Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku
Predmet: Operativni sistemi 1(SI2OS1, IR2OS1)
Nastavnik: Prof. dr Dragan Milićev
Asistent: Bojan Furlan

Laboratorijska vežba

Razvojno okruženje Borland C++ 3.1

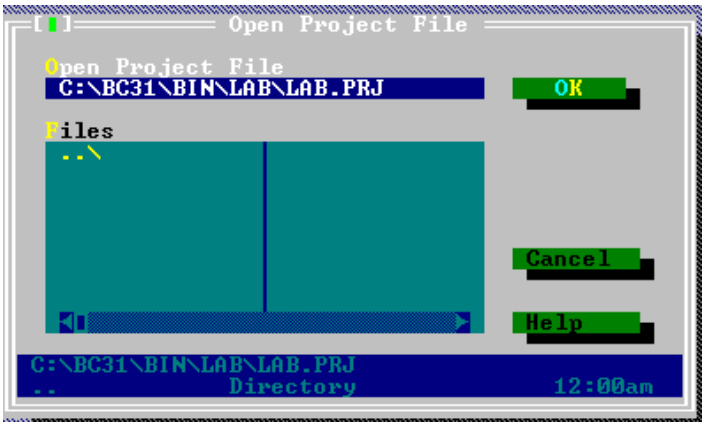
Važne napomene: Pre dolaska na vežbu pročitati i upoznati se sa ovim tekstom **u celini i pažljivo**. Ukoliko u vežbi nešto nije dovoljno precizno navedeno ili student naiđe na neki problem, prvo treba da pokuša sam da ga razreši, pa tek onda ukoliko ne uspe da zatraži pomoći od demonstratora ili predmetnog asistenta, jer se očekuje kreativnost i profesionalni pristup u rešavanju praktičnih problema!

Sadržaj

Podešavanje okruženja

Napomena: Razvojno okruženje Borland C++ 3.1 je instalirano u direktorijumu c:\bc31\

1. Raspakovati fajl lab.zip (sa adrese http://os.etf.bg.ac.rs/OS1/vezbe/lab/) u direktorijum c:\bc31\bin\lab\
2. Pokrenuti razvojno okruženje c:\bc31\bin\bc.exe
3. Kreirati novi projekat – iz glavnog menija odabrati Project->Open
4. Zatim odabrati putanju c:\bc31\bin\lab\ i uneti ime na c:\bc31\bin\lab\lab.prj (kao na slici) i kliknuti na OK.



5. Unutar direktorijuma c:\bc31\bin\lab\ kreirati poddirektorijume: OUTPUT i LIB (npr. pomocu File Explorera OS domaćina)
6. Odabrati Options->Directories… i zatim dopuniti putanje kao na slici:

7. Sačuvati projekat: Options->Save…

Napomena: Na ovaj način sva podešavanja biće sačuvana u okviru datog projekta.

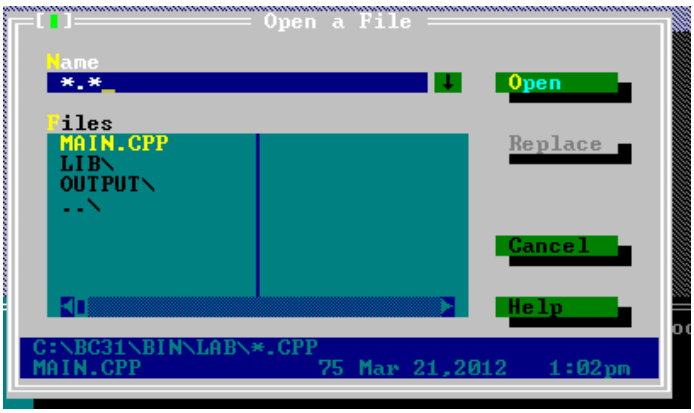
Pretprocesor

Pretprocesor obrađuje sve direktive u ulaznim fajlovima. Najčešće korišćene direktive:

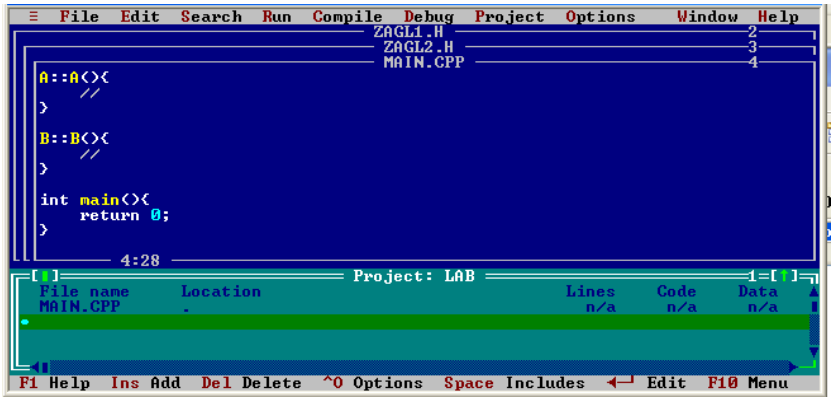
- #include "fajl" – direktivu zamenjuje sadržajem navedenog fajla; fajl traži i u lokalnom direktorijumu;
- #include <fajl> – direktivu zamenjuje sadržajem navedenog fajla; fajl ne traži u lokalnom direktorijumu;
- #define simbol sadržaj – definiše navedeni simbol i dodeljuje mu zadati sadržaj;
- #ifndef simbol – početak bloka koji se prepisuje u izlazni fajl samo ukoliko dati simbol nije definisan;
- #endif – kraj #ifndef direktive.

