**Elektrotehnički fakultet u Beogradu**

**Katedra za računarsku tehniku i informatiku**

**PROJEKAT IZ PREDMETA**

**OPERATIVNI SISTEMI 1**

## Jun 2008 Vidosavljević Nikola 279/06

### **SADRZAJ**

### bounded.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

### bounded.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

### event.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

### event.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

### ithread.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

### ithread.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

### kernel.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

### kernel.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

### kernelev.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

### kernelev.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

### kersem.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

### kersem.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

### keyevent.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

### keyevent.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

### krobject.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

### lock.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

### main.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

### pcb.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

### pcb.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

### queue.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

### queue.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

### schedule.h. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

### semaphor.cpp. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

### semaphor.h. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

### thread.cpp. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

### thread.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

### user.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

### user.cpp. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: bounded.cpp

// Date: June 2007

//===========================================================================//

#include <stdio.h>

#include "bounded.h"

#include "lock.h"

BoundedBuffer::BoundedBuffer () :

mutex(1), spaceAvailable(BUFFER\_SIZE), itemAvailable(0),

head(0), tail(0) {}

void BoundedBuffer::append (char d) {

spaceAvailable.wait();

mutex.wait();

buffer[tail] = d;

tail = (tail+1)%BUFFER\_SIZE;

mutex.signal();

itemAvailable.signal();

}

char BoundedBuffer::take () {

while (itemAvailable.wait() == 0);

mutex.wait();

char d = buffer[head];

head = (head+1)%BUFFER\_SIZE;

mutex.signal();

spaceAvailable.signal();

return d;

### }

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: bounded.h

// Date: June 2007

//===========================================================================//

#ifndef \_BOUNDED\_H\_

#define \_BOUNDED\_H\_

#include "semaphor.h"

const int BUFFER\_SIZE = 70;

class BoundedBuffer {

public:

BoundedBuffer ();

void append (char);

char take ();

int size(){return itemAvailable.val();}; // potrebno consumeru

private:

Semaphore mutex;

Semaphore spaceAvailable, itemAvailable;

char buffer[BUFFER\_SIZE];

int head, tail;

};

#endif // \_OS1\_BOUNDED\_BUFFER\_

//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 "kernelev.h"

#include "lock.h"

#include <stdlib.h>

Event::Event(IVTNo n, InterruptHandler h){

\_lock;

myImpl = new KernelEv(n, h);

\_unlock;

}

Event::~Event(){

\_lock;

delete myImpl;

\_unlock;

}

int Event::wait(){

\_lock;

int rez = myImpl->wait();

\_unlock;

return rez;

}

void Event::signal(){

\_lock;

myImpl->signal();

\_unlock;

}

void Event::oldHandler(){

myImpl->oldHandler();

}

//ithread.h

#ifndef \_ITHREAD\_H\_

#define \_ITHREAD\_H\_

#include "thread.h"

class Thread;

class IdleThread : public Thread {

public:

IdleThread();

virtual void run();

void start();

};

#endif

//ithread.cpp

#include "ithread.h"

#include "pcb.h"

#include "thread.h"

#include "lock.h"

IdleThread::IdleThread() : Thread("IdleThread", 256, minTimeSlice()){}

void IdleThread::run(){

while(1);

}

void IdleThread::start(){

\_lock

myPCB->state = READY;

myPCB->createStack();

\_unlock

}

//kernel.cpp

#include "lock.h"

#include "kernel.h"

#include "schedule.h"

#include "queue.h"

#include <stdlib.h>

#include <dos.h>

void interrupt (\*Kernel::oldRoutine)() = NULL;

PCB\* Kernel::running = NULL;

Thread\* Kernel::starting = NULL;

IdleThread \* Kernel::idle = NULL;

Action Kernel::action = nothing;

void Kernel::initialization() {

\_lock;

oldRoutine = initializeInterrupt(0X08, timerIR);

//pocetnu nit koja ce prva biti pokrenta

starting = new Thread("Starting", 0x10000, minTimeSlice());

starting->myPCB->state = READY;

running = starting->myPCB;

//nit koja ne radi nista, dodeljuje se prcesoru kada je scheduler prazan

//tj. nema spremnih niti

idle = new IdleThread();

idle->start();

\_unlock;

