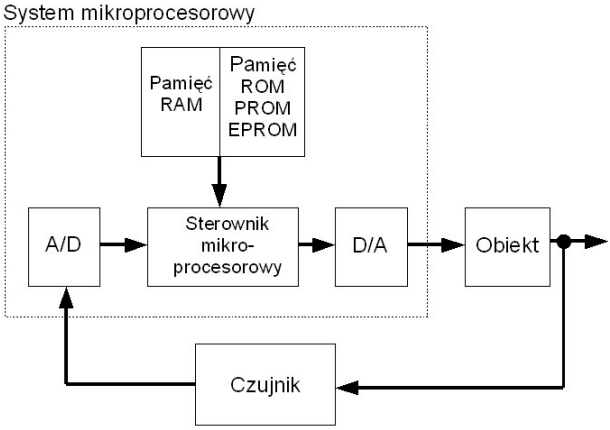
AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Magdalena Szymczyk*, Piotr Szymczyk*

Techniki programistyczne stosowane w niezawodnych komputerowych systemach sterowania

1. Wprowadzenie

W wyniku rozwoju technologii mikroprocesorowej implementacja algorytmów sterowania jest dzisiaj realizowana z pomocą wbudowanych mikrokontrolerów, a nie analogowo. Typowy system sterowany mikroprocesorowo jest przedstawiony na rysunku 1.



Rys. 1. Mikroprocesorowe sterowanie obiektem

Algorytmy sterowania są realizowane w różnych językach programowania, zarówno tych niskiego poziomu jak kod maszynowy danej jednostki CPU, jak i trochę wyższego – w asemblerze. Ich zaletą jest szybkość działania. Częściej niż dawniej (ze względu na wydajność procesorów) stosuje się teraz języki wysokiego poziomu takie jak C czy C++, które powoli stają się standardami dla tego typu systemów. Wykorzystanie mikroprocesorów do

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

systemów sterowania daje możliwość ich szybkiego i w miarę prostego modyfikowania. Inną zaletą jest to, że algorytmy sterowania mogą być bardzo złożone [1].

Jednym z atrybutów wiarygodności (*dependability*) funkcjonowania systemu komputerowego (własność pozwalająca jego użytkownikom mieć uzasadnione zaufanie do oferowanych przez niego usług) jest niezawodność, czyli zdolność systemu do dostarczenia poprawnej usługi, w konkretnym przedziale czasu.

Niezawodność systemów komputerowych związana jest z następującymi technikami:

- unikania błędów lub ich prewencja,
- detekcja i eliminacja błędów,
- przewidywanie obecności i występowaniem błędów i określanie ich konsekwencji,
- tolerancją uszkodzeń, czyli możliwość poprawnej pracy systemu (zapewnienie usługi zgodnej ze specyfikacją) przy obecności błędów.

W tym artykule przedstawione zostaną mechanizmy dotyczące detekcji błędów oraz te dające możliwość poprawnej pracy sytemu przy ich wystąpieniu.

2. Tolerancja uszkodzeń

Uszkodzenia w wbudowanych systemach sterowania mogą być trwałe (*permanent*) – powstałe w wyniku np. mechanicznych uszkodzeń komponentów systemu i przemijające (zewnętrzne – *transient*, ich skutki wychodzą poza granice systemu oraz wewnętrzne – *intermittent*, skutki pozostają wewnątrz systemu). Czas trwania tych ostatnich jest krótki, ale efekt ich działania może być równie poważny jak w przypadku uszkodzeń trwałych. Mogą one na przykład zmieniać sposób przepływu sterowania w programie, czy też modyfikować dane zapisane w pamięci układu, a to może skutkować jego nieodpowiednim działaniem. Częstotliwość wystąpienia uszkodzeń przemijających jest dużo większa niż trwałych (na przykład w systemach wbudowanych stosowanych w motoryzacji), w związku z tym nie stosuje się tutaj kosztownych rozwiązań narzędziowych dla zapewnienia odporności na uszkodzenia trwałe.