Otvaranje i prevođenje fajlova

1. Otvoriti fajlove `zagl1.h` i `zagl2.h` i `main.cpp`
    a. File->Open, zatim u polje filtera uneti *.* kao na slici



    b. Zatim, iz liste odabrati fajl `zagl1.h`
    c. Ponoviti za `zagl2.h`
2. Dodati fajl main.cpp u listu fajlova projekta
    a. Window->Project, pa zatim pritisnuti taster INSERT
    b. Iz liste odabrati fajl `main.cpp` i pritisnuti dugme ADD pa DONE
    c. Lista fajlova projekta izgledaće kao na slici



3. Prevesti fajl `main.cpp` pomoću naredbe Compile->BuildAll

   Koju će grešku javiti kompajler? Zašto?

   Pogledati sadržaj ova 3 fajla.

4. Pozvati preprocessor iz komandne linije pomoću sledećih naredbi:
   `cd c:\bc31\bin\lab`
   `c:\bc31\bin\cpp -Ic:\bc31\include;. main.cpp`
5. Pomoću tekst editora otvoriti fajl `main.i` i pogledati generisani sadržaj.

Zaštita od višestrukog uključivanja

1. Izmeniti date fajlove kao u nastavku

```
"zagl1.h":
-----------------------------------------------------------------
#ifndef _ZAGL1_H_
#define _ZAGL1_H_

#include "zagl2.h"

class A{
  public:
    A();
};

#endif
-----------------------------------------------------------------

"zagl2.h":
-----------------------------------------------------------------
#ifndef _ZAGL2_H_
#define _ZAGL2_H_

#include "zagl1.h"

class B {
  public:
    B();
```

```
};

#endif
-----------------------------------------------------------------

"main.cpp"
-----------------------------------------------------------------
#include "zagl1.h"
#include "zagl2.h"

A::A(){
    //
}

B::B(){
    //
}

int main(){
    return 0;
}
-----------------------------------------------------------------
```

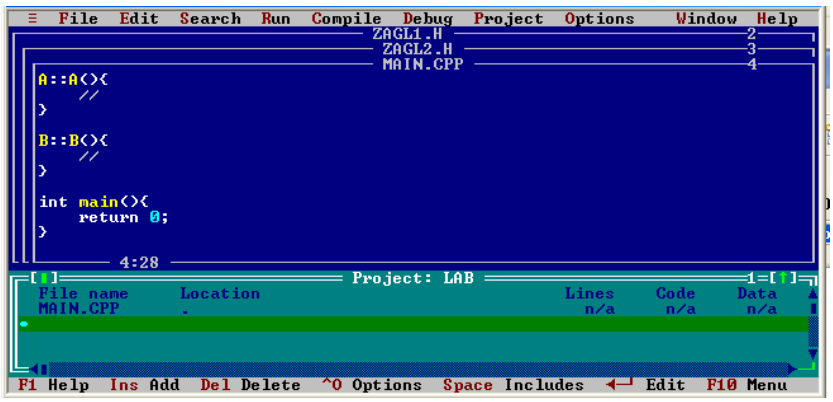
Opis generisanog izlaza "main.i" dat je u nastavku:

```
                                        komentari (nisu deo izlaza)
-----------------------------------------------------------------
main.cpp 1:                             na pocetku main.cpp
ukljucen zagl1.h
.\zagl1.h 1:                            nije definisano _ZAGL1_H_
...                                     definise se _ZAGL1_H_
.\zagl1.h 4:                            ukljucuje se zagl2.h (iz
zagl1.h)
zagl2.h 1:                              nije definisano _ZAGL2_H_
...                                     definise se _ZAGL2_H_
zagl2.h 4:                              ukljucuje se zagl1.h (iz
zagl2.h)
zagl1.h 1:                              definisano _ZAGL1_H_ i zato
...                                     se ostatak fajla zagl1.h
zagl1.h 12:                             preskače
zagl2.h 5:                              zavrseno ukljucivanje
zagl1.h iz zagl2.h
zagl2.h 6: class B {                    ostatak sadržaja zagl2.h
zagl2.h 7: public:
zagl2.h 8: B();
zagl2.h 9: };
...
zagl2.h 12:
.\zagl1.h 5:                            završeno uključivanje
zagl2.h iz zagl1.h
.\zagl1.h 6: class A{                   ostatak sadržaja zagl1.h
.\zagl1.h 7: public:
.\zagl1.h 8: A();
.\zagl1.h 9: };
...
```

```
.\zagl1.h 12:
main.cpp 2:                              završeno uključ. zagl1.h iz
main.cpp
.\zagl2.h 1:                             uključuje se zagl2.h iz
main.cpp
...                                      ali je sada definisano
_ZAGL2_H_
.\zagl2.h 12:                            pa se ostatak zagl2.h
preskače
main.cpp 3:                              ostatak sadržaja main.cpp
main.cpp 4: A::A(){
main.cpp 5:
main.cpp 6: }
main.cpp 7:
main.cpp 8: B::B(){
main.cpp 9:
main.cpp 10: }
main.cpp 11:
main.cpp 12: int main(){
main.cpp 13: return 0;
main.cpp 14: }
main.cpp 15:
-----------------------------------------------------------------
```

