



KARTA OPISU PRZEDMIOTU - SYLABUS

Nazwa przedmiotu

Metody bezpiecznego programowania [S2Inf1-SRC>MBP]

Przedmiot

Kierunek studiów
Informatyka

Rok/Semestr
1/1

Studia w zakresie (specjalność)
Systemy rozproszone i chmurowe

Profil studiów
ogólnoakademicki

Poziom studiów
drugiego stopnia

Język oferowanego przedmiotu
polski

Forma studiów
stacjonarne

Wymagalność
obligatoryjny

Liczba godzin

Wykład
30

Laboratorium
30

Inne
0

Ćwiczenia
0

Projekty/seminaria
0

Liczba punktów ECTS

5,00

Koordynatorzy

dr hab. inż. Paweł Wojciechowski prof. PP
pawel.t.wojciechowski@put.poznan.pl

Wykładowcy

Wymagania wstępne

Student rozpoczynający ten przedmiot powinien posiadać podstawową wiedzę z dziedziny systemów współbieżnych i rozproszonych oraz znajomość co najmniej jednego współczesnego języka programowania. Powinien posiadać umiejętność rozwiązywania podstawowych problemów synchronizacji współbieżnych wątków lub procesów. Powinni też mieć umiejętność pozyskiwania informacji ze wskazanych źródeł angielskojęzycznych. Powinien również rozumieć konieczność poszerzania swoich kompetencji oraz mieć gotowość do podjęcia współpracy w ramach zespołu. Ponadto w zakresie kompetencji społecznych student musi prezentować takie postawy jak uczciwość, odpowiedzialność, wytrwałość, ciekawość poznawczą, kreatywność, kulturę osobistą, szacunek dla innych ludzi.

Cel przedmiotu

Celem wykładów jest omówienie wybranych metod programowania, które pozwalają pisać poprawne i wydajne programy współbieżne. Po pierwsze, zobaczymy, jak poprawnie i wydajnie używać monitorów. Następnie omówimy alternatywną metodę synchronizacji, czyli pamięć transakcyjną. Po drugie, przedstawimy wybrane metody automatycznej weryfikacji. Po trzecie, omówimy niektóre zagadnienia dotyczące poprawności programów współbieżnych przeznaczonych dla nowoczesnych architektur komputerowych. Na koniec pokażemy przykładowe metody i języki, które ułatwiają pisanie programów równoległych i rozproszonych. W ramach zająć lab. studenci realizują ćwiczenia i projekty programistyczne. Pozostałe cele przedmiotu: - rozwijanie u studentów umiejętności rozwiązywania problemów programowania współbieżnego oraz umiejętności stosowania zaawansowanych mechanizmów synchronizacji do rozwiązania problemów synchronizacji, - kształtowanie u studentów umiejętności pracy zespołowej w trakcie realizacji ćwiczeń i projektów na zajęciach laboratoryjnych.

Przedmiotowe efekty uczenia się

Wiedza:

ma uporządkowaną i podbudowaną teoretycznie wiedzę ogólną w zakresie języków i paradygmatów bezpiecznego programowania (k2st_w2)
ma zaawansowaną wiedzę szczegółową dotyczącą wybranych zagadnień z zakresu informatyki, takich jak współczesne metody, języki i narzędzia programowania współbieżnego i rozproszonego (k2st_w3)
ma wiedzę o trendach rozwojowych i najistotniejszych nowych osiągnięciach informatyki i innych, wybranych, pokrewnych dyscyplin naukowych w zakresie języków i paradygmatów bezpiecznego programowania (k2st_w4)
ma zaawansowaną i szczegółową wiedzę o procesach zachodzących w cyklu życia systemów informatycznych programowych (k2st_w5)
zna zaawansowane metody, techniki i narzędzia stosowane przy rozwiązywaniu złożonych zadań inżynierskich i prowadzeniu prac badawczych w obszarze informatyki, który dotyczy programowania współbieżnego (k2st_w6)

Umiejętności:

potrafi pozyskiwać informacje z literatury, baz danych oraz innych źródeł (w języku polskim i angielskim), integrować je, dokonywać ich interpretacji i krytycznej oceny, wyciągać wnioski oraz formułować i wyczerpująco uzasadniać opinie (k2st_u1)
potrafi wykorzystać do formułowania i rozwiązywania zadań inżynierskich i prostych problemów badawczych metody analityczne, symulacyjne oraz eksperymentalne (k2st_u4)
potrafi — przy formułowaniu i rozwiązywaniu zadań inżynierskich — integrować wiedzę z różnych obszarów informatyki (a w razie potrzeby także wiedzę z innych dyscyplin naukowych) oraz zastosować podejście systemowe, uwzględniające także aspekty pozatechniczne (k2st_u5)
potrafi ocenić przydatność i możliwość wykorzystania nowych osiągnięć (metod i narzędzi) oraz nowych produktów informatycznych (k2st_u6)
potrafi dokonać krytycznej analizy istniejących rozwiązań technicznych oraz zaproponować ich ulepszenia (usprawnienia) (k2st_u8)
potrafi ocenić przydatność metod i narzędzi służących do rozwiązania zadania inżynierskiego, polegającego na budowie lub ocenie systemu informatycznego lub jego składowych, w tym dostrzec ograniczenia tych metod i narzędzi (k2st_u9)
potrafi - stosując m.in. koncepcyjnie nowe metody - rozwiązywać złożone zadania informatyczne, w tym zadania nietypowe oraz zadania zawierające komponent badawczy (k2st_u10)

