Elektrotehnički fakultet
Univerziteta u Beogradu
Katedra za računarsku tehniku i informatiku

Praktikum iz Operativnih sistema

- rešenja za 2006. godinu -

Verzija: 3.6.2009 18:39

1. Otvorena knjiga

1. **Program test se sastoji iz nekoliko** `.cpp` **i** `.h` **fajlova čije su zavisnosti date u priloženoj tabeli. Napraviti** `makefile` **koji će omogućiti optimalno pravljenje izvršnog fajla** `test` **(obezbediti da se kompajliraju samo one datoteke, koje su promenjene nakon poslednjeg kompajliranja).**

File	Includes
main.cpp	z.h x.h
x.cpp	x.h y.h
y.cpp	y.h
z.cpp	z.h

```
test: main.o x.o y.o z.o
     g++ -o test main.o x.o y.o z.o

y.o: y.cpp y.h
     g++ -c y.cpp -o y.o

x.o: x.cpp x.h y.h
     g++ -c x.cpp -o x.o

z.o: z.cpp z.h
     g++ -c z.cpp -o z.o

main.o: main.cpp z.h x.h
     g++ -c main.cpp

clean:
     rm -rf *.o *~ test
```

2. **Napisati** `bash shell script join` **koji kreira datoteku, čije je ime zadato prvim argumentom, a čiji sadržaj predstavlja konkatenaciju (nadovezivanje) sadržaja datoteka navedenih kao svi ostali argumenti skripta. Pretpostaviti da postoji skript** `x` **koji kopira sadržaj jedne datoteke zadate kao drugi argument, u datoteku zadatu kao prvi argument. Ako ne postoji datoteka, zadata kao prvi argument, onda skript** `x` **kreira tu datoteku, inače na već postojeću datoteku konkatenira sadržaj.**

```
#!/bin/bash
# Resenje treceg zadatka sa ispita
OUTPUT=$1
shift
cat $* >> $OUTPUT
```

3. **Šta tačno radi komanda** `cat`**?**

   cat ispisuje sadržaj datoteka, zadatih kao argumente, na standardni izlaz.

4. **Napisati skript** `x`**, opisan u pitanju 1, koristeći komandu** `cat`**.**

```
#!/bin/bash
cat $2 >> $1
```

5. **Napisati** `bash shell` **skript** `join` **koji kreira datoteku, čije je ime zadato prvim argumentom, a čiji sadržaj predstavlja konkatenaciju (nadovezivanje) sadržaja datoteka navedenih kao svi ostali argumenti skripta. Ako datoteka sa identičnim imenom već postoji treba je obrisati. Ako neka od datoteka koje treba spojiti ne postoji onda je samo preskočiti.**

```
#!/bin/bash
OUTPUT=$1
shift
if [ -e $OUTPUT ]; then
     rm -rf $OUTPUT
fi
cat $* >> $OUTPUT 2> /dev/null
```

6. **Dat je program** `encrypt` **koji čita tekstualni fajl sa standardnog ulaza i na standardni izlaz izbacuje njegovu šifrovanu reprezentaciju. Napisati** `bash shell` **skript** `join` **koji kreira datoteku, čije je ime zadato prvim argumentom, a čiji sadržaj predstavlja konkatenaciju (nadovezivanje) šifrovanog sadržaja datoteka navedenih kao svi ostali argumenti skripta. Ako datoteka sa identičnim imenom već postoji treba je obrisati. Ako neka od datoteka koje treba šifrovati ne postoji onda je samo preskočiti.**

```
#!/bin/bash
OUTPUT=$1
shift
if [ -e $OUTPUT ]; then
     rm -rf $OUTPUT
fi
cat $* | encrypt >> $OUTPUT 2> /dev/null
```

   6.1. **Napisati** `shell` **komandu koja pokreće skript** `join` **da šifruje fajlove** `file1`**,** `file2` **i** `file3` **i rezultat ispisuje u fajlove** `out1`**,** `out2` **i na standardni izlaz.**

```
join out1 file1 file2 file3; cat out1 | tee out2
```

