  
Univerzitet u Beogradu  
Elektrotehnički fakultet  
Katedra za računarsku tehniku i informatiku

**Projekat**iz predmeta  
Operativni sistemi 1   
(IR2OS1)

|  |  |  |
| --- | --- | --- |
|  | Septembar 2007. | **Student:** Radoš Lazarević  **Broj indeksa:** 160/05 |

**Napomene:** Datoteke su date po abecednom redu. Svaka datoteka je započeta na novoj strani. Sintaksne reči su istaknute različitim bojama.

**Sadržaj:**

|  |  |
| --- | --- |
| event.h | 3 |
| event.cpp | 4 |
| idlethr.h | 5 |
| idlethr.cpp | 6 |
| kernel.h | 7 |
| kernel.cpp | 8 |
| kevent.h | 12 |
| kevent.cpp | 13 |
| ksemafor.h | 14 |
| ksemafor.cpp | 15 |
| pcb.h | 17 |
| pcb.cpp | 19 |
| queue.h | 22 |
| queue.cpp | 23 |
| semaphor.h | 25 |
| semaphor.cpp | 26 |
| sleeppq.h | 27 |
| sleeppq.cpp | 28 |
| spqlist.h | 31 |
| spqlist.cpp | 32 |
| thread.h | 34 |
| thread.cpp | 35 |

**//event.h**

#ifndef \_\_EVENT\_H\_\_

#define \_\_EVENT\_H\_\_

typedef unsigned int IVTNo;

typedef void interrupt (\*InterruptHandler) (...);

class KernelEv;

class Event {

public:

Event (IVTNo, InterruptHandler);

~Event();

int wait();

void signal();

void oldHandler();

private:

KernelEv\* myImpl;

};

#endif

**//event.cpp**

#include "event.h"

#include "kernel.h"

#include "kevent.h"

Event::Event (IVTNo brojUlaza, InterruptHandler noviHendler) {

lock

myImpl = new KernelEv (brojUlaza, noviHendler);

unlock

}

Event::~Event() {

lock

delete myImpl;

unlock

}

int Event::wait() {

lock

return myImpl->wait();

unlock

}

void Event::signal() {

lock

myImpl->signal();

unlock

}

void Event::oldHandler() {

lock

myImpl->oldHandler();

unlock

}

**//idlethr.h (Idle Thread)**

#ifndef \_\_IDLETHREAD\_H\_\_

#define \_\_IDLETHREAD\_H\_\_

class IdleThread: public Thread{

//onemogucavam ostale metode:

void Sleep(int timeToSleep){}

void Interrupt() {}

public:

IdleThread(char\* ime);

void run();

};

#endif

**// idlethr.cpp**

#include "thread.h"

#include "idlethr.h"

#include "kernel.h"

#include "pcb.h"

IdleThread::IdleThread(char\* ime): Thread(ime, defaultStackSize, minkv) {}

void IdleThread::run(){

while(1){

if (unfTh == 1) break;   
// IdleThread je jedna od nedovrsenih!

dispatch();

}

}

**//kernel.h**

#ifndef \_\_KERNEL\_H\_\_

#define \_\_KERNEL\_H\_\_

// Zabranjuje prekide

#define lock asm cli

// Dozvoljava prekide

#define unlock asm sti

#define bn 5

#define kv 100

#define broj 30000

#define minkv 5

#define MAX 64512

#include "event.h"

//extern int semPreempt;

extern int unfTh;

int userMain(int argc, char\* argv[]);

int main(int argc, char\*\* argv);

void dispatch();

InterruptHandler swapRoutine(IVTNo entryNum, InterruptHandler addr);

#endif

**//kernel.cpp**

#include <iostream.h>

#include <dos.h>

#include "pcb.h"

#include "thread.h"

#include "schedule.h"

#include "kernel.h"

#include "sleeppq.h"

#include "user.h"

#include "event.h"

#include "idlethr.h"

#include "spqlist.h"

#define reci(A) { cout << A ;}

// Globalne promenljive:

InterruptHandler otp;

volatile int exp = 0;

int unfTh = 0;

//int semPreempt = 0;

// Globalne funkcije i strukture:

