시스템 프로그래밍 숙제 보고서

2013-11392 김지현

OS의 세마포어와, 직접 구현한 필터락 알고리즘을 이용해 2, 4, 8개의 스레드에서 +1 연산을 1000만번 수행한 결과, 퍼포먼스는 아래와 같았다

OS 세마포어를 쓴 코드는 다음과 같다 (C++)

|  |
| --- |
| #include <iostream>  #include <vector>  #include <thread>  #include <mutex>  #include <unistd.h>  using namespace std;  static int var = 0;  int main(int argc, char \*argv[]) {  size\_t n = 8;  int opt;  while((opt = getopt(argc, argv, "n:")) != -1) {  n = atoi(optarg);  }  cout << "Calculating with " << n << " threads... " << flush;  mutex mutex;  auto threads = vector<thread>();  for (size\_t i = 0; i < n; ++i) {  threads.emplace\_back([&](){  for (size\_t \_ = 0; \_ < 10000000/n; ++\_) {  mutex.lock();  var += 1;  mutex.unlock();  }  });  }  for (auto& thread : threads) {  thread.join();  }  cout << "Done!" << endl;  cout << "var : " << var << endl;  return 0;  } |

직접 구현한 필터락 알고리즘 코드는 아래와 같다 (Rust)

|  |
| --- |
| use std::thread;  use std::thread::yield\_now;  use std::sync::atomic::fence;  use std::sync::atomic::Ordering::\*;  use std::marker::Sync;  static mut var: i32 = 0;  fn main() {  let n = 8;  let filter = Filter::new(n);  let addr = &filter as \*const Filter as usize;  let mut threads = Vec::new();  for id in 0..n {  threads.push(thread::spawn(move || {  let filter = unsafe { &mut \*(addr as \*mut Filter) };  for \_ in 0..1000\_0000/n {  filter.lock(id);  unsafe { var += 1 }  filter.unlock(id);  }  }))  }  for thread in threads.into\_iter() {  let \_ = thread.join();  }  unsafe {  println!("var : {}", var);  }  }  /// Slow starvation-free binary semaphore  struct Filter {  thread\_count: usize,  levels: Vec<usize>,  victims: Vec<usize>  }  impl Filter {  fn new(thread\_count: usize) -> Self {  Filter {  thread\_count: thread\_count,  levels: vec![0; thread\_count],  victims: vec![0; thread\_count]  }  }  fn lock(&mut self, id: usize) {  for level in 1..self.thread\_count {  self.levels[id] = level;  self.victims[level] = id;  fence(SeqCst);  let cond1 = || {  for other in 0..self.thread\_count {  if id == other { continue; }  fence(SeqCst);  if self.levels[other] >= level { return true; }  }  false  };  let cond2 = || { fence(SeqCst); self.victims[level] == id };  while cond1() && cond2() { yield\_now(); }  }  }  fn unlock(&mut self, id: usize) {  self.levels[id] = 0;  fence(SeqCst);  }  }  unsafe impl Sync for Filter {} |