Testowanie programów współbieżnych

CODERS SCHOOL

https://coders.school

Łukasz Ziobroń
lukasz@coders.school

Łukasz Ziobroń

Not only programming experience:

- C++ and Python developer @ Nokia & Credit Suisse
- Scrum Master @ Nokia & Credit Suisse
- Code Reviewer @ Nokia
- Webmaster (HTML, PHP, CSS) @ StarCraft Area

Training experience:

- C++ trainings @ Coders School
- Practial Aspects Of Software Engineering @ PWr, UWr
- Nokia Academy @ Nokia
- Internal corporate trainings

Public speaking experience:

- Academic Championships in Team Programming
- code::dive conference
- code::dive community

Testowanie programów współbieżnych - Agenda

- Testowanie współbieżnych programów - problemy
- Wielokrotne powtórzenia
- Testy na różnych platformach
- Testy obciążeniowe
- Symulowane środowisko
- Narzędzia do testowania współbieżności
- Inne problemy

Testowanie współbieżnych programów - problemy

- Jeśli testy na programach wielowątkowych przechodzą, to nie ma 100% pewności, że są poprawne
- Często do przetestowania różnych scenariuszy potrzeba napisać dodatkowe programy / moduły. Np. aplikację łączącą się przez sieć trzeba przetestować w poniższych przypadkach:
  - udane połączenie
  - nieudane połączenie
  - utracone połączenie
  - utracone pakiety
  - uszkodzone pakiety
  - inne?
- Często nie można przewidzieć wszystkich możliwych scenariuszy zachowań lub zależności czasowych
- Najczęstszymi problemami są:
  - wyścigi
  - zakleszczenia
  - zagłodzenia
  - fałszywe współdzielenie

Wielokrotne uruchomienia

- Wielokrotne i bardzo częste uruchamianie testów

- Problemy:
  - Uruchamianie w pętli tych samych zestawów testowych często (~99,9%) jest tak samo kolejkowane za każdym razem
  - Koszty energii elektrycznej

Testy na różnych platformach

- Uruchamianie testów na różnych architekturach sprzętowych
  - Inna architektura procesora może skutkować przykładowo innym sposobem dostępu do współdzielonych danych
- Uruchamianie testów na różnych systemach operacyjnych
  - Schedulery mogą inaczej kolejkować zadania

- Problemy:
  - Koszty sprzętu

Testy obciążeniowe (Stress tests)

- Uruchamianie testów przy bardzo wysokim obciążeniu procesora
  - Brak zasobów do wykonania zadań często skutkuje innym ich zakolejkowaniem
- Uruchamianie testów przy zmiennym obciążeniu procesora

- Problemy:
  - Koszty energii elektrycznej
  - Szybsze zużywanie się sprzętu (RAM, dyski twarde)

Symulowane środowisko

- Symulacja wolnego łącza internetowego, zerwanych połączeń, utraconych pakietów
- Symulacja pełnych buforów sieciowych
- Symulacja pełnego dysku twardego (/dev/full),
- Gotowe rozwiązania (np. docker swarm)

- Problemy:
  - odpowiednia infrastruktura
  - czas na konfigurację infrastruktury

Narzędzia do testowania współbieżności

- Thread Sanitizer / Address Sanitizer
  - W pakiecie z kompilatorem (g++, clang++)
  - Nie wymaga dodatkowych konfiguracji / ustawień / pracy
  - Nie działa pod Windowsem ☹
  - Używaj **zawsze** pod Linuxem / Mac jeśli program jest wielowątkowy
  - Thread Sanitizer – typowe problemy wielowątkowości
  - Address Sanitizer – problemy z pamięcią, ale też mogą dotyczyć wielowątkowości
  - Użycie: dodanie flagi kompilacji `-fsanitize=thread` lub `-fsanitize=address`
  - https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual
  - https://github.com/google/sanitizers/wiki/AddressSanitizer

Narzędzia do testowania współbieżności

- Helgrind / DRD / Valgrind
  - Valgrind podmienia standardowe alokatory i dealokatory pamięci (operatory new i delete) na swoje własne, służące do sprawdzania wycieków pamięci
  - Valgrind jest jednowątkowy / synchroniczny, przez co dostęp do pamięci w programie pod valgrindem jest synchroniczny i wszystkie operacje są kolejkowane
  - Uruchamianie programów wielowątkowych pod samym Valgrindem w ramach CI (Continuous Integration) może pomóc wykryć wiele problemów
  - Helgrind (Thread Error Detector) oraz DRD (Data Race Detector) to część pakietu narzędzi valgrind
  - Działanie podobne do TSANa
  - Nie działa pod Windowsem ☹
  - Użycie:
    - Valgrind: `valgrind path/to/bin`
    - Helgrind: `valgrind --tool=helgrind path/to/bin`
    - DRD: `valgrind --tool=drd path/to/bin`
  - http://valgrind.org/docs/manual/hg-manual.html
  - http://valgrind.org/docs/manual/drd-manual.html

Inne problemy

- Nie ma programów testujących fałszywe współdzielenie, bo może ono zależeć od architektury procesora
- W testach unikaj jakiegokolwiek czekania za pomocą timeoutów na innych procedurach, jeśli nie ma dokładnych wymagań czasowych ich działania.
  - Używaj zmiennych warunkowych lub future do komunikacji

```
template<typename T>
bool waitForFuture(std::future<T>& f, std::chrono::seconds timeout = std::chrono::seconds{5})
{
    return f.wait_for(timeout) == std::future_status::ready;
}
```

- Testowanie pod Windowsem – MS CHESS

- Przykłady testów jednostkowych do aplikacji współbieżnych: https://github.com/el-bart/but/tree/master/src/But/Threading

CODERS SCHOOL

https://coders.school

Łukasz Ziobroń
lukasz@coders.school