}

void Kernel::termination(){

//da li postoji neka nit koja nije zavrsena, osim pocetne

if ((PCB\*) running != starting->myPCB) return;

\_lock;

Queue \*q = PCB::indetificationQueue;

delete q;

q = PCB::sleepQueue;

delete q;

initializeInterrupt(0X08, oldRoutine);

delete starting;

\_unlock

}

void Kernel::dispatch(){

\_lock;

action = explicitDispatch;

timerIR();

action = nothing;

\_unlock;

}

void interrupt Kernel::timerIR(){

static volatile unsigned int tempSP, tempSS;

static PCB \*newThread;

//ukoliko nije eksplicitni dispatch pozivamo staru prekidnu rutinu

if (action != explicitDispatch){

tick();

(\*oldRoutine)();

}

\_lock;

//Ako postoje niti koje su uspavane azuriramo njihova vremena

//jos koliko dugo treba da budu suspendovane

if ((action != explicitDispatch) && !(PCB::sleepQueue->empty()))

PCB::sleepQueue->update();

//ukoliko nije eksplicitni dispatch a vreme niti nije beskonacno azuriramo brojac

if(action != explicitDispatch && running->pcbTimeSlice != 0)

running->timePassed += (LongTime) minTimeSlice();

//ukoliko nije eksplicitni dispatch, a vreme niti nije isteklo vratimo se nazad

if((action != explicitDispatch) && (running->timePassed < running->pcbTimeSlice || running->pcbTimeSlice == 0))

return;

if (action == explicitDispatch)

action = nothing;

//ukoliko je nit spremna a nije idle ili starting vracamo je u red spremnih

if ((running->state == READY) && (running != idle->myPCB) && (running != starting->myPCB))

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

while (1){

newThread = Scheduler::get();

//Ako je prazan radi idle

if(newThread == NULL){

if(!PCB::indetificationQueue->empty()){

newThread = idle->myPCB;

}else{

newThread = starting->myPCB;

}

}

if (newThread->state != READY)

continue;

//promena konteksta

asm {

mov tempSP, sp

mov tempSS, ss

}

running->sp = tempSP;

running->ss = tempSS;

running = newThread;

tempSP = running->sp;

tempSS = running->ss;

asm {

mov sp, tempSP

mov ss, tempSS

}

//postavljamo njeno proteklo vreme na nulu

running->timePassed = 0;

break;

}

\_unlock;

return;

}

void Kernel::runWrapper(){

running->myThread->run();

\_lock;

PCB::indetificationQueue->deleteFromQ(running);

running->state = OVER;

PCB\* temp;

while(!running->waitQueue->empty()){

temp = running->waitQueue->get();

temp->state = READY;

}

dispatch();

}

InterruptRoutine Kernel::initializeInterrupt(IVTNo n, InterruptRoutine newRoutine){

\_lock;

unsigned int tempseg = FP\_SEG(newRoutine);

unsigned int tempoff = FP\_OFF(newRoutine);

unsigned int oldseg, oldoff;

InterruptRoutine old;

n \*= 4;

asm {

push es

push ax

push bx

mov ax, 0

mov es, ax

mov bx, word ptr n

mov ax, word ptr es:bx+2

mov word ptr oldseg, ax

mov ax, word ptr es:bx

mov word ptr oldoff, ax

mov ax, word ptr tempseg

mov word ptr es:bx+2, ax

mov ax, word ptr tempoff

mov word ptr es:bx, ax

pop bx

pop ax

pop es

}

old = (InterruptRoutine) MK\_FP(oldseg, oldoff);

\_unlock;

return old;

}

//kernel.h

typedef enum {nothing, explicitDispatch} Action;

#ifndef \_KERNEL\_H\_

#define \_KERNEL\_H\_

#include "pcb.h"

#include "thread.h"

#include "ithread.h"

typedef void (interrupt \*InterruptRoutine)();

typedef unsigned int IVTNo;

class Kernel {

public:

static void initialization();

static void termination();

static void dispatch();

private:

friend class Thread;

friend class PCB;

friend class KernelSem;

friend class KernelEv;

static Thread \*starting;

static IdleThread \*idle;

static PCB\* running;

static Action action;

static void interrupt timerIR();

static void interrupt (\*oldRoutine)();

static void runWrapper();

static InterruptRoutine initializeInterrupt(IVTNo n, InterruptRoutine newRoutine);

