NADZOR PROCESA U SISTEMIMA ZA RAD U REALNOM VREMENU

Milun Jevtić, Milunka Damnjanović, Vladimir Živković, *Elektronski fakultet Niš*

Sadržaj - *U radu se razmatraju tehnike realizacije nadzora procesa u mikroračunarskim sistemima koji rade u realnom vremenu. Dato je i jedno rešenje realizacije nadzora u osnovi zasnovano na softveru, a sa ciljem da se omogući testiranje vremenskih karakteristika sistema u toku razvoja i detektuje svako vremensko prekoračenje u toku rada sistema. Razvijeno rešenje minimalno angažuje procesorskog vremena je pogodno da se implementira i u programski kod i u jezgro operativnog sistema za rad u realnom vremenu.*

1. UVOD

Dodatna poteškoća pri projektovanju sistema za rad u realnom vremenu (Real-Time System - RTS) je dokazivanje korektnog zadovoljavanja vremenskih ograničenja, što je inače posebno važno za sisteme za rad u realnom vremenu sa rigidnim ograničenjima (Hard RTS).

U postupku projektovanja RTS-a često se prave pretpoostavke o ponašanju sistema i njegove okoline. Ove pretpostavke mogu imati različite forme, ali su posebno važne one koje se odnose na vremenska ograničenja. Tako izražavanje vremenskih ograničenja može biti u obliku: gornje granice kašnjenja u komunikaciji između procesa, rokova za izvršenje zadataka, ili minimalnog intervala razdvajanja između pojave dva događaja. Ograničenja se tako definišu da se često odnose na nepredvidivost spoljašnje okoline ili da bi se pojednostavio problem koji je inače nemoguće ili vrlo teško rešiti. Takve pretpostavke mogu se izraziti kao deo formalne specifikacije sistema ili kao zahtevi vremenskog raspoređivanja zadataka u realnom vremenu. I pored uvođenja metoda formalne verifikacije i sadašnjih rezultata u vremenskom planiranju u realnom vremenu, potreba za nadzorom u realnom vremenu ovih sistema nije umanjena iz nekoliko razloga:

1. Okolina izvršenja za mnoge sisteme je nesavršena i interakcija sa spoljašnjim svetom uvodi dodatnu nepredvidivost;
2. Pretpostavke pri projektovanju mogu biti narušene u toku rada sistema zbog neočekivanih uslova, kao što je na primer prekoračenje broja promena;
3. Primena formalnih tehnika ili algoritama vremenskog planiranja zahteva pretpostavke o osnovnom sistemu;
4. Može biti neostvarivo verifikovati formalno neka svojstva u vreme projektovanja.

Dakle, neophodne su provere u toku rada sistema [1] za dokazivanje korektnosti funkcionisanja sistema u realnom vremenu. I to kako kod procesa projektovanja nakon faze integracije, tako i u toku eksploatacije sistema radi povećane pouzdanosti i predvidivog ponašanja sistema prema okolini.

Ovde će biti izložene strategije testiranja RTS-a. Zatim se razmatraju tehnike realizacije nadzora procesa i događaja u njima kod RTS-a. Cilj je obezbediti generalni okvir za nadzor vremenski kritičnih sistema u toku rada. Može se predvideti sistem u kome nadzor u toku rada obezbeđuje povratnu informaciju, tako da se sistem može adaptirati na promene u okolini ili izuzetne uslove koji mogu da se steknu i u njemu.

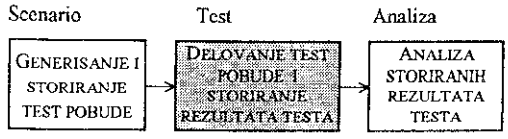
Dato je i jedno rešenje realizacije nadzora u osnovi zasnovano na softveru, a sa ciljem da se omogući testiranje vremenskih karakteristika sistema u toku razvoja i detektuje svako vremensko prekoračenje u toku rada sistema.

2. STRATEGIJE TESTIRANJA

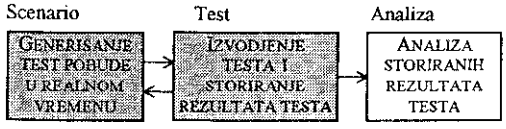
Mnoge sisteme za rad u realnom vremenu nije moguće testirati u pravom okruženju gde treba da funkcionišu. Nije moguće jer su to okruženja sa procesima često opasnim po život ljudi ili velike materijalne vrednosti. Zato će se ovde razmotriti strategije testiranja sistemi za dokazivanje njihovog korektnog funkcionisanja u realnom vremenu.

