Zadaća 3
Sistemsko programiranje

Jun, 2019

Sadržaj

1 Problem 1

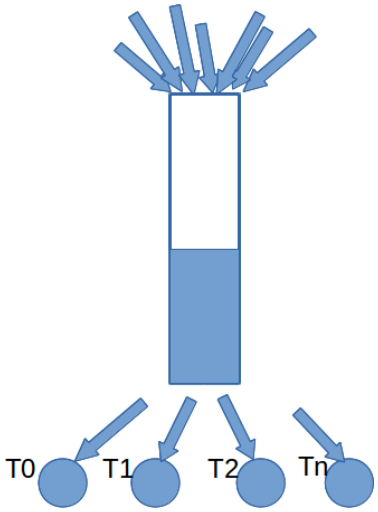
U fajlu `thread_pool.h`, u prilogu zadaće, data je pojednostavljena implementacija thread pool-a. U datoj implementaciji zadaci koji se mogu izvršavati u thread pool-u predstavljeni su `std::function<void()>` tipom. Zadaci se dodjeljuju thread pool-u pozivom metoda `async`. Da bi se u thread pool mogla poslati funkcija koja prima parametre, moguće je koristiti funckiju `std::bind`, koja kao parametre prima `Callable` objekat (bilo šta što podržava operator `()`), te argumente koji će biti proslijeđeni prilikom poziva navedenog operatora. Alternativno, umjesto `std::bind`, moguće je koristiti lambda izraze za sličnu namjenu. Ovo je prikazano u sljedećem primjeru korištenja thread pool-a. Alternativna varijanta prikazana je u zakomentiranom dijelu koda.

```
int main(int argc, char *argv[])
{
  thread_pool tp;

  for(int i = 0; i < 40; ++i) {
    tp.async(std::bind(call_fib, i));
  }

  // for(int i = 0; i < 40; ++i) {
  //   tp.async( [i](){ call_fib(i);} );
  // }

  std::this_thread::sleep_for(std::chrono::seconds{5});
  print_calculated_values();

  return 0;
}
```

Kompletan primjer korištenja thread pool-a dat je u fajlu `main.cpp`, u prilogu zadaće. Funkcije `add_calculated_value` i `print_calculated_values` su dodane da bi se moglo potvrditi da thread pool zaista izvršava zadatke koji su mu dodijeljeni.

Prilikom inicijalizacije thread pool-a specificira se broj niti. Ukoliko se ne specificira, koristi se onoliko niti koliko je hardverski podržano na datoj platformi. U datoj implementaciji, svi zadaci, koji mogu doći sa različitih niti, se smještaju u jedan queue (opisan klasom `task_queue`). Taj queue

je dijeljen između svih niti u thread pool-u, tako da je neophodno da bude sinhronizovan, za što su korišteni `std::mutex` i `std::condition_variable`. Niti iz pool-a će preuzimati zadatke iz queue-a sve dok queue nije prazan. Ukoliko je queue prazan, niti će blokirati, osim ako nije prethodno signalizirano da se queue zaustavi. U slučaju da je signalizirano da se queue zaustavi, prije samog zaustavljanja svi zadaci se moraju završiti. Opisana implementacija je simbolično prikazana na sljedećoj slici (1).



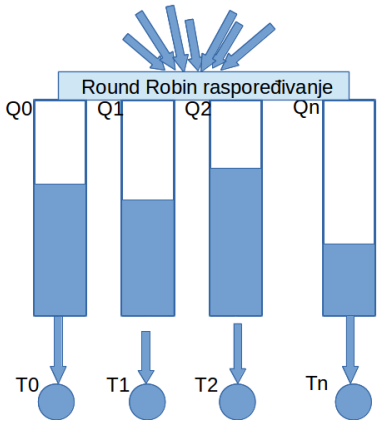
Slika 1: Inicijalna implementacija

Strelica predstavlja zadatak, pravougaonik queue, a krug nit. Dužina strelice simbolično predstavlja dužinu zadatka, a različiti smjerovi označavaju da zadaci mogu doći sa različitih niti.

U zadaći je potrebno implementirati sljedeće:

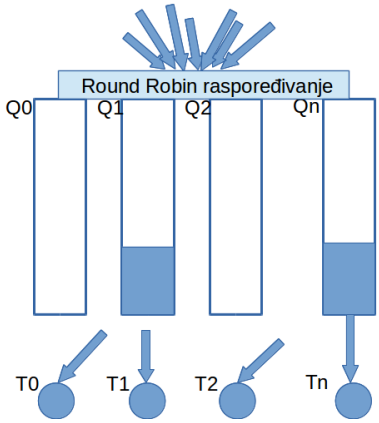
1. Modificirati opisanu implementaciju thread pool-a tako da se interno, umjesto jednog, koristi po jedan queue za svaku nit. Zadaci koji se dodjeljuju thread pool-u raspoređuju se po round-robin principu u različite queue-ove (2). Princip rada sa svakim pojedinačnim queue-on je identičan kao i u datoj implementaciji.

2. Dodatno na stavku 1, implementirati funkcionalnost krađe zadataka (*task stealing*, 3). Naime, ukoliko neka nit izvrši sve zadatke iz svoga queue-a, tada će pokušati "ukrasti" zadatak iz queue-a sljedeće niti, ukoliko postoji, u suprotnom će nastaviti tražiti zadatak u queue-u svake sljedeće niti, sve dok ne pronađe zadatak ili ustanovi da nema



Slika 2: Implementacija sa više internih queue-ova

zadataka ni u jednom queue-u. Ukoliko nit pronađe zadatak, preuzet će ga iz queue-a u kojem ga je pronašla, a ukoliko se ispostavi da nema zadataka koji čekaju na izvršenje, nit ide na spavanje, dok je ne probudi novi zadatak koji se doda u njen queue ili dok se thread pool ne zaustavi.



Slika 3: Implementacija sa *task stealing*-om