JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 10 (2019) 49-53
Wysłane: 2018-11-22
Przyjęte: 2018-12-02

Programowanie wielowątkowe w językach strukturalnych i obiektowych

Mateusz Łukasz Wiśniewski
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono sposoby programowania wielowątkowego w wybranych językach programowania takich jak: C podejście strukturalne oraz obiektowe C++ przy użyciu bibliotek Posix Threads dla języka C oraz bibliotekę Boost dla języka C++. Opisano również charakterystykę wybranych bibliotek. Przedstawiono przykładowe rozwiązania typowych problemów programistycznych wykorzystujących wątki. Starano odpowiedzieć na pytanie czym kierować się przy wyborze języka do nauki programowania wielowątkowego oraz programowania w ogólnym tego słowa znaczeniu.

Słowa kluczowe: programowanie; wielowątkowość; pthreads; boost

Adres e-mail: wmmateusz@gmail.com

Multithreaded programming in structural and object-oriented languages

Mateusz Łukasz Wiśniewski
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents multithreaded programming in selected programming languages such as: C structural and C ++ object oriented approach using Posix Threads libraries for C language and Boost library for C ++ language. The characteristics of selected libraries are also described. Examples of typical programming problems using threads are presented. An effort was made to answer the question of what to do when choosing a language for learning multithread programming and programming in the general meaning of this word.

Keywords: programming; multithreading; pthreads; boost

E-mail address: wmmateusz@gmail.com

1. Wstęp

Programowanie wielowątkowe jest dobrym sposobem na pisanie programów, które w tym samym czasie wykonują równolegle różne operacje. Wielowątkowość jest cechą systemu operacyjnego a nie samego języka programowania. Obecne systemy operacyjne oraz produkowane procesory posiadające kilka rdzeni obsługują wiele wątków. Możliwość programowania wielowątkowego dostarcza wiele języków programowania. Wybór zależy od wielu czynników często bardzo subiektywnych. Ciekawą kwestią wydaje się być wybór środowiska do nauki programowania wielowątkowego. Prostszy strukturalny język C może okazać się lepszym rozwiązaniem. Dodatkowo wielowątkowość w C z biblioteką Posil Threads ma dłuższą historię niż w języku C++, w którym jest dostępna od wersji 11 ew. w postaci popularnej ale stosunkowo młodej biblioteki Boost.

2. Przegląd języków programowania

W tym rozdziale przedstawiono krótką charakterystykę języków programowania C oraz C++.

Język C

Język C tworzony jako język łączący języki wysokiego poziomu z niskopoziomowymi. Nowe kompilatory języka mogły być szybko tworzone na nowe platformy. Niskopoziomowa natura języka C pozwala programiście sprawować kontrolę na tym, co robi komputer, jednocześnie pozwalając na specjalne dostosowanie i optymalizacje na konkretnych platformach. Pozwala to na szybkie działanie kodu nawet na ograniczonym sprzęcie, na przykład w systemach wbudowanych. Język C nie zawiera wielu właściwości dostępnych w innych językach programowania m. in. brak bezpośredniej obsługi programowania wielowątkowego. Brak jest natomiast w nim obsługi wyjątków czy obsługi programowania obiektowego, a w szczególności polimorfizmu czy dziedziczenia.[1]

Język C++

Język C++ jest językiem wieloparadygmatowym. Znaczy to, że można w nim stosować różne style programowania: programowanie proceduralne, obiektowe czy generyczne. C++ zakłada statyczną kontrolę typów, umożliwia bezpośrednie zarządzanie wolną pamięcią.[7]

3. Przegląd użytych bibliotek

W rozdziale opisano dwie biblioteki programistyczne Posix Threads oraz Boost.

POSIX Threads

Biblioteki wątków POSIX są standardowymi API (Interfejs programowania aplikacji) interfejs programistyczny aplikacji wielowątkowych dla języka C / C ++.

