Instrukcja laboratoryjna **4**	**Przetwarzanie / Systemy odporne na błędy**
	Temat: Programowanie wielowersyjne
	Przygotował: mgr inż. Michał Młodawski

1. Definicja i koncepcja

Programowanie wielowersyjne, znane również jako N-version programming (NVP) lub multi-version programming, to zaawansowana technika inżynierii oprogramowania, która ma na celu zwiększenie niezawodności i bezpieczeństwa systemów komputerowych. Koncepcja ta została po raz pierwszy zaproponowana przez Algirdasa Avižienisa w 1977 roku jako metoda tolerowania błędów w oprogramowaniu.

Główna idea programowania wielowersyjnego polega na niezależnym opracowaniu kilku różnych wersji lub implementacji tego samego komponentu oprogramowania. Te różne wersje są następnie wykonywane równolegle, a ich wyniki są porównywane w celu wykrycia i potencjalnego skorygowania błędów.

1.1.Cel i uzasadnienie

Główne cele

1. **Zwiększenie niezawodności**: Poprzez wykorzystanie wielu implementacji, programowanie wielowersyjne ma na celu zmniejszenie prawdopodobieństwa awarii systemu spowodowanej pojedynczym błędem w oprogramowaniu.
2. **Poprawa bezpieczeństwa**: W systemach krytycznych dla bezpieczeństwa, takich jak kontrola lotów czy elektrownie jądrowe, nawet drobne błędy mogą mieć katastrofalne konsekwencje. Programowanie wielowersyjne służy jako dodatkowa warstwa ochrony.
3. **Wykrywanie błędów**: Porównując wyniki różnych implementacji, można wykryć rozbieżności, które mogą wskazywać na błędy w jednej lub więcej wersji.
4. **Tolerancja błędów**: System może kontynuować działanie nawet w przypadku awarii jednej z wersji, opierając się na wynikach pozostałych implementacji.

Uzasadnienie stosowania

Programowanie wielowersyjne opiera się na założeniu, że niezależnie opracowane wersje oprogramowania będą miały różne błędy. To założenie wynika z obserwacji, że programiści, pracując niezależnie, prawdopodobnie popełnią różne błędy, a nie te same. Dzięki temu, gdy jedna wersja zawiedzie, inne mogą nadal działać poprawnie.

1.2.Metodologia i proces implementacji

Kluczowe etapy

1. **Specyfikacja wymagań**: Tworzenie szczegółowej i jednoznacznej specyfikacji funkcjonalnej dla komponentu.
2. **Niezależny rozwój**: Kilka zespołów lub programistów niezależnie implementuje tę samą specyfikację.
3. **Różnorodność w implementacji**: Zachęca się do używania różnych:
    - Języków programowania
    - Algorytmów
    - Struktur danych
    - Narzędzi programistycznych
4. **Integracja**: Łączenie różnych wersji w jeden system z mechanizmem głosowania lub porównywania wyników.
5. **Testowanie**: Rygorystyczne testowanie całego systemu, w tym mechanizmu porównywania/głosowania.

Mechanizmy porównywania wyników

1. **Głosowanie większościowe**: Najprostszy mechanizm, gdzie wynik pojawiający się najczęściej jest uznawany za poprawny.
2. **Średnia z odrzuceniem skrajnych**: Obliczanie średniej z wyników, odrzucając wartości skrajne.
3. **Konsensus**: Wymaganie, aby wszystkie lub większość wersji zgadzały się co do wyniku.

1.3.Zastosowania w różnych dziedzinach

Lotnictwo i astronautyka

- **Systemy kontroli lotu**: Krytyczne systemy w samolotach i statkach kosmicznych często wykorzystują programowanie wielowersyjne.
- **Przykład**: System kontroli lotu Boeinga 777 wykorzystuje trzy różne wersje oprogramowania.

Energetyka jądrowa

- **Systemy bezpieczeństwa reaktorów**: Monitorowanie i kontrola parametrów reaktora jądrowego.
- **Systemy awaryjnego wyłączania**: Zapewnienie niezawodnego wyłączenia reaktora w sytuacjach krytycznych.

Medycyna

- **Systemy wspomagania decyzji medycznych**: Diagnostyka i zalecenia leczenia oparte na wielu algorytmach.
- **Kontrola urządzeń medycznych**: Zapewnienie bezpieczeństwa pacjentów korzystających z zaawansowanych urządzeń medycznych.

