



Operativni Sistemi

II Semestar – 2022/23 – Vježbe

Sedmica 5

Handout za Vježbe

Agenda:

- <threads> i <atomic>
- semafori
- mutexi
- Linux za sistemsku administraciju

Kontakt:

Narcisa.hadzajlic@size.ba (B Grupa)


Adin.jahic2019@size.ba (A Grupa)

Include <thread>

Biblioteka <thread> uključena je u standardnu C++ biblioteku i koristi se za rad s nitima. Niti su korisni kada želimo da se dio koda izvršava u pozadini, bez blokiranja glavne niti. Niti se mogu koristiti za izvršavanje dugotrajnih zadataka, primjerice učitavanje velikih datoteka ili obradu velikih količina podataka.

Osnovne funkcije i klase biblioteke <thread> uključuju sljedeće:

- **std::thread** - Klasa koja predstavlja novu nit koji će izvršavati određeni kod.
- `std::thread::join()` - Metoda koja čeka da se nit završi s izvršavanjem prije nego što se program nastavi izvršavati.
- `std::thread::detach()` - Metoda koja omogućuje zadanoj niti da se izvršava u pozadini bez čekanja na završetak izvršavanja.
- `std::this_thread::get_id()` - Metoda koja vraća identifikator trenutne niti.
- `std::this_thread::sleep_for()` - Metoda koja blokira izvođenje trenutne niti za određeni vremenski period.
- `std::thread::hardware_concurrency()` - Metoda koja vraća broj niti koje podržava hardver.



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

void funkcija_za_novu_nit() {
    std::cout << "Pozdrav iz nove niti!" << std::endl;
}

int main() {
    std::thread nova_nit(funkcija_za_novu_nit);

    nova_nit.join();

    return 0;
}
```

Include <atomic>

Biblioteka <atomic> u C++ omogućava rad s atomičnim tipovima podataka. Atomički tipovi podataka su tipovi podataka koji se mogu čitati i pisati u jednoj operaciji, a to osigurava da se niti neće natjecati za pristup zajedničkim varijablama.

Ova biblioteka pruža nekoliko različitih klasa za rad s atomičnim tipovima podataka. Neke od tih klasa uključuju

- std::atomic,
- std::atomic_flag,
- std::atomic_int,
- std::atomic_long,
- std::atomic_bool, itd.

Sve ove klase pružaju metode i operatore koji se koriste za sigurno čitanje i pisanje atomičnih varijabli. Ove metode i operatori osiguravaju da se operacije izvode atomički, tj. u jednoj operaciji, čime se sprječava preklapanje i nejasnoće u slučaju kada se više niti natječe za pristup istoj varijabli.

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

std::atomic<int> brojac(0);

void povecaj_brojac() {
    for (int i = 0; i < 1000000; i++) {
        brojac++;
    }
}

int main() {
    std::thread nit1(povecaj_brojac);
    std::thread nit2(povecaj_brojac);

    nit1.join();
    nit2.join();

    std::cout << "Vrijednost brojaca: " << brojac << std::endl;

    return 0;
}
```

Mutexi

Klasična implementacija mutexa je nastala kao rezultat rješavanja „stanja trke“ u takozvanim kritičnim sekcijama procesa. Cilj je obezbjediti konkurentan pristup podacima te istovremeno održati njihov integritet. Mutexi su implementirani („u svom najprimitivnijem obliku“) sa 2 funkcije „LOCK“ i „UNLOCK“ na osnovu kojih se može kontrolisati pristup procesima tj ulasku u kritičnu sekciju.

```
#include <atomic>
#include <thread>

struct Mutex {
    std::atomic<bool> locked = false;

    void lock() {
        bool expected = false;
        while (!locked.compare_exchange_strong(expected, true, std::memory_order_acquire))
        {
            expected = false;
            std::this_thread::yield();
        }
    }

    void unlock() {
        locked.store(false, std::memory_order_release);
    }
};
```

Ovdje se koristi struktura Mutex, koja ima jednu jedinu varijablu locked koja označava da li je zaključavanje aktivno. lock() metoda pokušava dobiti pristup zaključavanju, koristeći compare_exchange_strong() metodu koja uzima dva argumenta: očekivani i željeni vrijednosti. Ako trenutna vrijednost locked varijable odgovara očekivanoj vrijednosti, compare_exchange_strong() će postaviti novu vrijednost na true i metoda će vratiti true. Inače, metoda će postaviti očekivanu vrijednost na trenutnu vrijednost locked varijable i vratiti false.

Ako compare_exchange_strong() metoda vrati false, to znači da neki drugi proces ili nit već ima zaključavanje, pa će se metoda yield() pozvati kako bi se drugim nitima dao pristup procesoru. Ovaj postupak će se nastaviti sve dok compare_exchange_strong() metoda ne vrati true.

unlock() metoda jednostavno postavlja vrijednost locked varijable na false, što označava da je zaključavanje završeno i da druge niti mogu pristupiti zaštićenom resursu.

Ova implementacija koristi memory_order_acquire za compare_exchange_strong() u lock() metodi, kako bi osigurala da se sve promjene u memoriji učitaju prije nego što se blokira kritični dio koda. Također koristi memory_order_release za store() metodu u unlock() metodi, kako bi osigurala da se sve promjene u memoriji izvrše prije nego što se otključa kritični dio koda.