};

#endif

//kernelev.cpp

#include "kernelev.h"

#include "event.h"

#include "kernel.h"

#include "queue.h"

KernelEv::KernelEv(IVTNo n, InterruptHandler h){

ivtEntry = n;

oldHandle =(InterruptHandler) Kernel::initializeInterrupt(n, (InterruptRoutine)h);

}

KernelEv::~KernelEv(){

Kernel::initializeInterrupt(ivtEntry, (InterruptRoutine)oldHandle);

}

int KernelEv::wait() {

unsigned int rez = !((Kernel::running->BlockingFlag == 1) && (Kernel::running->waiting == this));

Kernel::running->state = BLOCKED;

Kernel::running->BlockingFlag = 0;

Kernel::running->waiting = this;

waitQueue.put((PCB\*)Kernel::running);

Kernel::dispatch();

return rez;

}

void KernelEv::signal() {

PCB\* temp;

while (!waitQueue.empty()) {

temp = waitQueue.get();

temp->state = READY;

Scheduler::put(temp);

}

Kernel::dispatch();

}

void KernelEv::oldHandler() {

(\*oldHandle)();

}

int KernelEv::iCall(PCB \*pcb){

return waitQueue.deleteFromQ(pcb);

}

//kernelev.h

#ifndef \_KERNELEV\_H\_

#define \_KERNELEV\_H\_

#include "queue.h"

#include "event.h"

#include "schedule.h"

#include "krobject.h"

class KernelEv : public KernelObject {

public:

int wait();

void signal();

void oldHandler();

KernelEv(IVTNo, InterruptHandler);

~KernelEv();

virtual int iCall(PCB \*pcb);

private:

IVTNo ivtEntry;

InterruptHandler oldHandle;

Queue waitQueue;

};

#endif

//kersem.cpp

#include "kersem.h"

#include "kernel.h"

#include "schedule.h"

#include "semaphor.h"

KernelSem::KernelSem(int init){

value = init;

}

int KernelSem::wait() {

value--;

int rez = !((Kernel::running->BlockingFlag == 1) && (Kernel::running->waiting == this));

if (value < 0){

Kernel::running->state = BLOCKED;

Kernel::running->BlockingFlag = 0;

Kernel::running->waiting = this;

waitQueue.put((PCB\*) Kernel::running);

Kernel::dispatch();

} else if (semPreempt)Kernel::dispatch();

return rez;

}

void KernelSem::signal() {

value++;

if (value <= 0) {

PCB\* temp = waitQueue.get();

temp->state = READY;

Scheduler::put(temp);

}

if (semPreempt) Kernel::dispatch();

}

int KernelSem::val() const {

return value;

}

int KernelSem::iCall(PCB \*pcb){

if(waitQueue.deleteFromQ(pcb)){

value++;

return 1;

}else return 0;

}

//kersem.h

#ifndef \_KERSEM\_H\_

#define \_KERSEM\_H\_

#include "queue.h"

#include "semaphor.h"

#include "krobject.h"

class KernelSem : public KernelObject {

public:

friend class Semaphore;

int wait();

void signal();

int val() const;

KernelSem (int init);

virtual int iCall(PCB \*pcb);

private:

int value;

Queue waitQueue;

};

#endif

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: keyevent.cpp

// Date: June 2005

//===========================================================================//

#include <dos.h>

#include "keyevent.h"

#include "event.h"

#include "bounded.h"

#include "user.h"

#define KEYBOARD\_INT\_NUM 0x09

//---------------------------------------------------------------------------//

// Uvek definisati globalno da se moze pozivati iz INT rutine

//---------------------------------------------------------------------------//

Event\* pEV;

//---------------------------------------------------------------------------//

// Podeseno za qwerty tastature

// Tabela nije potpuna

//---------------------------------------------------------------------------//

char keymap[128] = {

0 , 27,'1','2','3','4','5','6','7','8','9','0','-','=', 8 , 9,

'q','w','e','r','t','y','u','i','o','p','[',']', 13, 0 ,'a','s',

'd','f','g','h','j','k','l',';',0,0,'`','\\','z','x','c','v','b',

'n','m',',','.','/', 0 ,'\*', 0 ,' '

};

void interrupt keyboardHandler(...)

