Operativni Sistemi

VEŽBE 04 – MEĐUSOBNA ISKLJUČIVOST

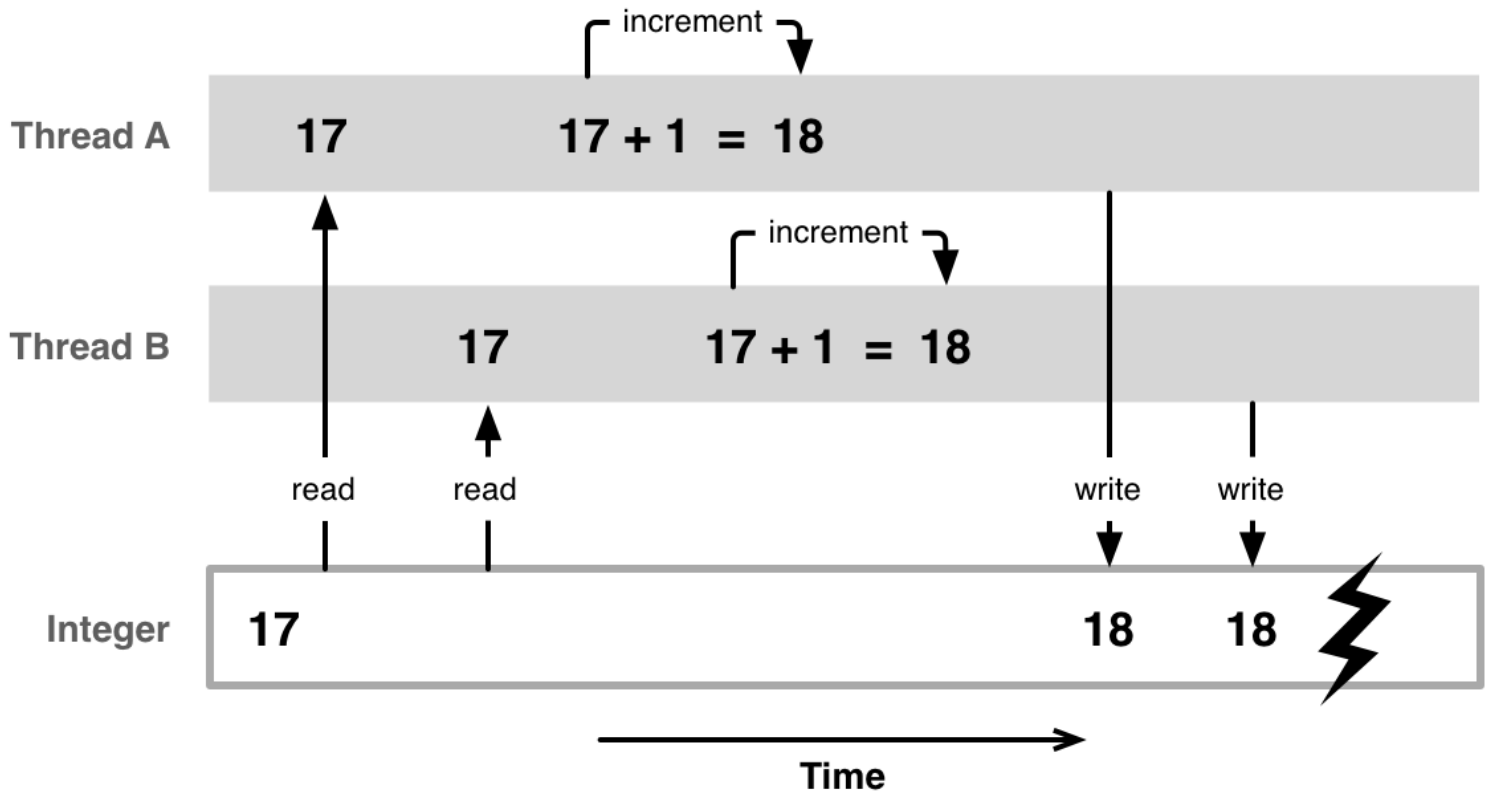
Međusobna isključivost

- Međusobno isključivo pristupanje **niti zajedničkim/deljenim** resursima je neophodno da bi se zaštitila konzistentnost tih resursa.
  - Šta je to **resurs** u (konkurentnom) programu?
  - Šta je to **zajednički** (ili **deljeni**) resurs?
  - Šta znači biti **konzistentan**?
  - 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), odnosno toga *ko je pobedio u trci.*
  - To znači da je naš deterministički program upravo postao ne-deterministički.

