



Универзитет у Београду
Електротехнички факултет
Катедра за рачунарску технику и информатику

ИР2ОС1
ОПЕРАТИВНИ СИСТЕМИ 1
Пројектни задатак

Момир Ђекић
03/174

фебруар 2007.

0. УВОД

Систем

Систем пројектног задатка је реализован у 22 датотеке. Код је за потребе овог документа, посебно преформатиран, урађен је комплетан *[syntax highlighting](#)*, са посебно визуелно издвојеним кодом у асемблеру и коментарима.

Постоје две врсте коментара у коду:

- 1) коментар дела кода као наслов
- 2) коментар дела кода као додатно појашњење

Први се односе на чисто именовање дела кода или неког његовог својства, док други додатно појашњавају извршавање и разлоге због којих се део кода налази баш на том месту. Код писан у асемблеру је додатно обојен да би читалац могао јасније да чита ток кода, али и да би се могла обратити пажња на поједине делове пројекта који су од највеће важности.

Тест

За потребе тестирања пројектног задатка, написан је тест *[Producer/Consumer](#)* типа, где се симулира рад на виртуелном складишту реализованом и виду стандардног бројачког семафора који представља истовремено резерве, али и семафор за синхронизацију нити произвођача и потрошача. Свакој од нити је додељен сопствени *[timeslice](#)*, те се разлике јасно могу видети у табели резултата где су нити раздвојене именом и резултатима свог рада на стање (семафор) виртуелног складишта. Овај тест показује је концепт нити и семафора сисетма урађен у складу са потребама пројектног задатка.

1. thread.h

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: thread.h                          ****
//****      Opis: Zaglavlje klase Thread             ****
//****      ****
//****      Student: Momir Djekic 03/174             ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _THREAD_H_
#define _THREAD_H_

class PCB;

typedef float Time;
typedef unsigned long StackSize;

const StackSize defaultStackSize = 4096;
const Time defaultTimeSlice = 100;

class Thread {
public:
    void start();
    void waitToComplete();

    virtual ~Thread();

protected:
    friend class System;
    friend class IdleThread;
    friend class PCB;

    Thread (StackSize stackSize=defaultStackSize, Time
timeSlice=defaultTimeSlice);
    virtual void run() {}

private:
    PCB* myPCB;
};

void dispatch();
extern void tick();
Time minTimeSlice();

#endif
```

2. thread.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: thread.cpp                        ****
//****      Opis: Definicije metoda klase Thread      ****
//****                                             ****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include "system.h"
#include "pcb.h"
#include "thread.h"
#include "schedule.h"
#include <dos.h>
#include <stdlib.h>

class System;
class Scheduler;

Thread::Thread(StackSize stackSize, Time timeSlice) {
    lock();
    myPCB = new PCB(this, stackSize, (LongTime)timeSlice);
    unlock();
}

Thread::~Thread() {
    lock();
    waitToComplete();
    delete myPCB;
    unlock();
}

void Thread::start() {
    lock();
    myPCB->state = PCB::READY;
    myPCB->createStack();
    Scheduler::put(myPCB);
    unlock();
}
```

```

void Thread::waitToComplete() {
    lock();
    /*
    =====
    Nema potrebe da se nit ceka u sledecim situacijama:
    1. nit je zavrсила svoje izvršavanje
    2. ne može nit sama sebe da ceka
    Takodje, pocetna nit (starting) se poslednja završava te nju ne treba cekati
    ali ni nit koja ne radi nista (idle) koja se nikada i neće završiti
    =====
    */
    //|1.|
    if (myPCB->state == PCB::OVER) { unlock(); return; }

    //|2.|
    if (myPCB == (PCB*)System::running) { unlock(); return; }

    //|pocetna nit (starting)|
    if (this == System::starting) { unlock(); return; }

    //|nit koja ne radi nista (idle)|
    if (this == System::idle) { unlock(); return; }

    System::running->state = PCB::BLOCKED;
    myPCB->waitQueue->put((PCB*)System::running); //prijavi running za cekanje
    System::dispatch();
    unlock();
}

void dispatch() {
    lock();
    System::dispatch();
    unlock();
}

//void tick() {};

Time minTimeSlice() { return 55; } // |minimalni time slice, 55 ms|

```

3. pcb.h

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: pcb.h                               ****
//****      Opis: Zaglavlje klase PCB                 ****
//****                                             ****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                  ****
//*****
//*****

#ifndef _PCB_H_
#define _PCB_H_

#include "thread.h"
#include "queue.h"

typedef unsigned long LongTime;
class Thread;
class Queue;

class PCB {
public:
    PCB(Thread* myT, StackSize stackSize, LongTime timeSlice);
    ~PCB();