Postoje tri osnovne strategije za testiranje - procenu korektnog funkcionisanja sistema koji rade u realnom vremenu. Svaka od ovih zahteva da se formira odgovarajuće okruženje i ispita ponašanje RTS-a u njemu.

Sekvencijalno test okruženje prikazano na slici 1. je najmanje složeno. U "off-line" režimu se formira scenario testiranja i generiše test pobuda pre izvršenja test procedure. Podaci o tome se storiraju i onda reprodukuju u toku rada sistema u realnom vremenu. Pri tome se rezultat testa - odziv sistema storira u realnom vremenu a kasnije analizira i donosi zaključak. Nedostatak ovog pristupa je u nemogućnosti dinamičke promene test scenarija i praćenja tok testa. Dalje, uspeh ili neuspeh testa je poznat tek posle analiza. Ali i pored navedenih nedostataka on je često zbog prirode okruženja jedino moguć.

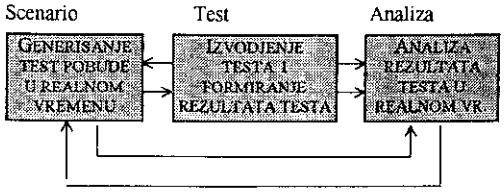


Sl. 1. Sekvencijalna organizacija procesa testiranja



Sl. 2. Testiranje sa generisanjem pobude u realnom vremenu

Komplikovaniji sistem za testiranje odvijanju testa u realnom vremenu pridružuje i generianje podataka za test scenario (slika 2.). U "off-line" režimu se samo analiziraju rezultati testa.



Sl. 3. Testiranje sa odvijanjem svih procesa u realnom vremenu

Jedino konfiguracija na slici 3. obezbeđuje kompletno testiranje u realnom vremenu, odnosno omogućuje "on-line" testiranje sistema za rad u realnom vremenu.

Kod složenijih RTS-a ukazuje se potreba za korišćenje sva tri režima testiranja tokom procesa projektovanja.

3. NADZOR U REALNOM VREMENU

Cilj nadzora u realnom vremenu je da sačuva performanse sistema u opsegu u kome se ne menja redosled i tajming događaja. Sam sistem nadzora je sistem sa ograničenjima u realnom vremenu. Nadzor se može ostvariti na različitim nivoima. Nadzor niskog nivoa pribavlja svaki signal koji se prenosi na basu. Nadzor visokog nivoa detektuje samo događaje na nivou procesa. Instrumentacioni kod insertuje se manuelno ili automatski kao kod tehnika prekidnih tačaka. Za prepoznavanje događaja, instrumentacioni kod se insertuje u odgovarajućim tačkama gde može da generiše informaciju vezanu za događaje od interesa. Nadzirani događaji bili bi događaji koji su indikativni za ponašanje sistema.

Treba praviti razliku između sistemskih ograničenja koja su ugrađena u program za rad u realnom vremenu i ograničenja koje se nadgledaju asinhrono pomoću posebnog zadatka Tako nadzor može da se javi ili sinhrono sa izvršenjem aplikacije ili asinhrono sa izvršenjem. Sinhrona provera, ili provera izjava, zahteva da korisnik doda izjave aplikacionom kodu. Izjave su provere da bi se odredilo da li, za posebne delove softvera, relevantni delovi stanja sistema (na primer vrednosi promenljivih ili I/O signali) su unutar granica potrebnih za taj deo da bi ispravno funkcionisao. Izjave se stavljaju direktno u aplikaciju i mogu biti proverene kad se pojave u toku izvršenja. Ako je potrebna češća provera, mora se koristiti asinhrona provera. Asinhrona provera obavlja se na spoljašnjem procesu koji prima događaje od aplikacije.

Proračun RTS-a može biti viđen kao niz pojava događaja. Informaciono, događaji predstavljaju stvari koje se dešavaju u sistemu. Jedna pojava događaja definiše jednu tačku u vremenu u proračunu u kome se događa poseban primerak događaja. Dakle, vremenske osobine mogu se izraziti kao odnos između pojave događaja u proračunu (izračunavanju).

U ovom modelu razlikujemo dve klase događaja koji mogu da se osmatraju: događaji oznake (Label Events - LE) i događaji prelaza (transition events - TE). LE se koriste za označavanje početka i završetka niza programskih naredbi (oni odgovaraju start/stop događajima u RTL). Oni se definišu insertovanjem specijalizovanih oznaka na odgovarajuća mesta u kodu.

