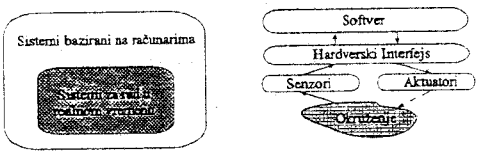
MOGUĆNOST REALIZACIJE VREMENSKI KRITIČNIH ZADATAKA NA PLATFORMAMA OPŠTE NAMENE

Vladimir D. Živković, Milun S. Jevtić, Bojan A. Leković, *Elektronski fakultet u Nišu*

Sadržaj - *U ovom radu se razmatraju mogućnosti korišćenja PC-hardvera i multitasking operativnih sistema opšte namene prilikom realizacije sistema za rad u realnom vremenu. Pored osnovnih problema izložene su i mogućnosti jednostavnog prevazilaženja nekih od njih.*

1. UVOD

Niska cena integrisanog hardvera i fleksibilnost softvera su glavni razlozi zašto se u mnogim sferama ljudske delatnosti primenjuju sistemi bazirani na računarima (engl. *Computation Based Systems - CBS*). Posebna grupa CBS-a jesu sistemi za rad u realnom vremenu (engl. *Real-Time Systems - RTS*)(slika 1.a.). Osnovna karakteristika sistema za rad u realnom vremenu je da korektnost njihovog rada ne zavisi samo od logičke ispravnosti rezultata obrade, već i od vremenskog trenutka u kome je ovaj rezultat generisan [1].



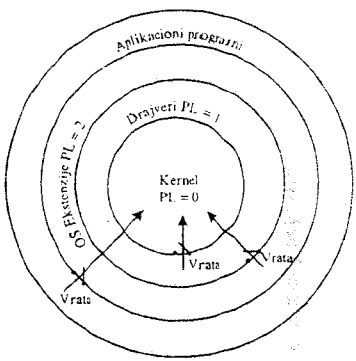
a b

Sl.1. (a) Odnos RTS-CBS, (b) Bazični elementi RTS-a.

Veza između okruženja i računara na kome je baziran RTS (slika 1.b.) proističe iz vremenskih ograničenja i ograničenja koje postavlja zahtev za pouzdan rad sistema[1]. Zahtev pouzdanosti se rešava redudansom, odnosno ugradnjom rezervnih fizičkih resursa koja se može uvoditi u hardver, u softver. Sa druge strane, brzina opsluživanja zadataka ne zavisi samo od arhitekture računara, već i od opterećenja sistema. Kako se u svakom trenutku kod uniprocesorskih sistema samo po jedan zadatak (engl. *task*) može opslužiti, znači da svi ostali čekaju na opsluživanje. Broj zadataka koji čekaju na opsluženje pretstavlja opterećenje sistema. Opterećenje sistema generiše ozbiljan problem u smislu kada koji zadatak opslužiti kako bi se minimizirala verovatnoća greške (tj. da bi se izvršili svi zadaci u postavljenom roku). Iz svega prehodno navedenog dolazi se do logičnih zaključaka da su RTS-i sistemi visoke cene sa specifičnim zahtevima kako u pogledu hardvera, tako i u pogledu softvera. Ipak, opravdano je postavljati pitanje da li postoji mogućnost da se specifični zahtevi RTS-a pokriju jeftinim hardverom koji se koristi kod personalnih računara (engl. *Personal Computer - PC*) i postojećim operativnim sistemima koji poseduju sposobnost konkurentnog praćenja i izvršenja više zadataka (engl. *multitasking Operating Systems - multitasking OS*)? Odgovor zavisi od unutrašnje prirode multitasking OS-a i fizičkih ograničenja koja postavlja PC-hardver.

2. HARDVERSKA PODRŠKA

Savremeni PC-računari, bazirani na Intelovim 32-bitnim procesorima, imaju mogućnosti da podrže vremenski kritične zadake do određene granice. Počev od Intelovog 16-bitnog 80286 procesora pa na dalje, PC procesori poseduju u sebi mehanizam koji štiti zadatke multitasking OS-a od nepravilnog ili nekorektnog pristupa memorijskim prostorima [2]. Podrška se ogleda i kroz definisanje posebnog režima rada procesora, poznatijeg kao zaštićeni mod (engl. *protected mode*).