    Thread* myThread;

    unsigned int sp, ss;

    static const int NEW, READY, BLOCKED, OVER;
    volatile int state;

    volatile LongTime timePassedCounter;
    LongTime pcbTimeSlice;

    StackSize pcbStackSize;

    // |StackPointer|
    unsigned char* pcbSP;

    // |Red cekanja|
    Queue* waitQueue;

    // |f-ja koja obezbedjuje stvaranje pocetnog steka niti|
    void createStack();
};

#endif

```

4. pcb.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: pcb.cpp                          ****
//****      Opis: Definicije metoda klase PCB         ****
//****                                             ****
//****      Student: Momir Djekic 03/174             ****
//****      jun 2006.                                ****
//*****
//*****

#include "pcb.h"
#include "system.h"
#include <stdlib.h>
#include <dos.h>

// |Postavljanje konstanti stanja niti: NOVA, SPREMNA, BLOKIRANA, ZAVRSENA|
const int PCB::NEW = 0;
const int PCB::READY = 1;
const int PCB::BLOCKED = 2;
const int PCB::OVER = 3;

PCB::PCB(Thread* myT, StackSize stackSize, LongTime timeSlice) {
    lock();
    state = NEW;
    pcbStackSize = stackSize;
    myThread = myT;
/*
=====
Potrebno je napomenuti da se stek dodeljuje tek kada se pozove run() odredjene niti
radi uштеde u prostoru koji ce nit zauzeti. Postoji mogucnost da se neka nit nikada
nece pokrenuti, vec ce zauvek ostati u stanju NEW ili READY ili ce vecito biti blo-
kirana.
=====
*/
    pcbSP = NULL;
    pcbTimeSlice = timeSlice;
    timePassedCounter = 0.;
    waitQueue = new Queue();
    unlock();
}

PCB::~~PCB() {
    lock();
    delete waitQueue;
    delete pcbSP;
    unlock();
}
```

```

void PCB::createStack() {
    lock();
/*
=====
        tsp ---> sp
        tss ---> ss
        tip ---> ip
        tcs ---> cs
        oldss -> temporary ss
        oldsp -> temporary sp
=====
*/

    static volatile unsigned tsp, tss, tip, tcs, oldss, oldsp;
    static unsigned char *tempStack;

// |Alokacija prostora i dodela pokazivaca na formirani stek niti|
//*****
    tempStack = new unsigned char[this->pcbStackSize];
    this->pcbSP = tempStack;
//*****

// |Dohvatanje segment-a i offset-a novoformiranog steka|
//*****
    tsp = this->sp = FP_OFF(tempStack+pcbStackSize);
    tss = this->ss = FP_SEG(tempStack+pcbStackSize);
//*****

/*
=====
Simuliramo poziv omotacke f-je wrapper() zarad pamcenja mesta povratka kada se
zavrshi opsluzivanje interrupt rutine koja je pozvana kao sistemska usluga.
=====
*/

//*****
    tip = FP_OFF(&(System::wrapper));
    tcs = FP_SEG(&(System::wrapper));
//*****

/*
=====
Potrebno je sa steka f-je preci na novovofrmirani stek same niti, te to obavljamo
koriscenjem asemblera.
=====
*/
    asm {
        mov oldss, ss
        mov oldsp, sp
        mov ss, tss
        mov sp, tsp

/*
=====
Na stek stavljamo programsku statusnu rec (PSW) i eksplicitno postavljamo bit I
(enable interrupt) na 1 kako bismo omogucili prekide nakon povratka u omotacku f-ju
wrapper().
=====
*/

```

```

        pushf
        pop ax

/*
=====
Koriscenjem logicke operacije ILI (OR) postavljamo bit I PSW-a na 1
bbbbbbbbbbbbbbbb OR 0000001000000000 = bbbbbbb1bbbbbbbbbb
=====
*/
        or ax, 0000001000000000b
        push ax

// |Obezbedjujemo adekvatnu adresu povratka u omotacku f-ju wrapper(|
        mov ax, tcs
        push ax
        mov ax, tip
        push ax

/*
=====
Cuvanje konteksta...pozvana je interrupt f-ja koja bi trebalo da sve dole navedene
registre baca na stek.
=====
*/
        mov ax, 0
        push ax
        push bx
        push cx
        push dx
        push es
        push ds
        push si
        push di
        push bp

        // |Vracamo StackPointer niti|
        mov tsp, sp
        // |Vracamo StackPointer f-je|
        mov sp, oldsp
        // |StackSegment...nije bilo promene|
        mov ss, oldss
    }

// |Pozicija od koje ce se nastaviti izvorsavanje pri promeni konteksta|
    this->sp = tsp;
    unlock();
}