InterruptHandler swapRoutine(IVTNo entryNum, InterruptHandler addr){

unsigned oldSEG, oldOFF, newSEG, newOFF;

newSEG = FP\_SEG((\*addr));

newOFF = FP\_OFF((\*addr));

IVTNo adr = entryNum \* 4;

asm{

cli

push es

push ax

push bx

mov bx,0 // inicijalizuje rutinu

mov es,bx

mov bx, adr

push word ptr es:bx // pamti staru rutinu

pop word ptr oldOFF

add bx, 2

push word ptr es:bx

pop word ptr oldSEG

mov ax, newSEG

mov word ptr es:bx, ax

mov ax, newOFF

sub bx, 2

mov word ptr es:bx, ax

pop bx

pop ax

pop es

sti

}

return (InterruptHandler) MK\_FP(oldSEG, oldOFF);

}

void dispatch(){

lock

exp = 1;

asm int 08h

unlock

}

// Prekidna rutina za timer i njene promenljive:

static volatile unsigned tsp=0;

static volatile unsigned tss=0;

static volatile unsigned tbp=0;

static volatile unsigned tax=0;

static volatile unsigned tbx=0;

static volatile unsigned tcx=0;

static volatile unsigned tdx=0;

static volatile unsigned tes=0;

static volatile unsigned tds=0;

static volatile unsigned tsi=0;

static volatile unsigned tdi=0;

static volatile unsigned tcs=0;

static volatile unsigned tip=0;

static volatile unsigned tPSW=0;

static volatile int brojac = 20;

void interrupt timer(...){

(\*otp)();

asm cli

//tick();

if(!exp) SpqList::knock();

if (--brojac == 0 || exp ==1){

asm {

pop word ptr tbp

pop word ptr tdi

pop word ptr tsi

pop word ptr tds

pop word ptr tes

pop word ptr tdx

pop word ptr tcx

pop word ptr tbx

pop word ptr tax

pop word ptr tip

pop word ptr tcs

pop word ptr tPSW

mov word ptr tsp, sp

mov word ptr tss, ss

}

PCB::running->sp = tsp;

PCB::running->ss = tss;

PCB::running->bp = tbp;

PCB::running->ax = tax;

PCB::running->bx = tbx;

PCB::running->cx = tcx;

PCB::running->dx = tdx;

PCB::running->es = tes;

PCB::running->ds = tds;

PCB::running->si = tsi;

PCB::running->di = tdi;

PCB::running->cs = tcs;

PCB::running->ip = tip;

PCB::running->PSW = tPSW;

if (((PCB\*)PCB::running)->status == RUNNING) {

((PCB\*)PCB::running)->status = READY;

Scheduler::put((PCB\*)PCB::running);

}

PCB::running = Scheduler::get();

((PCB\*)PCB::running)->status = RUNNING;

tsp = PCB::running->sp;

tss = PCB::running->ss;

tbp = PCB::running->bp;

tax = PCB::running->ax;

tbx = PCB::running->bx;

tcx = PCB::running->cx;

tdx = PCB::running->dx;

tes = PCB::running->es;

tds = PCB::running->ds;

tsi = PCB::running->si;

tdi = PCB::running->di;

tcs = PCB::running->cs;

tip = PCB::running->ip;

tPSW = PCB::running->PSW;

brojac = PCB::running->kvant;

exp = 0;

asm {

mov sp, word ptr tsp

mov ss, word ptr tss

push word ptr tPSW

push word ptr tcs

push word ptr tip

push word ptr tax

push word ptr tbx

push word ptr tcx

push word ptr tdx

push word ptr tes

push word ptr tds

push word ptr tsi

push word ptr tdi

push word ptr tbp

}

}

asm sti

}

int main(int argc, char\*\* argv){

lock

int rez = 0;

otp = swapRoutine(0x08, timer);

PCB::createMainPCB("gla!", minkv);

IdleThread\* it = new IdleThread("idle!");

it->start();

unlock

rez = userMain(argc, argv);

swapRoutine(0x08, otp);

delete it;