4. REALIZACIJA NADZORA

Postoje tri široka pristupa nadzoru: hardverski, softverski i hibridni pristup.

4.1 SOFTVERSKI PRISTUP NADZORU

Postoje dve softverske tehnike nadzora.

1. Jedna tehnika ugrađuje nadzorni kod unutar ciljnog programa. Ova tehnika nije transparentna.
2. Druga tehnika ugrađuje nadzorni kod unutar sistemskog kernela ili tretira monitorski kod kao proces odvojen od procesa ciljnog koda. Ovaj metod je transparentan ali manje fleksibilan.

Softverski nadzor, koji zahteva detekciju događaja i sakupljanje podataka o događajima, može se opisati na sledeće načine:

- Ciljni program detektuje događaje i skuplja podatke o događajima.
- Ciljni program detektuje događaje a nadzor skuplja podatke o događajima.
- Kernel detektuje događaje a nadzor skuplja podatke o događajima.

Sve tri implementacije imaju svoje prednosti i nedostatke. Implementacija oba, i detekcije događaja i skupljanja događaja, unutar ciljnog programa je najlakši i najdirektniji način za nadzor sistema. Ostvarivanje detekcije događaja i prikupljanja događaja na nivou kernela ima dva nedostatka u odnosu na ono na programskom nivou: jedno je transparentnost, a drugo redukovanje nadzornih perturbacija (smetnje, uznemirenje). Transparentnost se lako postiže jer korisnici ne moraju da modifikuju ciljne programe zbog detekcije događaja i prikupljanja događaja. Perturbacije su redukovane jer je promena konteksta između ciljnog programa i kernela izbegnuta. Nasuprot tome, izvršavanje detekcije događaja na programskom nivou zahteva sistemski poziv. Ovo rezultuje u ekstenzivnoj promeni konteksta (izmeni) jer sistemski pozivi se implementiraju sa interaptima kernelu. Međutim, mada pristup na nivou kernela redukuje perturbacije, dodatni stalni nadzor na kernelu ipak usporava vreme izvršenja kernela i povećava vreme izvršenja ciljnog programa.

4.2 HARDVERSKI PRISTUP NADZORU

Generalno, nadzor hardvera koristi se da bi se monitorovalo ponašanje u toku izvršenja ili hardverskih elemenata ili softverskih modula. Prvo se koristi kod merenja performansi hardverskih elemenata, kao što je merenje pristupa kešu, vreme pristupa memoriji, totalno CPU vreme, totalno vreme izvršenja, I/O zahteva, vreme zauzetosti I/O. Nadzor softverskih modula koristi se kod debagovanja i kod analize performansi takvih softverskih karakteristika kao što su: programska uska grla, mrtve petlje i stepen paralelizma. Izgleda da su ove dve upotrebe nadzora sasvim različite, ali rezultati merenja hardverskih performansi mogu takođe da pomognu kod analize softverskih performansi i debagovanja. Na primer, može se uočiti da se uska grla javljaju zbog čestog obraćanja istoj memorijskoj lokaciji, a mrtve petlje zbog poruka zatvorenih unutar komunikacione mreže, dok stepen paralelizma može biti određen deljenjem ukupnog računarskog vremena ukupnim vremenom izvršenja.

4.3 HIBRIDNI PRISTUP NADZORU

Hibridni pristup sastoji se od softverske implementacije i hardverske detekcije - dakle komponenta između hardverskog i softverskog nadzora. Ciljni program koji se nadzire prvo se instrumentalizuje ručno ili automatski da bi generisao događaje, a zatim se hardverski nadzor koristi za detekciju događaja i prikupljanje odgovarajućih podataka o događajima. Hardverski nadzor može se projektovati kao stalni deo ciljnog sistema u toku faze projektovanja, ili on može biti individualni elemenat ili koprocesor integrisan u ciljni sistem u toku faze testiranja i debagovanja.