2. Pokušati ponovo prevođenje fajla `main.cpp`
   Da li je sada uspešno? Zašto?

3. Zatvoriti sve prozore i ukloniti fajl `main.cpp` iz projekta, tako što se odabere ovaj fajl iz liste i zatim obriše pritiskom na taster DELETE.



Postavka zadatka

U zadatku 3 sa vežbi (fajl z3n.cpp) realizovana je promena konteksta (engl. context switch) pomoću prekida od timera (08h). Kontekst procesora čuva se na steku i pri tom svakoj niti u sistemu može se dodeliti različit kvant vremena za izvršavanje.
Potrebno je izmeniti dato rešenje na sledeći način:

1. Obezbediti zabranu preuzimanja bez zabrane prekida
   - Izmeniti ispis `cout` u funkcijama `a()` i `b()` kao i u prekidnoj rutini.
2. Univerzalno rešiti problem alokacije lokalnih promenljivih u proceduri za promenu konteksta.
3. Izmeniti način raspoređivanja, odnosno izbor niti kojoj će biti dodeljen procesor iz skupa spremnih niti, tako što će biti upotrebljen modul koji je obezbeđen spolja i koji sadrži listu spremnih PCB-ova, kao i sam algoritam raspoređivanja.

   Deklaracije interfejsa ovog modula nalaze se u zaglavlju `schedule.h` i izgledaju ovako:

```
// File: schedule.h
class PCB;

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

   Operacija `get()` vraća onu nit iz reda spremnih koja je po algoritmu raspoređivanja izabrana da bude sledeća za izvršavanje i izbacuje tu nit iz reda spremnih. Operacija `put()` smešta datu nit u red spremnih.

Zabrana preuzimanja bez zabrane prekida

1. U projekat dodati fajl Z3N.CPP i otvoriti ga.
2. Dodati globalnu poromenljivu

```
volatile unsigned int lockFlag = 1;   //fleg za zabranu promene konteksta
```

3. Izmeniti ispis u funkciji `a()` na sledeći način:

```
//begin lock
  lockFlag=0;
//end lock

  cout<<"u a() i = "<<i<<endl;

//begin unlock
                                //ovakav unlock je potreban da bi se proverilo
```

```
  lockFlag=1;                   //da li je zatrazeno preuzimanje u toku ispisa;
  if (zahtevana_promena_konteksta) {    //ako jeste onda se sada vrsi eksplicitno
                                        //preuzimanje;
     dispatch();
  }
//end unlock

//voditi racuna da je ovo samo pomocno resenje za obezbjedjenje kriticne sekcije koje
//radi na jednoprocesorskim sistemima; iako su semafori jedno od univerzalnih resenje,
//ovakav pristup moze da bude bolji pri resavanju domaceg ukoliko se ispis koristi za
//debagovanje ...
```

4. Na isti način izmeniti i ispis u f-ji `b()`

```
//begin lock
  lockFlag=0;
//end lock

  cout<<"u b() i = "<<i<<endl;

//begin unlock
                              //ovakav unlock je potreban da bi se proverilo
  lockFlag=1;                 //da li je zatrazeno preuzimanje u toku ispisa;
  if (zahtevana_promena_konteksta) {    //ako jeste onda se sada vrsi eksplicitno
                                        //preuzimanje;
     dispatch();
  }
//end unlock
```

5. Izmeniti prekidnu rutinu `timer()` tako što će se promena konteksta vršiti samo ako je postavljen `lockFlag`. Takođe, dodati ispis vrednosti brojača unutar prekidne rutine nakon čuvanja konteksta, a pre poziva raspoređivača.