PCB::destroyMainPCB();

reci(endl << "Main kaze: Kraj!");

return rez;  
}

**//kevent.h (Kernel Event)**

#ifndef \_\_KEVENT\_H\_\_

#define \_\_KEVENT\_H\_\_

#include "event.h"

#include "ksemafor.h"

class KernelEv {

IVTNo brUlaza;

KernelSem\* event;

InterruptHandler ohp;

public:

KernelEv (IVTNo, InterruptHandler);

~KernelEv ();

void oldHandler() volatile { (\*ohp)(); }

int wait() { return event->wait(); }

void signal() volatile { event->signalAll(); }

};

#endif

**// kevent.cpp**

#include "kevent.h"

#include "kernel.h"

#include "dos.h"

KernelEv::KernelEv (IVTNo brUl, InterruptHandler nhp){

brUlaza = brUl;

event = new KernelSem (0);

ohp = swapRoutine(brUlaza, nhp);

}

KernelEv::~KernelEv () {

swapRoutine(brUlaza, ohp);

delete event;

}

**// ksemafor.h (Kernel Semaphore)**

#ifndef \_\_KSEMAFOR\_H\_\_

#define \_\_KSEMAFOR\_H\_\_

#include "semaphor.h"

#include "queue.h"

#include "kernel.h"

#include "thread.h"

#include "schedule.h"

class KernelSem {

KernelSem() {}   
// Zabrana pravljenja objekta ove klase bez inicijalnih vrednosti.

int val;

volatile Queue\* q;   
// Red niti koje su blokirane na ovom semaforu.

friend class Thread;

public:

KernelSem (int init);

~KernelSem ();

virtual int wait ();

virtual void signal () volatile ;

void signalAll (); // Aktivira sve niti koje cekaju!

const int value () const { return val; }

};

#endif

**// ksemafor.cpp**

#include "ksemafor.h"

#include "kernel.h"

#include "pcb.h"

KernelSem::KernelSem (int init): val(init){

q = new Queue();

}

KernelSem::~KernelSem (){

lock

delete q;

unlock

}

int KernelSem::wait (){

int temp = 0;

lock

if(--val<0) {

((Queue\*)q)->put((PCB\*)PCB::running);   
// Stavljam runninng PCB na red za cekanje ovog semafora.

PCB::running->status = BLOCKED; // Blokiram doticni PCB.

PCB::running->blockedAtSem = this; // Saopstava PCB-u da je blokiran na ovom semaforu.

dispatch();

}

else if (semPreempt) dispatch();

// Ako je zadnji put blokirana na ovom semaforu, to ce wait i saopstiti

if (PCB::running->blockedAtSem == this)   
{temp = 1;}

unlock

return temp;

}

void KernelSem::signal() volatile {

lock

if (val<0) {

volatile PCB\* temp = ((Queue\*)q)->get();

if(temp) { //q vraca NULL ako je prazan

((PCB\*)temp)->status = READY;

Scheduler::put((PCB\*)temp);

}

}

val++;

if (semPreempt) dispatch();

unlock

}

void KernelSem::signalAll() {

lock

volatile int duzina = ((Queue\*)q)->size();   
// Uzima se broj blokiranih niti i onda se vrsi budjenje niti kroz petlju.

for (int i=0; i<duzina; i++ ) signal();

unlock

}

**// pcb.h**

#ifndef \_\_PCB\_H\_\_

#define \_\_PCB\_H\_\_

#include <stdlib.h>

#include "thread.h"

#include "queue.h"

#include "schedule.h"

#include "ksemafor.h"

#include "kernel.h"

class SleepPQ;

struct elemIdNameList {

ID id;

char\* ime;

Thread\* thr;

elemIdNameList\* next;

elemIdNameList(ID idd, char\* imee, Thread\* threadd, elemIdNameList\* pok=NULL);

};

void dispatch();

enum ThreadStatus {NEW, RUNNING, READY, SLEEPING, BLOCKED, OVER};

class PCB{

volatile unsigned sp;

volatile unsigned ss;

volatile unsigned bp;

volatile unsigned ax;

volatile unsigned bx;

volatile unsigned cx;

volatile unsigned dx;

volatile unsigned es;

volatile unsigned ds;

volatile unsigned si;

volatile unsigned di;

volatile unsigned cs;

volatile unsigned ip;

volatile unsigned PSW;

volatile enum ThreadStatus status;

char\* ime;