Kompetencje społeczne:

rozumie, że w informatyce wiedza i umiejętności bardzo szybko stają się przestarzałe (k2st_k1), rozumie znaczenie wykorzystywania najnowszej wiedzy z zakresu informatyki w rozwiązywaniu problemów badawczych i praktycznych (k2st_k2)

Metody weryfikacji efektów uczenia się i kryteria oceny

Efekty uczenia się przedstawione wyżej weryfikowane są w następujący sposób:

Efekty uczenia się weryfikowane są w następujący sposób:

Ocena formująca:

a) w zakresie wykładów -- na podstawie odpowiedzi na pytania dotyczące materiału omówionego na

poprzednich wykładach,
b) w zakresie laboratoriów -- na podstawie oceny bieżącego postępu realizacji zadań i projektów programistycznych.

Ocena podsumowująca:

a) w zakresie wykładów weryfikowanie założonych efektów kształcenia realizowane jest przez sprawdzian pisemny. Do zaliczenia na ocenę dostateczną konieczne jest uzyskanie powyżej połowy możliwych do uzyskania punktów.

b) w zakresie laboratoriów weryfikowanie założonych efektów kształcenia realizowane jest przez projekty programistyczne oraz sprawdzian pisemny, który może być połączony ze sprawdzianem pisemnym na wykładzie.

Na ocenę końcową może wpływać aktywność podczas zajęć, np. omówienie dodatkowych aspektów zagadnienia, oraz udoskonaleniem materiałów dydaktycznych.

Treści programowe

Pisanie programów współbieżnych dla współczesnych architektur komputerowych, które nie gwarantują spójności sekwencyjnej, jest zazwyczaj znacznie trudniejsze niż pisanie analogicznych programów sekwencyjnych. Ponadto testowanie takich programów jest trudne, a wykrywanie i usuwanie wszystkich błędów programistycznych w kodzie nie zawsze jest możliwe. Celem wykładów jest omówienie niektórych metod programowania, które pozwalają pisać poprawne i wydajne programy współbieżne.

Tematyka zajęć

Podstawowy program przedmiotu jest podzbiorem poniższych tematów, z których niektóre są omawiane na więcej niż jednym zajęciach:

Wykłady

1. Klasyczne zagadnienia dotyczące programowania współbieżnego w kontekście współczesnych architektur sprzętowych, w szczególności niskopoziomowe konstrukcje zapewnienia atomowości (wzajemnego wykluczania), synchronizacji warunkowej i barier.
2. Programowanie współbieżne i synchronizacja na przykładzie monitorów w C#/Java:
 - podstawowe operacje monitorów, prawidłowy dostęp do danych współdzielonych, niezmienniki, poprawne wzorce projektowe, błędne praktyki (np. double-check locking),
 - zakleszczenie, zagłodzenie, problemy z efektywnością przy konfliktach na zamkach i odwróceniu priorytetów, zaawansowane problemy synchronizacji i optymalizacje (np. unikanie spurious wake-ups i spurious lock conflicts).
3. Detekcja warunku wyścigu w czasie wykonania programu na przykładzie narzędzia Eraser: algorytm lockset, optymalizacje (inicjalizacja zmiennych, dane współdzielone tylko do odczytu, read-write locks), poprawność i kompletność algorytmu.
4. Detekcja warunku wyścigu wysokiego rzędu (ang. high-level data race) przez analizę statyczną algorytm oraz jego poprawność i kompletność (false positives - niepotrzebne ostrzeżenia, false negatives - niezauważone błędy).
5. Detekcja błędów programistycznych (dzielenie przez zero, warunek wyścigu, zakleszczenie) w oparciu o model checking, na przykładzie narzędzia Java Pathfinder.
6. Warunkowe regiony krytyczne (CCR) i pamięć transakcyjna: niskopoziomowe operacje na pamięci transakcyjnej, implementacja CCR przy użyciu tych operacji, struktura stosu, struktury danych (ownership records i deskryptory transakcji), algorytm atomowego zapisu do pamięci transakcyjnej.
7. Poprawność współbieżnego dostępu do obiektów współdzielonych: historie sekwencyjne i współbieżne, własność liniowości (ang. linearizability), współbieżna kolejka FIFO i rejestr, własności liniowości (lokalność, blokowanie vs. nieblokowanie).
8. Model pamięci na przykładzie języka Java: model pamięci jako specyfikacja poprawnej semantyki programów współbieżnych oraz legalnych implementacji kompilatorów i maszyn wirtualnych, słabość vs. siła modelu pamięci, współczesne ograniczenia klasycznego modelu spójności sekwencyjnej, analiza globalna i optymalizacje kodu, model pamięci happens-before oraz jego słabość, model pamięci uwzględniający circular causality, przykłady kontrowersyjnych transformacji kodu programów.
9. Programowanie równoległe na przykładzie języka Cilk: konstrukcje programistyczne, graf skierowany acykliczny (DAG) jako model obliczeń wielowątkowych, miary wykonania na procesorze wielordzeniowym,