Sl.2. Koncept nivoa privilegija kod Intelovih procesora

Suština multitasking obrade kod uniprocesorskog sistema kakav je PC, jeste obezbediti simultano prisustvo više programa u memoriji, bez opasnosti da izvršavanje jednog od njih naruši integritet drugog, a samim tim i funkciju celokupnog sistema. U ovu svrhu procesor sadrži hardver za proveru pristupa podacima i kodu u memoriji, i obezbeđuje prava pristupa preko četiri nivoa privilegija (Slika 2.). Pristup iz segmenata jednog nivoa u segmente drugog nivoa odvija se preko tačno definisanih vrata (engl. *call gate, task-gate, interrupt-gate*). Zahtev koji nastaje iz prirode RTS-a - da se svi zadaci nalaze u memoriji, tj. da nema zadataka koji su odloženi u sekundarnu memoriju (engl. *swapped out*), nalaže relativno veliku količinu operativne memorije. Raniji PC računari, bazirani na 16-bitnim procesorima sa 20 adresnih linija, nisu mogli da adresiraju veliku količinu memorije. Današnji 32-bitni procesori rešavaju problem adresiranja veće memorije dodatnim adresnim linijama. Upravljanje memorijom se vrši uvođenjem posebnih registara i logike. Novo-uvedeni registri za upravljanje memorijom (engl. *Memory Management Registers - MMR*) i upravljački registri za i386 zaštićeni mod (engl. *Control Registers*) omogućavaju da se za svaki zadatak izdvoji segment čija je veličina 1 MBy (ili 4 GBy, u zavisnosti od granularnosti kojim se pristupa memoriji, bajtovski ili

stranično - 1 stranica = 4 KBy). Ukoliko se uzme u obzir da se preko MMR-a može podržati 16383 segmenata, dobija se da je ukupni "logički" ili "virtualni adresni prostor" jednak 16 GBy ili 64 TBy, bajtovski ili stranično respektivno gledano. Kako je adresna magistrala 32-bitna, memorija koja se fizički može adresirati je 2^{32} By = 4096 MBy. Osim cene, koja bi ovde bila primarni faktor, postoji i realna "nepotrebnost" tolike memorije. Postoji procena koja kaže da pouzdanost sistema jeste obrnuto proporcionalna količini memorije. Ipak, centralna procesorska jedinica (engl. *Central Processing Unit - CPU*) današnjih PC-ja može da adresira dovoljnu količinu memorije da bi svi zadaci RTS-a bili u njoj.

Pristup memoriji u zaštićenom modu odvija se preko:

1. selektora segmenta (engl. *segment selector*), što je u stvari segmentni registar;
2. deskriptora segmenta (engl. *segment descriptor*).

Deskriptori segmenata su smešteni u sledećim tabelama:

1. Lokalne tabele deskriptora segmenata - sadrži deskriptore za memoriju koju koristi specifikovani zadatak;
2. Globalna tabela deskriptora segmenata - sadrži adrese lokalnih tabela deskriptora;
3. Tabela prekidnih deskriptora segmenata - sadrži deskriptore segmenata gde se nalazi kod prekidnih funkcija (važno je napomenuti da u zaštićenom modu prekidna tabela deskriptora segmenata ne počinje sa najnižom adresom, kako je to slučaj u 16-bitnom 8086 modu ili realnom modu).

U skupu MMR-a, od posebnog značaja jeste registar tekućeg zadatka (engl. *Task Register - TR*). TR-om se u memoriji adresira posebni sistemski segment *Task State Segment - TSS*. TSS sadrži stanja internih CPU-ovih registara, adresu prethodnog TSS-a, adresu tabele gde se nalaze ulazne tačke za različite delove koda datog zadatka, kao i bitmapu kojom se dozvoljava/brani pristup ulazno-izlaznom uređaju. TR je zbog toga od posebnog značaja u smislu multitasking OS-a. Naime, multitasking OS treba da obezbedi task-gejtove, TSS deskriptore i TSS strukturu u memoriji da bi se obavila komutacija konteksta, tj. prebacivanje zadataka (engl. *task switch*). Procesor potpuno nezavisno i bez ikakve instrukcije multitasking OS-a "čuva" sadržaj starog TSS i postavlja novi TSS. Pri tome procesor hardverski iščitava i upisuje 104 By. Ono što bi preostalo kao zadatak multitasking OS-a jeste da alocira dovoljno procesorskog vremena za svaki od zadataka. Upravljanje prebacivanjem zadataka jeste posao operativnog sistema. Ovako se, osim postizanja efekta konkurentnosti zadataka usled istiskivanja zadataka, dobija i strožija kontrola u slučaju neregularnosti nekog od zadatakā - sistem je robustniji. Poštujući pravilo da je sistem brz i efikasan koliko je i njegov najsporiji deo [3], a u skladu sa konstatacijom da savremeni PC-računari imaju mogućnosti da podrže vremenski kritične zadake do određene granice, treba uočiti koja je to granica.

