Sveučilište u Zagrebu
PMF – Matematički odsjek

Objektno programiranje (C++)

Predavanja 09 - Multithreading

Vinko Petričević

unique_lock

- Za razliku od guard_lock predloška koji odmah zaključa mutex, predložak unique_lock ima više mogućnosti
- Kao drugi parametar u konstruktoru može primiti std::defer_lock, te tada ne zaključava mutex
- Možemo pozivati lock/unlock/try_lock, … eksplicitno, a na destruktoru će otključati mutex, ako ga je zaključao
- Proći detaljnije primjere sa condition_variable

Objektno orijentirani pristup

- Svaki objekt bi se sam trebao brinuti o sebi
- Zbog toga bi dobro bilo da svaka klasa koju namjeravamo koristiti u višedretvenom okruženju ima u sebi ugrađene sinhronizacijske mehanizme koji osiguravaju da su podatci kojima se pristupa u ispravnom stanju
- To su općenito vrlo teški problemi i lako je dobiti greške
- Prva verzija paralelnog linuxa je imala jedan mutex za kompletni kernel, pa je na dvoprocesorskom sustavu rijetko radilo jednako brzo kao dva računala s jednim procesorom
- Dobro bi bilo da svaka klasa u sebi ima mutex koji osigurava da ih neki drugi thread ne može istovremeno mijenjati sadržaj koji čitamo
- Takav mutex onda obično deklariramo kao mutabile, da bi se mogao koristiti i na konstantnim objektima
- Međutim, i u takvim situacijama, lako možemo dobiti neočekivano ponašanje
- Primjer – stog
- Različiti mehanizmi će biti obrađeni na kolegiju Napredni C++, pa onda neću sada detaljnije ulaziti u to kako pojedinu strukturu efikasno implementirati

Deadlock

- Ukoliko je jedan thread zaključao mutex, a čeka na drugi mutex, a drugi thread je zaključao taj muteks, a čeka na prvi, takva situacija se zove deadlock
- Neoprezna implementacija može dovesti do deadlocka i u krajnje jednostavnim situacijama, kao što je prvi zadatak sa prethodnih vježbi
- Pisati siguran i efikasan paralelan kod je općenito vrlo teško
- Zbog toga nam često treba više mutexa, ali tada trebamo biti izuzetno oprezni da ne dobijemo deadlock

- Primjer swap – potencijalni problem da ako prvo zaključamo mutex jednog, pa onda drugog, može doći do deadlocka ukoliko neki drugi thread pokušava napraviti suprotno

- Ukoliko jedan thread više puta zaključa mutex, ponašanje nije definirano. U takvim situacijama možemo koristiti recursive_mutex, kojeg možemo više puta zaključati iz jednog threada, ali ga isto toliko puta moramo otključati

shared_mutex (c++ 17)

- Ponekad želimo da imamo više dvije razine isključivanja, npr. više threadova može čitati zaštićene podatke, a da ih samo jedan thread može mijenjati
- Takve situacije rješava klasa shared_mutex, a predložak shared_lock služi za zaključavanje samo za čitanje, dok ga obični lock zaključava i za čitanje i pisanje
- Primjer mapa

Static

- Statičke i globalne varijable će biti dijeljene između threadova
- Ako to ne želimo, možemo varijablu navesti kao thread_local, te će svaki thread imati svoju takvu varijablu

Alokacija memorije

- Prve verzije linuxa su bile zaključane samo jednim mutexom, pa su ponekad na dva procesora radile sporije nego na jednom
- Slična situacija se događa prilikom alociranja memorije, jer je u našem programu (barem sadašnja verzija g++-a) postavljen mutex na malloc/free koji se implicitno poziva iz new/delete

- Ukoliko koristimo klasu koju smo sami napravili, možemo preopteretiti operator new/delete

- Ukoliko koristimo standardne spremnike, možemo napraviti svoju klasu allocator koja će se brinuti o tome