KARTA OPISUPRZEDMIOTU - SYLABUS

Nazwaprzedmiotu
Metody bezpiecznego programowania

Przedmiot

Kierunek studiów
Informatyka
Studia w zakresie(specjalność)
Systemy rozproszone
Poziom studiów
drugiego stopnia
Forma studiów
stacjonarne

Rok/semestr
1/1
Profil studiów
ogólnoakademicki
Język oferowanego przedmiotu
polski
Wymagalność
obligatoryjny

Liczba godzin

Wykład
30
Laboratoria
30
Inne (np. online)
Ćwiczenia
Projekty/seminaria

Liczba punktówECTS

5

Wykładowcy

Odpowiedzialny za przedmiot/wykładowca:
dr hab. inż. Paweł Wojciechowski, prof. nadzw.
email: Pawel.T.Wojciechowski@put.poznan.pl
tel: 61 665 3021
wydział: Wydział Informatyki
adres: ul. Piotrowo 3, 60-965 Poznań

Odpowiedzialny za przedmiot/wykładowca:

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) oraz umiejętność pozyskiwania informacji ze wskazanych źródeł angielskojęzycznych. Powinien również rozumieć konieczność poszerzania swoich

kompetencji / mieć gotowość do podjęcia współpracy w ramach zespołu. Ponadto w zakresie kompetencji społecznych student musi prezentować takie postawy jak uczciwość, odpowiedzialność, wytrwałość, ciekawość poznawcza, kreatywność, kultura osobista, szacunek dla innych ludzi.

Cel przedmiotu

1. Przekazanie studentom podstawowej wiedzy w zakresie współczesnych metod, języków i narzędzi bezpiecznego programowania (ang. safe programming), to jest takich, które gwarantują programowanie wolne od określonej klasy błędów programistycznych,
2. Omówienie przykładowych metod programowania funkcyjnego na przykładzie języka funkcyjnego OCaml, przykładowych metod i narzędzi bezpiecznego programowania systemów współbieżnych i rozproszonych, mechanizmów lub algorytmów zastosowanych w wybranych narzędziach, a także podstawowych własności poprawności programów współbieżnych, które mają zastosowanie w kontekście omawianych metod i narzędzi,
3. Rozwijanie u studentów umiejętności wnioskowania na temat poprawności programów współbieżnych,vna przykładach programów poprawnych i błędnych,
4. Kształtowanie u studentów umiejętności pracy zespołowej przez seminaryjny charakter niektórych zajęć, z naciskiem na dyskusję i wspólne wypracowywanie wniosków, a także przez realizację projektów programistycznych.

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 kształcenia przedstawione wyżej 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.

Ocena podsumowująca:
a) w zakresie wykładów weryfikowanie założonych efektów kształcenia realizowane jest przez:
- ocenę wiedzy i umiejętności wykazanych na egzaminie pisemnym o charakterze problemowym. Egzamin polega na odpowiedzi pisemnej na 3 pytania, wybrane z listy kilkudziesięciu pytań, która jest udostępniania wcześniej studentom. Za udzielenie poprawnych odpowiedzi na wszystkie pytania można otrzymać 9 punktów. Do zaliczenia na ocenę dostateczną należy zdobyć min. 4 punkty.
b) w zakresie laboratoriów weryfikowanie założonych efektów kształcenia realizowane jest przez:
- ocenę umiejętności związanych z realizacją projektu programistycznego; ocena ta obejmuje także umiejętność pracy w zespole, gdyż projekty przeważnie są realizowane przez dwóch studentów,
- ocenę i obronę przez studenta prezentacji na podstawie artykułów naukowych wskazanych przez prowadzącego lub przygotowanie ćwiczeń laboratoryjnych w których uczestniczy grupa studentów. W obu przypadkach dyskusja jest moderowana przez prowadzącego zajęcia. W przypadku prezentacji na ocenę składa się m.in. klarowność wytłumaczenia motywacji dla danego rozwiązania, posługując się odpowiednimi przykładami, oraz umiejętność pracy w zespole (w przypadku prezentacji dwuosobowej). W przypadku ćwiczeń laboratoryjnych na ocenę składa się m.in. klarowność wytłumaczenia problemu i jego rozwiązania oraz umiejętność pracy z grupą studentów realizujących ćwiczenia.
Ocena zaliczeniowa może zostać podwyższona za wyróżniającą aktywność podczas zajęć, a szczególnie

za:
- omówienia dodatkowych aspektów zagadnienia,
- efektywność zastosowania zdobytej wiedzy podczas rozwiązywania zadanego problemu,
- uwagi związane z udoskonaleniem materiałów dydaktycznych.