```
void interrupt timer(){     // prekidna rutina
       if (!zahtevana_promena_konteksta) brojac--;
       if (brojac == 0 || zahtevana_promena_konteksta) {
              if (lockFlag){
            zahtevana_promena_konteksta = 0;  //ako je eventualno i bila zahtevana
                    //promena konteksta, sada ce se obaviti pa treba obrisati
                    //fleg za zahtev
                    asm {
                           // cuva sp
                           mov tsp, sp
                           mov tss, ss
                    }

                    running->sp = tsp;
                    running->ss = tss;

                    //ispis unutar prekidne rutine
                    lockFlag=0;
                    cout<<"Promena konteksta! Brojac= "<<brojac<<endl;  // ako neko
                           //vec vrsi ispis, lockFlag je vec na 0 i zato se nece ni
                           //poceti promena konteksta, pa samim tim se ne moze desiti
                           //ni ovaj ispis (ne moze se desiti paralelan ispis iz neke
                           //od niti i prekidne rutine)

                    asm cli; // u nekim slucajevima se desi da se prekidi omoguce
                             // unutar cout<<...
```

```
                        lockFlag=1;
                        //kraj ispisa

                        running= getNextPCBToExecute();    // Scheduler

                        tsp = running->sp;
                        tss = running->ss;

                        brojac = running->kvant;

                        asm {
                               mov sp, tsp   // restore sp
                               mov ss, tss
                        }

                }
                else zahtevana_promena_konteksta = 1;


        }

        // poziv stare prekidne rutine koja se
        // nalazila na 08h, a sad je na 60h
        // poziva se samo kada nije zahtevana promena
        // konteksta – tako se da se stara
        // rutina poziva samo kada je stvarno doslo do prekida
        if(!zahtevana_promena_konteksta) asm int 60h;