szeregowanie zachłanne i górne ograniczenia na czas obliczeń.

10. Programowanie rozproszone z gwarancjami ostatecznej spójności na przykładzie Conflict-Free Replicated Types (CRDTs) i Cloud Types.

11. Programowanie rozproszone w modelu przesyłania komunikatów (ang. message-passing) na przykładzie języka Erlang (model aktorów, odporność na awarie) i/lub Nomadic Pict (mobilność procesów, weryfikacja poprawności komunikacji w trakcie kompilacji przez typy statyczne).

Zajęcia laboratoryjne

Na zajęciach laboratoryjnych studenci poznają różne konstrukcje programowania współbieżnego, ze szczególnym uwzględnieniem bezpieczeństwa (ang. safety) w programowaniu na współczesne architektury komputerowe, które nie gwarantują spójności sekwencyjnej. Omówione zostaną także przykładowe algorytmy implementujące mechanizmy synchronizacji oraz współbieżne, nieblokujące struktury danych. Jest to materiał komplementarny względem wykładów.

Jeśli czas pozwoli na zajęciach lab. omówiony zostanie także OCaml -- wzorcowy język programowania funkcyjnego, mający korzenie w języku ML z lat 70-tych. OCaml stał się inspiracją dla wielu innych współczesnych języków programowania, np. Rust. Omawiając konstrukcje języka zostanie położony nacisk na ekspresję i bezpieczeństwo programowania (ang. safety). OCaml jest eleganckim połączeniem najlepszych cech języków programowania, tj. wspiera moduły i interfejsy, funkcje pierwszego rzędu, typy abstrakcyjne, zmienne typów, dopasowanie do wzorca, niezmienność (ang. immutability), statyczną weryfikację typów oraz automatyczne zarządzanie pamięcią. Wszystkie te cechy razem w naturalny sposób wspierają bezpieczne programowanie.

Metody dydaktyczne

a) Wykłady -- prezentacja multimedialna ilustrowana przykładami na tablicy, demonstracja na komputerze, dyskusja moderowana przez prowadzącego.

b) Ćwiczenia laboratoryjne -- prezentacja multimedialna, omówienie przykładów na tablicy lub na komputerze, ćwiczenia praktyczne przy komputerze polegające na wykonaniu przez studentów zadań, praca w zespole, wykonywanie projektów programistycznych przez studentów, dyskusja moderowana przez prowadzącego.

Literatura

Przykładowe artykuły naukowe (wszystkie są dostępne przez Bibliotekę Główną Politechniki Poznańskiej i/ lub są udostępnione studentom przez prowadzącego zajęcia):

1. An Introduction to Programming with C# Threads. Andrew D. Birrell
2. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson
3. High-level Data Races. Cyrille Artho, Klaus Havelund, Armin Biere
4. Language Support for Lightweight Transactions. Tim Harris, Keir Fraser
5. Linearizability: a correctness condition for concurrent objects. Maurice P. Herlihy, Jeannette M. Wing
6. The Java Memory Model. Jeremy Manson, William Pugh, Sarita V. Adve
7. A Minicourse on Multithreaded Programming. Charles E. Leiserson, Harald Prokop
8. MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean, Sanjay Ghemawat
9. Erlang - A survey of the language and its industrial applications. Joe Armstrong
10. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages. Paweł T. Wojciechowski
11. Sinfonia: A New Paradigm for Building Scalable Distributed Systems. Marcos K. Aguilera, Arif Merchant, Mehul Sha
12. The Part-Time Parliament. Leslie Lamport
13. Cloud Types for Eventual Consistency. Sebastian Burckhardt¹, Manuel Fahndrich, Daan Leijen, and Benjamin P. Wood
14. Conflict-free Replicated Data Types. Nuno Preguica, Carlos Baquero, Marc Shapiro
15. Developing Applications with OCaml. Emmanuel Chailoux, Pascal Manoury and Bruno Pagano

Bilans nakładu pracy przeciętnego studenta

	Godzin	ECTS
Łączny nakład pracy	125	5,00
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	60	2,50
Praca własna studenta (studia literaturowe, przygotowanie do zajęć laboratoryjnych/ćwiczeń, przygotowanie do kolokwium/egzaminu, wykonanie projektu)	65	2,50