7. **Koristeći** `POSIX threads` **napisati kod na jeziku** `C++` **za dve niti** `P` **i** `C`**, koje rade po sledećem scenariju.** `P` **proizvodi neke cele brojeve u slučajnim vremenskim trenucima i stavlja ih u bafer.** `C` **uzima po tri broja iz bafera i obrađuje ih koristeći funkciju** `int f(int a, int b, int c)`**, čiji rezultat ispisuje na standardni izlaz. Funkciju** `f` **ne treba implementirati.** `P` **i** `C` **se jedino mogu sinhronizovati preko bafera.** `C` **čita tri broja iz bafera nezavisno od toga ko ih je u bafer upisao. Kada se brojevi jednom pročitaju od strane jedne niti, niko ih više ne može čitati.**

   buffer_cp.cpp

```
#include <iostream>
#include <cstdlib>
using namespace std;

int f(int a, int b, int c)
{
     return a + b + c;
}

#include "Bafer.h"

void* funkcijaC(void *bafer)
{
  Bafer<int> *pBafer = (Bafer<int> *) bafer;
  int i = 4;
  while (i--)
  {
    /// @warning Nije najsrecnije resenje!!!
    while ((pBafer->get_size() % 3) != 0 || pBafer->get_size() == 0)
      sleep((unsigned)(rand()/(RAND_MAX+.1)*10));

    int b1, b2, b3;
    b1 = pBafer->get();
    b2 = pBafer->get();
    b3 = pBafer->get();

    cout << "Procitani sledeci brojevi iz bafera: " << b1 << ", " << b2
<< ", " << b3 << endl;
    cout << "Rezultat funkcije je: "<< f(b1, b2, b3) << endl;
  }

  pthread_exit(NULL);
}

void* funkcijaP(void *bafer)
{
  Bafer<int> *pBafer = (Bafer<int> *) bafer;
  for (int i = 0; i < 12; i++)
  {
    sleep((unsigned)(rand()/(RAND_MAX+.1)*10));
    int podatak = (int)(rand()/(RAND_MAX+.1)*20);
```

```
    pBafer->put(podatak);
    cout << "Ubacen broj " << podatak << " u bafer" << endl;
  }

  pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
  Bafer<int> *pBafer = new Bafer<int>;
  int status;
  pthread_t nitC, nitP;
  pthread_attr_t attr;

  srand(time(NULL));

  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

  pthread_create( &nitP, &attr, funkcijaP, (void *) pBafer);
  pthread_create( &nitC, &attr, funkcijaC, (void *) pBafer);
  pthread_attr_destroy(&attr);

  pthread_join( nitP, (void **)&status);
  pthread_join( nitC, (void **)&status);

  delete pBafer;
  pthread_exit(NULL);
}
```

Bafer.h

```
#ifndef __BAFER_H__
#define __BAFER_H__

#include <pthread.h>

#include <queue>
using namespace std;

/// @class Bafer
/// thread-safe klasa za potrebe resavanja zadatka
```

```
template<typename T>
class Bafer : public queue<T>
{
private:
     /// mutex koji obezbedjuje sinhronizaciju izmedju niti
     pthread_mutex_t m_mutex;

public:
     /// konstruktor inicijalizuje mutex
     Bafer() { pthread_mutex_init(&m_mutex, NULL); }

     /// destruktor unistava mutex
     ~Bafer() { pthread_mutex_destroy(&m_mutex); }

     /// uzima podatak iz bafera
     T get()
     {
          pthread_mutex_lock(&m_mutex);
          T podatak = this->front();
          this->pop();
          pthread_mutex_unlock(&m_mutex);
          return podatak;
     }

     /// stavlja podatak u bafer
     void put(T& podatak)
     {
          pthread_mutex_lock(&m_mutex);
          this->push(podatak);
          pthread_mutex_unlock(&m_mutex);
     }

     /// vraca trenutni broj podataka u baferu
     size_type get_size()
     {
          pthread_mutex_lock(&m_mutex);
          size_type velicina = this->size();
          pthread_mutex_unlock(&m_mutex);
          return velicina;
     }
```

```
};
#endif
```