```

5. queue.h

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: queue.h                            ****
//****      Opis: Zaglavlje klase Queue              ****
//****            (realizovan preko ulancane liste)    ****
//****                                                    ****
//****      Student: Momir Djekic 03/174              ****
//****            jun 2006.                            ****
//*****
//*****

#ifndef _QUEUE_H_
#define _QUEUE_H_

#include "pcb.h"

class PCB;

// |Klasa cvora ulancane liste|
//*****
class Elem {
public:
    PCB* pcb;
    Elem* next;
};
//*****

class Queue {
public:
    int size() const;
    void put(PCB* pcb);
    PCB* get();

    Queue();
    ~Queue();

private:
    Elem* first;
    int len;
};

#endif

```

6. queue.cpp

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: queue.cpp                          ****
//****      Opis: Definicije metoda klase Queue      ****
//****              (realizovan preko ulancane liste) ****
//****
//****      Student: Momir Djekic 03/174              ****
//****              jun 2006.                          ****
//*****
//*****

#include <stdlib.h>
#include "queue.h"
#include "system.h"

Queue::Queue() {
    lock();
    first = NULL;
    len = 0;
    unlock();
}

Queue::~Queue() {
    lock();
    Elem* temp;
    while( first != NULL )
    {
        temp = first;
        first = first->next;
        delete temp;
        len--;
    }
    unlock();
}

int Queue::size() const {
    return len;
}

PCB* Queue::get() {
    if( first == NULL ) return NULL;
    lock();
    len--;
    Elem* temp = first;
    first = first->next;
    unlock();
    return temp->pcb;
}

```

```
void Queue::put(PCB* pcb) {  
    lock();  
    Elem **dp = &first;  
    while( (*dp) != NULL )  
    {  
        if( (*dp)->pcb == pcb ) { unlock(); return; }  
        dp = &((*dp)->next);  
    }  
    (*dp) = new Elem();  
    (*dp)->next = NULL;  
    (*dp)->pcb = pcb;  
    len++;  
    unlock();  
}
```

7. system.h

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: system.h                          ****
//****      Opis: Zaglavlje klase System              ****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _SYSTEM_H_
#define _SYSTEM_H_

#include "pcb.h"
#include "thread.h"
#include "idletred.h"
#include "event.h"

typedef void (interrupt *InterruptRoutine)();

// |Primitiva za zakljucavanje - zabrana asinhronog preuzimanja|
#define lock() {asm{ pushf; cli;}}

// |Primitiva za otkljucavanje|
#define unlock() {asm{ popf }}

class System {
public:
    static void init();
    static void term();
    static void dispatch();

private:
    friend class Thread;
    friend class PCB;
    friend class KernelSem;
    friend class KernelEv;

    static Thread* starting;
    static IdleThread* idle;
    static volatile PCB* running;

    // |Flag koji govori da li je pozvan eksplicitno dispatch()|
    static volatile char dispatched;

    static void interrupt (*oldRoutine)();
    static void interrupt timerIR();
    static InterruptRoutine replaceRoutine(IVTNo intN, InterruptRoutine
newRoutine);
    static void wrapper();
};

#endif

```

8. system.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: system.cpp                        ****
//****      Opis: Definicije metoda klase System      ****
//****      ****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include "system.h"
#include "schedule.h"
#include "kernelev.h"
#include <dos.h>
#include <stdlib.h>

// |Pocetni uslov kaze da nije bilo eksplicitnog dispatch()|
volatile char System::dispatched = 0;

void interrupt (*System::oldRoutine)() = NULL;
volatile PCB* System::running = NULL;
Thread* System::starting = NULL;
IdleThread* System::idle = NULL;

// |INIT sistema - njegovo pokretanje|
void System::init() {
    lock();
// |Na mesto sistemskog brojaca na ulazu 08 unosimo nasu prekidnu rutinu|
//*****
    oldRoutine = replaceRoutine(0x08, timerIR);
//*****

// |Pocetna nit|
//*****
    starting = new Thread(0x10000, minTimeSlice());
//*****

/*
=====
Pocetnu nit cemo pokrenuti, a nakon nekog vremenskog intervala doci ce do promene
konteksta te ce ova nit cekati (u Sheduler-u)na zavrsetak svih pokrenutih niti.
=====
*/

//*****
    starting->myPCB->state = PCB::READY;
    running = (volatile PCB*)starting->myPCB;
//*****