Standard POSIX Threads wchodzi w skład standardu POSIX określająca implementację wielowątkowości, która obejmuje podstawowe mechanizmy zarządzania wątkami, obiektami synchronizującymi oraz definiuje jednolity interfejs programistyczny dla języka C. Standard określa pewien podstawowy zestaw funkcji oraz szereg opcji, które mogą być

udostępnione przez implementację. Rozwiązanie to jest bardziej efektywne w systemach wieloprocesorowych lub wielordzeniowych, w których przepływ procesów można zaplanować na innym procesorze, zwiększając szybkość działania dzięki przetwarzaniu równoległym lub rozproszonym. Wątki wymagają mniejszej ilości zasobów niż tworzenie osobno nowego procesu.[3]

Boost

Boost to zestaw bibliotek dla języka programowania C++, który zapewnia wsparcie dla zadań i struktur, takich jak algebra liniowa, generowanie liczby pseudolosowej, wielowątkowość, przetwarzanie obrazu, wyrażeń regularnych i testowanie jednostkowe. Zawiera ponad osiemdziesiąt pojedynczych bibliotek. Biblioteki boost są objęte licencją Boost Software License, która pozwala każdemu używać, modyfikować i dystrybuować biblioteki za darmo. Biblioteki są niezależne od platformy i obsługują najpopularniejsze kompilatory.

Za tworzenie i publikowanie bibliotek Boost odpowiedzialna jest społeczność Boost składa się ona ze stosunkowo dużej grupy programistów C ++ z całego świata koordynowanych za pośrednictwem strony internetowej www.boost.org. Serwis GitHub jest używany jako repozytorium do przechowywania kodu bibliotek. Misją tej społeczności jest rozwijanie i gromadzenie wysokiej jakości bibliotek, które uzupełniają standardową bibliotekę. Biblioteki, które okażą się wartościowe i stają się ważne dla rozwoju aplikacji napisanych w języku C ++, mają dużą szansę na włączenie do standardowej biblioteki.[6]

Tabela 1. Zestawienie bibliotek.

Biblioteka	**Posix Threads**	**Boost**
Rok powstania	1985-1990	1999
Język	C	C++
Aktualna wersja stabilna	Sierpień 2018	Sierpień 2018
Platforma sprzętowa	GNU/Linux, Windows	Wieloplatformowa
Licencja	GNU Library	Boost Software License

4. Przegląd dostępnych funkcji w bibliotekach

Biblioteka Posix Threads posiada około 100 procedur obsługujących wątki, wszystkie poprzedzone pthread_ i można je podzielić na cztery grupy:

1) Zarządzanie wątkami - tworzenie, łączenie wątków itp.,
2) Mutexy,
3) Zmienne warunku,
4) Synchronizacja między wątkami za pomocą blokad i barier odczytu / zapisu.

Biblioteka Boost C ++ udostępnia kilka klas, które mogą być używane do pisania wielowątkowych programów w języku C++. Proces pisania wielowątkowego programu jest podobny do programów pisanych chociażby w Javie.

Tabela 2. Zestawienie wybranych funkcji dostępnych w bibliotekach.[8]

Posix Threads	**Boost**
Tworzenie i usuwanie wątków	
pthread_create (thread,attr,start_routine,arg) pthread_exit(status) pthread_cancel(thread) pthread_attr_init(attr) pthread_attr_destroy(attr)	boost::thread::thread() boost::thread::this_thread() boost::thread::thread_group()
Dołączanie i odłączanie wątków	
pthread_join (threadid,status) pthread_detach (threadid) pthread_attr_setdetachstate (attr,detachstate) pthread_attr_getdetachstate (attr,detachstate)	boost::thread::detach() boost::thread::join() boost::thread::join_all() boost::thread::joinable() boost::thread::timed_join() boost::thread::try_join_for() boost::thread::try_join_until()
Mutexy	
pthread_mutex_init (mutex,attr) pthread_mutex_destroy (mutex) pthread_mutexattr_init (attr) pthread_mutexattr_destroy (attr) pthread_mutex_lock (mutex) pthread_mutex_trylock (mutex) pthread_mutex_unlock (mutex)	boost::mutex::mutex() boost::mutex:recursive_mutex() boost::mutex::timed_mutex() boost::mutex:: lock_guard() boost::mutex:: unique_lock() boost::mutex::try_lock() boost::mutex::lock() boost::mutex::call_one()
Zmienne warunkowe	
pthread_cond_init (condition,attr) pthread_cond_destroy (condition) pthread_condattr_init(attr) pthread_condattr_destroy (attr) pthread_cond_wait (condition,mutex) pthread_cond_signal (condition) pthread_cond_broadcast (condition)	boost::condition_variable:: wait() boost::condition_variable:: timed_wait() boost::condition_variable:: wait_for() boost::condition_variable_ any::wait() boost::condition_variable_ any::wait_for() boost::condition_variable_ notify_one() boost::condition_variable_ notify_all()
Inne	
pthread_attr_getstacksize (attr, stacksize) pthread_attr_setstacksize (attr, stacksize) pthread_attr_getstackaddr (attr, stackaddr) pthread_attr_setstackaddr (attr, stackaddr) pthread_self () pthread_equal (thread1,thread2) pthread_once (once_control, init_routine)	boost::thread::sleep() boost::this_thread::sleep_for() boost::this_thread::sleep_until() boost::this_thread:: interruption_point() boost::thread::id()

