Ilham Nur Romdoni, M0520038

Synchronized Threads

Pemrograman Berorientasi Objek
S1 Informatika UNS

Pada suatu sistem multithreading, sinkronisasi adalah suatu proses pengendalian akses dari sumber daya terbagi pakai (shared resource) oleh banyak thread sedemikian sehingga hanya satu thread yang dapat mengakses suatu sumber daya tertentu pada satu waktu.

Pada modul kali ini akan dipraktikkan membuat thread tersinkronisasi.

C++

Tulis code berikut

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

using namespace std;

void func() {
    cout << "entered thread " << this_thread::get_id() << endl;
    this_thread::sleep_for(5s);
    cout << "leaving thread " << this_thread::get_id() << endl;
}

int main() {
    thread t1(func);
    thread t2(func);
    thread t3(func);

    t1.join();
    t2.join();
    t3.join();

    return 0;
}
```

Simpan file dengan nama Synchro.cpp, jelaskan output yang didapat dari source code tersebut!

```
ilham@ilham-VirtualBox:~/Documents$ g++ Synchro.cpp -o Synchro -pthread
ilham@ilham-VirtualBox:~/Documents$ ./Synchro
entered thread 140094085027392
entered thread 140094093420096
entered thread 140094076634688
leaving thread 140094093420096
leaving thread 140094085027392
leaving thread 140094076634688
```

Semua angka yang dimulai dengan 140 pada output adalah ID thread. Code di atas membuat tiga thread, yang ID thread uniknya ditetapkan masing-masing oleh library pthread. Memunculkan kalimat `"entered thread "` dan ID thread langsung dari ketiga thread. Kemudian `this_thread::sleep_for(5s)` akan mengistirahatkan sebentar thread selama 5 detik. Lalu menampilkan kalimat `"leaving thread "` dan ID thread dari ketiga thread. Code yang ditulis berjalan dalam konteks thread. Program sebelumnya memiliki total empat thread, termasuk main thread. Code memblokir main thread dengan `join()` untuk memaksanya menunggu tiga thread lainnya menyelesaikan tugas mereka terlebih dahulu, main thread akan gagal keluar sebelum ketiganya. Saat main thread keluar, aplikasi juga keluar, yang berakhir dengan menghancurkan thread yang baru dibuat sebelum waktunya.

Selanjutnya kita modifikasi source code dengan menambahkan library mutex untuk melakukan lock dan unlock resource, source code menjadi sebagai berikut

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

using namespace std;

mutex g_lock;

void func() {
    g_lock.lock();

    cout << "entered thread " << this_thread::get_id() << endl;
    this_thread::sleep_for(5s);
    cout << "leaving thread " << this_thread::get_id() << endl;
    g_lock.unlock();
}

int main() {
    thread t1(func);
    thread t2(func);
    thread t3(func);

    t1.join();
    t2.join();
    t3.join();

    return 0;
}
```

Jelaskan output yang didapat dari source code tersebut, apa yang berbeda dari hasil compile source code sebelumnya?

```
ilham@ilham-VirtualBox:~/Documents$ g++ Synchro#2.cpp -o Synchro#2 -lpthread
ilham@ilham-VirtualBox:~/Documents$ ./Synchro#2
entered thread 140307614832192
leaving thread 140307614832192
entered thread 140307606439488
leaving thread 140307606439488
entered thread 140307623224896
leaving thread 140307623224896
```

Perbedaannya terletak pada saat menampilkan kalimat "`entered thread `" dan ID thread dari satu thread. Kemudian `this_thread::sleep_for(5s)` akan mengistirahatkan sebentar thread selama 5 detik. Lalu menampilkan kalimat "`leaving thread `" dan ID thread dari thread tersebut. Thread selanjutnya menunggu giliran yang menyebabkan tampilan entered, sleep dan leaving dilakukan secara bergiliran. Thread akan diistirahatkan 5 detik setelah menampilkan kalimat entered untuk tiap thread di mana total istirahat program menjadi 15 detik. Berbeda dengan sebelumnya yang hanya 5 detik. Hal ini terjadi karena penambahan library mutex. Mutex lock adalah kunci eksklusif yang hanya mengizinkan satu thread untuk mengoperasikan critical section code dalam batas proses yang sama. Sampai thread yang telah memperoleh kunci melepaskan mutex lock, semua thread lainnya harus menunggu gilirannya. Setelah thread memperoleh mutex lock, thread dapat mengakses resource bersama dengan aman. Penguncian dilakukan oleh baris `g_lock.lock()` dan pelepasan dilakukan oleh baris `g_lock.unlock()`. Critical section code berada di antara kedua baris tersebut. Semua thread disinkronkan dengan mutex yang sama.