```

```

// |Nit koja ne radi nista - IdleThread|
/*
=====
Idle nit startujemo, pri чему joj je dodeljen stek ali je ne stavljamo u Sheduler,
jer nam je ona potrebna bas kada Sheduler nema nijednu nit iz reda spremnih da
prosledi sistemu.
=====
*/
//*****
    idle = new IdleThread();
    idle->start();
//*****
    unlock();
}

// |TERM sistema - njegovo gasenje|
void System::term() {

/*
=====
Prilikom gasenja niti, potrebno je proveriti da li su sve niti, osim pocetne
(starting) završene.
=====
*/
    if( (PCB*)running != starting->myPCB ) return;
    lock();

// |Vracamo prekidnu rutinu na svoje mesto|
    replaceRoutine(0x08, oldRoutine);
/*
=====
Ispunjeni su svi uslovi i sada mozemo obrisati pocetnu nit i na taj nacin završiti
u potpunosti rad sistema.
=====
*/
    delete starting;
    unlock();
}

void System::dispatch() {
    lock();
    // |Eksplicitni dispatch(), setujemo flag|
    dispatched = 1;
    timerIR();
    // |Resetujemo flag|
    dispatched = 0;
    unlock();
}

// |Prekidna rutina za promenu konteksta|
void interrupt System::timerIR() {
    static volatile unsigned int tsp, tss;
    static volatile PCB *newThread;

```

```

// |Ukoliko je poziv bio eksplicitan, pozovi staru prekidnu rutinu|
//*****
if(!dispatched ) {
    tick();
    (*oldRoutine)();
}
//*****

// |INTD - interrupt disable; CLI - clear interrupt |
asm { cli };

// |U zavisnosti od vrste dispatch()-a i isteka time slice-a:|

/*
=====
1. Ukoliko je dispatch() pozvan eksplicitno i vreme koje je dodeljeno niti nije
beskonacno, potrebno je azurirati brojac.
=====
*/
//*****
if (!dispatched && running->pcbTimeSlice != 0)
    running->timePassedCounter += (LongTime)minTimeSlice();
//*****

/*
=====
2. Ukoliko dispatch() nije pozvan eksplicitno i vreme koje je dodeljeno niti nije
isteklo, vratiti se.
=====
*/

//*****
if (!dispatched && (running->timePassedCounter < running->pcbTimeSlice ||
running->pcbTimeSlice == 0) ) return;
//*****

// |Ukoliko je flag bio setovan, resetujemo ga|
if (dispatched) dispatched = 0;

// |Ukoliko je nit spremna, a nije IdleThread, nit vracamo u Sheduler|
//*****
if( running->state == PCB::READY && running != idle->myPCB )
Scheduler::put((PCB*)running);

//*****
while(1)
{
    newThread = Scheduler::get();
    // |Ako je prazan Sheduler, eto posla za IdleThread|
    if (newThread == NULL) newThread = idle->myPCB;
    // |Ako dohvacena nit nije spremna, trazimo novu|
    if (newThread->state != PCB::READY) continue;

```

```

// |Promena konteksta|
asm {
    mov tsp, sp
    mov tss, ss
}
running->sp = tsp;
running->ss = tss;
running = newThread;
tsp = running->sp;
tss = running->ss;
asm {
    mov sp, tsp
    mov ss, tss
}
running->timePassedCounter = 0.;
break;
}
return;
}

/*
=====
Nova prekidna rutina ide na odgovarajuci ulaz u IVT (Interrupt Vector Table). Ova
f-ja, replaceRoutine() vraća staru prekidnu rutinu kada postavi nasu, korisnicku.
=====
*/

InterruptRoutine System::replaceRoutine(IVTNo intN, InterruptRoutine newRoutine) {
    lock();
    // |Segment i offset nove rutine|
    //*****
    unsigned int tseg = FP_SEG(newRoutine);
    unsigned int toff = FP_OFF(newRoutine);
    //*****

    unsigned int oldseg, odloff;
    InterruptRoutine old;

    // |Quote, source = http://www.cpu-world.com/Arch/8086.html|
    //
    =====
    When an interrupt occurs, the processor stores FLAGS register into stack, disables
    further interrupts, fetches from the bus one byte representing interrupt type, and
    jumps to interrupt routine address which is stored in location
    >>>4*<interrupt type><<<
    =====
    */

    intN *= 4;