{

// Ne pozivam zato sto ja citam kodove, inace je korisno

//pEV->oldHandler();

pEV->signal(); // obavezno za svaki user\_handler

// Extra user code here...

}

//---------------------------------------------------------------------------//

KeyboardEvent::KeyboardEvent(BoundedBuffer\* bb) : Thread("key"), myBuffer(bb)

{

theEnd = 0;

}

//---------------------------------------------------------------------------//

// pEV se mora kreirati i unistiti u funkciji run

// ne sme se "verovati" destruktoru zato sto je KeyboardEvent

// nasledjen iz Thread i destruktor moze da se izvrsi pre kraja run

// iz datog razloga je najbolje uopste ne definisati destruktor

//---------------------------------------------------------------------------//

void KeyboardEvent::run()

{

char scancode, status, znak;

// Create basic event & hook our handler

pEV = new Event(KEYBOARD\_INT\_NUM, keyboardHandler);

ID idCon;

idCon = Thread::getIdOf("con");

\_lock

printf("%d",idCon);

\_unlock

Thread \*tCon = getThreadById(idCon);

\_lock

printf("%s",tCon->getName());

\_unlock

while (!theEnd) {

pEV->wait();

status = inportb(0x64);

while (status & 0x01){ //dok se ocitava takav status da je pritisnut neki taster

//scancode = inportb(0x60);

if (status & 0x01){ // Can I read?

scancode = inportb(0x60);

};

//////////////////////

asm{

cli

in al, 61h //; Send acknowledgment without

or al, 10000000b //; modifying the other bits.

out 61h, al //;

and al, 01111111b //;

out 61h, al //;

mov al, 20h //; Send End-of-Interrupt signal

out 20h, al //;

sti

}

//////////////////////

if (status & 0x01){

// press ESC za theEnd

if (scancode==1)

theEnd = 1;

else {

if (scancode&0x80) {

tCon->Interrupt();

myBuffer->append(keymap[scancode&0x7F]);

};

};

};

status = inportb(0x64);

}//while

}// while

delete pEV;

//theEnd = 1; // Za producere i consumera

}

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: keyevent.h

// Date: June 2007

//===========================================================================//

#ifndef \_KEYEVENT\_H\_

#define \_KEYEVENT\_H\_

#include "thread.h"

class BoundedBuffer;

class Event;

class KeyboardEvent :public Thread{

public:

KeyboardEvent (BoundedBuffer\*);

protected:

virtual void run();

char znak;

private:

BoundedBuffer\* myBuffer;

};

#endif // \_OS1\_KEYBOARD\_EVENT\_

//krobject.h

#ifndef \_KROBJECT\_H\_

#define \_KROBJECT\_H\_

#include "pcb.h"

class PCB;

class KernelObject {

public:

virtual int iCall(PCB \*pcb) { return 0;}

};

#endif

//lock.h

#ifndef \_LOCK\_H\_

#define \_LOCK\_H\_

#define \_lock asm cli;

#define \_unlock asm sti;

#endif

//main.cpp

#include <iostream.h>

#include "kernel.h"

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

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

Kernel::initialization();

int value = userMain(argc, argv);

Kernel::termination();

printf("\nKRAJ!");

int i; cin >> i;

return value;

}

//pcb.cpp

#include "pcb.h"

#include "lock.h"

#include "kernel.h"

#include "schedule.h"

#include "krobject.h"

#include <stdlib.h>

#include <dos.h>

int PCB::countID = -2;

Queue \*PCB::indetificationQueue = new Queue();

Queue \*PCB::sleepQueue = new Queue();

PCB::PCB(char\* name, Thread\* myT, StackSize stackSize, LongTime timeSlice) {

//inicijalizacija svih clanica PCB-a, stek se formira kada se pozove metoda run()

\_lock;

pcbStackSize = stackSize;

pcbTimeSlice = timeSlice;

timePassed = 0;

timeToSleep = 0;

pcbSP = NULL;

waitQueue = new Queue();

waiting = NULL;

BlockingFlag = 0;

myThread = myT;

threadName = name;

threadID = ++countID;

if (threadID > 0) indetificationQueue->put(this);

\_unlock;

}

PCB::~PCB() {

\_lock

delete waitQueue;

delete pcbSP;