Kada se hardverski nadzor implementira kao deo ciljnog sistema, koraci nadzora su kao što sledi:

1. Definiše se klasa događaja koja će biti nadgledana i dodeljuje se memorija svakoj klasi.
2. Instrumentacioni kod se ugrađuje u ciljni program.
3. Ciljni program se kompajlira sa instrumentacionim kodom, i za instrumentacioni kod se konstruiše tabela nadzornih simbola.
4. Objektni kod instrumentalizovanog ciljnog programa linkuje se i loaduje tako da objektni kod odgovoran za nadzor markira pojavu događaja i zapisuje događaj, uključujući i klasu događaja i relevantnu informaciju o događaju, u unapred dodeljen memorijski prostor rezervisan u koraku 1. za tu klasu događaja.
5. Hardverski nadzorni elemenat osluškuje i uparuje signal na basu sa adresom dodeljenom toj klasi događaja. Jedan događaj je detektovan ako hardverski nadzorni elemenat detektuje adresni signal na basu koji odgovara jednoj od adresa unapred dodeljenih klasama događaja.

Kada se hardverski nadzor izvodi kao koprocesor, koraci nadzora su kao što sledi:

1. Definiše se klasa događaja koju treba nadzirati i svakoj klasi se dodeljuje memorija.
2. Instrumentacioni kod insertuje se u ciljni program.
3. Ciljni program se kompajlira sa instrumentacionim kodom, a konstruiše se tabela nadzornih simbola za instrumentacioni kod.
4. Objektni kod instrumentalizovanog ciljnog programa linkuje se i loaduje tako da se objektni kod koji je odgovoran za nadzor izvršava odgovarajućim nadzornim koprocesorom.
5. Nadzorni koprocesor izvršava specijalne nadzorne instrukcije kao na primer *write*, koja brzo konstruiše i storira trag događaja u brzi RAM unutar koprocesora.

5. FUNKCIJE IMPLEMENTIRANJA NADZORA

U softverskom inženjerstvu, nadzor (monitoring) je zapisivanje pojava specificiranih događaja u toku izvršavanja programa da bi se dobile informacije o izvršenju koje se ne mogu pouzdano sagledati proučavanjem teksta programa. Informacija nadzora uključuje ponašanje u toku izvršavanja nadziranog programa i značajne informacije operativnog sistema. U prošlosti, istraživači su koristili metode nadzora da bi skupili informacije kroz različite forme instrumentacionih tehnika za testiranje i debagovanje programa, dinamičko raspoređivanje zadataka, analizu performansi, i optimizaciju programa.

Ovde je nadzor u realnom vremenu razmatran kao alat za testiranje i debagovanje sistema. Naime, ako funkcijama za nadzor događaja dodamo mogućnost provere, što ne predstavlja veliki napor posle integracije HRTS i testiranja u toku procesa projektovanja, lako je postići testiranje vremenskih karakteristika događaja u toku funkcionisanja sistema.

I hardverski i softverski realizovane funkcije nadzora sa ugradjivanjem u aplikacioni kod ili u nivo kernela operativnog sistema, vide se kao skup bibliotečkih procedura, obično u C jeziku. U osnovni njima se funkcionisanje sistema sagledava kao niz pojava događaja.

Događaji koji se trebaju osmatrati u nekom RTS-u specificiraju se dodeljivanjem oznaka u programu za rad u realnom vremenu onim događajima koje treba nadzirati u toku izvršenja. Primeri ovih događaja su startovanje ili kraj programskog segmenta, i dodeljivanje promenljivoj stanja. Vremensko ograničenje sistema može se videti kroz tvrdnju (kao poruku) koja ukazuje na odnos izmedju pojava događaja koji se osmatraju.

Predloženi pristup pravi razliku izmedju sistemskog ograničenja koje je ugradjeno u RT program i ograničenja koje je nadzirano asinhrono pomoću odvojenog zadatka. Implementacija prototipa dopušta specifikaciju i nadzor oba tipa sistemskih ograničenja. Naša prototipska implmementacija dopušta specifikaciju i nadzor oba tipa sistemskih ograničenja. Naročito, implementacija podržava karakterisanje C programa sa događajima i specificiranje sistemskih ograničenja. Ona takođe obezbeđuje mehanizme za asinhroni ili sinhroni nadzor ovih ograničenja u toku rada.

Da bi se nadziralo funkcionisanje sistema u vremenu korišćenjem tačaka osmatranja, potrebna je funkcija *events*. Iniciranje i dovršenje niza programskih naredbi može se označiti insertovanjem ovih funkcija u izvorni kod kao labela.

```
...
Ei->(ek, tmin, tmax)
statement_S
Ek<-(ej, tmin, tmax)
...
```

Slika 4. Fragment koda sa oznakama događaja

Strelica na desno specificira događaj e_i koji označava start naredbe koja sledi (`statement_S`), a strelica na levo specificira kraj prethodne naredbe. Ako je $t_{min}=t_{max}=0$, funkcija samo inicira merenje vremenskog intervala do pojave događaja e_k. Za definisano $t_{min}, t_{max} \neq 0$ funkcija inicira proveru da li će se događaj e_k pojaviti unutar specificiranog vremenskog intervala $[t_{min}, t_{max}]$.