ID id;

Time kvant;

Thread\* myThread;

static void starter();

static volatile PCB\* running;

static PCB\* mainPCB;

KernelSem\* finished;

// staticke metode koje rade sa ulancanom listom id-eva i imena:

static ID setID(char\* imee, Thread\* thr);

static Thread\* getThread(ID idd);

static ID getIdOf(char\* imee);

static elemIdNameList \*prvi, \*zadnji;

static int duzinaListe;

static int ukID;

static SleepPQ\* spq;

KernelSem\* blockedAtSem;

friend void interrupt timer(...);

friend int userMain(int argc, char\* argv[]);  
 friend class Thread;

friend class IdleThread;

friend class SleepPQ;

friend class KernelSem;

public:

PCB(char\* name, Time timeSlice);

PCB(char\* name, StackSize stackSize, Time timeSlice, Thread\* myTh);

~PCB();

static void createMainPCB(char\* name, Time timeSlice);

static void destroyMainPCB() { delete mainPCB; }

char\* getIME() {return ime;}

ID getID() {return id;}

};

#endif

**// pcb.cpp**

#include <iostream.h>

#include <dos.h>

#include <string.h>

#include "pcb.h"

#include "sleeppq.h"

#include "kernel.h"

SleepPQ\* PCB::spq = new SleepPQ();

// --- Dodaci za rad za ID i imenima PCB-ova:

elemIdNameList::elemIdNameList(ID idd, char\* imee, Thread\* threadd, elemIdNameList\* pok){

id = idd;

ime = imee;

thr = threadd;

next = pok;

}

elemIdNameList\* PCB::prvi = NULL;

elemIdNameList\* PCB::zadnji = NULL;

int PCB::duzinaListe = 0;

int PCB::ukID = 0;

// ---

volatile PCB\* PCB::running;

PCB\* PCB::mainPCB;

void PCB::starter(){

running->myThread->run();

lock

((PCB\*)running)->status = OVER;

unfTh--;

((PCB\*)running)->finished->signalAll();

unlock

dispatch();

}

void PCB::createMainPCB(char\* name, Time timeSlice){

mainPCB = new PCB(name, timeSlice);

(PCB\*)running = mainPCB;

}

void PCB::PCB(char\* name, Time timeSlice){

if (timeSlice < minkv && timeSlice != 0 ) timeSlice = minkv;

if (timeSlice == 0) timeSlice = -1;

ime = name;

id = setID(name, NULL);

kvant = timeSlice;

status = RUNNING;

}

PCB::PCB(char\* name, StackSize stackSize, Time timeSlice, Thread\* myTh): sp(0), ss(0), bp(0), ax(0), bx(0), cx(0), dx(0), si(0), di(0), cs(0), ip(0)

{

if (stackSize > 32768) stackSize = 32768;

if (timeSlice < minkv && timeSlice != 0 ) timeSlice = minkv;

if (timeSlice == 0) timeSlice = -1;

myThread = myTh;

ime = name;

id = setID(name, myTh);

kvant = timeSlice;

status = NEW;

finished = new KernelSem(0);

blockedAtSem = NULL;

unsigned char \*tempStack = new unsigned char[stackSize];

sp = FP\_OFF(tempStack+stackSize);

ss = FP\_SEG(tempStack+stackSize);

volatile unsigned newPSW, newDS, newES;

ip = FP\_OFF(starter);

cs = FP\_SEG(starter);

asm{

push ax

pushf

pop ax

or ax, 0x00000200

push ax

pop word ptr newPSW

push ds

pop word ptr newDS

push es

pop word ptr newES

pop ax

}

es = newES;

ds = newDS;

PSW = newPSW;

}

PCB::~PCB(){

lock

myThread = NULL;

unlock

}

ID PCB::setID(char\* imee, Thread\* thr){

ID temp = ukID++;

elemIdNameList\* telem = new elemIdNameList(temp, imee, thr, NULL);

if(prvi==NULL)

prvi = telem;

else

zadnji->next = telem;

zadnji = telem;

duzinaListe++;

return temp;