Techniki programistyczne tolerancji uszkodzeń zdobywają coraz większą popularność. Takie rozwiązania zazwyczaj zmniejszają wydajność systemu komputerowego znacznie mniej w porównaniu z rozwiązaniami sprzętowymi, ze względu na nakłady czasowe związane z wykonaniem zaimplementowanej techniki programistycznej. Czasami też takie metody nie posiadają odpowiednich możliwości wpływu na sprzęt, aby pokrycie błędów było wystarczające. Mimo to metody programistyczne są atrakcyjnym rozwiązaniem. Do podstawowych zalet technik softwarowych należy zaliczyć możliwość wykorzystania sprzętu ogólnie dostępnego lub po jego niewielkiej modyfikacji. Takie technologie są dużo tańsze, jeśli chodzi o koszty zastosowania. Dla tak zwanych błędów miękkich (przejściowych) czas odtwarzania stanu systemu nie jest wartością krytyczną, gdyż pojawiają się one

w przeciągu dni, miesięcy, lat, a nie w każdej mikrosekundzie. Poziom zabezpieczenia też może być dowolnie dopasowywany do potrzeb konkretnego układu, gdy bierze się pod uwagę wpływ na wydajność systemu jako całości. Techniki programistyczne mające na celu tolerancję uszkodzeń mogą być wdrażane na etapie tworzenia oprogramowania lub też rozwijania go. Mogą one radzić sobie nie tylko z błędami projektowymi oprogramowania, ale też z błędami przejściowymi. Gdy taki błąd się pojawi, to posiadają one mechanizmy, które dają możliwość tolerowania błędu.

Jest aktualnie wiele technik programistycznych, które implementują tolerancję uszkodzeń. Można je ogólnie podzielić je na:

- bazujące na jednym środowisku programistycznym (monitory, niepodzielne akcje, weryfikacja decyzji czy też obsługa przerwań);
- bazujące na wielu środowiskach programistycznych – zróżnicowanie sposobu projektowania środowiska programistycznego (bloki nawrotu, programowanie N-krotne, programowanie N-krotne samosprawdzające);
- bazujące na zróżnicowaniu sposobu dostarczenia danych wejściowych (bloki powtarzania, programowanie N-kopii).

Techniki programistyczne dotyczą zarówno sposobów detekcji błędów, jak i metod przywracania stanu poprawnej pracy po zaistniałym błędzie i mogą one być wdrażane na różnych poziomach systemu [2]. Moga one być realizowane na poziomie aplikacji, systemu operacyjnego czy też maszyny wirtualnej, która jest abstrakcją sprzętu i może obsługiwać kilka systemów operacyjnych. Każde z rozwiązań ma swoje plusy i minusy, ale w tym artykule przedstawione zostaną metody poziomu aplikacji, ze względu na łatwą dostępność dla programisty.

Metody przywracania poprawnej pracy po błędzie dzielą się na te typu:

- odtwarzanie wsteczne,
- odtwarzanie wyprzedzające.

Głównym ich założeniem jest zastosowanie czasowej oraz informacyjnej redundancji w kodzie wykonywanego programu.

3. Odtwarzanie wsteczne

W momencie wystąpienia błędu system zazwyczaj wchodzi w nieprawidłowy stan, techniki nawrotu próbują odtworzyć stan wolny od błędu lub poprawny. Metody odtwarzania wstecznego dokonują tego poprzez powrót do ostatnio zachowanego stanu, jeśli założy się, że ten jest wolny do błędu. Informacje o stanie systemu zbierane są w ściśle określonych momentach czasu i określa się je jako punkty kontrolne. Informacje te zapisywane są w pamięci stałej wolnej od błędów. Po detekcji błędu stan systemu jest odtwarzany dla ostatniego zachowanego punktu kontrolnego.