5. Przykładowe programy

Rozdział przedstawia opis oraz fragmenty rozwiązań typowych programistycznych problemów synchronizacji.

Problem producent - konsument

Problem producenta i konsumenta to przykład synchronizacji zasobów. W zagadnieniu tym występują dwa rodzaje procesów, pomiędzy którymi są współdzielone pewne zasoby (pudełko). Pierwszy proces – producent generuje dane a drugi

– konsument je pobiera. Zadaniem producenta jest dostarczenie danych do pudełka, natomiast zadaniem konsumenta jest ich pobieranie z pudełka.

Listing 1. Kod funkcji - pthreads.[10]

```
void wloz(int m) {
    pthread_mutex_lock(&mutex);
    if (iloscElem == ILOSC_ELEMENTOW) {{
       printf ("Pudelko jest pelne.\n");
    }
    while (iloscElem == ILOSC_ELEMENTOW) {;
       pthread_cond_wait(&cond, &mutex);
    }
    }
    zawartosc[p] = m;
    p = (p + 1) % ILOSC_ELEMENTOW;
    ++iloscElem;
    pthread_mutex_unlock(&mutex);
    pthread_cond_signal(&cond);
}
```

W pierwszym przykładzie (Listing 1) wykorzystano funkcję pthread_mutex_lock() przysyłając w parametrze zmienną mutex. Funkcja ta pozwala zablokować zmienną oraz dalej wykonywać na niej operację. Po wszystkim należy ręcznie odblokować zmienną używając funkcji pthread_mutex_unlock(). Należy pamiętać, że ilość wywołań funkcji otwierającej blokadę musi być taka sama jak ilosć funkcji zamykającej tę blokadę. W przeciwnym razie może dojść do tzw. zakleszczenia.

W programie użyty został obiekt typu pthread_cond_t, który udostępnia funkcję do komunikacji między wątkami. Funkcja pthread_cond_wait() jako parametr przyjmuję referencję do obieku mutex oraz cond i zatrzymuje działanie wątku do czasu, aż inny wątek wyśle powiadomienie. Po wywołaniu funkcji pthread_cond_wait(), blokada obiektu podanego jako parametr jest zdjęta. Po otrzymaniu powiadomienia od innego wątku następuje wznowienie działania wątku oraz ponowne ustawienie blokady na obiekcie mutex. Do komunikacji między wątkami służy metoda pthread_cond_signal(). Wznawia ona wszystkie wątki, które wywołały wcześniej metodę wait() danego obiektu condition.

Listing 2. Kod funkcji - boost.[2]

```
class Pudelko {
private:
    boost::mutex mutex;
    boost::condition cond;
    unsigned int p, c, iloscElem;
    int zawartosc[ILOSC_ELEMENTOW];
public:
    Pudelko()
    : p(0), c(0), iloscElem(0){}

void wloz(int m)    {
      mutex.lock();
if (iloscElem == ILOSC_ELEMENTOW) {{
    boost::mutex::scoped_lock lock(ioMutex);
    cout << "Pudelko jest pelne." << std::endl;
    }
    while (iloscElem == ILOSC_ELEMENTOW) {
    cond.wait(mutex);
    }
}
    this->zawartosc[p] = m;
    p = (p+1) % ILOSC_ELEMENTOW;
    ++iloscElem;
    cond.notify_one();
    mutex.unlock();
}
Pudelko pudelko;
```

W przykładzie drugim (Listing 2) wykorzystano funkcje lock() obiektu boost::mutex, która jest alternatywą dla funkcji pthread_mutex_lock() służy ona do zamknięcia blokady. Otwarcie blokady następuje po wywołaniu funkcji unlock() na zmiennej mutex. Również w tym przypadku należy wystrzegać się zakleszczania i pamietać o ilości wywołań funkcji lock() oraz unlock().