Semafori

Semafori su alat koji se koristi u operativnim sistemima i paralelnom programiranju kako bi se sinkronizirali procesi i threadovi. Semafor je vrsta varijable koja se koristi za upravljanje pristupom zajedničkim resursima.

Semafor drži broj dozvola, koje threadovi moraju dobiti prije nego što mogu pristupiti zajedničkom resursu. Kada se thread pokušava pristupiti zajedničkom resursu, on prvo pokušava dobiti dozvolu od semafora. Ako su sve dozvole već u upotrebi, thread čeka dok se jedna od njih ne oslobodi. Kada se thread završi s upotrebom zajedničkog resursa, on oslobađa dozvolu tako da je ponovno dostupna drugim threadovima.

Semafori su korisni kada se radi o kritičnim sekcijama, gdje više threadova želi pristupiti istom zajedničkom resursu. Korištenje semafora osigurava da samo jedan thread može pristupiti kritičnoj sekciji u isto vrijeme, sprječavajući probleme poput utrke za uvjetnim varijablama i stanja spavanja.

```
#include <iostream>
#include <thread>
#include <atomic>
#include <vector>

struct Semaphore {
    std::atomic<int> count;

    Semaphore(int initial_count = 1) : count(initial_count) {}

    void wait() {
        int expected = 1;
        while (!count.compare_exchange_strong(expected, 0))
        {
            expected = 1;
            std::this_thread::yield();
        }
    }

    void signal() {
        count.store(1);
    }
};

locked.store(false, std::memory_order_release);
};
```

Ovaj primjer koristi Semaphore strukturu koja ima dva metoda, `wait()` i `signal()`, koje čekaju i signaliziraju semafor. `wait()` metoda smanjuje vrijednost brojača semafora i čeka dok se ne poveća na 1, a `signal()` metoda povećava vrijednost brojača za 1.

Linux za sistemsku administraciju

- Sistemski Logovi
- Komande za manipulaciju procesa.
- Komande za procese
- Administracija grupa i korisnika
- Varijable okruženja
- Monitoranje korisnika

Sistemski Logovi

Logovi se nalaze na putanji „**/var/log/**“. Čitanje logova u Linuxu je važan postupak za praćenje različitih događaja i problema na sistemu. Postoje različiti logovi koje Linux sustav generiše, a neki od najčešćih su sljedeći:

- Syslog - glavni dnevnik događaja na sistemu koji sadrži sve važne događaje vezane uz sistem, uključujući kernel poruke, poruke aplikacija, poruke sistema sigurnosti.
- Auth.log - dnevnik koji sadrži informacije o prijavi i autentikaciji korisnika na sistemu, uključujući i neuspjele prijave.
- Apache access i error logovi - dnevnik koji sadrži informacije o pristupu web browseru, uključujući korisničke zahtjeve i greške.
- Mail.log - dnevnik koji sadrži informacije o e-pošti, uključujući slanje, primanje i greške.
- Cron.log - dnevnik koji sadrži informacije o agendama na sistemu, uključujući i pokretanje skripti i programskih alata na rasporedu.

Za čitanje logova u Linuxu možete koristiti naredbu "**tail**", koja prikazuje zadnjih nekoliko linija dnevnika. Primjerice, "**tail -f /var/log/syslog**" prikazat će zadnjih 10 linija syslog datoteke i nastaviti s prikazivanjem novih linija koje se dodaju u dnevnik dok se ne prekine naredba (**CTRL+C**). Ako želite pregledati cijelu datoteku, možete koristiti naredbu "**less**", koja omogućuje pomicanje naprijed i natrag kroz dnevnik.

Komande za održavanje sistema

- Shutdown
- Reboot
- Halt
- Init

Komande za manipulaciju procesa

- Systemctl
- Ps
- Top
- Kill

Administracija grupa i korisnika

- Useradd
- Groupadd
- Userdel
- Groupdel
- Usermod

Specifčne lokacije

1. /etc/passwd
2. /etc/group
3. /etc/shadow

Varijable okruženja

Environment variables (varijable okruženja) su vrijednosti koje su dostupne u okruženju operativnog sistema i koriste se za razne potrebe aplikacija i sistema. One sadrže podatke kao što su putanja do datoteka, korisnička imena, postavke sistema, i drugo.

U Linuxu, neke od uobičajenih okružnih varijabli uključuju:

- HOME - putanja do mape matičnog direktorija trenutno prijavljenog korisnika.
- PATH - popis mapa u kojima operativni sustav traži izvršne datoteke
- USER - korisničko ime trenutno prijavljenog korisnika.
- SHELL - putanja do ljuske koju koristi trenutni korisnik.
- LANG - postavka jezika sustava.

Naredba "env" prikazuje sve okružne varijable na vašem sistemu.

Primjer: "env | grep USER" će izlistati sve varijable okruženja koje sadrže riječ "USER" u njihovom imenu.

Također, naredba "export" da bi se postavila vrijednost okružne varijable.

Primjer: "export MY_VAR=my_value" će postaviti okružnu varijablu "MY_VAR" na vrijednost "my_value". Ova varijabla će biti dostupna u svim sljedećim procesima koji se pokrenu u trenutnom okruženju.

Lokacija ovih varijabli se nalazi u /etc/.bashrc

**Za sve eventualne primjedbe, komentare, sugestije obratiti se na mail:
adin.jahic2019@size.ba**