Systemy finansowe

- **Transakcje giełdowe**: Zapewnienie integralności i bezpieczeństwa transakcji o wysokiej wartości.
- **Systemy bankowe**: Ochrona przed błędami w przetwarzaniu danych finansowych.

1.4.Zalety i korzyści

1. **Zwiększona niezawodność**: Poprzez wykorzystanie wielu implementacji, system staje się bardziej odporny na awarie.
2. **Lepsza detekcja błędów**: Rozbieżności między wersjami mogą wskazywać na błędy, które w przeciwnym razie mogłyby pozostać niewykryte.
3. **Ciągłość działania**: System może kontynuować pracę nawet w przypadku awarii jednej z wersji.
4. **Ochrona przed błędami systematycznymi**: Różnorodność implementacji zmniejsza ryzyko błędów wynikających z konkretnego podejścia do rozwiązania problemu.
5. **Zwiększone zaufanie**: W systemach krytycznych, programowanie wielowersyjne może zwiększyć zaufanie użytkowników i organów regulacyjnych.
6. **Potencjalne oszczędności długoterminowe**: Mimo wyższych kosztów początkowych, zapobieganie poważnym awariom może przynieść znaczące oszczędności w dłuższej perspektywie.

1.5.Najlepsze praktyki i rekomendacje

1. **Jasna i szczegółowa specyfikacja**: Kluczowe jest stworzenie jednoznacznej specyfikacji funkcjonalnej, która minimalizuje ryzyko wspólnych błędów interpretacyjnych.
2. **Maksymalizacja różnorodności**: Zachęcanie do stosowania różnych języków programowania, narzędzi i podejść do rozwiązywania problemów.
3. **Niezależne zespoły**: Zapewnienie, że zespoły pracujące nad różnymi wersjami nie komunikują się ze sobą w kwestiach implementacyjnych.
4. **Rygorystyczne testowanie**: Każda wersja powinna przejść dokładne testy jednostkowe i integracyjne przed połączeniem w system.
5. **Zaawansowane mechanizmy porównywania**: Inwestycja w skuteczne algorytmy porównywania wyników i wykrywania anomalii.
6. **Ciągła walidacja**: Regularne sprawdzanie i walidacja systemu w trakcie jego eksploatacji.
7. **Dokumentacja i śledzenie zmian**: Dokładne dokumentowanie każdej wersji i wszystkich zmian wprowadzanych w systemie.

Studia przypadków

System kontroli lotu Airbus A330/340

Airbus zastosował programowanie wielowersyjne w systemie kontroli lotu dla swoich modeli A330 i A340. System składa się z pięciu niezależnych komputerów, z których każdy używa różnego sprzętu i oprogramowania. Trzy z tych komputerów są podstawowe, a dwa służą jako zapasowe. Każdy komputer jest zdolny do samodzielnego sterowania samolotem, a system głosowania decyduje, który z nich ma kontrolę w danym momencie.

Reaktor jądrowy Olkiluoto 3 w Finlandii

W elektrowni jądrowej Olkiluoto 3 zastosowano programowanie wielowersyjne w systemach bezpieczeństwa. Cztery niezależne podsystemy bezpieczeństwa, każdy zaprojektowany przez inny zespół, monitorują i kontrolują krytyczne parametry reaktora. System wykorzystuje głosowanie 2-z-4, co oznacza, że co najmniej dwa z czterech podsystemów muszą zgodzić się na podjęcie działania.

Mars Pathfinder

NASA zastosowała elementy programowania wielowersyjnego w misji Mars Pathfinder. Łazik wykorzystywał trzy różne metody nawigacji: dead reckoning, odometria wizualna i śledzenie słońca. Każda z tych metod była implementowana niezależnie, a system nawigacyjny wykorzystywał kombinację ich wyników do określenia pozycji łazika.

Programowanie wielowersyjne stanowi potężne narzędzie w arsenale inżynierów oprogramowania, szczególnie w kontekście systemów o krytycznym znaczeniu dla bezpieczeństwa. Chociaż wiąże się z wieloma wyzwaniami, w tym zwiększonymi kosztami i złożonością, korzyści w postaci zwiększonej niezawodności i bezpieczeństwa mogą być nieocenione w odpowiednich zastosowaniach.