Race-condition



Rezultat: Nekako posle dva i++, promenljiva i se povećala za samo 1.

Ispravan program

- Da bi konkurentni program bio **ispravan** svi pristupi **deljenim** promenljivama moraju biti **ekskluzivni i (po potrebi) sinhronizovani**
- Ekskluzivnost (radi se na ovim vežbama) se postiže zabranom '**istovremenog**' pristupa deljenom resursu od strane više niti.
- Sinhronizacija (radi se na sledećim vežbama) pristupa će se baviti **redosledom pristupa** niti deljenom resursu.

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.

Kritična sekcija (KS)

- KS je deo koda u kojem se pristupa deljenom resursu
- KS treba da je što kraća
  - Jer u njoj dolazi do serijalizacije izvršavanja niti.
- Ulaz u i izlaz iz KS se štiti mehanizmom za sinhronizaciju (mutex)

Kritična sekcija (KS)

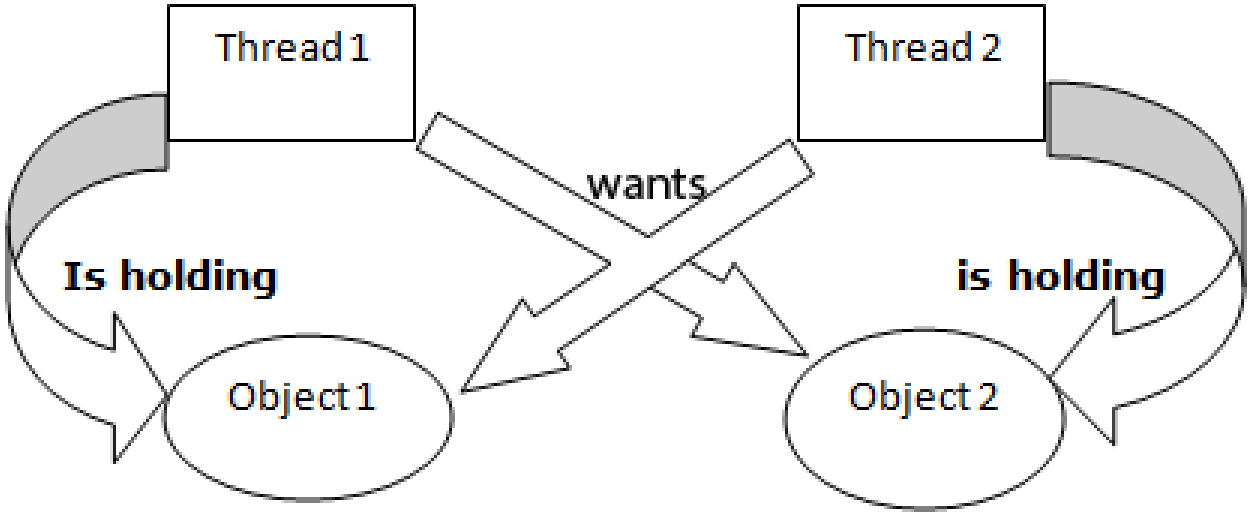
```
void visina() {
	int v;
	m.lock();
	cout << "Koliko ste visoki [cm]?" <<endl;
	cin >> v;
	cout << "Vasa visina je " << v << " cm." << endl;
	cout << endl;
	m.unlock();
}
```

Deadlock (mrtva petlja)

Mora se paziti kako se formiraju kritične sekcije

U slučaju korišćenja više od jednog muteksa u programu moguće je izazivanje mrtve petlje

Najbolja praksa je da ukoliko ima više muteksa u programu kritične sekcije tih muteksa budu razdvojene



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.

Ključna reč delete

- Novi standard pruža mogućnost da se kompajleru eksplicitno zabrani da stvori neku od metoda koju po defaultu može da automatski generiše (konstruktor ili operator dodele).

```
class mutex {
```

. . .

```
mutex(const mutex&) = delete;
```

. . .

```
}
```

- **Objekte klase mutex je zabranjeno kopirati**. Čak i da je to moguće program semantički ne bi bio ispravan jer bi niti zaključavale različite mutekse (kopije) umesto jedinstvenog muteksa (originala).