acs
applied computer science

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Paralelno računarstvo

Računarske vežbe

Letnji semestar 2020/2021.
Studijski program: Informacioni inženjering

Operativni sistemi

STL

Standard Template Library (STL)

- Deo standardne biblioteka C++ jezika
- STL se sastoji iz nekoliko delova:
  - Kontejneri – kolekcije elemenata
  - Iteratori – za pristup elementima kontejnera
  - Algoritmi – manipulacija podacima u kontejneru

Klase kontejnera

- Sekvencijalni

  vector, deque, list

- Asocijativni

  set, multiset, map, multimap

- Adapteri
  - stack – LIFO
  - queue – FIFO
  - priority_queue

Iteratori

- Karakteristike
  - Pokazuje na element u kolekciji (npr. Niz) *it
  - Iterira kroz elemente kolekcije ++it
- Razlikovati:
  - iterator
  - const_iterator
  - reverse_iterator
  - const_reverse_iterator

Zadatak - STL

Napraviti funkciju:

vector<int> min_n(const vector<int>& v, int n);

Ova funkcija vraća vektor koji sadrži n najmanjih elemenata iz vektora v.

Podrazumeva se: v.size()>=n

Niti

Od čega nastaju niti?

- Niti nastaju od funkcija
- Svaka nit po svom kreiranju počinje da izvršava funkciju koja je prosleđena konstuktoru niti

Pozdrav iz niti

```
#include <iostream>
#include <thread>

using namespace std;

void kod_niti() {                    // Funkcija niti.
   cout << "Pozdrav iz niti!" << endl;
}

int main() {
        thread t(kod_niti);
        t.join();                    // Poziv operacije join() na niti t.
```

Funkcija main

- Ulazna tačka (*entry point*) programa tj. početak korisničkog dela programa je funkcija **`main()`**
- Završetak korisničkog dela programa nastupa kada se završi funkcija **`main()`**
- Ovo **važi** kako za sekvencijalni tako i za konkurentni program!
- Kada se završi nit **`main()`** završava se ceo program (bez obzira na stanja ostalih niti)

Objekat klase **thread (nit)**

- Služi za stvaranje niti (toka izvršavanja).
- Kada se stvori nit, objekat klase thread je u stanju 'joinable' (pokazuje na tok izvršavanja). Ovo znači da je nit krenula da se izvršava.
- se prevodi iz stanja 'joinable' operacijama:
  - **`join()`**
  - **`detach()`**.
- Ukoliko se nit ne prevede iz stanja 'joinable' nekom od dve metode (**`join`** ili **`detach`**) dobiće se greška: ”***terminate called without an active exception. Core dumped*** ”.

Razlika između `join()` i `detach()`

- operacija **join**() blokira nit pozivaoca dok se nit na kojoj je operacija join pozvana ne završi
- *Koristi se kada nit main čeka rezultat rada niti koje je stvorio*
- Operacija **detach**() razdvaja nit pozivaoca od niti na kojoj je operacija detach pozvana, tako da nit pozivaoc ne čeka da se nit na kojoj je operacija detach pozvana završi
- *Koristi se kada postoje ciklične niti (daemon) koje izvšavaju neku funkciju u beskonačnoj petlji. Nit main tada uz pomoć detach može da završi rad, a u suprotnom bi čekala beskonačno*

Pokretanje više niti

```cpp
#include <thread>
#include <iostream>

using namespace std;

void f(int rbr) {    //funkcija niti
   cout << rbr;
}

const int BROJ_NITI = 10;

int main() {
   thread t[BROJ_NITI];              //Deklaracija niza niti.
   for(int i=0; i<BROJ_NITI; ++i)
      t[i] = thread(f, i);
   for(int i=0; i<BROJ_NITI; ++i)  //Main ceka sve niti da se zavrse.
                                   //Mora u odvojenoj petlji jer bi u suprotnom bio sekvencijalni program.
      t[i].join();
   cout << endl;
}
```

Zadatak - Suma vektora N niti

- Napraviti konkurentni program koji izračunava sumu svih elemenata vektora, koristeći funkciju f():
- void f(ci pocetak, ci kraj, double& zbir);
- Funkcija f nalazi se u fajlu suma_vektora.h.

Međusobna isključivost

Međusobna isključivost

- Međusobno isključivo pristupanje **niti zajedničkim/deljenim** resursima je neophodno da bi se zaštitila konzistentnost tih resursa.

- Konzistentnost se narušava stihijskim pristupanjem deljenim resursima (dolaz do race condition-a).