Technika ta nie jest panaceum na wszystkie problemy związane z niezawodnością oprogramowania, ale w połączeniu z innymi metodami, takimi jak formalna weryfikacja, testowanie i rygorystyczne procesy rozwoju, może znacząco przyczynić się do tworzenia bardziej niezawodnych i bezpiecznych systemów.

W miarę jak systemy komputerowe stają się coraz bardziej zintegrowane z krytycznymi aspektami naszego życia, od transportu po opiekę zdrowotną, znaczenie technik takich jak programowanie wielowersyjne będzie prawdopodobnie rosło. Jednocześnie, postęp w dziedzinach takich jak sztuczna inteligencja i obliczenia kwantowe otwiera nowe możliwości i wyzwania w kontekście programowania wielowersyjnego.

Ostatecznie, skuteczne wdrożenie programowania wielowersyjnego wymaga starannego rozważenia kompromisów między kosztami, złożonością a potencjalnymi korzyściami. W odpowiednich kontekstach, gdzie stawką jest ludzkie życie lub krytyczne zasoby, inwestycja w tę technikę może okazać się nie tylko uzasadniona, ale wręcz niezbędna.

2. Mechanizmy porównywania wyników w programowaniu wielowersyjnym

2.1.Głosowanie większościowe

Głosowanie większościowe to najprostszy i najczęściej stosowany mechanizm porównywania wyników w programowaniu wielowersyjnym.

Zasada działania:

- System zbiera wyniki ze wszystkich dostępnych wersji oprogramowania.
- Wynik, który pojawia się najczęściej (ma największą liczbę "głosów"), jest uznawany za poprawny.
- W przypadku równej liczby głosów, system może mieć zdefiniowaną dodatkową regułę decyzyjną.

Przykład:

Załóżmy, że mamy 5 wersji oprogramowania kontrolującego temperaturę w reaktorze:

- Wersja A: 100°C
- Wersja B: 100°C
- Wersja C: 101°C

- Wersja D: 100°C
- Wersja E: 105°C

W tym przypadku, wynik 100°C zostałby uznany za poprawny, ponieważ pojawia się najczęściej (3 razy).

Zalety:

1. Prostota implementacji i zrozumienia.
2. Efektywność w przypadku jednoznacznych wyników.
3. Szybkość podejmowania decyzji.

Wady:

1. Może zawodzić w przypadku subtelnych błędów, które wpływają na większość wersji.
2. Nie uwzględnia stopnia rozbieżności między wynikami.
3. Może być problematyczny przy parzystej liczbie wersji.

Zastosowania:

- Systemy, gdzie wyniki są dyskretne lub binarne.
- Przypadki, gdzie szybkość decyzji jest kluczowa.
- Jako podstawowy mechanizm w prostszych systemach wielowersyjnych.

2.2.Średnia z odrzuceniem skrajnych

Ten mechanizm jest bardziej zaawansowany i może być szczególnie użyteczny w systemach, gdzie wyniki są ciągłe (np. pomiary fizyczne) i mogą występować sporadyczne duże odchylenia.

Zasada działania:

1. System zbiera wyniki ze wszystkich dostępnych wersji.
2. Identyfikuje i odrzuca wartości skrajne (outliers).
3. Oblicza średnią z pozostałych wyników.
4. Ta średnia jest uznawana za poprawny wynik.

Metody identyfikacji wartości skrajnych:

1. **Metoda percentyli**: Odrzucanie wyników poniżej 10-go i powyżej 90-go percentyla.
2. **Metoda odchylenia standardowego**: Odrzucanie wyników oddalonych o więcej niż 2 lub 3 odchylenia standardowe od średniej.

3. **Metoda IQR (Interquartile Range)**: Odrzucanie wyników poniżej Q1 - 1.5*IQR i powyżej Q3 + 1.5*IQR, gdzie Q1 i Q3 to odpowiednio pierwszy i trzeci kwartyl, a IQR to różnica między nimi.

Przykład:

Załóżmy, że mamy 7 wersji oprogramowania mierzących ciśnienie w zbiorniku:

- Wersja A: 10.2 MPa
- Wersja B: 10.1 MPa
- Wersja C: 10.3 MPa
- Wersja D: 10.2 MPa
- Wersja E: 10.2 MPa
- Wersja F: 9.5 MPa (wartość skrajna)
- Wersja G: 11.5 MPa (wartość skrajna)

Po odrzuceniu wartości skrajnych (F i G), średnia z pozostałych wyników wynosi 10.2 MPa, co zostałoby uznane za poprawny wynik.