Granica je postavljena perifernim jedinicama koje podržavaju rad centralnog procesora. Na primer, neki od RTS-a zahtevaju vremensku preciznost koju je nemoguće postići programabilnim tajmerom (engl. *Programmable Interval Timer - PIT*), koji se nalazi kao sastavni deo IBM-PC konfiguracije. Fundamentalno hardversko ograničenje za obradu u realnom vremenu ogleda se u obradi prekida, gde je za Pentium procesor potrebno najmanje 61 ciklus da se uđe i izađe iz prekida [4], uz dodatne cikluse za interakciju sa programabilnim prekidnim kontrolerom (engl. *Programmable Interrupt Controler - PIC*). Još jedno ograničenje jeste prethodno opisani *task-switching*. Naime, prilikom komutacije konteksta procesor pamti i učitava po 104 By, što je vremenski zahtevno. Ipak, i pored toga, moderni PC-hardver jeste u stanju da obradi dosta široki skup zadataka sa kritičnim vremenskim ograničenjima.

3. OBRADA VREMENSKI KRITIČNIH ZADATAKA

Na prvi pogled UNIX-orjentisani operativni sistemi bi mogli da zadovolje stroge kriterijume koje postavljaju RTS-i. Ovi sistemi podržavaju multitasking sa istiskivanjem, "portovani" su na PC računare i potpuno koriste 32-bitnu prirodu CPU-a. Neki od njih ispunjavaju kriterijume koje propisuje POSIX.1b-1993 standard, kao što su planiranje po prioritetu, zadržavanje korisničkih stranica u memoriji, poboljšana komunikacija između zadataka, itd. Međutim, priroda svih operativnih sistema iz UNIX - dijalekta jeste da se baziraju na deobi vremena (engl. *time sharing*). To znači da se vrši usrednjavanje performansi sistema u smislu balansiranja vremena odziva i protoka podataka radi fer deobe procesorskog vremena između zadataka. Kroz prizmu RTS-a ovakvo planiranje zadataka (engl. *scheduling*) nije korektno, jer se u istu poziciju stavljaju neki kritični periodičan zadatak i nekritičan zadatak. Problem je sasvim jasan, ukoliko se zna da su ovi operativni sistemi orjentisani čoveku, pa se samim tim favorizuju interaktivni zadaci.

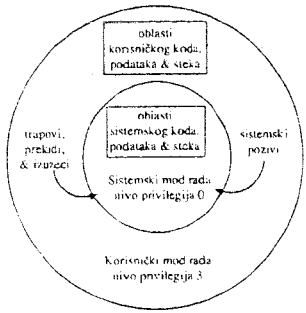
Međutim, čak i promena politike planiranja ili kompletnog planera ne bi mnogo poboljšala situaciju u korist ostvarivanja rokova vremenski kritičnih zadataka. Slika 3. prikazuje moguća stanja i prelaze između mogućih stanja jednog zadatka kod UNIX - orjentisanog operativnog sistema. Takođe, sl.3. prikazuje da UNIX koristi samo dva od ukupno četiri definisana nivoa privilegija:

- "nulti" privilegijski nivo, definisan kao "kernel mod", i
- nivo broj 3, koji je definisan kao korisnički ili "user mod".

