

Spis treści

Słowo wstępne (11)

Podziękowania (13)

O tej książce (15)

O autorze (19)

Rozdział 1. Witaj, świecie współbieżności w C++! (21)

1.1. Czym jest współbieżność? (22)

1.1.1. Współbieżność w systemach komputerowych (22)

1.1.2. Modele współbieżności (25)

1.1.3. Współbieżność kontra równoległość (27)

1.2. Dlaczego warto stosować współbieżność? (27)

1.2.1. Stosowanie współbieżności do podziału zagadnień (27)

1.2.2. Stosowanie współbieżności do podniesienia wydajności - równoległość zadań i danych (28)

1.2.3. Kiedy nie należy stosować współbieżności (30)

1.3. Współbieżność i wielowątkowość w języku C++ (31)

1.3.1. Historia przetwarzania wielowątkowego w języku C++ (31)

1.3.2. Obsługa współbieżności w nowym standardzie (32)

1.3.3. Większa obsługa współbieżności i równoległości w standardach C++14 i C++17 (33)

1.3.4. Efektywność biblioteki wątków języka C++ (33)

1.3.5. Mechanizmy związane z poszczególnymi platformami (34)

1.4. Do dzieła (35)

1.4.1. Witaj, świecie współbieżności! (35)

1.5. Podsumowanie (36)

Rozdział 2. Zarządzanie wątkami (39)

2.1. Podstawowe zarządzanie wątkami (40)

2.1.1 Uruchamianie wątku (40)

2.1.2. Oczekiwanie na zakończenie wątku (43)

2.1.3. Oczekiwanie w razie wystąpienia wyjątku (44)

2.1.4. Uruchamianie wątków w tle (46)

2.2. Przekazywanie argumentów do funkcji wątku (47)

2.3. Przenoszenie własności wątku (50)

2.4. Wybór liczby wątków w czasie wykonywania (55)

2.5. Identyfikowanie wątków (57)

2.6. Podsumowanie (59)

Rozdział 3. Współdzielenie danych przez wątki (61)

3.1. Problemy związane ze współdzieleniem danych przez wątki (62)

3.1.1. Sytuacja wyścigu (64)

3.1.2. Unikanie problematycznych sytuacji wyścigu (65)

3.2. Ochrona współdzielonych danych za pomocą muteksów (66)

3.2.1. Stosowanie muteksów w języku C++ (66)

3.2.2. Projektowanie struktury kodu z myślą o ochronie współdzielonych danych (68)

3.2.3. Wykrywanie sytuacji wyścigu związanych z interfejsami (70)

3.2.4. Zakleszczenie: problem i rozwiązanie (77)

3.2.5. Dodatkowe wskazówki dotyczące unikania zakleszczeń (80)

3.2.6. Elastyczne blokowanie muteksów za pomocą szablonu `std::unique_lock` (87)

3.2.7. Przenoszenie własności muteksu pomiędzy zasięgami (89)

3.2.8. Dobór właściwej szczegółowości blokad (90)

3.3. Alternatywne mechanizmy ochrony współdzielonych danych (93)

3.3.1. Ochrona współdzielonych danych podczas inicjalizacji (93)

3.3.2. Ochrona rzadko aktualizowanych struktur danych (97)

3.3.3. Blokowanie rekurencyjne (99)

3.4. Podsumowanie (100)

Rozdział 4. Synchronizacja współbieżnych operacji (101)