4. Odtwarzanie wyprzedzające

Jak wcześniej powiedziano, w momencie wystąpienia błędu system zazwyczaj wchodzi w nieprawidłowy stan i techniki nawrotu próbują odtworzyć stan wolny od błędu lub poprawny. Metody odtwarzania wyprzedzającego próbują zrealizować to poprzez znalezienie stanu, z którego system mógłby kontynuować działanie. Ten stan to może być zdegenerowany stan poprzedniego stanu, wolnego od błędu. Kompensacja błędu bazuje na redundancji sprzętu, informacji, oprogramowania czy też redundancji czasu. Przykładami takich technik są NVP, NCP, DRB (*distributed recovery block*) [3]. Metodę tę stosuje się, gdy nie ma czasu koniecznego przy stosowaniu odtwarzania wstecznego.

5. Typy redundancji stosowane w technikach programistycznych tolerancji uszkodzeń

Jedną z najpowszechniejszych redundancji jest replikacja sprzętowa. Na tych urządzeniach może być zainstalowane to samo oprogramowanie lub też zróżnicowane (ta sama specyfikacja wymagań, ale różne grupy programistów, tworzenie wariantów oprogramowania), co daje możliwość poprawy błędów zarówno programistycznych, jak i sprzętowych. Ta redundancja może istnieć w dowolnych konfiguracjach (pojedyncza, podwójna, itp.) i to zarówno jeśli chodzi o sprzęt, jak i oprogramowanie.

Redundancja softwarowa związana jest z replikacją na poziomie programu, modułu, funkcji, obiektu czy też procesu.

Często stosowane techniki to też:

- replikacja instrukcji przetwarzających dane – zduplikowanie wszystkich zmiennych programu oraz wszystkich instrukcji przetwarzających dane, co w wyniku daje duplikaty rezultatów;
- statyczna kontrola przebiegu sterowania programu – do każdego podstawowego bloku programu przyporządkowuje się odpowiednią sygnaturę oraz także na początku i na końcu każdego bloku, zamieszcza dodatkowe instrukcje sprawdzające aktualną sygnaturę umieszczoną w zmiennej globalnej z tą statyczną;
- replikacja instrukcji rozgałęziających – powielenie wykonania instrukcji rozgałęziającej w każdym z kierunków w celu określenia przewidywanego czasu wykonania i porównania go z rzeczywistym czasem wykonania uruchamianej aktualnie instrukcji.

Redundancja informacyjna lub danych związana jest często z redundancją softwarową. Dane dla systemu mogą być podawane w różny sposób, z różnych źródeł. Przykładem redundancji informacyjnej są kody detekcji błędów lub kody poprawiające błędy.

Czasowa redundancja wykorzystuje dodatkowy czas dla wykonania zadań związanych z tolerancją uszkodzeń. Wykorzystuje się ją dla obydwu typów redundancji programistycznej i sprzętowej.

6. Techniki programistyczne związane z technikami programistycznymi tolerancji uszkodzeń

Detektory błędów na poziomie aplikacji bazują na własnościach określonego algorytmu wykonywanego przez system. Są to między innymi asercje i znaczniki, bazujące na stałych relacjach pomiędzy zmiennymi w programie.

Asercje (*assertions*) – czyli analiza ciągła przeprowadzana jest na poziomie aplikacji lub dodatkowego zewnętrznego urządzenia testującego (dodatkowy procesor) i polega na wprowadzeniu do kodu źródłowego programu wyrażeń warunkowych typu ”if”. Spełnienie wyrażenia powoduje kontynuację działania programu, niespełnienie – przejście do odpowiedniej obsługi błędu.

Sposób wymaga dokładnej znajomości implementacji, dla każdego programu muszą one być zaprojektowane indywidualnie, przez co zapewniają większe współczynniki pokrycia błędów.

Takie rozwiązanie zazwyczaj testuje jakąś własność, na przykład czy nie przekroczono zakresu dostępnej pamięci, co może być przyczyną zarówno błędu programistycznego, jak i sprzętowego.