\_unlock

}

void PCB::Interrupt(){

\_lock;

if (state == BLOCKED){

if(waiting->iCall(this)){

BlockingFlag = 1;

state = READY;

Scheduler::put(this);

}

}

\_unlock;

}

void PCB::Sleep(Time SleepTime){

\_lock;

sleepQueue->put(this);

timeToSleep = SleepTime;

state = BLOCKED;

Kernel::dispatch();

\_unlock;

}

void PCB::createStack() {

\_lock

static volatile unsigned newSP, newSS, newIP, newCS, oldSP, oldSS, tax;

static unsigned char \*Stack;

//alokacija meorije i azuriranje SP

Stack = new unsigned char[this->pcbStackSize];

this->pcbSP = Stack;

newSP = this->sp = FP\_OFF(Stack + pcbStackSize);

newSS = this->ss = FP\_SEG(Stack + pcbStackSize);

newIP = FP\_OFF(&(Kernel::runWrapper));

newCS = FP\_SEG(&(Kernel::runWrapper));

asm {

mov tax, ax

mov oldSS, ss

mov oldSP, sp

mov ss, newSS

mov sp, newSP

pushf //postavljamo PSW na stek

pop ax //skidamo PSW sa steka

or ax, 1000000000b //setujemo bit I reci PSW kako bi omogucili prekide

push ax //vracamo PSW na stack

mov ax, newCS //stavljamo CS na stek

push ax

mov ax, newIP //stavljamo IP na stek

push ax

//stavljamo na stek ostale programski dostupne registre

mov ax, 0

push ax

push bx

push cx

push dx

push es

push ds

push si

push di

push bp

mov newSP, sp

mov sp, oldSP

mov ss, oldSS

}

this->sp = newSP; //azuriramo SP

\_unlock

}

//pcb.h

#ifndef \_PCB\_H\_

#define \_PCB\_H\_

#include "thread.h"

#include "queue.h"

typedef unsigned long LongTime;

typedef enum {NEW, READY, BLOCKED, OVER} State;

class Queue;

class KernelObject;

class PCB {

public:

PCB(char\* name, Thread\* myT, StackSize stackSize, LongTime timeSlice);

~PCB();

int threadID;

static int countID;

Thread \*myThread;

char \*threadName;

Time timeToSleep;

volatile LongTime timePassed;

LongTime pcbTimeSlice;

StackSize pcbStackSize;

unsigned char\* pcbSP;

unsigned int sp, ss;

void Interrupt();

void Sleep(Time SleepTime);

Queue \*waitQueue;

void createStack();

State state;

static Queue \*indetificationQueue;

static Queue \*sleepQueue;

KernelObject \*waiting;

unsigned int BlockingFlag;

};

#endif

//queue.cpp

#include "queue.h"

#include "pcb.h"

#include "thread.h"

#include "schedule.h"

#include <string.h>

#include "lock.h"

//trazi ID niti na osnovu njenog imena ako ne nadje nit vraca nulu

//prva stvorena korisnicka nit u sistemu ima ID = 1

ID Queue::findIdOf(char \*name){

Element \*current = first;

while(current != NULL){

if (!strcmp(name, current->pcb->threadName))

return current->pcb->threadID;

else current = current->next;

}

return 0;

}

//trazi nit na osnovu zadatog id,

Thread\* Queue::findThreadWithID(ID idToFind){

Element \*current = first;

while(current != NULL){

if (idToFind == current->pcb->threadID)

return current->pcb->myThread;

else current = current->next;

}

return NULL;

}

Queue::~Queue() {

Element \*current = first, \*toDelete;

while(current != NULL){

toDelete = current;

current = current->next;

delete toDelete;

}

}

PCB\* Queue::get(){

if (!first) return 0;

Element\* current = first;

first = first->next;

PCB\* pcb = current->pcb;

delete current;

return pcb;

}

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

Element \*fresh = new Element(pcb);

if (!first) first = fresh;

else last->next = fresh;

last = fresh;

}

//brise nit iz reda na osnovu zadatok pcb-a

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

if (first == NULL) return 0;

Element \*current = first->next, \*previous = first, \*temp = NULL;

if (first->pcb == pcb){

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

else first = current;

delete previous;

return 1;

}

if (!current) return 0;

while ((current->pcb != pcb) && (current->next != NULL)){

previous = previous->next;

current = current->next;