}

Thread\* PCB::getThread(ID idd){

elemIdNameList\* t;

t = prvi;

while(t != NULL){

if (t->id == idd) break;

t = t->next;

}

if (t == NULL) return NULL;

else return t->thr;

}

ID PCB::getIdOf(char\* imee){

elemIdNameList\* t;

t = prvi;

while(t != NULL){

if (!strcmp(t->ime, imee)) break;

t = t->next;

}

if (t == NULL) return -1;

else return t->id;

}

**// queue.h**

#ifndef \_\_QUEUE\_H\_\_

#define \_\_QUEUE\_H\_\_

#include <stdlib.h>

class PCB;

struct Elem {

PCB\* pcb;

Elem\* next;

Elem(PCB\* p, Elem\* n=NULL);

};

class Queue {

int duz;

Elem \*first, \*last;

friend class PCB;

public:

Queue();

~Queue();

int size() const;

void put(PCB\* pcb);

PCB\* get();

int remove(PCB \*);

};

#endif

**// queue.cpp**

#include <iostream.h>

#include <dos.h>

#include "queue.h"

#include "pcb.h"

#include "kernel.h"

Elem::Elem(PCB\* p, Elem\* n){

pcb = p;

next = n;

}

Queue::Queue(){

first = last = NULL;

duz = 0;

}

Queue::~Queue(){

Elem\* temp;

while(first!=NULL){

temp = first;

first = first->next;

delete temp;

}

}

int Queue::size() const{

return duz;

}

void Queue::put(PCB\* pcb){

lock

Elem \*temp = new Elem(pcb);

if(first==NULL) first = temp;

else last->next = temp;

last = temp;

duz++;

unlock

}

PCB\* Queue::get(){

lock

if(first == NULL){

unlock

return NULL;

}

duz--;

Elem\* temp = first;

first = first->next;

if(first==NULL) last = NULL;

PCB\* pcb = temp->pcb;

delete temp;

unlock

return pcb;

}

int Queue::remove(PCB \*pcb){

lock

Elem \*temp = first, \*prev = NULL;

while ((temp!=NULL)&&(temp->pcb!=pcb)){

prev = temp;

temp = temp->next;

}

if(temp!=NULL) {

if(prev==NULL) first = first->next;

else prev->next = temp->next;

if(temp==last) last = prev;

duz--;

delete temp;

unlock

return 1;

}

unlock

return 0;

}

**// semaphor.h**

#ifndef \_SEMAPHOR\_H\_

#define \_SEMAPHOR\_H\_

class KernelSem;

extern int semPreempt;

class Semaphore {

public:

Semaphore (int init=1);

~Semaphore ();

virtual int wait ();

virtual void signal();

int val () const;

private:

KernelSem\* myImpl;

};

#endif

**// semaphor.cpp**

#include "semaphor.h"

#include "kernel.h"

#include "ksemafor.h"

Semaphore::Semaphore (int init){

lock

myImpl = new KernelSem(init);

unlock

}

Semaphore::~Semaphore (){

lock

delete myImpl;

myImpl = 0;

unlock

}

int Semaphore::wait () {

lock

return myImpl->wait();

unlock

}

void Semaphore::signal() {

lock

myImpl->signal();

unlock

}

int Semaphore::val () const {

lock

return myImpl->value();

unlock

}

**// sleepq.h (Sleeping Priority Queue)**

#ifndef \_SLEEPPQ\_H\_

#define \_SLEEPPQ\_H\_

#include "pcb.h"

//#include <stdlib.h>

#include "thread.h"

#include "kernel.h"

class SleepPQ {

struct ElemPQ {

PCB\* pcb;

ElemPQ\* next;

int rang;

ElemPQ(PCB\* novi, int rangg, ElemPQ\* sled=NULL): pcb(novi), rang(rangg), next(sled) {}

~ElemPQ() { pcb=NULL; next=NULL; }

};

Semaphore\* mutex;

ElemPQ \*first, \*last;

int duz;

int qSleepingFor;

void put (PCB\* pcb, Time rang);

PCB\* get();

void wakeOneUp ();

volatile const int size() {return duz;}

public:

SleepPQ();