```

```

// |Kod dat na vezbama|
asm {
    push es
    push ax
    push bx

    mov ax,0
    mov es,ax
    mov bx, word ptr intN
    //pamti staru
    mov ax, word ptr es:bx+2
    mov word ptr oldseg, ax
    mov ax, word ptr es:bx
    mov word ptr odloff, ax

    //postavlja novu
    mov ax, word ptr tseg
    mov word ptr es:bx+2, ax
    mov ax, word ptr toff
    mov word ptr es:bx, ax

    pop bx
    pop ax
    pop es
}
old = (InterruptRoutine) MK_FP(oldseg,odloff);
unlock();
return old;
}

// |Omotacka f-ja metode run() niti|
void System::wrapper() {
    running->myThread->run();
    lock();
    // |Proglasi nit gotovom|
    running->state = PCB::OVER;
/*
=====
Potrebno je aktivirati sve niti koje su cekale na ovoj, te iz odgovarajuceg reda
punimo Sheduler.
=====
*/
    PCB* temp;
    while( running->waitQueue->size() > 0 )
    {
        temp = running->waitQueue->get();
        temp->state = PCB::READY;
        Scheduler::put(temp);
    }
    // |Predajemo procesor|
    dispatch();
}

```

9. semaphor.h

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: semaphor.h                        ****
//****      Opis: Zaglavlje klase Semaphore          ****
//****      ****
//****      Student: Momir Djekic 03/174             ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _SEMAPHOR_H_
#define _SEMAPHOR_H_

extern int semPreempt;

class KernelSem;

class Semaphore
{
public:
    Semaphore (int init=1);
    ~Semaphore ();

    void wait();
    void signal();

    int val() const;

private:
    KernelSem* myImpl;
};

#endif
```

10. semaphor.cpp

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: semaphor.cpp                      ****
//****      Opis: Definicije metoda klase Semaphore  ****
//****                                             ****
//****      Student: Momir Djekic 03/174             ****
//****      jun 2006.                                ****
//*****
//*****

#include <stdlib.h>
#include "semaphor.h"
#include "krnlsem.h"
#include "system.h"
#include "schedule.h"

// |Da li treba da dodje do preuzimanja|
int semPreempt = 0;

Semaphore::Semaphore(int init) {
    lock();
    myImpl = new KernelSem(init);
    unlock();
}

Semaphore::~Semaphore() {
    lock();
    delete myImpl;
    unlock();
}

// |wait()|
//*****
void Semaphore::wait() {
    lock();
    myImpl->wait();
    unlock();
}

//*****

// |signal()|
//*****
void Semaphore::signal() {
    lock();
    myImpl->signal();
    unlock();
}

//*****

int Semaphore::val() const {
    return myImpl->val();
}

```

11. idletred.h

```

//*****
//*****
//****          Predmet:Operativni sistemi 1 IR2OS1          ****
//****          Fajl: idletred.h                               ****
//****          Opis: Zaglavlje klase IdleThread              ****
//****                                                    ****
//****          Student: Momir Djekic 03/174                  ****
//****          jun 2006.                                     ****
//*****
//*****

#ifndef _IDLETRED_H_
#define _IDLETRED_H_

#include "thread.h"

class Thread;

class IdleThread : public Thread {
public:
    IdleThread();
    virtual void run();
    void start();
};

#endif
```

12. idletred.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: idletred.cpp                      ****
//****      Opis: Definicije metoda klase IdleThread ****
//****                                             ****
//****      Student: Momir Djekic 03/174            ****
//****      jun 2006.                               ****
//*****
//*****

#include "system.h"
#include "idletred.h"
#include "pcb.h"

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

/*
=====
IdleThread je systemska besposlena nit, ona u sustini ne radi nista, vec se samo
vrti u beskonacnoj petlji. Ova nit ne ide u Sheduler i ne rasporedjuje se.
Koristimo je onda kada Sheduler-u ponestane spremnih niti.
=====
*/

void IdleThread::run() {
    while (1);
}

void IdleThread::start() {
    lock();
    myPCB->state = PCB::READY;
    myPCB->createStack();
    unlock();
}
```

13. event.h

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: event.h                             ****
//****      Opis: Sistemska klasa dogadjaja (interfejs) ****
//****      zaglavlje                                   ****
//****                                               ****
//****      Student: Momir Djekic 03/174               ****
//****      jun 2006.                                  ****
//*****
//*****

#ifndef _EVENT_H_
#define _EVENT_H_

typedef unsigned int IVTNo;
typedef void interrupt (*InterruptHandler)(...);

class KernelEv;

class Event
{
public:
    Event(IVTNo, InterruptHandler);
    ~Event();

    void wait();
    void signal();

    void oldHandler();

private:
    KernelEv* myImpl;
};

#endif
```