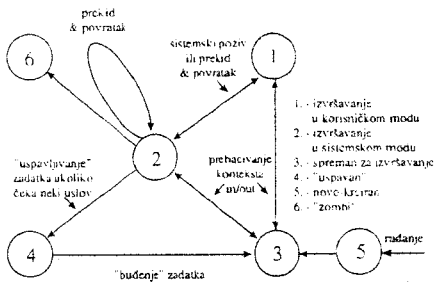
Kod najvećeg dela UNIX - orjentisanih sistema važi da se zadatak koji prilikom svog izvršenja dospe u "kernel-mod" ne može istisnuti. Drugim rečima, ako zadatak sistemskim pozivom počne izvršenje rutine iz kernela (drajvera), a drugi zadatak višeg prioriteta zatraži CPU-resurs, zadatak višeg prioriteta moraće da sačeka kompletiranje ranije unesenog sistemskog poziva. Sa stanovišta sigurnosti sistema ovaj prisup je korektan jer se obezbeđuje zaštita podataka koji mogu biti promenjeni ukoliko bi se dozvolilo istiskivanje u "kernel-modu". Međutim, ovakvim banalnim pojednostavljivanjem politike planiranja gubi se na performansama sistema. Jedan od načina da se zaštiti integritet kernela operativnog sistema jeste uvođenje sinhronizacionih struktura - semafora, kritičnih regija i sl. Ali ni ovi mehanizmi u globalu ne rešavaju ono što bi trebalo da bude glavna karakteristika RTS-a, a to je minimalna latentnost prekida.

Velika mana UNIX - orijentisanih operativnih sistema jeste vremenska neodređenost koju virtualni memorijski sistem unosi prilikom straničenja memorije. Ukoliko neki od tekućih programa zahteva da se memorijska stranica iz

sekundarne memorije prebaci u primarnu memoriju, zavisno od opterećenja sistema ovaj proces može da potraje duže. Sa stanovišta RTS-a to je potpuno neprihvatljivo.



a.



b.

Sl.3. Ponašanje zadatka u UNIX - orjentisanom operativnom sistemu: (a) - Modovi zadatka, (b) - stanja zadatka.

I na kraju, problem koji postoji kod ovih operativnih sistema jeste nepreciznost u ostvarivanju tražene rezolucije tajmera. Naime, da bi planer korektno obavljao prebacivanje zadataka neophodno je vrlo precizno određivati vreme kada istiskivanje treba da se obavi. Nepreciznosti u ovom pogledu dovode do džitera u vremenu javljanja zadataka, što može da dovede do nekorektnog rada sistema.

U svetu postoji trend da se Windows NT koristi za obradu zadataka u realnom vremenu. Ipak, kako je pokazano [5], nezgrapan Windows NT kernel nije sposoban za procesiranje zadataka sa čvrstim vremenskim ograničenjima:

1. Win32 API nije projektovan za zahteve realnog vremena;
2. Obrada prekida može biti odlagana do u nedogled;
3. Memorijska prealokacija je problematična; i
4. Značajno zauzeće memorije od strane Windows NT-a može biti veliki problem kod ugrađenih sistema.

Posebnu grupu RTS-a čine oni koji se baziraju na QNX mikrokernelu. Mikrokerneli imaju veći broj prednosti nad tradicionalnim monolitnim multitasking OS-ima:

- Debagiranje korisničkih zadataka je lakše nego debagiranje kernel komponenti;
- U slučaju greške u korisničkom zadatku, pošto se izvršenje odvija u posebnom adresnom prostoru, greške su izolovane;
- Zadatak može istisnuti izvršavanje drajverske-rutine;
- Mikrokerneli su mali, pa se lakše računaju parametri obrade u najgorem slučaju, na primer latentnost prekida;
- Brzo prebacivanje zadataka;
- Podržani mehanizmi komunikacije između zadataka (engl. *Inter Process Communication - IPC*).

Slaba tačka mikrokernela jesu performanse [6]. Mikrokerneli svu težinu postavljaju na IPC i prebacivanje zadataka. Sami mikrokerneli obezbeđuju samo nekoliko servisa direktno. To znači da QNX-u treba više sistemskih poziva u odnosu na monolitni kernel da bi realizovao isti zadatak. To je razlog zbog čega monolitni kerneli još uvek prosperiraju.