Treści programowe

Podstawowy program przedmiotu obejmuje następujące zagadnienia:

1. 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),

2. Programowanie współbieżne i synchronizacja na przykładzie monitorów w C#/Java: 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. Języki funkcyjne: weryfikacja przez silne typowanie, polimorfizm typów, składanie funkcji, częściowe wykonanie (currying), obiekty funkcyjne (closures), referencje i bezpieczne zarządzanie pamięcią, funkcje anonimowe [lab],

4. Języki funkcyjne: przykładowe typowane struktury danych, konstrukcje agregacyjne (np. map-reduce czyli mapowanie/odwzorowywanie z redukcją, filtrowanie, oraz folding), dopasowanie do wzorca, poprawna rekurencja (tail recursion) [lab],

5. Poprawność współbieżnego dostępu do obiektów współdzielonych: historie sekwencyjne i współbieżne, własność liniowości (linearizability), przykłady: kolejka FIFO i rejestry, formalizacja i własności liniowości (np. lokalność, blokowanie vs. nieblokowanie),

6. Dynamiczna detekcja błędów w programowaniu współbieżnym na przykładzie Eraser: relacja happens-before i jej ograniczenia, algorytm lockset do detekcji warunków wyścigu, optymalizacje algorytmu uwzględniające inicjalizację zmiennych, dane współdzielone tylko do odczytu oraz read-write locks,

7. Warunek wyścigu wysokiego rzędu (high-level data race): definicja, algorytm detekcji, poprawność i kompletność algorytmu (false positives - niepotrzebne ostrzeżenia, false negatives - niezauważone błędy),

8. Pamięć transakcyjna: warunkowe regiony krytyczne (CCR) i inne konstrukcje językowe, niskopoziomowe operacje pamięci transakcyjnej, implementacja CCR przy użyciu tych operacji, struktura stosu, ownership records i deskryptory transakcji, algorytm atomowego zapisu do pamięci transakcyjnej (na przykładach),

9. Poprawność programowej pamięci transakcyjnej: klasyczne własności i ich ograniczenia w kontekście pamięci transakcyjnej: liniowość, szeregowalność (serializability), globalna atomowość (niepodzielność) i odtwarzalność (recoverability), model pamięci transakcyjnej, opacity jako własność bezpieczeństwa (safety) pamięci transakcyjnej,

10. Ograniczenia pamięci transakcyjnej i transakcji: problemy przy zamianie zamków na optymistyczne transakcje, silna vs. słaba atomowość a poprawność programów, problem z metodami natywnymi (operacje nieodwracalne), problem z sekwencyjną kompozycją transakcji. Porównanie teoretycznej prędkości wykonania transakcji atomowych i sekcji krytycznych,

11. 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, formalizacja modelu, przykłady kontrowersyjnych transformacji kodu programów,

12. Bezpieczne programowanie równoległe na przykładzie Cilk (ze wsparciem dla C/C++): abstrakcje programistyczne w Cilk, model obliczeń wielowątkowych w oparciu o graf skierowany acykliczny (DAG), miary wykonania na procesorze wielordzeniowym, szeregowanie zachłanne i górne ograniczenia na czas obliczeń,

13. Bezpieczne wsadowe obliczenia rozproszone w dużej skali na przykładzie Google: technika rozproszonego (równoległego) mapowania i redukcji (map-reduce), podstawowe konstrukcje programistyczne, architektura systemu, odporność na awarie, transparentność,

14. Bezpieczne programowanie rozproszone w modelu przesyłania komunikatów: model rozproszonych aktorów (Erlang) lub obiektów (NPict), operacje przesyłania komunikatów wbudowane w język programowania, weryfikacja poprawności komunikacji sieciowej przez typy, mobilność procesów (NPict), [lab]

15. Minitransacje - alternatywne względem message passing podejście do budowy systemów rozproszonych na przykładzie Sinfonia: semantyka i przykłady minitranskacji, caching i spójność, tolerancja awarii, architektura systemu, protokół zatwierdzania minitransakcji.

Co roku powyższa lista jest rozszerzana o tematy dodatkowe z zakresu najnowszej wiedzy. Zagadnienia oznaczone przez [lab] są realizowane w ramach ćwiczeń laboratoryjnych.

Metody dydaktyczne:
1. wykład: prezentacja multimedialna, prezentacja ilustrowana przykładami na tablicy, demonstracja na komputerze,

2. ćwiczenia laboratoryjne: ćwiczenia praktyczne, dyskusja, praca w zespole, studium przypadków, demonstracja na komputerze.