8. **Napisati glavni program koji kreira** `n` **instanci niti tipa** `P`**, i** `m` **instanci niti tipa** `C`**, koje sve komuniciraju preko zajedničkog bafera.** `n` **i** `m` **se zadaju preko komandne linije kao argumenti programa. Obezbediti da se niti tipa** `P` **blokiraju kada je bafer pun, a da se niti tipa** `C` **blokiraju kada je bafer prazan.**

```
/// @todo Videti zadatke sa vežbi. Iskoristiti klase Thread i
BoundedBuffer.
```

9. **Koristeći** `POSIX threads` **napisati kod na jeziku** `C++` **za dve niti, proizvođača i potrošača, koji razmenjuju podatke preko dva bafera koji naizmenično zamenjuju uloge: dok proizvođač upisuje podatke u bafer** `A`**, potrošač uzima podatke iz bafera** `B`**; kada proizvođač napuni bafer** `A` **a potrošač isprazni bafer** `B`**, ova dva bafera zamenjuju uloge. Napisati kod proizvođača i potrošača uz sinhronizaciju pomoću semafora. Pretpostaviti da je bafer običan niz. Dozvoljeno je korišćenje koda sa vežbi. Napomena: kod sa vežbi ne treba prepisivati već samo treba navesti šta se koristi (Primer: "Izvodim iz klase** `X` **implementirane na vežbama" ili "Koristim funkciju** `f`**" ili ...).**

```
/// @todo
```

10. **Napisati na programskom jeziku Java program servera koji svaku primljenu poruku, od bilo kog klijenta, prosleđuje svim svojim klijentima (eho). Koristiti** `Socket`**.**

```
/// @todo videti primer servera sa vezbi
```

11. **Napisati na programskom jeziku Java kod servera koji može istovremeno da opslužuje više klijenata. Pretpostaviti da je dat metod** `void serveClient(Socket s)` **koji služi za opsluživanje pojedinačnog klijenta. Napraviti posebnu nit za opsluživanje svakog klijenta koji se prijavi. Implementirati metod** `void serveClient(Socket s)`**, koji će da radi eho, odnosno svaku poruku koju primi, samo prosleđuje nazad istom klijentu. Kada stigne poruka** `Kraj`**, metod treba da je pošalje nazad klijentu i da se završi.**

2. Zatvorena knjiga

1. **Šta je rezultat rada sledećih komandi:**

```
alias lm="ls -l"
```

Postavlja smenu simbola, tako da kad god korisnik otkuca **`lm`**, to će se pre izvršavanja smeniti sa **`ls -l`**.

```
pwd
```

Ispisuje tekući direktorijum.

```
ls -l
```

Lista sadržaj tekućeg direktorijuma, zajedno sa dodatnim informacijama o fajlovima: prava pristupa, vlasnika, grupu, tip fajla, ...

```
ls –la
```

Lista sadržaj tekućeg direktorijuma uključujući i skrivene fajlove (one čija imena počinju tačkom), zajedno sa dodatnim informacijama o fajlovima: prava pristupa, glasnika, grupu, tip fajla, ...

```
ls ~ -l
ls ~/. -l
```

Lista sadržaj **home** direktorijuma, zajedno sa dodatnim informacijama o fajlovima: prava pristupa, vlasnika, grupu, tip fajla, ...

```
mkfifo x1
```

Pravi **named pipe** pod imenom **x1** u tekućem direktorijumu.

```
rm –rf *
```

Bezuslovno briše kompletan sadržaj tekućeg direktorijuma.

```
kill -9 42456
```

Bezuslovno ubija proces čiji je **PID** jednak **42456**.

```
chmod +x *.sh
```

Za sve fajlove u tekućem direktorijumu, čije se ime završava sa **.sh**, postavlja pravo pristupa **x** (**execute**), za sve korisnike.

```
chmod 744 *
```

