se čuva u globalnom pokazivaču.

Na kraju, postavlja se pitanje kako obezbediti konkurentno izvršavanje niti (što je i osnovni zadatak projekta). To se radi tako što se „ukrade“ prekidna rutina tajmera, koji se okida na svakih 55ms, i izmeni tako da se u njoj izvrši promena konteksta. Svakoj niti biće pridružen timeSlice, tj. Broj kvantova od 55ms tokom kojeg ona sme koristiti procesor, a kada to vreme istekne ona prepušta procesor narednoj niti. Kako iskoristiti prekidnu rutinu tajmera objašnjeno je u sekciji ispod.

Globalni podaci u funckije

- `allPCBs` je lista svih kreiranih PCB-ova
- `running` je pokazivač na tekuću nit
- `allSems` je lista svih semafora. Zašto je potrebna objašnjeno je u sekciji o semaforima
- `idle` je nit koja uposleno čeka ako su sve ostale niti blokirane
- `lockFlag` i funkcije `lock()` i `unlock()` služe za zabranu promene konteksta bez zabrane prekida; koriste se kod kritičnih sekcija na sledeći način:

```
lock();
--- kritična sekcija ---
unlock();
```

- brojač (`counter`) pokazuje koliko još kvantova tekuća nit sme da zauzima procesor. Kada postane 0, dolazi do promene konteksta i u njega se upisuje `timeSlice` niti koja sledeća dobija procesor
- `zahtevana_promena_konteksta` (`context_switch_on_demand`) je fleg koji označava da je korisnik eksplicitno zatražio promenu konteksta; koristi se u funkciji `dispatch()`
- `tsp`, `tss` i `tbp` su pomoćne promenljive koje služe za promenu konteksta u prekidnoj rutini tajmera
- šta se dešava u prekidnoj rutini tajmera?
    - Smanjuje se brojač
    - Ažurira se lista svih semafora (detaljnije u sekciji o semaforima)
    - Ukoliko je dozvoljeno, vrši se promena konteksta:
        - Čuva se kontekst tekuće niti (`ss`, `sp`, `bp`)
        - Uzima se nova nit iz raspoređivača
        - Podmeće se kontekst te nove niti (`ss`, `sp` i `bp`) i u brojač je upisuje `timeSlice` te nove niti
- funckije `init()` I `restore()` služe da postave novu prekidnu rutinu tajmera (na početku programa), odnosno da vrate staru (na kraju programa)

Klase Thread i PCB

Klasa `Thread` je omotačka klasa `PCB`-a. `PCB` je struktura podataka u kojoj se nalazi sve što je potrebno za funckionisanje jedne niti:

- Stek i veličina steka (`stack` i `stackSize`)
- Registri `ss`, `sp` i `bp`. Zašto je potreban registar `bp` (base pointer)? Potreban je zbog alokacije lokalnih promenljivih. Na primer, ako se u nekoj prekidnoj rutini koriste lokalne promenljive koje se alociraju na steku, sp se pomera. Zbog toga se pri ulasku u prekidnu rutinu čuva početni `sp` u `bp` registru, kako bi se kasnije restaurirao
- Flegovi stanja niti `NEW`, `READY`, `BLOCKED` i `FINISHED`

- Uvezana nit (`myThread`) i njen `timeSlice`
- `waitingList` je lista svih niti koje čekaju na završetak ove niti i koje treba probuditi kada se ova nit završi
- fleg `semStatus` biće objašnjen u delu o semaforima

Što se tiče funckija članica `PCB` i `Thread`, najbitnije su:

1. konstruktor – služi za inicijalizaciju članova. Treba voditi računa o tome da veličina steka ne sme biti više od 64k, a zatim ga podeliti sa dva, jer unsigned zauzima 2 bajta. Ono što možda deluje komplikovano jeste inicijalizacija steka. Prvo, on se ne inicijalizuje za nit (tj. `PCB`) glavnog programa, jer se taj deo automatski inicijalizuje pri pokretanju programa (zbog toga postoji provera `myTh != 0`). Zatim:

```
stack[stackSize - 1] = 0x200; //postavlja I bit PSW-a na 1
//punjenje PC regista, pc = code_seg + instruction_pointer_offs
stack[stackSize - 2] = FP_SEG(PCB::wrapper); //postavlja code segment
stack[stackSize - 3] = FP_OFF(PCB::wrapper); //postavlja ip offset
```

2. `wrapper(void)` funkcija – poziva run funkciju uvezane niti. Kada izvršavanje run dođe do kraja bude se sve niti koje su čekale na kraj ove, označava se da je ova nit završena i traži se promena konsteksta
3. destruktor – ovaj `PCB` se briše iz liste svih `PCB`-ova, briše se stek i lista čekanja

Definicija klase `Thread` data je u specifikaciji projekta. Ono o čemu treba voditi računa u implementaciji njenih metoda jeste:

