Multithreading în C++

# Introducere

Multithreading-ul este o tehnică de programare prin care mai multe fire de execuție (threads) sunt create în cadrul unui program pentru a rula în paralel, permițând astfel executarea simultană a mai multor sarcini. Acest lucru poate îmbunătăți performanța aplicațiilor, în special pe sistemele cu mai multe nuclee de procesor.

# Declarația unui thread

În C++, un thread poate fi declarat folosind clasa `std::thread`. Un thread poate fi creat pentru a executa o funcție globală, o metodă dintr-o clasă sau chiar o funcție lambda. După ce un thread este creat, acesta poate rula în paralel cu alte thread-uri până la finalizarea sarcinii.

# Exemplul 1: Utilizarea funcțiilor globale

În acest exemplu, sunt definite două funcții globale `ThreadFunction1` și `ThreadFunction2`. Fiecare dintre ele execută un ciclu de la un anumit început până la un sfârșit, afișând mesaje simple pe ecran. Apoi, sunt create două thread-uri care rulează aceste funcții în paralel.

void ThreadFunction1(int start, int end) {  
 for (int i = start; i <= end; i++) {  
 cout << "Hello \n";  
 }  
 }  
   
 void ThreadFunction2(int start, int end) {  
 for (int i = start; i <= end; i++) {  
 cout << "Hi \n";  
 }  
 }  
   
 int main() {  
 thread funcTest1(ThreadFunction1, 1, 25000);  
 thread funcTest2(ThreadFunction2, 25001, 50000);  
 funcTest1.join();  
 funcTest2.join();  
 }

În acest cod, `funcTest1` și `funcTest2` sunt thread-urile care rulează funcțiile globale. Metoda `join()` este folosită pentru a aștepta ca fiecare thread să își termine execuția înainte de a continua programul.

# Exemplul 2: Utilizarea metodelor dintr-o clasă

În acest exemplu, thread-urile sunt folosite pentru a apela metode dintr-o clasă. Metodele `ThreadFunction1` și `ThreadFunction2` fac parte din clasa `test`. În interiorul `main`, sunt create două thread-uri care apelează aceste metode folosind un obiect instanțiat al clasei.

class test {  
 public:  
 void ThreadFunction1(int start, int end) {  
 for (int i = start; i <= end; i++) {  
 cout << "Hello \n";  
 }  
 }  
   
 void ThreadFunction2(int start, int end) {  
 for (int i = start; i <= end; i++) {  
 cout << "Hi \n";  
 }  
 }  
 };  
   
 int main() {  
 test obj;  
 thread t1(&test::ThreadFunction1, &obj, 1, 50);  
 thread t2(&test::ThreadFunction2, &obj, 50, 100);  
 t1.join();  
 t2.join();  
 }

În acest cod, `t1` și `t2` sunt thread-uri care rulează metodele clasei `test`. Metodele sunt apelate folosind un pointer la funcția membru și un obiect instanțiat `obj` al clasei.

# Exemplul 3: Funcții lambda

În acest exemplu, thread-urile sunt create folosind funcții lambda. Lambda-urile sunt funcții anonime care pot fi definite direct în locul în care sunt utilizate.

int main() {  
 thread th1([] {std::cout << "Ilie "; });  
 thread th2([] {std::cout << "Ioan"; });  
 th1.join();  
 th2.join();  
 }

Acest cod creează două thread-uri `th1` și `th2`, fiecare dintre ele executând câte o funcție lambda. Fiecare lambda afișează un nume diferit pe ecran.