Zalety:

1. Większa odporność na pojedyncze duże błędy lub awarie.
2. Uwzględnia stopień rozbieżności między wynikami.
3. Dobrze radzi sobie z ciągłymi danymi wyjściowymi.

Wady:

1. Może być mniej efektywny przy małej liczbie wersji.
2. Wymaga starannego doboru metody identyfikacji wartości skrajnych.
3. Może być wrażliwy na subtelne, systematyczne błędy wpływające na większość wersji.

Zastosowania:

- Systemy pomiarowe w przemyśle.
- Kontrola procesów w chemii i inżynierii.
- Systemy nawigacyjne, gdzie dokładność jest kluczowa.

3. Konsensus

Mechanizm konsensusu wymaga, aby wszystkie lub zdecydowana większość wersji zgadzała się co do wyniku. Jest to najbardziej rygorystyczne podejście, które może zapewnić najwyższy poziom pewności, ale też może być najbardziej restrykcyjne.

Zasada działania:

1. System zbiera wyniki ze wszystkich dostępnych wersji.
2. Sprawdza, czy wszystkie (lub określona większość) wersje zgadzają się co do wyniku.
3. Jeśli warunek konsensusu jest spełniony, wynik jest akceptowany.
4. Jeśli konsensus nie zostanie osiągnięty, system może:
    - Zażądać ponownego obliczenia
    - Przejść w tryb bezpiecznego zatrzymania
    - Uruchomić alternatywną procedurę decyzyjną

4. Warianty konsensusu:

1. **Pełny konsensus**: Wymaga zgody wszystkich wersji.
2. **Konsensus większościowy**: Wymaga zgody określonego procentu wersji (np. 80% lub więcej).
3. **Konsensus z tolerancją**: Wymaga, aby wyniki mieściły się w określonym przedziale tolerancji.

Przykład:

Załóżmy, że mamy system sterowania reaktorem jądrowym z 5 wersjami oprogramowania decydującymi o konieczności wyłączenia awaryjnego:

- Wersja A: Wyłączyć
- Wersja B: Wyłączyć
- Wersja C: Wyłączyć
- Wersja D: Nie wyłączać
- Wersja E: Wyłączyć

Przy wymogu 80% konsensusu, decyzja o wyłączeniu zostałaby podjęta (4 z 5 wersji zgadzają się).

Zalety:

1. Najwyższy poziom pewności w przypadku osiągnięcia konsensusu.
2. Minimalizuje ryzyko podjęcia błędnej decyzji.

3. Szczególnie użyteczny w systemach krytycznych dla bezpieczeństwa.

Wady:

1. Może prowadzić do impasu decyzyjnego, jeśli konsensus nie zostanie osiągnięty.
2. Wymaga precyzyjnego zdefiniowania warunków konsensusu.
3. Może być mniej efektywny w systemach wymagających szybkiego podejmowania decyzji.

Zastosowania:

- Systemy kontroli w elektrowniach jądrowych.
- Krytyczne systemy w lotnictwie i kosmonautyce.
- Systemy finansowe wysokiej częstotliwości, gdzie błędy mogą mieć katastrofalne skutki.

Porównanie mechanizmów

Mechanizm	Złożoność	Odporność na błędy	Szybkość decyzji	Zastosowania
Głosowanie większościowe	Niska	Średnia	Wysoka	Systemy binarne, szybkie decyzje
Średnia z odrzuceniem skrajnych	Średnia	Wysoka	Średnia	Pomiary ciągłe, systemy z potencjalnymi dużymi odchyleniami
Konsensus	Wysoka	Bardzo wysoka	Niska	Systemy krytyczne dla bezpieczeństwa, gdzie pewność jest kluczowa

Wybór odpowiedniego mechanizmu porównywania wyników w programowaniu wielowersyjnym zależy od specyfiki systemu, wymagań dotyczących bezpieczeństwa, rodzaju danych wyjściowych oraz kontekstu operacyjnego. W praktyce, zaawansowane systemy często łączą różne mechanizmy lub stosują je hierarchicznie, aby zoptymalizować równowagę między niezawodnością, szybkością i dokładnością podejmowania decyzji.