4.1. Oczekiwanie na zdarzenie lub inny warunek (102)

4.1.1. Oczekiwanie na spełnienie warunku za pomocą zmiennych warunkowych (103)

4.1.2. Budowa kolejki gwarantującej bezpieczne przetwarzanie wielowątkowe przy użyciu zmiennych warunkowych (106)

4.2. Oczekiwanie na jednorazowe zdarzenia za pomocą przyszłości (111)

4.2.1. Zwracanie wartości przez zadania wykonywane w tle (112)

4.2.2. Wiązanie zadania z przyszłością (114)

4.2.3. Obietnice (szablon `std::promise`) (117)

4.2.4. Zapisywanie wyjątku na potrzeby przyszłości (119)

4.2.5. Oczekiwanie na wiele wątków (121)

4.3. Oczekiwanie z limitem czasowym (124)

4.3.1. Zegary (124)

4.3.2. Okresy (125)

4.3.3. Punkty w czasie (127)

4.3.4. Funkcje otrzymujące limity czasowe (129)

4.4. Upraszczanie kodu za pomocą technik synchronizowania operacji (131)

4.4.1. Programowanie funkcyjne przy użyciu przyszłości (131)

4.4.2. Synchronizacja operacji za pomocą przesyłania komunikatów (136)

4.4.3. Współbieżność w stylu kontynuacji dzięki użyciu Concurrency TS (141)

4.4.4. Łączenie kontynuacji ze sobą (143)

4.4.5. Oczekiwanie na więcej niż tylko jedną przyszłość (146)

4.4.6. Oczekiwanie za pomocą `when_any` na pierwszą przyszłość w zbiorze (148)

4.4.7. Zasuwy i bariery w Concurrency TS (151)

4.4.8. Zasuwa typu podstawowego - `std::experimental::latch` (151)

4.4.9. Podstawowa bariera - `std::experimental::barrier` (153)

4.4.10. `std::experimental::flex_barrier`, czyli elastyczniejszy przyjaciel `std::experimental::barrier` (155)

4.5. Podsumowanie (156)

Rozdział 5. Model pamięci języka C++ i operacje na typach atomowych (157)

5.1. Podstawowe elementy modelu pamięci (158)

5.1.1. Obiekty i miejsca w pamięci (158)

5.1.2. Obiekty, miejsca w pamięci i przetwarzanie współbieżne (159)

5.1.3. Kolejność modyfikacji (161)

5.2. Operacje i typy atomowe języka C++ (161)

5.2.1. Standardowe typy atomowe (162)

5.2.2. Operacje na typie `std::atomic_flag` (165)

5.2.3. Operacje na typie `std::atomic` (167)

5.2.4. Operacje na typie `std::atomic` - arytmetyka wskaźników (170)

5.2.5. Operacje na standardowych atomowych typach całkowitoliczbowych (172)

5.2.6. Główny szablon klasy `std::atomic<>` (172)

5.2.7. Wolne funkcje dla operacji atomowych (174)

5.3. Synchronizacja operacji i wymuszanie ich porządku (176)

5.3.1. Relacja synchronizacji (178)

5.3.2. Relacja poprzedzania (179)

5.3.3. Porządkowanie pamięci na potrzeby operacji atomowych (181)

5.3.4. Sekwencje zwalniania i relacja synchronizacji (201)

5.3.5. Ogrodzenia (204)

5.3.6. Porządkowanie operacji nieatomowych za pomocą operacji atomowych (206)

5.3.7. Porządkowanie operacji nieatomowych (207)

5.4. Podsumowanie (210)

Rozdział 6. Projektowanie współbieżnych struktur danych przy użyciu blokad (211)

6.1. Co oznacza projektowanie struktur danych pod kątem współbieżności? (212)

6.1.1. Wskazówki dotyczące projektowania współbieżnych struktur danych (213)

6.2. Projektowanie współbieżnych struktur danych przy użyciu blokad (214)

6.2.1. Stos gwarantujący bezpieczeństwo przetwarzania wielowątkowego przy użyciu blokad (215)

6.2.2. Kolejka gwarantująca bezpieczeństwo przetwarzania wielowątkowego przy użyciu blokad i zmiennych warunkowych (218)

6.2.3. Kolejka gwarantująca bezpieczeństwo przetwarzania wielowątkowego przy użyciu szczegółowych blokad i zmiennych warunkowych (222)