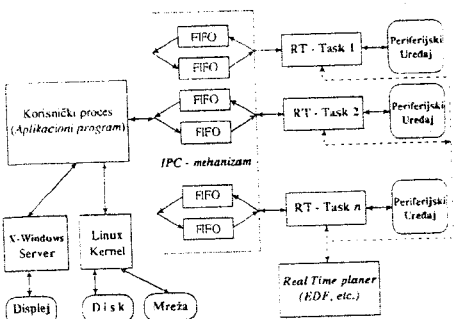
4. MODIFIKACIJE U LINUX OPERATIVNOM SISTEMU

Osnovni nedostatak UNIX - orjentisanih operativnih sistema za primenu u RTS-ima je moguća vremenska neodređenost u izvršenju nekih zadataka. Zbog toga je neophodno modifikovati izvršenje tih zadataka, odnosno ako je moguće takve zadatke u okviru RTS-a ugraditi kao vremenski nekritične zadatke sa najnižim prioritetom.

Problem nepoštovanja urgentnosti zadataka RTS-a od strane multitasking OS-a se može rešiti dogradnjom multitasking OS-a. To naravno podrazumeva postojanje mogućnosti izmene izvornog koda kernela operativnog sistema. *Linux* operativni sistem spada u UNIX grupu operativnih sistema, otvoren je i ima povoljna distribuciona pravila. Modifikacija *Linux* kernela [6] može rešiti neke probleme, kao što je napr. latentnost prekida.

Latentnost prekida jeste posledica zabrane prekida koja se postavlja kada zadatak pređe u "kernel-mod". Problem je moguće rešiti postavljanjem softverskog emulatora prekida između kernela operativnog sistema i PIC-a. Konkretno, emulacijom se *cli*, *sti*, i *iret* instrukcije (engl. *Clear Interrupt Flag*, *Set Interrupt Flag*, i *Interrupt Return*, respektivno) iz izvornog koda kernela multitasking OS-a zamenjuju sa emulacijonim makroima [6]. Pri tome se podrazumeva da se jednostavni i brzi zadaci (zadaci čiji su algoritmi sekvencijalne prirode) formulišu kao zadaci realnog vremena (engl. *real-time tasks*), dok se vremenski nekritični i kompleksni korisnički zadaci (engl. *user's tasks*) i dalje tretiraju klasično - preko kernela multitasking OS-a. Emulacioni softver će u slučaju prekida proveriti da li je prekid namenjen vremesnki kritičnom zadatku ili korisničkom zadataku. U slučaju da je namenjen zadatku realnog vremena, emulacioni softver pokreće *real-time* prekidne rutinu, nakon čega planer zadataka realnog vremena odlučuje koji će zadatak biti izvršen (treba imati na umu da su zadaci realnog vremena i *real-time* prekidne rutine vremenski kratke i jednostavne funkcije). U suprotnom, prekid je namenjen multitasking OS-u, tj. korisničkom zadatku, i emulira se *cli* instrukcija kojom se brani prekid. Time se dozvoljavaju svi eventualni prekidi, odnosno ulančavaju se prekidi namenjeni multitasking OS-u. Kada se napusti prekidna rutina multitasking OS-a, emulira se *sti* instrukcija, čime se najpre "prozivaju" eventualno ulančani prekidi, a nakon toga se izvršava instrukcija povratka iz prekida. Naime, eksperimenti [6] su pokazali da se na povratak iz "kernel moda" i eventualno ponovno unošenje istog, u slučaju prekida, troši dosta

dragocenog vremena. Tome doprinose hardverski mehanizmi vezani za *iret* instrukciju. Ideja emulacije *iret* instrukcije jeste da se pre povratka iz prekida najpre osmotri da li postoji neki ulančani prekid. Ovo je vrlo značajno jer se pri svakom povratku iz "kernel-moda" poziva planer *Linux*-a. Ukoliko *Linux* operativni sistem tretira prekid kao da je nastao u "kernel-modu", planer *Linux* operativnog sistema neće biti pozvan na izvršenje, čime se štedi vreme. Svi zadaci realnog vremena bi trebalo da se nalaze u zajedničkom adresnom prostoru u kome je i kernel *Linux*-a. Razlog za to jeste problem sa performansama CPU-a, jer se u slučaju različitih adresnih prostora za zadatke realnog vremena menja registar koji pomaže straničenje memorije, a samim tim i keš memorija koja sadrži najskorije korišćene memorijske stranice proglašava nevažećom. Kako je prebacivanje zadataka kod RTS-a veoma frekventno, opisani sled događaja uzrokuje pad performansi CPU-a. Takođe, performanse mogu da obore i sistemski pozivi, gde bi u slučaju različitih adresnih prostora isti izvršavali sedam puta duže [6].