14. event.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: event.cpp                          ****
//****      Opis: Sistemska klasa dogadjaja (interfejs) ****
//****      definicije                                ****
//****                                              ****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include "event.h"
#include "thread.h"
#include "system.h"
#include "kernelev.h"
#include "pcb.h"
#include "schedule.h"
#include <stdlib.h>

Event::Event(IVTNo n, InterruptHandler h) {
    lock();
    myImpl = new KernelEv(n, h);
    unlock();
}

Event::~Event() {
    lock();
    delete myImpl;
    unlock();
}

void Event::wait() {
    lock();
    myImpl->wait();
    unlock();
}

void Event::signal() {
    lock();
    myImpl->signal();
    unlock();
}

void Event::oldHandler() {
    myImpl->oldHandler();
}
```

15. kernelev.h

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: kernelev.h                        ****
//****      Opis: Sistemska klasa dogadjaja - zaglavlje ****
//****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _KERNELEV_H_
#define _KERNELEV_H_

#include "event.h"
#include "schedule.h"
#include "queue.h"

class KernelEv {
public:
    void wait();
    void signal();

    void oldHandler();

    KernelEv(IVTNo, InterruptHandler);
    ~KernelEv();

private:
    IVTNo ivtEntry;
    InterruptHandler oldHandle;

    Queue waitQueue;
};

#endif

```

16. kernelev.cpp

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: kernelev.cpp                      ****
//****      Opis: Sistemska klasa dogadjaja - definicije ****
//****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include "system.h"
#include "kernelev.h"
#include "event.h"
#include "queue.h"

KernelEv::KernelEv(IVTNo n, InterruptHandler h) {
    ivtEntry = n;
    oldHandle = (InterruptHandler)System::replaceRoutine(n, (InterruptRoutine)h);
}

KernelEv::~KernelEv() {
    System::replaceRoutine(ivtEntry, (InterruptRoutine)oldHandle);
}

void KernelEv::wait() {
    // |running = BLOCKED|
    System::running->state = PCB::BLOCKED;
    waitQueue.put((PCB*)System::running );
    // |dispatch()|
    System::dispatch();
}

void KernelEv::signal() {
    PCB* temp;
    // |deblock all|
    while( waitQueue.size() > 0 ) {
        temp = waitQueue.get();
        temp->state = PCB::READY;
        Scheduler::put(temp);
    }
    // |dispatch()|
    System::dispatch();
}

void KernelEv::oldHandler() {
    (*oldHandle)();
}

```

17. krnlsem.h

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: krnlsem.h                          ****
//****      Opis: Sistemska klasa semafora - zaglavlje ****
//****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _KRNLSEM_H_
#define _KRNLSEM_H_

#include "queue.h"

class KernelSem {
public:
    friend class Semaphore;

    void wait();
    void signal();

    int val() const;

    KernelSem(int init=1);

private:
    int value;
    Queue waitQueue;
};

#endif
```

18. krnlsem.cpp

```

//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: krnlsem.cpp                        ****
//****      Opis: Sistemska klasa semafora - definicije ****
//****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include <stdlib.h>
#include "krnlsem.h"
#include "semaphor.h"
#include "system.h"
#include "schedule.h"
#include "queue.h"

KernelSem::KernelSem(int init) {
    value = init;
}

void KernelSem::wait() {
    value--;
    if( value < 0 )
    {
        System::running->state = PCB::BLOCKED;
        waitQueue.put((PCB*)System::running);
        System::dispatch();
    }
    // |Ukoliko je trazeno preuzimanje, dispatch()|
    else if (semPreempt)
        System::dispatch();
}

void KernelSem::signal() {
    value++;
    if( value <= 0 )
    {
        PCB* t = waitQueue.get();
        t->state = PCB::READY;
        Scheduler::put(t);
    }
    if( semPreempt )
        System::dispatch();
}

int KernelSem::val() const { return value; }

```

19. schedule.h

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: schedule.h                        ****
//****      Opis: SCHEDULER                          ****
//****      zaglavlje                                ****
//****                                              ****
//****      Student: Momir Djekic 03/174             ****
//****      jun 2006.                                ****
//*****
//*****

#ifndef _SCHEDULE_H_
#define _SCHEDULE_H_

#include "pcb.h"
#include "queue.h"

class Scheduler {
public:
    static void put (PCB*);
    static PCB* get ();

private:
    static Queue q;
};