6.3. Projektowanie złożonych struktur danych przy użyciu blokad (235)

6.3.1. Implementacja tablicy wyszukiwania gwarantującej bezpieczeństwo przetwarzania wielowątkowego przy użyciu blokad (235)

6.3.2. Implementacja listy gwarantującej bezpieczeństwo przetwarzania wielowątkowego przy użyciu blokad (241)

6.4. Podsumowanie (246)

Rozdział 7. Projektowanie współbieżnych struktur danych bez blokad (247)

7.1. Definicje i ich praktyczne znaczenie (248)

7.1.1. Rodzaje nieblokujących struktur danych (248)

7.1.2. Struktury danych bez blokad (249)

7.1.3. Struktury danych bez oczekiwania (250)

7.1.4. Zalety i wady struktur danych bez blokad (250)

7.2. Przykłady struktur danych bez blokad (252)

7.2.1. Implementacja stosu gwarantującego bezpieczeństwo przetwarzania wielowątkowego bez blokad (253)

7.2.2. Eliminowanie niebezpiecznych wycieków - zarządzanie pamięcią w strukturach danych bez blokad (257)

7.2.3. Wykrywanie węzłów, których nie można odzyskać, za pomocą wskaźników ryzyka (262)

7.2.4. Wykrywanie używanych węzłów metodą zliczania referencji (271)

7.2.5. Zmiana modelu pamięci używanego przez operacje na stosie bez blokad (277)

7.2.6. Implementacja kolejki gwarantującej bezpieczeństwo przetwarzania wielowątkowego bez blokad (282)

7.3. Wskazówki dotyczące pisania struktur danych bez blokad (295)

7.3.1. Wskazówka: na etapie tworzenia prototypu należy stosować tryb `std::memory_order_seq_cst` (295)

7.3.2. Wskazówka: należy używać schematu odzyskiwania pamięci bez blokad (296)

7.3.3 Wskazówka: należy unikać problemu ABA (296)

7.3.4. Wskazówka: należy identyfikować pętle aktywnego oczekiwania i wykorzystywać czas bezczynności na wspieranie innego wątku (297)

7.4. Podsumowanie (298)

Rozdział 8. Projektowanie współbieżnego kodu (299)

8.1. Techniki dzielenia pracy pomiędzy wątki (300)

8.1.1. Dzielenie danych pomiędzy wątki przed rozpoczęciem przetwarzania (301)

8.1.2. Rekurencyjne dzielenie danych (302)

8.1.3. Dzielenie pracy według typu zadania (307)

8.2. Czynniki wpływające na wydajność współbieżnego kodu (310)

8.2.1. Liczba procesorów (310)

8.2.2. Współzawodnictwo o dane i ping-pong bufora (311)

8.2.3. Fałszywe współdzielenie (314)

8.2.4. Jak blisko należy rozmieścić dane? (315)

8.2.5. Nadsubskrypcja i zbyt intensywne przełączanie zadań (316)

8.3. Projektowanie struktur danych pod kątem wydajności przetwarzania wielowątkowego (317)

8.3.1. Podział elementów tablicy na potrzeby złożonych operacji (317)

8.3.2. Wzorce dostępu do danych w pozostałych strukturach (319)

8.4. Dodatkowe aspekty projektowania współbieżnych struktur danych (321)

8.4.1. Bezpieczeństwo wyjątków w algorytmach równoległych (321)

8.4.2. Skalowalność i prawo Amdahla (329)

8.4.3. Ukrywanie opóźnień za pomocą wielu wątków (330)

8.4.4. Skracanie czasu reakcji za pomocą technik przetwarzania równoległego (332)

8.5. Projektowanie współbieżnego kodu w praktyce (334)

8.5.1. Równoległa implementacja funkcji `std::for_each` (334)

8.5.2. Równoległa implementacja funkcji `std::find` (337)

8.5.3. Równoległa implementacja funkcji `std::partial_sum` (343)

8.6. Podsumowanie (353)

Rozdział 9. Zaawansowane zarządzanie wątkami (355)