}

if (current->pcb == pcb){

temp = current;

previous->next = current->next;

if (current == last) last = previous;

delete temp;

return 1;

} else return 0;

}

//koristi se za red uspavani niti, ova operacije azurira sve uspavane niti

//tj. azurira njihova vremena - jos koliko treba da budu blokirane

void Queue::update(){

Element \*current = first;

while (current != NULL) {

Element \*temp = current->next;

if (current->pcb->timeToSleep <= 0){

current->pcb->state = READY;

Scheduler::put(current->pcb);

deleteFromQ(current->pcb);

}

current = temp;

}

current = first;

while (current != NULL){

current->pcb->timeToSleep -= minTimeSlice();

current = current->next;

}

}

//queue.h

#ifndef \_QUEUE\_H\_

#define \_QUEUE\_H\_

#include "pcb.h"

#include "thread.h"

#include <stdio.h>

class PCB;

class Thread;

class Queue{

struct Element {

PCB\* pcb;

Element\* next;

Element(PCB\* p, Element\* s=0){

pcb = p;

next = s;

}

};

Element\* first, \*last;

public:

Queue() : first(0) {}

~Queue();

void put(PCB\* pcb);

PCB\* get();

int empty() const { return first == NULL;}

int deleteFromQ(PCB\* pcb);

void update();

ID findIdOf(char \*name);

Thread\* findThreadWithID(ID idToFind);

};

#endif

//schedule.h

#ifndef \_SCHEDULE\_H\_

#define \_SCHEDULE\_H\_

#include "pcb.h"

class PCB;

class Scheduler {

public:

static void put(PCB\*);

static PCB\* get();

};

#endif

//semaphor.cpp

#include "semaphor.h"

#include "kersem.h"

#include <stdlib.h>

#include "lock.h"

Semaphore::Semaphore(int init){

\_lock

myImpl = new KernelSem(init);

\_unlock

}

Semaphore::~Semaphore(){

\_lock;

delete myImpl;

\_unlock;

}

int Semaphore::wait(){

\_lock;

int rez = myImpl->wait();

\_unlock;

return rez;

}

void Semaphore::signal(){

\_lock;

myImpl->signal();

\_unlock;

}

int Semaphore::val() const{

return myImpl->val();

}

//semaphor.h

#ifndef \_SEMAPHOR\_H\_

#define \_SEMAPHOR\_H\_

extern int semPreempt;

class KernelSem;

class Semaphore {

public:

Semaphore(int init = 1);

~Semaphore();

virtual int wait();

virtual void signal();

int val() const;

private:

KernelSem\* myImpl;

};

#endif

//thread.cpp

#include "kernel.h"

#include "thread.h"

#include "pcb.h"

#include "schedule.h"

#include <stdlib.h>

#include <dos.h>

#include "lock.h"

Time minTimeSlice() { return 55;} // 1s/18.2

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

\_lock

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

\_unlock

}

Thread::~Thread(){

\_lock

waitToComplete();

delete myPCB;

\_unlock

}

void Thread::start(){

\_lock

myPCB->state = READY; //menjamo stanje, PCB ulazi u red spremnih

myPCB->createStack(); //kreiramo stek

Scheduler::put(myPCB);

\_unlock

}

ID Thread::getId(){

return myPCB->threadID;

}

char\* Thread::getName(){

return myPCB->threadName;

}

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

return PCB::indetificationQueue->findThreadWithID(id);

}

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

return PCB::indetificationQueue->findIdOf(name);

}

void dispatch() {

\_lock

Kernel::dispatch();

\_unlock

}

void Thread::sleep(Time timeToSleep){

\_lock;

myPCB->Sleep(timeToSleep);

\_unlock;

}

void Thread::Interrupt(){

\_lock;

myPCB->Interrupt();

\_unlock;

}

void Thread::waitToComplete(){

\_lock;

//ne cekamo ako je nit zavrsila

if (myPCB->state == OVER){

\_unlock

return;

}

//nit ne ceka samu sebe

if (myPCB ==(PCB\*) Kernel::running){

\_unlock

return;

}

//nit ne radi nista

if(this == Kernel::idle){

\_unlock

return;

}

//pocetna nit ne treba da se ceka

if(this == Kernel::starting){

\_unlock

return;