        //zahtevana_promena_konteksta = 0; - sada se menja u zavisnosti od lockFlag-a
}
```

6. Prevesti projekat pomoću naredbe Compile->BuildAll
7. Pokrenuti program pomoću naredbe Run->Run (ili Ctrl+F9)
   Kada će se vršiti promena konteksta i za koje vrednosti brojača?
   Zašto?

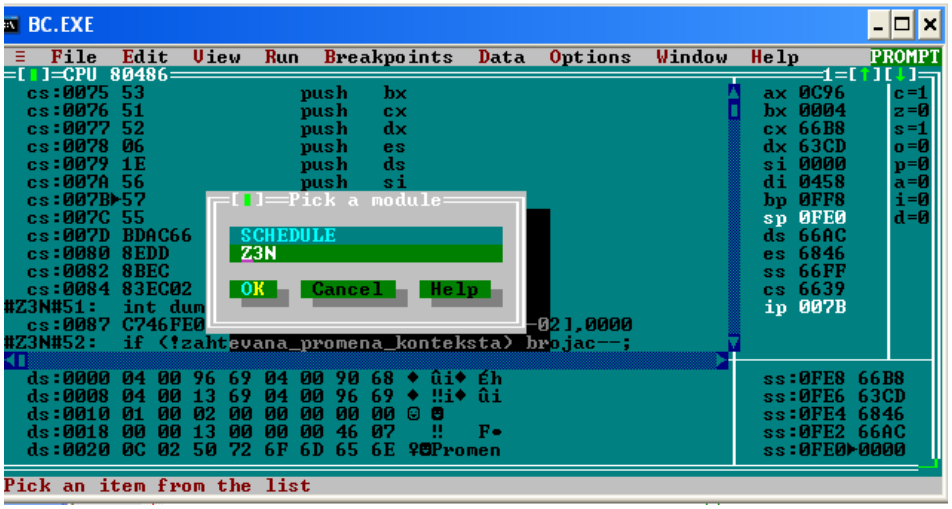
Problem alokacije lokalnih promenljivih

1. Dodati na pocetku prekidne rutine jednu lokalnu promenljivu

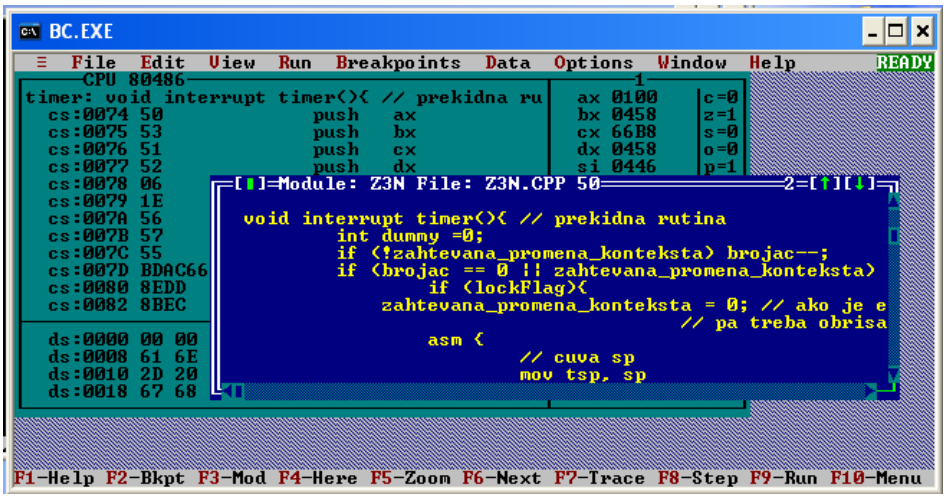
```
int dummy =0;
```

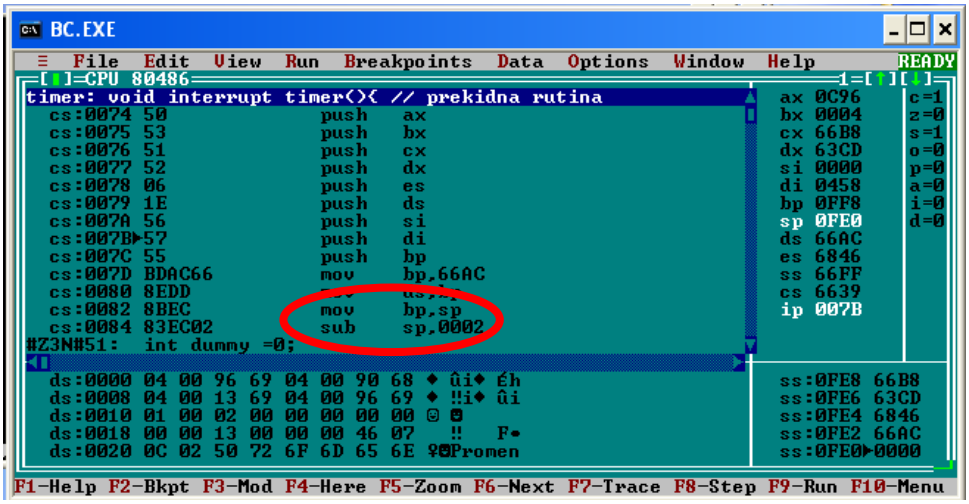
2. Prevesti program i pokrenuti ga (Umesto iz razvojnog okruzenja pokrenuti program direktno iz komandne linije `c:\> c:\bc31\bin\lab\output\LAB.EXE`).
   U kom trenutku će doći do kraha? Zašto?

3. Pokrenuti Turbo Debugger (SHIFT+F4)
4. Pritisnuti taster F3 (ili View->Module) i odabrati modul Z3N.



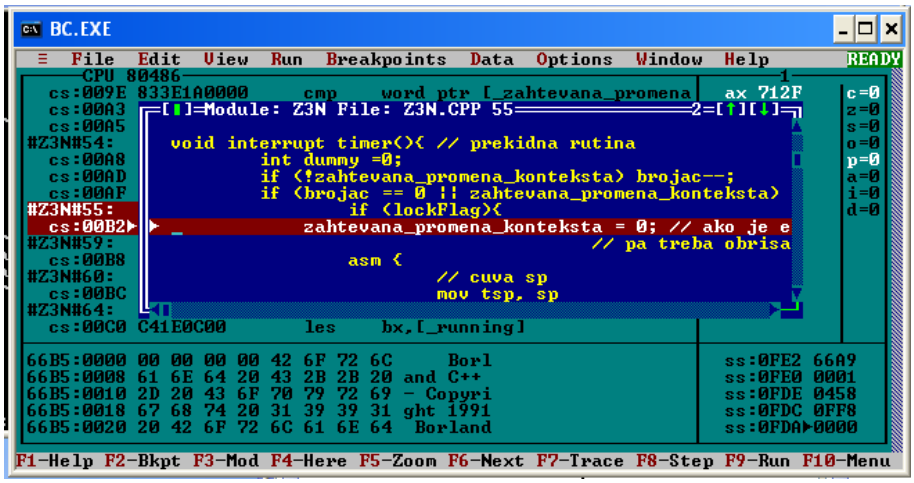
5. Postaviti kursor na pocetak prekindne rutine i pogledati generisan asemblerski kod sa View ->CPU





Koji registri se čuvaju? Zašto? Na koju lokaciju pokazuje registar SP nakon čuvanja konteksta? Zašto?

6. Postaviti Breakpoint u prekindnoj rutini na trenutak kada se započinje promena kontesta.
    - Postaviti kursor na datu liniju i pritisnuti taster F2 (ili Breakpoints->Toggle)



Zašto ne na početak prekidne rutine? Koji bi se onda trenuci detektovali u debbuger-u?

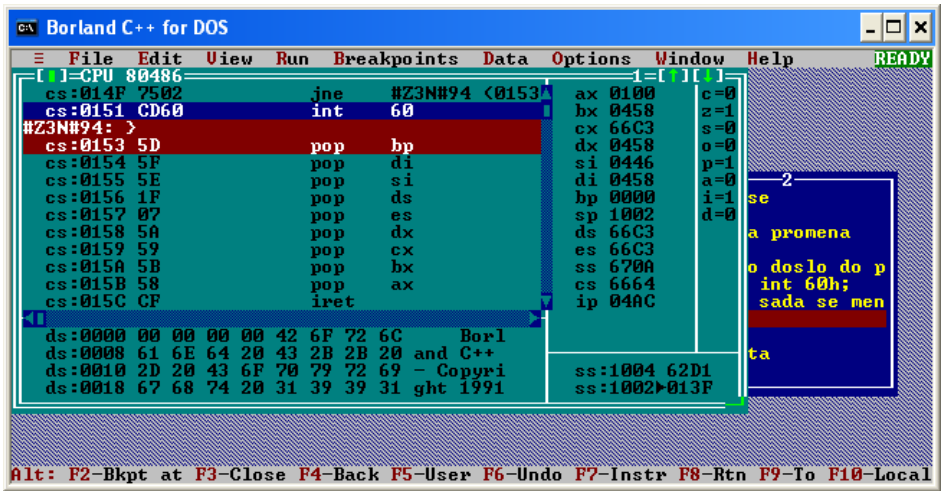
7. Pokrenuti program do zadate linije (Taster F9 ili Run->Run) do trenutka zaustavljanja (BreakPoint).