9.1. Pule wątków (356)

9.1.1. Najprostsza możliwa pula wątków (356)

9.1.2. Oczekiwanie na zadania wysyłane do puli wątków (359)

9.1.3. Zadania oczekujące na inne zadania (363)

9.1.4. Unikanie współzawodnictwa w dostępie do kolejki zadań (366)

9.1.5. Wykradanie zadań (368)

9.2. Przerywanie wykonywania wątków (372)

9.2.1. Uruchamianie i przerywanie innego wątku (373)

9.2.2. Wykrywanie przerwania wątku (375)

9.2.3. Przerywanie oczekiwania na zmienną warunkową (375)

9.2.4. Przerywanie oczekiwania na zmienną typu `std::condition_variable_any` (379)

9.2.5. Przerywanie pozostałych wywołań blokujących (381)

9.2.6. Obsługa przerwania (382)

9.2.7. Przerywanie zadań wykonywanych w tle podczas zamykania aplikacji (383)

9.3. Podsumowanie (384)

Rozdział 10. Algorytmy równoległości (385)

10.1. Algorytmy równoległe w bibliotece standardowej (385)

10.2. Polityki wykonywania (386)

10.2.1. Ogólny efekt wyboru polityki wykonywania (386)

10.2.2. `std::execution::sequenced_policy` (388)

10.2.3. `std::execution::parallel_policy` (388)

10.2.4. `std::execution::parallel_unsequenced_policy` (389)

10.3. Algorytmy równoległości w bibliotece standardowej C++ (390)

10.3.1. Przykłady używania algorytmów równoległych (392)

10.3.2. Licznik odwiedzin (394)

10.4. Podsumowanie (396)

Rozdział 11. Testowanie i debugowanie aplikacji wielowątkowych (397)

11.1. Rodzaje błędów związanych z przetwarzaniem współbieżnym (398)

11.1.1. Niechciane blokowanie (398)

11.1.2. Sytuacje wyścigu (399)

11.2. Techniki lokalizacji błędów związanych z przetwarzaniem współbieżnym (400)

11.2.1. Przeglądanie kodu w celu znalezienia ewentualnych błędów (401)

11.2.2. Znajdowanie błędów związanych z przetwarzaniem współbieżnym poprzez testowanie kodu (403)

11.2.3. Projektowanie kodu pod kątem łatwości testowania (405)

11.2.4. Techniki testowania wielowątkowego kodu (407)

11.2.5. Projektowanie struktury wielowątkowego kodu testowego (410)

11.2.6. Testowanie wydajności wielowątkowego kodu (413)

11.3. Podsumowanie (414)

Dodatek A. Krótki przegląd wybranych elementów języka C++11 (415)

A.1. Referencje do r-wartości (415)

A.2. Usunięte funkcje (420)

A.3. Funkcje domyślne (421)

A.4. Funkcje constexpr (425)

A.5. Funkcje lambda (430)

A.6. Szablony zmiennoargumentowe (436)

A.7. Automatyczne określanie typu zmiennej (440)

A.8. Zmienne lokalne wątków (441)

A.9. Ustalanie argumentu szablonu klasy (442)

A.10. Podsumowanie (443)

Dodatek B. Krótkie zestawienie bibliotek przetwarzania współbieżnego (445)

Dodatek C. Framework przekazywania komunikatów i kompletny przykład implementacji systemu bankomatu (447)

Dodatek D. Biblioteka wątków języka C++ (465)

D.1. Nagłówek (465)

D.2. Nagłówek (481)

D.3. Nagłówek (498)

D.4. Nagłówek (536)

D.5. Nagłówek (561)

D.6. Nagłówek (613)

D.7. Nagłówek (619)

Skorowidz (631)