#endif
```

20. schedule.cpp

```
//*****
//*****
//****          Predmet:Operativni sistemi 1 IR2OS1          ****
//****          Fajl: schedule.cpp                          ****
//****          Opis: SCHEDULER                              ****
//****          definicije                                   ****
//****                                                    ****
//****          Student: Momir Djekic 03/174                ****
//****          jun 2006.                                    ****
//*****
//*****

#include "schedule.h"
#include "system.h"

Queue Scheduler::q;

void Scheduler::put(PCB* p) {
    lock();
    q.put(p);
    unlock();
}

PCB* Scheduler::get() {
    lock();
    PCB* temp = q.get();
    unlock();
    return temp;
}
```

21. status.h

```
//*****  
//*****  
//****          Predmet:Operativni sistemi 1 IR2OS1          ****  
//****          Fajl: main.cpp                                ****  
//****          Opis: F-ja main() startuje i stopira sistem   ****  
//****          ****  
//****          Student: Momir Djekic 03/174                 ****  
//****          jun 2006.                                     ****  
//*****  
//*****  
  
#include <iostream.h>  
#include "system.h"  
#include "semaphor.h"  
  
Semaphore PrintStatus(1);  
  
#define PRINT(X) { PrintStatus.wait(); cout << X << endl; PrintStatus.signal(); }
```

22. main.cpp

```
//*****
//*****
//****      Predmet:Operativni sistemi 1 IR2OS1      ****
//****      Fajl: main.cpp                          ****
//****      Opis: F-ja main() startuje i stopira sistem ****
//****
//****      Student: Momir Djekic 03/174              ****
//****      jun 2006.                                ****
//*****
//*****

#include <iostream.h>
#include "system.h"
#include "status.h"

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

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

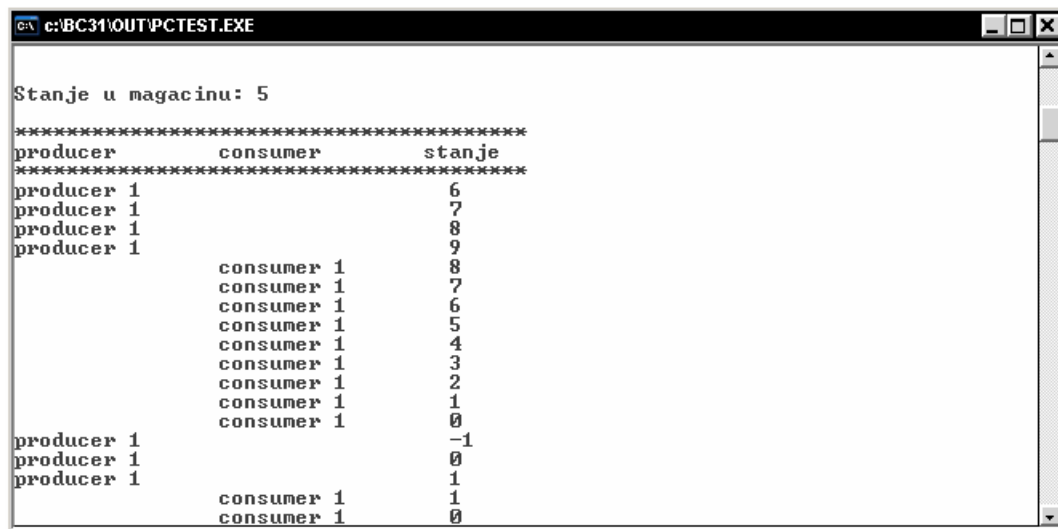
    // |SYSTEM ----> ON|
    System::init();

    PRINT("*****\n");
    PRINT("*****\n");
    PRINT("****      Predmet: Operativni sistemi 1 (IR2OS1)      ****\n");
    PRINT("****      Projektni zadatak - TEST                      ****\n");
    PRINT("****      Student: Momir Djekic 03/174                    ****\n");
    PRINT("****      Profesor: doc. dr. Dragan Milicev              ****\n");
    PRINT("****      Asistent: dipl. ing. Sasa Stojanovic            ****\n");
    PRINT("*****\n");
    PRINT("*****\n");

    // |Start korisnicke glavne f-je|
    PRINT("Start test f-je:\n");
    int value = userMain(argc, argv);
    // |SYSTEM ----> OFF|
    System::term();
    return value;
}
```

23. ПРИЛОГ - тест

У даљем тексту су дате датотеке *main.cpp* и *test.h* које се користе за потребе теста Producer/Consumer.