8. Iterirati kroz asemblerski kod (Taster F8) sve do izlaska iz prekidne rutine (instrukcija IRET) i posmatrati promene vrednosti registara; ili postaviti trenutak zaustavljanja na kraj prekidne rutine i zatim pritisnuti taster F9.

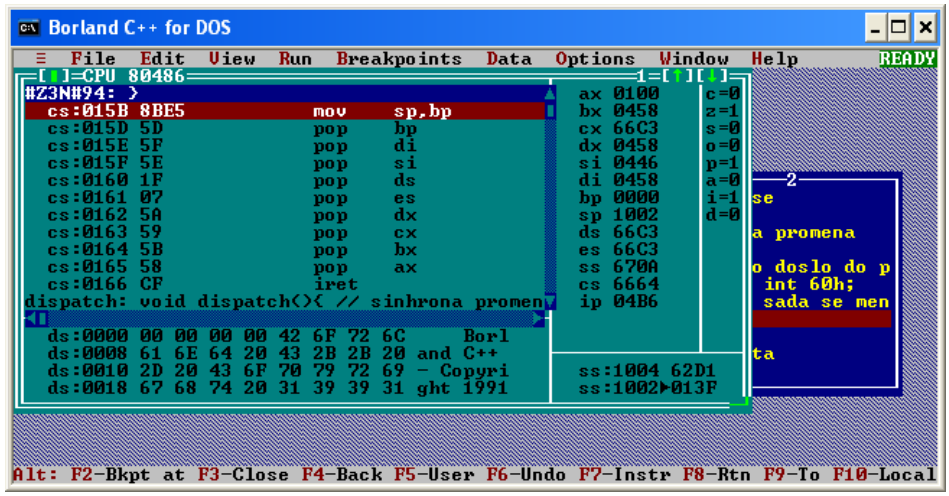
Gde će se vratiti tok izvršavanja programa? Da li je korektan povratak iz prekidne rutine? Napomena: Moguće je da će doći do kraha programa. Taj trenutak je nepredvidiv, jer zavisi od ostalih parametra na koje programsko okruženje ne utiče (npr. trenutni sadržaj proizvoljnih mem. lokacija)

9. Ugasiti program (ALT+X) ili ukoliko je potrebno ponovo pokrenuti okruženje.

Rešenje problema alokacije lokalnih promenljivih

Obratiti pažnju na razliku između generisanog koda za izlazak iz prekidne rutine u slučaju kada je isključena (zakomentarisana) promenljiva `dummy` (prva naredna slika) i slučaj kada je ona uključena (druga naredna slika)





U drugom slučaju prevodilac je generisao još jednu instrukciju `mov sp, bp` koja sa steka skida prostor rezervisan za ovu promenjivu. S obzirom da vrednost ovog registra nije adekvatno promenjena tako da ukazuje na stek niti čiji se kontekst restaurira (već pokazuje na staru vrednost) ovo će dovesti do nepredvidivog ponašanja. Iz tog razloga sada je neophodno prilikom čuvanju konteksta tekuće niti sačuvati i vrednost registra BP, a restaurirati novu vrednost ovog registra za nit koja dobija procesor.
Na ovaj nači će prilikom restauracije sačuvanog konteksta korektno biti dealociran prostor za pomoćne lokalne promenljive koje su prevodilac ili korisnik alocirali (SP = BP).

1. Dodati globaljnu promenljivu u kojoj će se privremeno čuvati vrednost registra BP