Race-condition

Kada 2 (ili više) niti istovremeno pristupaju nezaštićenom resursu:

- One se trkaju (*race*), koja će pre da pristupi resursu.
- Otuda naziv: ***race-condition***.
- Posledica: rezultat izvršavanja neočekivano zavisi od redosleda događaja (pristupa).
- Primer Data race

Klasa **mutex**

- Primitiva koja obezbeđuje međusobnu isključivost, na engleskom *MUTual EXclusion* – otuda naziv.
- Nezaobilazan koncept u konkurentnom programiranju.
- Da bi se koristila klasa mutex potrebno je uključivanje zaglavlja **<mutex>**
- … nudi operacije:
  - **lock()**
  - **Unlock()**
- Treba **izbegavati** direktno korišćenje ovih operacija!
  - Nisu *exception safe*.
- Treba paziti kod korišćenja više muteks-a u programu. Moguće je izazivanje mrtve petlje (tzv. deadlock)!

Klasa **unique_lock**

- Služi za ”zaključavanje” mutex-a.
- Templejt klasa.
  - Templejt parametar za naše zadatke je **mutex**.
- Konstruktoru se kao argument prenosi referenca mutex-a koji treba zaključati.
- Obezbeđuje ”automatsko” oslobađanje muteksa (u destruktoru) kada objekat ove klase završi svoj životni vek.
- Omogućava privremeno otpuštanje propusnice (operacija **unlock**), što se može koristiti radi otpuštanja propusnice radi ispunjenja uslova čekanja ili povremenog otpuštanja propusnice radi sprečavanja ”izgladnjivanja” niti.

Zadatak – data race

- Kreirati globalnu celobrojnu promeljivu brojač. Nakon toga kreirati 2 funkcije inkrement i dekrement koje povećavaju i smanjuju dati brojač ITERACIJA puta. ITERACIJA je konstanta koja predstavlja proizvoljno velik broj (npr. 100.000.000).
- Kreirati jednu nit od funkcije inkrement i jednu nit od funkcije dekrement.
- Na kraju realizovati zaštitu pristupa brojaču uz pomoć klasa mutex i unique_lock. Primetiti koliko sada traje izvršavanje programa.

Deljena promenljiva

Dobra praksa je da se kao deljene promenljive koriste objekti klasa koje:

- enkapsuliraju atribute
- uključujući i objekte za sinhronizaciju (`mutex`)
- pristup obezbeđuju preko **ekskluzivnih i sinhronizovanih** metoda.

Zadatak - Banka

- Napraviti program koji simulira prenos novca unutar jedne banke, sa jednog bankovnog računa na drugi. Iznosi na računima u banci su predstavljeni datim nizom racuni. Taj niz sadrži onoliko elemenata koliko iznosi konstanta UKUPNO_RACUNA. Svaki element niza predstavlja jedan račun u banci.

- Funkcija main() popunjava sve račune u banci iznosom POCETNI_IZNOS. Potom ona kreira dve niti koje vrše transfer novca sa jednog računa na drugi. Ovaj prenos obavlja funkcija transfer().

- Data funkcija transfer() predstavlja telo niti koje vrše prenos novca. Funkcija 10 puta na slučajan način bira dva računa i neki iznos novca i poziva funkciju prebaci() koja skida novac sa prvog racuna i dodaje ga na drugi račun. Ovu funkciju je potrebno implementirati.

Uslovna sinhronizacija

Klasa **condition_variable**

- Primitiva za sinhronizaciju koja se koristi za blokiranje jedne ili više niti istovremeno. Niti su blokirane sve dok druga nit ne završi rad nad deljenim resursom i obavesti contion variable o završetku posla.
- Nit koja želi da pristupi deljenom resuru mora da:
  - dobije std::mutex
  - izvrši promene sve dok drži propusnicu
  - pozove notify_one ili notify_all nad objektom condition_variable.

Zadatak – **a** i **b** pre **c**

- Napraviti konkurentni program sa tri niti. Nit a ispisuje: Ja sam nit a. Nit b ispisuje: Ja sam nit b. Nit c ispisuje: Ja sam nit c.
- Obezbediti da se niti a i b, uvek izvrše pre niti c. Implementirati označene metode u klasi Koordinator.

Zadatak - parking

- Definisati klasu Parking koja modeluje Parking prostor kapaciteta jednog mesta.
- Ova klasa ima operacije: udji() i izadji().
- Automobili koji dolaze na Parking su predstavljeni programskim nitima. Za ulazak na Parking, automobil poziva metodu udji(). Za izlazak sa Parkinga, automobil poziva metodu izadji().
- Automobil se na Parkingu zadržava 3 sekunde. Pri ulasku, ukoliko je Parking zauzet, automobil mora da sačeka da se Parking oslobodi.
- Implementirati označene metode u klasi Parking u fajlu parking.h koristeći klasu mutex I klasu condition_variable.