W programie użyty został obiekt typu boost:condition, który udostępnia funkcję do komunikacji między wątkami, podobnie jak w przypadku biblioteki pthread – obiekt pthread_cond oraz jego funkcję - wait() wywołana na obiekcie boost::conditio jako parametr przyjmuje obiekt typu boost::mutex i zatrzymuje działanie wątku do czasu, aż inny wątek wyśle stosowane powiadomienie. Po wywołaniu funkcji wait(), blokada obiektu podanego jako parametr jest zdjęta. Po otrzymaniu powiadomienia od innego wątku następuje wznowienie działania wątku oraz ponowne ustawienie blokady na obiekcie boost::mutex. Do komunikacji między wątkami służy metoda notify_one() obiektu boost::condition. Wznawia ona wszystkie wątki, które wywołały wcześniej metodę wait() danego obiektu boost::condition.

Problem ucztujących filozofów

Problem 5-filozofów to klasyczny przypadek prezentacji problemu synchronizacji pracujących współbieżnie procesów. Teoretyczne wyjaśnienie zakleszczania i uniemożliwienia innym jednostkom korzystania z zasobów, przy założeniu takim, że każdy z filozofów bierze po jednym widelcu, a następnie próbuje zdobyć drugi. Zakłada się też, że jedzenie jednym widelcem jest niemożliwe.

Listing 3. Kod funkcji - pthreads.[9]

```
void *funkcja_pieciu(int n) {
    printf ("Filozof %d mysli\n", n);
    pthread_mutex_lock(&widelec[n]);
    pthread_mutex_lock(&widelec[(n + 1) % 5]);

    printf ("Filozof %d je\n", n);
    usleep(3);

    pthread_mutex_unlock(&widelec[n]);
    pthread_mutex_unlock(&widelec[(n + 1) % 5]);

    printf ("Filozof %d skonczyl jesc\n", n);
    return(NULL);
}
```

Poza opisywaną do tej pory funkcją pthread_mutex_lock(), która w tym przypadku ma za zadanie blokować zmienną widelec (lewy oraz prawy) tak by w danym momencie filozof mógł korzystać z dwóch widelców jednocześnie, w funkcji została użyta funkcja usleep(), która umożliwia zatrzymanie działania wątku na pewien określony czas przesłany w parametrze tej funkcji w milisekundach.

Listing 4. Kod funkcji main - pthreads.[11]

```
int main() {
   int i, j;
   for( j = 1; j <= POSILKI; j++) {
      for(i = 0; i < 5; i++)
         pthread_mutex_init(&widelec[i],NULL);
      for(i = 0; i < 5; i++)
         pthread_create(&filozofowie[i],NULL,(void
*)funkcja_pieciu,(void *)i);
      for(i = 0; i < 5; i++)
         pthread_join(filozofowie[i],NULL);
      for(i = 0; i < 5; i++)
         pthread_mutex_destroy(&widelec[i]);
   printf ("Filozof %d skonczyl %d posilek\n", i, j);
 }
```

Do tworzenia wątków w bibliotece Posix Threads służy funkcja pthread_create(), która w argumencie przyjmuje referencję do zmiennej pthread_t oraz funkcję, która ma zostać wywołana w tworzonym wątku.

Listing 5. Kod funkcji - boost.[5]

```
class Filozof {
private:
   int numer;
   int lewyWidelec;
   int prawyWidelec;
   int zjedzonychPosilkow;
public:
   Filozof(int n) : numer(n), zjedzonychPosilkow(0) {
   lewyWidelec = numer;
   prawyWidelec = (numer+4) % FILOZOFOWIE;
   }
   void operator()() {
      int n = FILOZOFOWIE;
      cout << "Filozof " << numer << " mysli\n";
      widelec[n].lock();
      widelec[(n + 1) % 5].lock();
      cout << "Filozof " << numer << " je\n";
      Sleep(3);
      widelec[n].unlock();
      widelec[(n + 1) % 5].unlock();
   }
};
```

Przy pomocy języka C++ stworzona została klasa Filozof posiadająca prywatne pola i przeciążony konstruktor. Posiada również przeciążony operator w klasie wykonujący operację jak zwykła funkcja. Znajduje się tu również funkcja Sleep działająca podobnie do funkcji usleep z poprzedniego przykładu.