Ovaj pristup ima dve važne korisne osobine: on odvaja vremenska razmatranja od funkcionalne specifikacije programa i dopušta izražavanje svojstava krajnjeg roka i kašnjenja. Inače on pretpostavlja:

1) Da se dve pojave istog događaja ne mogu desiti istovremeno;
2) Da su imena događaja jedinstvena u sistemu zadataka;
3) Da postoji jedan jedinstven takt za monotono rastući tok vremena, dostupan svim zadacima.

Izložen pristup razmatra korišćenje informacije nadzora u vremenskom rasporedjivanju u okolnostima rada u realnom vremenu. Informacija nadzora može se vraćati operativnom sistemu radi postizanja adaptivnog ponašanja.

Poseban tip događaja koji mogu da se osmatraju su dodeljivanja vrednosti promenljivima koje su deklarisane i kao osmotrive (watchable) promenljive. Ako je promenljiva v deklarisana pomoću

```
watchable_V V(vmin, vmax, Δv)
```

onda se bilo koje dodeljivanje vrednosti njoj razmatra kao događaj. Ako je definisano $v_{min}, v_{max} \neq 0$ i $\Delta v \neq 0$, provera je izvršena bez detektovanja greške ako je vrednost unutar definisanog intervala i ako je apsolutna promena vrednosti u odnosu na zadnju prethodnu vrednost, veća od Δv.

Istorija događaja generiše se u sistemu storiranjem vremena i/ili vrednosti osmotrivih promenljivih. Za pregled istorije u toku testiranja ponašanja prototipa i za rekonstrukciju događaja u toku funkcionisanja sistema,

potrebne su dve druge funkcije (na softverskom nivou kernela): `@(e,i)` i `@val(v,i)`. `@(e,i)` je funkcija pojave koja vraća vreme *i*te pojave događaja `e` (*i* može biti negativno za događaje koji su referencirani unazad. `@val(v,i)` je funkcija koja vraća vrednost osmotrive promenljive v na *i*tom događaju njene promene.

6. ZAKLJUČAK

Realizacija što boljeg, vernijeg nadzora RTS-a je sve značajniji uslov za uspešno projektovanje RTS-a. Polazeći od toga i ideje da se adekvatnim nadzorom za RTS-e može testirati RTS u toku rada u samoj eksploataciji izložen je jedan softverski zasnovan nadzor. Date su funkcije kojima se u procesu projektovanja mogu sagledavati vremenske karakteristike RTS-a, a u eksploataciji otkrivati greške u radu sistema. Greške koje se zbog nekog otkaza u sistemu manifestuju kao vremenska nesaglasnost, odnosno narušavanje vremenskih ograničenja. Definisane funkcije su pogodno da se implementira i u programski kod i u jezgro operativnog sistema za rad u realnom vremenu. Uz implementaciju jednostavnog hardverskog modula nadzor sa definisanim funkcijama minimalno angažuje procesorsko vreme.

LITERATURA

[1] Beth A. Schroeder, *"On-line Monitoring: A Tutorial"*, IEEE Computer, Vil. 28, No. 6, June 1995, pp.72-78.

[2] Bernhard Platner, *"Real-Time Execution Monitoring"*, IEEE Trans. Software Eng., Vol. SE-10, No. 6, Nov. 1984, pp. 756-764.

[3] Milun Jevtić, Milunka Damnjanović , *"An Approach to Design for Testability in Hard Real-Time Systems"*, Proceedings 21th International Conference on Microelectronics - MIEL'97, Vol.2, pp. 849-852, Niš, Serbia, September 1997.

Abstract - The implementation techniques for monitoring the processes in real-time microcomputer systems are considered in this paper. A monitoring implementation based on software is presented. It is intended for system timing characteristics testing during the system development, and time overflow detection during runtime. The solution requires minimal processor time and can be implemented in program code as well as in real-time operating system kernel.

PROCESS MONITORING IN REAL-TIME SYSTEMS

Milun Jevtić, Milunka Damnjanović, Vladimir Živković