Slika 4. Protok podataka u Real-Time aplikaciji baziranoj na Linux operativnom sistemu

Komunikacija između korisničkih zadataka i zadataka realnog vremena obavlja se preko *Linux*-IPC mehanizama[7]. Ukoliko je pomenuti IPC mehanizam FIFO (engl. *First In Eirst Qut - FIFO* [6]), komunikacija se vrši kao što je prikazano na Sl.4. FIFO-baferi su alocirani u adresnom prostoru kernela, a njihov broj je statički definisan i promenljiv u toku kompilacije kernela. Zadaci realnog vremena poseduju zasebne funkcije preko kojih se vrši kreacija, destrukcija, čitanje, upisivanje, i promena veličine FIFO-bafera (čitanje i upisivanje su atomične i neblokirajuće funkcije). Korisnički zadatak vidi FIFO-bafere kao karakter-uređaje.

Na Sl.4. je isprekidanom linijom prikazana veza prema planeru zadataka realnog vremena. To je zbog toga što se planer implicitno poziva od strane zadataka realnog vremena.

5. ZAKLJUČAK

U radu je predstavljen jedan od trendova koji su prisutni na polju RTS-a (opisana problematika je u svetu proverena u klasi centralizovanih sistema za rad u realnom vremenu sa čvrstim vremenskim rokovima i dinamičkim planiranjem izvršenja zadataka [6]).On se zasniva na adaptaciji već postojećeg multitasking operativnog sistema u smeru poštovanja čvrstih vremenskih zahteva. Naravno, predloženi pristup nije jedino rešenje. Međutim, svakako pretstavlja najjeftinije jer se bazira na kompletno slobodno dostupnom softveru i na klasičnim PC računarima. Prednost se takođe ogleda u mogućnosti korišćenja već razvijenih softvera za *Linux* operativni sistem, a u svrhu obrade vremenski nekritičnih zadataka. Pošto su vremenski kritični zadaci i njihov planer realizovani kao instalacioni kernel-moduli, zamena planera je olakšana u odnosu na neke operativne sisteme za rad u realnom vremenu. U većini RTS-a planeri pretstavljaju veliki i kompleksni deo koda koji se ne može nikako proširivati. Ponašanje planera se menja podešavanjem parametara planera, što nekada nije dovoljno. Nasuprot tome planeri koji se realizuju kao instalacioni kernel-modul su pogodni za eksperimentisanje sa različitim politikama i algoritmima planera. Dakle, primena platformi opšte namene kao podrške RTS-ima je moguća, ali uz izmene u pojedinim modulima kernela multitasking OS-a i uz minimalni dodatni hardver (*watch-dog* kartica ili neki drugi hardver-softver *monitoring* sistem) koji treba da obezbedi zahtevanu pouzdanost u radu RTS-a.

LITERATURA

[1] John A. Stankovic, "Real-Time Computing Systems: The Next Generation," *TUTORIAL Hard Real-Time Systems, Computer Society Press Editoria Board*, 1988.

[2] Hans-Peter Messmer, "The Indispensable PC Hardware Book - Your Hardware Questions Answered," *Addison - Wesley Longman*, 1997.

[3] James Mohr, "LINUX user's resource guide," *Prentice Hall*, 1997.

[4] Victor Yodaiken and Michael Barbanov, "A Real-Time Linux," *Technical Report, Department of Computer Science - New Mexico Tech*, 1997.

[5] Martin Timmerman and Jean-Christophe Monfret, "Windows NT as real-time OS?," *Real-Time Magazine*, 1997.

[6] Michael Barbanov, "A Linux-based Real-Time Operating System - MSc. thesis," *New Mexico Institute of Minig and Technology, Socorro, New Mexico*, June 1, 1997.

[7] David A. Rusling, "The Linux Kernel," david.rusling@reo.mts.dec.com, July 6, 1997

Abstract - *In this paper is presented one of the possibilities of implementing the real-time systems on the general purpose hardware and software platforms. Also, simple solutions for the basic problems are presented.*

ONE DIRECTION IN IMPLEMENTATION OF THE TIME CRITICAL TASKS ON THE GENERAL PURPOSE PLATFORMS

Vladimir D. Živković, Milun S. Jevtić, Bojan A. Leković