Metody dydaktyczne

Wykład: prezentacja multimedialna ilustrowana przykładami na tablicy, demonstracja na komputerze, dyskusja moderowana przez prowadzącego.
Ćwiczenia laboratoryjne: prezentacja multimedialna, omówienie przykładów na tablicy lub na komputerze, ćwiczenia praktyczne przy komputerze polegające na wykonaniu przez studentów zadań, praca w zespole, dyskusja moderowana przez prowadzącego, praca własna studentów w celu przygotowania projektu programistycznego oraz analiza powstałego kodu przy udziale prowadzącego.

Literatura

Podstawowa

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

1. Developing Applications with OCaml, Emmanuel Chailloux, Pascal Manoury and Bruno Pagano, O'Reilly France, English translation
2. An Introduction to Programming with C# Threads. Andrew D. Birrell
3. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. Leslie Lamport
4. Linearizability: a correctness condition for concurrent objects. Maurice P. Herlihy, Jeannette M. Wing
5. Language Support for Lightweight Transactions.Tim Harris, Keir Fraser
6. On the Correctness of Transactional Memory Rachid Guerraoui, Michał Kapałka
7. General and Efficient Locking without Blocking. Yannis Smaragdakis, Anthony Kay, Reimer Behrends, Michal Young
8. Subtleties of Transactional Memory Atomicity Semantics. Colin Blundell, E Christopher Lewis, Milo M. K. Martin
9. Deconstructing Transactional Semantics: The Subtleties of Atomicity. Colin Blundell, E Christopher Lewis, Milo M. K. Martin
10. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson
11. High-level Data Races. Cyrille Artho, Klaus Havelund, Armin Biere
12. A Minicourse on Multithreaded Programming. Charles E. Leiserson, Harald Prokop
13. Erlang - A survey of the language and its industrial applications. Joe Armstrong
14. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages. Paweł T. Wojciechowski
15. The Java Memory Model. Jeremy Manson, William Pugh, Sarita V. Adve
16. Sinfonia: A New Paradigm for Building Scalable Distributed Systems. Marcos K. Aguilera, Arif Merchant, Mehul Sha

17. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages, Paweł T. Wojciechowski

Uzupełniająca

1. Threading in C#. Joseph Albahari
2. Exceptions and side-effects in atomic blocks. Tim Harris
3. Process structuring, synchronization, and recovery using atomic actions. David Lomet
4. Transactions are Back-but How Different They Are? Relating STM and Database Consistency Conditions Hagit Attiya, Sandeep Hans
5. Pathological Interaction of Locks with Transactional Memory. Haris Volos, Neelam Goyal, Michael M. Swift
6. Ad Hoc Synchronization Considered Harmful Weiwei Xiong, Soyeon Park, Jiaqi Zhang, Yuanyuan Zhou, Zhiqiang Ma
7. Mnesia A Distributed Robust DBMS for Telecommunications Applications. Hakan Mattsson, Hans Nilsson, Claes Wikstrom
8. Threads Cannot be Implemented as a Library. Hans-J. Boehm
9. The Java Memory Model is Fatally Flawed. William Pugh
10. A Classification of Concurrency Failures in Java Components. Brad Long, Paul Strooper
11. Google's MapReduce Programming Model -- Revisited. Ralf Lammel
12. Atomizer: A Dynamic Atomicity Checker For Multithreaded Programs. Cormac Flanagan, Stephen N. Freund
13. Concurrent Haskell. Simon Peyton Jones, Andrew Gordon, Sigbjorn Finne
14. Developing a High-Performance Web Server in Concurrent Haskell. Simon Marlow
15. Beautiful concurrency. Simon Peyton Jones
16. Transactions with Isolation and Cooperation. Yannis Smaragdakis Anthony Kay Reimer Behrends Michal Young
17. Ownership Types for Safe Programming: Preventing data races and deadlocks. Chandrasekhar Boyapati, Robert Lee, Martin Rinard
18. Types for Atomicity: Static Checking and Inference for Java. Cormac Flanagan, Stephen N. Freund, Marina Lifshin, Shaz Qadeer
19. Semantics of Transactional Memory and Automatic Mutual Exclusion. Martin Abadi, Andrew Birrell,

Bilans nakładu pracy przeciętnego studenta

	Godzin	ECTS
Łączny nakład pracy	125	5
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	64	2,5
Praca własna studenta (studia literaturowe, przygotowanie do zajęć laboratoryjnych/ćwiczeń, przygotowanie do kolokwiów/egzaminu, wykonanie projektu)[1]	61	2,5

[1]niepotrzebne skreślić lub dopisać inne czynności