~SleepPQ();

int minRang () volatile {

if (first && duz>0) return first->rang;

else return 0;

}

void putToSleep(PCB\* pcb, int timeToSleep);

void incTime ();

int wakeUp(PCB\* thisOne);

};

#endif

**// sleeppq.cpp**

#include "sleeppq.h"

#include "spqlist.h"

void SleepPQ::put (PCB\* pcb, Time rang) {

if (pcb!=NULL && rang>0) {

ElemPQ\* temp = new ElemPQ(pcb, rang);

mutex->wait();

if (first == NULL) first = last = temp;

else {

ElemPQ\* p1 = first;

ElemPQ\* p0 = NULL;

while(p1 != NULL && rang > p1->rang) {

p0 = p1;

p1 = p1->next;

}

p0->next=temp;

temp->next=p1;

if (p1 == NULL) last = temp;

}

duz++;

mutex->signal();

}

}

PCB\* SleepPQ::get() {

PCB\* temp = NULL; // Vraca NULL ako je red prazan!

mutex->wait();

while (first!=NULL && temp == NULL) {

ElemPQ\* p1 = first;

first = p1->next;

if (p1 == last) last = NULL;

temp = p1->pcb;

duz--;

}

mutex->signal();

return temp;

}

int SleepPQ::wakeUp(PCB\* thisOne) {   
// Ova metoda se poziva prilikom nasilnog budjenja - vadim trazeni PCB iz liste

int rez = 0;

ElemPQ\* p1 = first;

ElemPQ\* p0 = NULL;

while(p1 != NULL) {

if (p1->pcb == thisOne) {

if(!p0) {

if (p1 == last) last = NULL;

first = first->next;

p1->next = NULL;

p1 = NULL;

rez = 1;

duz--;

if(!duz) SpqList::knockRemove(this);

}

else {

if (p1 == last) last = p0;

p0->next = p1->next;

p1->next = NULL;

p1 = NULL;

rez = 1;

duz--;

if(!duz) SpqList::knockRemove(this);

}

}

else {

p0 = p1;

p1 = p1->next;

}

}

return rez;

}

void SleepPQ::wakeOneUp () {

PCB\* temp = this->get();

if (temp) {

temp->status = READY;

Scheduler::put(temp);

}

if (!size()) SpqList::knockRemove(this);

}

void SleepPQ::putToSleep(PCB\* pcb, int timeToSleep){

if (!pcb || timeToSleep <=0) return;

lock

if (!size()) {qSleepingFor = 0; SpqList::knockAdd(this);}

Time wakeUpIn = (qSleepingFor + timeToSleep) % MAX;

put(pcb, wakeUpIn);

pcb->status = SLEEPING;

dispatch();

unlock

}

void SleepPQ::incTime () { // Protok vremena...

qSleepingFor = (++qSleepingFor) % MAX;

while (size() && qSleepingFor >= minRang()) wakeOneUp();

}

SleepPQ::SleepPQ() {

mutex = new Semaphore(1);

first = last = NULL;

duz = 0;

qSleepingFor = 0;

}

SleepPQ::~SleepPQ(){

first = last = NULL;

duz = 0;

delete mutex;

}

**// spqlist.h (Sleeping Priority Queue List)**

#ifndef \_\_SPQLIST\_H\_\_

#define \_\_SPQLIST\_H\_\_

#include "sleeppq.h"

class SpqList {

struct ElemSPQ {

SleepPQ\* spqp;

ElemSPQ\* next;

ElemSPQ(SleepPQ\* p, ElemSPQ\* n=NULL){ spqp = p; next = n; }

};

static ElemSPQ\* first, \*last;

static int ElemSPQduz;

public:

static void knockAdd(SleepPQ \*newOne);

static void knockRemove(SleepPQ \*thisOne);

static void knock();

};

#endif

**// spqlist.cpp**

#include "spqlist.h"

ElemSPQ\* SpqList::first=NULL;

ElemSPQ\* SpqList::last=NULL;

int SpqList::ElemSPQduz = 0;

void SpqList::knockAdd(SleepPQ \*newOne){

ElemSPQ\* temp = new ElemSPQ(newOne, NULL);

if (first == NULL) first = last = temp;

else last->next = temp;

last = temp;