Znaczniki (*signatures*) – są jednymi z efektywniejszych metod detekcji błędów, w tej technice do kodu programu dokłada się pewien zbiór operacji logicznych wraz z związanymi z nimi, wcześniej ustalonymi znakami sprawdzającymi czy sumami kontrolnymi. W przypadku wystąpienia błędu instrukcje logiczne sygnalizują nieprawidłowy przebieg programu. Tę technikę realizuje się zarówno programistycznie, jak i sprzętowo, poprzez zastosowanie równolegle działającego układu testującego. Ogromną zaletą tego rozwiązania jest to, że wykorzystanie tej metody nie wiąże się z dogłębną znajomością samego programu.

Do innych stosowanych technik programistycznych można zaliczyć metodę punktów kontrolnych i niepodzielne akcje.

Metoda punktów kontrolnych (*checkpointing*) – stosowana jest w technice odtwarzania wstecz. Metoda polega na odtworzeniu stanu systemu w ostatnim prawidłowym momencie pracy. Praca procesora jest ustawiana na poprzedni punkt pracy za pomocą zapisanego wcześniej w pamięci statycznej układu, stanu dla punktu kontrolnego. Takie rozwiązanie zmniejsza znacznie czas potrzebny do powrotu do normalnej pracy. Istnieją różne odmiany tej metody ze względu na moment wykonania punktów kontrolnych. Mogą one być wykonywane równomiernie, w równych odcinkach czasowych (rozwiązanie niezależne od specyfiki aplikacji) lub też w momentach najbardziej do tego właściwych (najkrótszy czas zapisu). Składowanie danych dla punktów kontrolnych zawsze wiąże się z pewnym narzutem czasowym.

Niepodzielne akcje (*atmic actions*) – stosowane są w rozwiązaniach dla systemów równoległych. Grupa komponentów tworzy niepodzielną akcję, gdy nie ma przepływu in-

formacji pomiędzy tymi grupami oraz resztą programu podczas jego działania. Akcja niepodzielna charakteryzuje się [3]:

- niepodzielnością – wszystkie kroki obliczeniowe tej akcji są wykonane albo żaden nie jest wykonany,
- określonym położeniem w uszeregowaniu kroków programu,
- odtwarzalnością – zewnętrzny efekt wykonania kroków obliczeniowych akcji niepodzielnej jest albo go brak.

Te założenia umożliwiają bezpieczne tworzenie metod tolerancji uszkodzeń dla systemów równoległych.

7. Podsumowanie

W większości techniki programistyczne oparte na redundancji softwarowej, informacyjnej i ewentualnie czasowej. Dobór odpowiedniej metody uzależniony jest od rodzaju aplikacji, dostępnych zasobów czy też dostępnych technik. Redundancja czasowa może być stosowana w aplikacjach typu interakcyjnego, lecz dla aplikacji czasu rzeczywistego o twardych wymaganiach czasowych, raczej rozwiązania bazujące na technikach odtwarzania w przód. Na pewno techniki te nie rozwiązują wszystkich problemów z oprogramowaniem, ale wydaje się, że w najbliższej przyszłości wszystkie systemy predestynujące do określenia bezpieczne będą miały wbudowane takie rozwiązania. Wymagają one dokładnej i starannej analizy wymagań dla każdego systemu indywidualnie, wówczas też zastosowane techniki będą lepsze.

Literatura

[1] Burns R., *Advanced Control Engineering*. Butterworth-Heinemann, 2001.
[2] Mukherjee S., *Architecture design for soft errors*. Elsevier, 2008.
[3] Pullum L., *Software Tolerance Techniques and Implementation*. Artech House, 2001.
[4] Wiliams R., *Real-Time Systems Development*. Elsevier, 2006.
[5] Szymczyk M., Szymczyk P., *Programistyczne technologie zwiększające niezawodność systemów czasu rzeczywistego*. Materiały konferencjyjne SCR 2009.
[6] Szymczyk M., Szymczyk P., *Mechanizmy wykrywania anomalii pracy procesora jako metody zwiększające niezawodność systemów czasu rzeczywistego.* Materiały konferencjyjne SCR 2009.