Za sve fajlove u tekućem direktorijumu postavlja da vlasnik može da ih čita/piše/izvršava, a grupni vlasnik i ostali korisnici samo mogu da čitaju.

```
chmod 660 child
```

Za fajl pod imenom **child** u tekućem direktorijumu postavlja prava pristupa takva da vlasnik i grupni vlasnik mogu da čitaju i da pišu, dok ostali korisnici nemaju nikakva prava pristupa do tog fajla.

```
chmod 664 child1 child2
```

Za fajlove pod imenom **child1** i **child2** u tekućem direktorijumu postavlja prava pristupa takva da vlasnik i grupni vlasnik mogu da čitaju i da pišu, dok ostali korisnici mogu samo da čitaju te fajlove.

```
chmod o=rw *
```

Za sve fajlove u tekućem direktorijumu postavlja da svi ostali korisnici (**other**), osim vlasnika i grupnog vlasnika, mogu samo da ih čitaju i da u njih pišu.

```
chmod o=r *
```

Za sve fajlove u tekućem direktorijumu postavlja da svi ostali korisnici (**other**), osim vlasnika i grupnog vlasnika, mogu samo da ih čitaju.

```
cat pera | grep -n galeb.etf.bg.ac.yu
```

U fajlu **pera** pronalazi sve redove koji sadrže string **galeb.etf.bg.ac.yu**, i štampa ih na standardni izlaz, zajedno sa rednim brojem reda.

```
cat /etc/passwd | grep milos
```

U fajlu **/etc/passwd** pronalazi sve redove koji sadrže string **milos** i štampa ih na standardni izlaz. Preciznije, ova će komanda ispisati osnovne podatke o korisniku **milos** koji se nalaze u pomenutom fajlu.

```
cat /etc/passwd | grep milos > log
```

U fajlu **/etc/passwd** pronalazi sve redove koji sadrže string **milos** i štampa ih u fajl pod imenom **log**. Preciznije, ova će komanda ispisati osnovne podatke o korisniku **milos** koji se nalaze u pomenutom fajlu.

```
clear;logout
```

Briše ekran i odjavljuje korisnika sa sistema.

```
ps –a
```

Lista sve procese koji se izvršavaju.

```
less /etc/passwd
```

Ispisuje sadržaj datoteke **/etc/passwd**, u kojoj se nalaze podaci o korisnicima sistema.

```
less /etc/inittab
```

Ispisuje sadržaj datoteke **/etc/inittab**, u kojoj se nalaze programi koje treba izvršiti prilikom startovanja sistema.

2. Šta je i čemu služi MPI (Message Passing Interface)?

MPI je skup API funkcija koje omogućavaju programerima da pišu efikasne paralelne programe koji komuniciraju razmenom poruka između procesa u cilju obavljanja celokupnog posla.

Citat sa http://www.lam-mpi.org/using/docs/: The Message Passing Interface (MPI), is a set of API functions enabling programmers to write high-performance parallel programs that pass messages between processes to make up an overall parallel job.

Napomena: MPI više nije u nastavnom programu IR2POS. Videti materijal za predmet Multiprocesorski sistemi.

3. Šta je `socket`?

socket je koncept koji obezbeđuje OS i koji služi za međuprocesnu komunikaciju, između procesa na udaljenim računarima.

3.1. Ko obezbeđuje ovaj koncept?

socket je koncept koji obezbeđuje OS i koji služi za međuprocesnu komunikaciju, između procesa na udaljenim računarima.

3.2. Demonstrirati kako se pravi `socket` na programskom jeziku Java.

Na Javi se **socket** pravi sa **new Socket(IPAdresa, port)**, gde su **IPAdresa** i **port**, adresa i port na kome osluškuje proces sa kojim se želi uspostaviti komnikacija.

4. Da li programi pisani na jezicima C++ i Java mogu da komuniciraju preko ovog koncepta? Ko to obezbeđuje?

Da. To obezbeđuje OS.

5. Da li programi na različitim (udaljenim) računarima mogu da komuniciraju preko ovog koncepta? Ko to obezbeđuje?

Da. To obezbeđuje OS.