```
unsigned tbp;
```

2. Dodati polje *bp* u PCB

```
struct PCB{
	unsigned sp;
	unsigned ss;
	unsigned bp;
	unsigned zavrsio;
	int kvant;
};
```

3. Izmeniti prekidnu rutinu tako da se pri promeni konteksta čuva registar BP i restaurira njegova vrednost

```
void interrupt timer(){     // prekidna rutina
	int dummy =0;
	if (!zahtevana_promena_konteksta) brojac--;
	if (brojac == 0 || zahtevana_promena_konteksta) {
		if (lockFlag){
		zahtevana_promena_konteksta = 0;
		asm {
			// cuva sp i bp
			mov tsp, sp
			mov tss, ss
			mov tbp, bp
		}

		running->sp = tsp;
		running->ss = tss;
		running->bp = tbp; //izmena

		lockFlag=0;
		cout<<"Promena konteksta! Brojac: "<<brojac<<endl;

		asm cli;
		lockFlag=1;

		running= getNextPCBToExecute();    // Scheduler

		tsp = running->sp;
		tss = running->ss;
		tbp = running->bp;   //izmena

		brojac = running->kvant;

		asm {
			mov sp, tsp   // restore sp
			mov ss, tss
			mov bp, tbp  //izmena
		}

	 }
	 else zahtevana_promena_konteksta = 1;


	}

	// poziv stare prekidne rutine koja se
      // nalazila na 08h, a sad je na 60h
      // poziva se samo kada nije zahtevana promena
      // konteksta – tako se da se stara
      // rutina poziva samo kada je stvarno doslo do prekida
	if(!zahtevana_promena_konteksta) asm int 60h;

	//zahtevana_promena_konteksta = 0;
}
```

Napomena: na početku izvršavanja prekidne rutine će biti izvršena instrukcija `mov bp, sp` (pogledati generisan asemblerski kod), a prilikom restauracije konteksta pre instrukcije `pop`

`bp` će biti izvršena naredba `mov sp, bp` pa će prostor alociran za lokalne promenljive na ovaj način biti oslobođen.

4. U createThread inicijalizovati polje BP unutar PCB-a (ova vrednost ranije nije bila značajna i pri kreiranju početnog kontesta je bila proizvoljna).

```
void createProcess(PCB *newPCB, void (*body)()){
	unsigned* st1 = new unsigned[1024];

	st1[1023] =0x200;//setovan I fleg u
			// pocetnom PSW-u za nit
	st1[1022] = FP_SEG(body);
	st1[1021] = FP_OFF(body);

	newPCB->sp = FP_OFF(st1+1012);  //svi sacuvani registri
					//pri ulasku u interrupt
					//rutinu
	newPCB->ss = FP_SEG(st1+1012);
	newPCB->bp = FP_OFF(st1+1012); // Izmena: pocetni bp treba da pokazuje na
				       //poziciju na kojoj se cuva stara vrednost bp

	newPCB->zavrsio = 0;
}
```

5. Prevesti ponovo program i pokrenuti ga.
6. Da li će sada doći do kraha? Zašto?

Izmena načina raspoređivanja

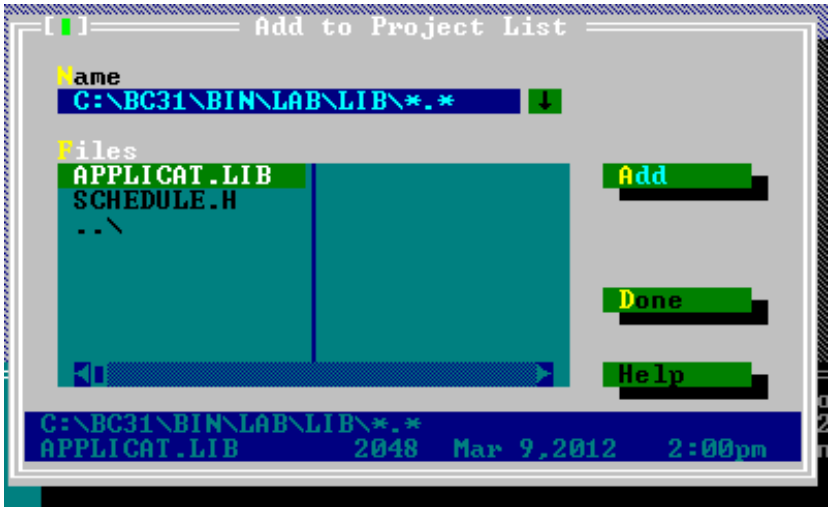
1. Dodati u zaglavlje direktivu `#include <schedule.h>`
2. Izmeniti struct PCB u class PCB

```
class PCB{
public:
	unsigned sp;
	unsigned ss;
	unsigned bp;
	unsigned zavrsio;
	int kvant;
};
```

3. Raspakovati `sheduler.zip` sa adrese http://os.etf.bg.ac.rs/OS1/projekat/ u direktorijum `lab\lib`

Napomena: Obavezno proveriti da li je odabran HUGE mem. model kao što je prikazano u uputstvu za podesavanje BC31 okruzenja. Takođe, proveriti da li su podešene putanje ka .lib i .h fajlovima!

4. Dodati `APLICAT.LIB` u projekat: (Project->Add Item...),
   - promeniti Name filter u `*.*` kako bi se videli svi fajlovi
   - odabrati `APLICAT.LIB`



5. Umesto trenutnog načina raspoređivanja postaviti sledeći kod:

```
//running= getNextPCBToExecute();  // Scheduler

if (! running->zavrsio) Scheduler::put((PCB *) running);
running=Scheduler::get();
```

6. Izmeniti f-ju `doSomething()`tako da prijavljuje 2 kreirane korisničke niti raspoređivaču. Takođe, dodeliti vremenski kvant main niti.