1. konstruktor – napraviti uvezani `PCB`, a kao pokazivač na nit proslediti `this`
2. `start(void)` – proveriti da li je nit već pokrenuta. Ako nije, resetovati `NEW` fleg uvezanog PCB-a i staviti ga u raspoređivač
3. `waitToComplete(void)` – proveriti da li je nit ugašena. Ako nije, blokirati tekuću nit (`running`), dodati je u listu čenja uvezanog `PCB`-a i zatražiti promenu konteksta

Klase Semaphore i KernelSem

Ponovo, Semaphore je omotačka klasa, koja poziva odgovarajuće funkcije članice `KernelSem`. Rad semafora je detaljno objašnjen na predavanjima, a glavne razlike su sledeće:

1. nit se pri pozivu `wait(unsigned int)` može blokirati na neodređeno vreme, ili na neki broj kvantova (1 kvant = 55ms), koji je prosleđen kao parametar.
2. Pomoću `signal(int)` se može deblokirati više od jedne niti

Stoga je potrebno u konstruktoru `KernelSem` objekat semafora dodati u globalnu listu svih semafora (`allSems`) i tu listu ažurirati prilikom svakog ulaska u prekidnu rutinu tajmera.

Naravno, niti blokirane na neograničeno vreme ne treba uzimati u obzir. Postoje i drugi načini implementacije liste čekanja, recimo globalna lista svih uspavanih niti, ali u ovoj implementaciji je korišćena lista semafora.

Još jedna razlika je u tome što `wait` treba da vrati vrednost 1 ili 0, u zavisnosti od toga da li je deblokirana zbog isteka vremena čekanja ili kao rezultat operacije `signal`. Taj podatak se čuva u flegu `semStatus` u `PCB`-u, pa njegovu vrednost treba vratiti kao rezultat funkcije `wait`.

Klase Event i KernelEv

Događaj (`Event`) treba realizovati kao **binarni semafor**, na kome se može blokirati samo jedna nit i to ona koja je napravila taj događaj

1. `wait()` funckija – ako je tekuća nit vlasnik, uraditi sledeće: ako je `val = 0` blokirati nit i postaviti pokazivač `blocked` na vlasnika, u suprotnom samo postaviti `val` na 0
2. `signal()` funkcija – ako je pokazivač na blokiranu nit 0, postaviti `val` na 1; u suprotnom deblokirati nit i postaviti pokazivač `blocked` na 0

U konstruktoru klase `KernelEv` treva uvezati napravljeni događaj sa ulazom u tabelu prekidnih rutina. Taj ulaz je već inicijalizovan (u samom javnom testu) i potrebno je samo pozvati funkciju `setEvent(KernelEv*)` i kao parametar proslediti `this`.

Klasa IVTEntry

`IVTEntry` predstavlja ulaz u tabelu prekidnih rutina, a određuju ga:

1. `numEntry` - broj ulaza
2. `oldRoutine()` – pokazivač na staru prekidnu rutinu vezanu za taj ulaz
3. `myEvent` – događaj uvezan sa ovim ulazom
4. `static IVTEntry* ivTable[]` – tabela prekidnih rutina

U konstruktoru `IVTEntry` radi se ledeće:

1. sačuva se je stara prekidna rutina
2. inicijalizuje se broj ulaza i postaviti `myEvent` na 0
3. doda se ovaj ulaz u tabelu svih prekidnih rutina
4. postavlja se nova prekidna rutina

Prilikom kreiranja objekta takođe treba pozvati funkciju `setEvent(KernelEv*)`, kako bi se sa ovim ulazom uvezao odgovarajući događaj (ovo se radi u konstruktoru `KernelEv`). Na kraju, u destruktoru treba **OBAVEZNO** restaurirati staru prekidnu rutinu, jer ako se to ne uradi može se javiti problem sa emulatorom 8086 procesora.

Makro PREPAREENTRY

Ovaj makro služi da pripremi prekidnu rutinu za tastaturu, stavi je u odgovarajući ulaz u tabeli prekidnih rutina i napravit pripadajući objekat `klase IVTEntry`. Izgleda ovako:

```
#define PREPAREENTRY(numEntry, callOld)\
void interrupt inter##numEntry(...); \
IVTEntry newEntry##numEntry(numEntry, inter##numEntry);\
void interrupt inter##numEntry(...) {\
newEntry##numEntry.signal();\
if (callOld == 1)\
newEntry##numEntry.callOldRoutine();\
}
```

- `\` na kraju svake linije označava da je u pitanju višelinijski makro
- `##` se tekstualno zamenjuju sa prosleđenim parametrima (npr. kada se pozove `PREPARERENTRY(9, 0)` tada `inter##numEntry` postaje `inter9NumEntry`)
- `callOld` parameter naznačava da li je potrebno pozvati staru prekidnu rutinu vezanu za konkretan ulaz ili ne. Bitan je za ispravno funkcionisanje emulatora.