6. Napisati na jeziku C++ deo koda koji upisuje poruku `"Polozicu POS"` **u imenovani cevovod pod nazivom** `Slavina`**.**

```
#include <fstream>
#include <sys/stat.h>

int main()
{
    __mode_t nacin_pristupa = 0644;
    int fd = mkfifo("Slavina", nacin_pristupa);

    std::fstream slavina("Slavina", std::ios::out);
    slavina << "Polozicu POS!" << std::endl;
    slavina.close();

    return 0;
}
```

7. Napisati shell komandu koja će sadržaj cevovoda `Slavina` **ispisati na ekran.**

```
cat Slavina
```

8. Implementirati na jeziku C++ procese `A` **i** `B` **koji u beskonačnoj petlji konkurentno izvršavaju opisane operacije. Proces** `A` **čita dva broja pomoću funkcije** `int getData()`**, pročitane brojeve šalje procesu** `B` **i čeka rezultat od procesa** `B` **koji ispisuje na ekran. Proces** `B` **čeka dva broja od procesa** `A`**, obrađuje ih pomoću funkcije** `int process(int a, int b)`**; i šalje rezultat procesu** `A`**. Na raspolaganju su imenovani cevovodi** `cev0` **i** `cev1`**.**

Makefile

```
all: procesA procesB
    @mkfifo cev0
    @mkfifo cev1

procesA: procesA.cpp
    g++ -o procesA procesA.cpp

procesB: procesB.cpp
```

```
	g++ -o procesB procesB.cpp

clean:
	@rm -rf procesA procesB cev0 cev1
```

procesA.cpp

```
#include <fstream>
#include <iostream>
using namespace std;

int getData()
{
	int broj;
	cin >> broj;
	return broj;
}

int main()
{
	int broj1, broj2, rezultat;
	fstream cev0("cev0", ios::in);
	fstream cev1("cev1", ios::out);

	while(1)
	{
		cout << "Unesite dva broja" << endl;
		broj1 = getData();
		broj2 = getData();
		cout << "Saljem " << broj1 << " i " << broj2 << endl;
		cev1 << broj1 << endl << broj2 << endl;
		cout << "Poslao sam " << broj1 << " i " << broj2 << endl;
		cev0 >> rezultat;
		cout << "Primio sam " << rezultat << endl;
	}

	return 0;
}
```

procesB.cpp

```
#include <fstream>
#include <iostream>
```

```
using namespace std;

int process(int a, int b)
{
     return a + b;
}

int main()
{
     int broj1, broj2, rezultat;
     fstream cev0("cev0", ios::out);
     fstream cev1("cev1", ios::in);

     while(1)
     {
          cev1 >> broj1 >> broj2;
          cout << "Primio sam " << broj1 << " i " << broj2 << endl;
          rezultat = process(broj1, broj2);
          cev0 << rezultat << endl;
          cout << "Poslao sam " << rezultat << endl;
     }

     return 0;
}
```

9. Dati program `A` **čita brojeve sa standardnog ulaza (dokle god ih ima), obrađuje svaki od njih pomoću funkcije** `void FA(int)`**, i rezultate ispisuje na standardni izlaz. Napisati shell komandu koja će pročitati sve brojeve iz fajla** `in`**, na svakog od njih primeniti funkciju** `FA(FA(FA(int)))` **i rezultat upisati u fajl** `out`**.**

```
./A <in | ./A | ./A > out
```

10. Napisati `shell` **komandu koja će u pozadini da pokrene program** `A` **koji će da pročita sve brojeve iz cevovoda** `cev0`**, da ih obradi i rezultate ispiše u cevovod** `cev1`**.**

```
./A <cev0 >cev1
```

11. Napisati shell komandu koja će da upiše brojeve `3`**,** `4`**,** `5` **i** `6` **u cevovod** `cev0`**.**

```
echo 3 4 5 6 > cev0
```

12. Napisati shell komandu koja će da ispiše brojeve iz cevovoda `cev1` **na standardni izlaz.**

```
cat cev1
tail -F cev1
```