ElemSPQduz++;

}

void SpqList::knockRemove(SleepPQ \*thisOne){

ElemSPQ\* p1 = first;

ElemSPQ\* p0 = NULL;

while(p1 != NULL) {

if (p1->spqp == thisOne) {

if(!p0) {   
// ako ne postoji p0, p1 sigurno pokazuje na first!

if (p1 == last) last = NULL;   
// ako je to ujedno i last, last = NULL, jer ce ionako da se isprazni lista

first = first->next;   
// first->next (ako je bio i jedini, sad je first = NULL)

p1->spqp = NULL; // brisem ga

p1->next = NULL;

p1 = NULL;

}

else {

if (p1 == last) last = p0;   
// ako postoji p0 a p1 = last, onda smo stigli do kraja, pa last mora da se vrati za jedno

p0->next = p1->next; // opkoracujem ga

p1->spqp = NULL; // brisem ga

p1->next = NULL;

p1 = NULL;

}

}

else {

p0 = p1;

p1 = p1->next;

} // kraj glavnog if-else-a

} // kraj while-a

} // kraj knockRemove

void SpqList::knock(){

ElemSPQ\* p1 = first;

while(p1 != NULL){

p1->spqp->incTime();

p1 = p1->next;

}

}

**// thread.h**

#ifndef \_\_THREAD\_H\_\_

#define \_\_THREAD\_H\_\_

typedef unsigned Time;

typedef unsigned long StackSize;

typedef int ID;

const int defaultStackSize = 4096;

const Time defaultTimeSlice = 100;

class PCB;

class Thread{

public:

void start();

void waitToComplete();

void Sleep(int timeToSleep);

void Interrupt();

ID getId();

static Thread\* getThreadById(ID id);

static ID getIdOf(char\* name);

char\* getName();

virtual ~Thread();

protected:

friend class PCB;

Thread(char\* name, StackSize stackSize=defaultStackSize, Time timeSlice=defaultTimeSlice);

virtual void run() {}

private:

PCB\* myPCB;

friend void userMain();

};

#endif

**// thread.cpp**

#include <iostream.h>

#include "thread.h"

#include "pcb.h"

#include "scheduler.h"

#include "kernel.h"

#include "sleeppq.h"

Thread::Thread(char\* name, StackSize stackSize, Time timeSlice){

lock

this->myPCB = new PCB(name, stackSize, timeSlice, this);

unfTh++;

unlock

}

Thread::~Thread(){

lock

waitToComplete();

delete myPCB;

unlock

}

void Thread::start(){

lock

myPCB->status = READY;

Scheduler::put(myPCB);

unlock

}

void Thread::waitToComplete(){

lock

if (myPCB->status == OVER) { // "Zavrsena sam - ne cekaj me"

unlock

return;

}

if (myPCB == (PCB\*)PCB::running) { // Ne mogu sama sebe da cekam

unlock

return;

}

myPCB->finished->wait();

// Stavljam running nit da ceka na mom semaforu dok se ne zavrsim

unlock

}

ID Thread::getId(){ return myPCB->getID(); }

Thread\* Thread::getThreadById(ID id){

lock

return PCB::getThread(id);

unlock

}

ID Thread::getIdOf(char\* name){

lock

return PCB::getIdOf(name);

unlock

}

char\* Thread::getName(){ return myPCB->getIME(); }

void Thread::Sleep(int timeToSleep){

// Samo doda u Sleeping Priority Queue

lock

(PCB::spq)->putToSleep(myPCB, timeToSleep);

unlock

}

void Thread::Interrupt() {

if (myPCB) {

lock

// Nasilno budjenje ako je blokirana:

if (myPCB->status==BLOCKED) {

if (myPCB->blockedAtSem)   
{ myPCB->blockedAtSem->val++; }

myPCB->status = READY;

Scheduler::put (myPCB);

}

// Nasilno budjenje ako spava:

if(myPCB->status == SLEEPING){

if(PCB::spq->wakeUp(myPCB)){

myPCB->status = READY;

Scheduler::put (myPCB);

}

}

unlock

}

}