```
void doSomething(){
	lock
	p[1] = new PCB();
	createProcess(p[1],a);
	cout<<"napravio a"<<endl;
	p[1]->kvant = 40;
	Scheduler::put(p[1]); //prijavljena kao spremna

	p[2] = new PCB();
	createProcess(p[2],b);
	cout<<"napravio b"<<endl;
	p[2]->kvant = 20;
	Scheduler::put(p[2]); //prijavljena kao spremna

	p[0] = new PCB();
	p[0]->kvant = 20; //dodeljen kvant i main niti
	p[0]->zavrsio = 0;

	running = p[0];
	unlock

	…
```

7. Prevesti projekat pomoću naredbe Compile->BuildAll

8. Pokrenuti program pomoću naredbe Run->Run (ili Ctrl+F9)

Kada će se vršiti promena konteksta i za koje vrednosti brojača? Zašto?
Da li se sada prepliće izvršavanje i main niti? Zašto?

Samostalan rad

Postavljanje i restauracija ulaza u IVT

Definisati tip `pInterrupt`

```
typedef void interrupt (*pInterrupt)(...);
```

Gde je potrebno zapamtiti staru rutinu postaviti sledeći kod:
```
pInterrupt oldRoutine = getvect(entryNumber);
```

Izmeniti potpis prekidne rutine `timer` tako da zadovoljava potpis `pInterrupt`

```
void interrupt timer (...)
```

Gde je potrebno postaviti prekidnu rutinu u IVT postaviti sledeći kod:

```
setvect(entryNumber,timer);
```

Više detalja o `setvect, getvect` iz zaglavlja pogledati u uputstvu (Meni Help)

Definisanje makroa za ispis

Definisati sledeće makroe i pozvati ih na svakom mestu gde je to potrebno:

```
// Zabrana ispisa
#define lockCout lockFlag=0;

// Dozvola ispisa
#define unlockCout lockFlag=1;\
        if (zahtevana_promena_konteksta) {\
                    dispatch();\
}
```

Definisanje parametrizovanih makroa

Proces zamene poziva nekog makroa odgovarajućim sadržajem naziva se (makro)ekspanzija. Ukoliko su argumenti makroi, na mestima gde je upotrebljen parametar prosleđuje se ekspandovani sadržaj argumenta, osim u slučaju upotrebe operatora ##, kada se prvo ugradi originalni argument (linija 9 u primeru u nastavku), pa se tek u ekspandovanom sadržaju pokušava pronaći i ekspandovati neki makro (linija 21 u primeru u nastavku – poziv makro3(makro) se menja u makro1 koji se zatim ekspanduje).

Poziv makroa mora imati jedan blanko znak ispred i jedan blanko znak iza. Ukoliko je definicija makroa predugačka, moguće je prelomiti makro u više linija, s tim da se na kraju svake linije osim poslednje doda znak '\' (ovaj znak mora biti neposredno ispred znaka za prelazak u novi red).

"makroi.cpp"

```
---------------------------------------------------------------
#define naziv_makroa(param1, param2) param1param2 param1 param2 param1##param2 \
  param1##nastavak

naziv_makroa(argument1, argument2)

#define makro1 telo1
#define makro2 telo2

naziv_makroa(makro1, makro2)

#define novi_makro(p1,p2) naziv_makroa(p1, p2)

novi_makro(makro1,makro2)

#define makro3(param) param 1

neki tekst makro3(makro3(makro)) ostatak teksta

#define makro3(param) param##1

neki tekst makro3(makro) ostatak teksta
---------------------------------------------------------------
```

Opis generisanog izlaza " makroi.i" dat je u nastavku:

"makroi.i":

```
---------------------------------------------------------------
makroi.cpp 1:
makroi.cpp 2:
makroi.cpp 3:
makroi.cpp 4: param1param2 argument1 argument2 argument1argument2
argument1nastavak
makroi.cpp 5:
makroi.cpp 6:
makroi.cpp 7:
makroi.cpp 8:
makroi.cpp 9: param1param2 telo1 telo2 makro1makro2 makro1nastavak
makroi.cpp 10:
makroi.cpp 11:
makroi.cpp 12:
makroi.cpp 13: param1param2 telo1 telo2 telo1telo2 telo1nastavak
makroi.cpp 14:
makroi.cpp 15:
makroi.cpp 16:
makroi.cpp 17: neki tekst makro 1 1 ostatak teksta
makroi.cpp 18:
makroi.cpp 19:
makroi.cpp 20:
makroi.cpp 21: neki tekst telo1 ostatak teksta
makroi.cpp 22:
---------------------------------------------------------------
```