```
c:\BC31\OUT\PCTEST.EXE

Stanje u magacinu: 5

*****
producer      consumer      stanje
*****
producer 1          6
producer 1          7
producer 1          8
producer 1          9
               consumer 1      8
               consumer 1      7
               consumer 1      6
               consumer 1      5
               consumer 1      4
               consumer 1      3
               consumer 1      2
               consumer 1      1
               consumer 1      0
producer 1          -1
producer 1          0
producer 1          1
               consumer 1      1
               consumer 1      0
```

23.1 test.h

```

//*****
//*****
//****          Predmet:Operativni sistemi 1 IR2OS1          ****
//****          Fajl: test.h                                ****
//****          Opis: Definicije klasa i metoda testova      ****
//****                                                    ****
//****          Student: Momir Djekic 03/174                 ****
//****          februar 2007.                                ****
//*****
//*****

#include <iostream.h>
#include "thread.h"
#include "semaphor.h"
#include "status.h"

// |U skladistu pre rada niti|
Semaphore rezerve(5);

class Worker : public Thread{
public:
    Worker(char* ime, StackSize size=defaultStackSize, Time
slice=defaultTimeSlice): Thread(size, slice), ime(ime) {};
    ~Worker() { waitToComplete(); };
protected:
    virtual void run();
    char* ime;
};

void Worker::run()
{
    for(int i=1; i<=20; i++)
    {
        PRINT(ime << ".run(): " << i);

        for(long j=1; j<=10000000; j++);
    }
}

class Producer : public Worker{
public:
    Producer(char* ime, int nn, StackSize size=defaultStackSize, Time
slice=defaultTimeSlice): Worker(ime, size, slice), n(nn) {};
    ~Producer() { waitToComplete(); };

protected:
    int n; // proizvodni kapaciteti Producer-a
    virtual void run();
};

```


23.2 main.cpp

```
//*****
//*****
//****          Predmet: Operativni sistemi 1 IR2OS1          ****
//****          Fajl: main.cpp                                ****
//****          Opis: F-ja main() za potrebe testiranja        ****
//****                                                    ****
//****          Student: Momir Djekic 03/174                  ****
//****          februar 2007.                                ****
//*****
//*****

#include <iostream.h>
#include "system.h"
#include "test.h"

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

    PRINT("*****\n");
    PRINT("****   Testing: Producer/Consumer   ****\n");
    PRINT("*****\n");

    PRINT("\nStanje u magacinu: " << rezerve.val() << "\n");
    PRINT("*****")
    PRINT("producer\t" << "consumer\t" << "stanje");
    PRINT("*****");
    Producer Producer1("producer 1",25,5000,55);
    Consumer Consumer1("consumer 1",18,5000,110);
    Consumer Consumer2("consumer 2",2,5000,110);
    Producer1.start();
    Consumer1.start();
    Consumer2.start();

    return 1;
};

int main(int argc, char* argv[]) {
    char p;
    // |SYSTEM ----> ON|
    System::init();
    PRINT("*****\n");
    PRINT("*****\n");
    PRINT("****          Predmet: Operativni sistemi 1 (IR2OS1)          ****\n");
    PRINT("****          Projektni zadatak - TEST                          ****\n");
    PRINT("****          Student: Momir Djekic 03/174                      ****\n");
    PRINT("****          Profesor: doc. dr. Dragan Milicev                ****\n");
    PRINT("****          Asistent: dipl. ing. Sasa Stojanovic              ****\n");
    PRINT("*****\n");
    PRINT("*****\n");
}
```

```
// |Start korisnicke glavne f-je|
PRINT("Start test f-je:\n");
int value = userMain(argc, argv);
// |SYSTEM ----> OFF|
System::term();
PRINT("\n*****\n");
PRINT("Pritisnite bilo koji znak i ENTER za kraj!");
cin >>p;
return value;

}
```

САДРЖАЈ

0. УВОД.....	2
1. thread.h.....	3
2. thread.cpp	4
3. pcb.h	6
4. pcb.cpp	7
5. queue.h	10
6. queue.cpp.....	11
7. system.h	13
8. system.cpp	14
9. semaphor.h	19
10. semaphor.cpp	20
11. idletred.h.....	21
12. idletred.cpp	22
13. event.h	23
14. event.cpp	24
15. kernelev.h	25
16. kernelev.cpp.....	26
17. krnlsem.h.....	27
18. krnlsem.cpp.....	28
19. schedule.h	29
20. schedule.cpp.....	30
21. status.h	31
22. main.cpp	32
23. ПРИЛОГ - тест	33
23.1 test.h.....	33
23.2 main.cpp	36