Listing 6. Kod funkcji main - boost.[4]

```
int main() {
   thread_group filozofowie;
   int i, j;
   for( j = 1; j <= POSILKI; j++) {
      for(i = 0; i < 5; i++)
         boost::lock_guard<boost::mutex> lock{widelec[i]};
      for(i = 0; i < 5; i++){
         filozofowie.create_thread(Filozof(i));
         filozofowie.join_all();
         cout << "Filozof " << i << " skonczyl " << j << "
posilek\n";      }
```

Wątki w bibliotece Boost są reprezentowane przez klasę boost::thread. Przeciążony konstruktor klasy wątku pobiera jako parametr adres funkcji, która ma zostać wykonywana w tworzonym wątku. Przykład utworzenia wątku: boost::thread nazwa_watku(&nazwa_funkcji);

Funkcja main() jest wykonywana w głównym wątku programu. Po utworzeniu grupy wątków przy pomocy klasy boost::thread_group, w pętli tworzy wątki przy pomocy funkcji create_thread(). W programie działa już niezależnie pięć wątków, które reprezentują 5 – filozofów.

Stworzone wątki zostają dołączone do wątku głównego w funkcji main() za pomocą funkcji boost::thread::join_all(). Wszystkie wątki kończą swoją pracę.

6. Wnioski

Przedstawione dwie biblioteki programistyczne Posix Threads dla języka C oraz Boost dla języka C++ mimo różnic w implementacji mają wiele wspólnych cech i funkcji za sprawą których możemy tworzyć, synchronizować wątki, a w konsekwencji rozwiązywać problemy informatyczne. Posiadają też własne funkcje niedostępne w drugiej bibliotece.

Zaletą języka C w stosunku do C++ jest jego prostota. Zyskał on popularność na systemach wbudowanych dzięki wydajności działania i dostępności operacji niskopoziomowych. Dodatkowo biblioteka Pthreads posiada większą ilość dostępnych funkcji.

C++ - jest językiem obiektowym ogólnego przeznaczenia, dynamicznie rozwijanym, stanowiącym rozszerzenie popularnego języka C. Podejście obiektowe jest trudniejsze, ale oferuje nowe funkcjonalności m.in. hermetyzację danych. Biblioteka Boost jest popularna w programowaniu równoległym.

Programowanie wielowątkowe jest kluczem do tworzenia programów, aplikacji działających szybko i wydajnie na urządzeniach. Wybór języka wraz z biblioteką zależy od przeznaczenia programów jak również preferencji czy doświadczenia programisty właściwie w analogiczny sposób jak odpowiedź na pytanie czy lepiej uczyć się języka C czy C++. Najlepiej znać oba standardy. Podobnie jak biblioteki Posix Threads i Boost. Warto też zauważyć, że często programowania wielowątkowego uczy się w oparciu o standardy OpenMP i MPI. Niewątpliwie obie biblioteki zawierają bardzo bogaty zestaw gotowych i podobnych funkcji pozwalających na efektywne tworzenie aplikacji wielowątkowych .

Literatura

[1] Stephen Prata, Język C. Szkoła programowania. Wydanie VI, Helion, 2016.

[2] Alex Allain, C++. Przewodnik dla początkujących, Helion, 2014.

[3] Dick Buttlar, Jacqueline Farrell, Bradford Nichols, Pthreads Programming A POSIX Standard for Better Multiprocessing, Wydawnictwo O'Reilly Media, 2013.

[4] Douglas C. Schmidt, Stephen D. Huston, C++ Network Programming Volume 1 Addison-Wesley Professional; 1 edition, 2001.

[5] Boris Sch Ling, The Boost C++ Libraries, Wydawnictwo XML Press, 2011.

[6] Bjarne Stroustrup, "Why C++ is not just an Object-Oriented Programming Language", Sigplan, 1995.

[7] LukasEinkemmer, A resistive magnetohydrodynamics solver using modern C++ and the Boost library, Elsevier B.V., 2016.

[8] https://theboostcpplibraries.com/introduction[20.11.2018]

[9] https://mortoray.com/2011/12/16/how-does-a-mutex-work-what-does-it-cost/[20.11.2018]

[10] Bil Lewis, Multithreaded Programming With PThreads Prentice Hall; 136th ed. edition (9 December 1997)

[11] David R. Butenhof Programming with POSIX Threads, Addison Wesley Longman, Inc,1997