}

Kernel::running->state = BLOCKED;

myPCB->waitQueue->put((PCB\*)Kernel::running);

Kernel::dispatch();

\_unlock;

}

//thread.h

#ifndef \_THREAD\_H\_

#define \_THREAD\_H\_

typedef unsigned long StackSize;

const StackSize defaultStackSize = 4096;

typedef float Time; //vreme u milisekundama

const Time defaultTimeSlice = 100;

typedef int ID;

void dispatch();

Time minTimeSlice();

extern void tick();

class PCB;

class Thread {

public:

void start(); //pokretanje niti

void waitToComplete();

void sleep(Time timeToSleep);

void Interrupt();

ID getId();

static Thread \* getThreadById(ID id);

static ID getIdOf(char\* name);

char \* getName();

virtual ~Thread();

protected:

friend class PCB;

friend class IdleThread;

friend class Kernel;

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

virtual void run() {}

private:

PCB \*myPCB;

};

#endif

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: user.h

// Date: June 2007

//===========================================================================//

#ifndef \_USER\_H\_

#define \_USER\_H\_

#include <stdio.h>

#include "thread.h"

#include "lock.h"

//---------------------------------------------------------------------------//

// Ovo se menja u testu

//---------------------------------------------------------------------------//

const Time TIME\_SLICE = 120; // 0 ili defaultTimeSlice

const int N = 3; // 1 <= N <= 20

//---------------------------------------------------------------------------//

// Signalizira kraj za Producer i Consumer preko Keyevent

//---------------------------------------------------------------------------//

extern int theEnd;

class BoundedBuffer;

//---------------------------------------------------------------------------//

class Producer : public Thread {

public:

Producer (char\* name, BoundedBuffer\* bb, char y) : Thread(name, defaultStackSize, TIME\_SLICE),

myBuffer(bb), x(y) {}

~Producer() {waitToComplete();}

protected:

virtual void run ();

char produce() {return x;}; // Produce an item

private:

BoundedBuffer\* myBuffer;

char x;

};

//---------------------------------------------------------------------------//

class Consumer : public Thread {

public:

Consumer (char\* name, BoundedBuffer\* bb) : Thread(name, defaultStackSize, 0), myBuffer(bb) {}

~Consumer() {waitToComplete();}

protected:

virtual void run ();

void consume(char p); // Consume an item

private:

BoundedBuffer\* myBuffer;

};

//---------------------------------------------------------------------------//

#endif // \_OS1\_USER\_H\_

//===========================================================================//

// Project: Projekat iz Operativnih sistema 1

// File: user.cpp

// Date: June 2007

//===========================================================================//

#include <dos.h>

#include "user.h"

#include "keyevent.h"

#include "bounded.h"

int semPreempt = 0;

//---------------------------------------------------------------------------//

int theEnd=0;

//---------------------------------------------------------------------------//

// Bez ovoga je ispis previse brz

//---------------------------------------------------------------------------//

void outputDelay()

{

delay(10);

}

//---------------------------------------------------------------------------//

void Consumer::consume(char p) {

\_lock

printf("%c ",p);

\_unlock;

} // Consume an item

void Producer::run () {

int i;

while(!theEnd) {

outputDelay();

char d = produce();

myBuffer->append(d);

}

}

//---------------------------------------------------------------------------//

void Consumer::run () {

int i = 0;

while(!theEnd) {

outputDelay();

char d = myBuffer->take();

consume(d);

if (i++ == 40) {

sleep(5000);

i = 0;

}

}

while (myBuffer->size()){

outputDelay();

char d = myBuffer->take();

consume(d);

}

printf("\nHappy End");

}

//---------------------------------------------------------------------------//

// Korisnicki program mora obavezno da definise ovu f-ju

//---------------------------------------------------------------------------//

void tick()

{};

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

{

BoundedBuffer buff;

char\* st = "con";

Consumer con(st, &buff);

Producer\* pro[N];

int i;

con.start();

for (i=0; i<N; i++){

char\* str = new char[7];

sprintf(str,"pro%d",i);

pro[i] = new Producer(str, &buff, '0'+i);

pro[i]->start();

}

KeyboardEvent\* kev = new KeyboardEvent(&buff);

kev->start();

for (i=0; i<N; i++){

delete pro[i];

}

delete kev;

return 0;

}

//---------------------------------------------------------------------------//