VxWORKS i VIŠEJEZGRENI PROCESORI

VIŠE JEZGRENI PROCESORI I VxWorks SMP

Multiprocesiranje je upotreba dva ili više procesora u sistemu u kojem procesori sarađuju i između sebe distribuiraju ukupni posao. Multiprocesiranje nije novo. Serveri, poslovni sistemi i oprema telekom operatera koriste multiprocesiranje već nekoliko godina. Neki ugrađeni uređaji koriste mnogo štampanih ploča na kojima se može nalaziti i po nekoliko procesora. Današnja nova tehnologija sama po sebi nije multiprocesirajuća, već multiprocesiranje koristi procesore sa više jezgara. Višejezgreni procesori imaju više od jedne jezgre na istom čipu. Svaka jezgra ima svoj vlastiti programski brojač. Arhitektura čipa dozvoljava jezgrima da „pričaju“ preko medija određenog tipa, obično preko lokalne memorije. Svaki višejezgreni procesor dozvoljava simultano izvršavanje više od jednog dijela softvera.

VxWORKS i VIŠEJEZGRENI PROCESORI

Multiprocesiranje nam daje pravu konkurentnost, što znači da se više od jednog dijela softvera izvršava na tačno identičan način na jednom ili više procesora, ali njihovo ponašanje kada se izvršavaju je u osnovi različito. Višezadačnost (multitasking) i višenitnost na jednoprocesorskom sistemu obično se preklapa sa izvršavanjem više od jedne niti u različitim vremenskim trenucima.

Pojmovi kao što su hyperthreading, simultani multithreading i čip multithreading su arhitekturalne tehnike primjenjene na jedan procesor. Instrukcije ulaze u procesor kroz cijev (pipe) i čekaju da budu izvršene. Procesori su često blokirani dok čekaju da instrukcije budu dohvačene iz memorije. Iako se povlači više instrukcija (multifetch), još uvijek imamo sekvencijalni tok. Na čipu, koncept višenitnosti je proširen malo više iz razloga što mogu da postoje zavisnosti između instrukcija u različitim nitima.

VxWORKS i VIŠEJEZGRENI PROCESORI

Višenitnost čipa se prebacuje između višestrukih niti na nivou hardvera, na sličan način na koji to radi operativni sistem sa softverom. Još uvijek ovo nije isto kao višejezgrenost iz razloga što različite hardverske niti bivaju multipleksirane u jedno jezgro. Koncepti hyperthreadinga, simultanog multithreadinga i chip multithreadinga nisu uzajamno isključivi. Možemo imati chip multihreaded višejezgreni procesor nad kojim se izvršava višezadačni operativni sistem.

Ovaj tehnološki razvoj čini multiprocesiranje jeftinijim po pitanju cijene, utroška energije u odnosu na višestruke procesore na jednoj ploči. Ovo je bitno iz razloga što će sve više i više proizvođača uređaja biti motivirano da uđe u svijet multiprocesiranja iz razloga povećanja performansi uređaja, uštede u prostoru i novcu i bogatijih mogućnosti usljed korištenja višejezgrenih procesora.

VxWORKS i VIŠEJEZGRENI PROCESORI

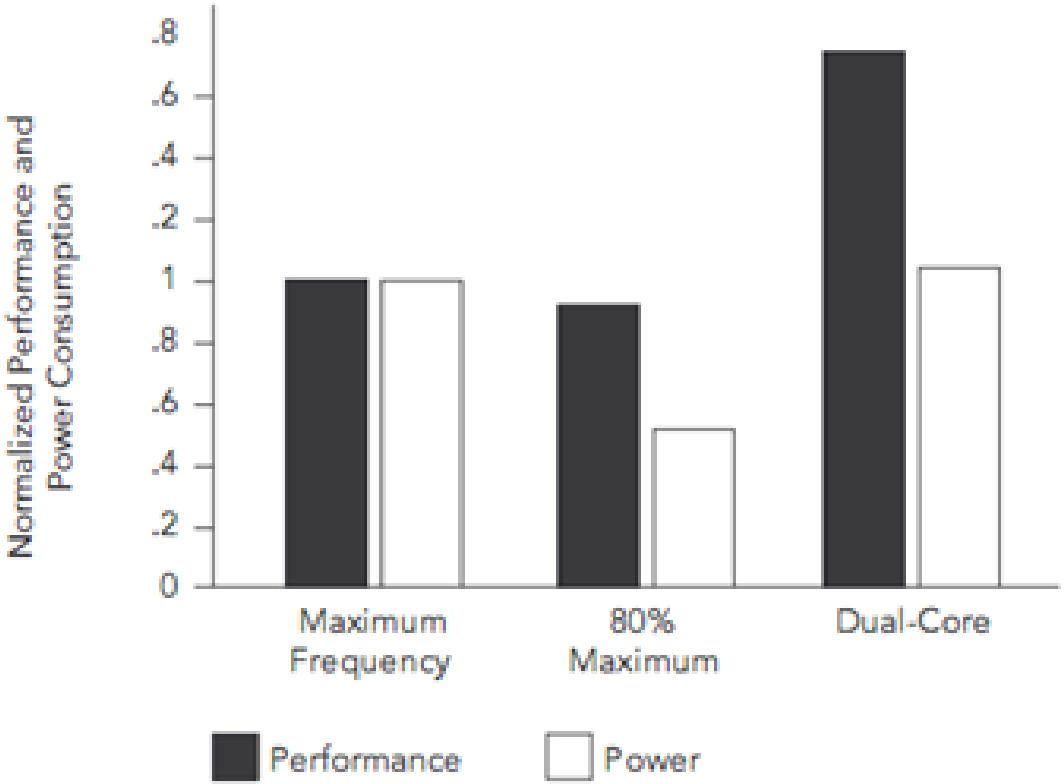
Trendovi kod višejezgrenih procesora

Proizvođači poluvodiča idu ka trendu višejezgrenih procesora. Godinama su veće performanse ostvarivane pakovanjem sve više i više tranzistora na jedan čip i povećanjem frekvencija sata. Međutim, poluvodička industrija je sada na mjestu gdje fizika više ne dozvoljava povećanje broja tranzistora i povećanje frekvencije sata. Proizvođači poluvodiča ne mogu povećavati frekvenciju a da pri tome zadrže potrošnju energije na razumnom nivou. Slika prikazuje efekte smanjivanja frekvencije na smanjenje potrošnje električne energije.

Potrošnja energije u procesoru je proporcionalna trečem stepenu napona, a napon je proporcionalan frekvenciji. Dakle, na nižoj frekvenciji možemo pokrenuti procesor sa niskim naponom tako da se potrošnja energije bitno smanjuje. Ako se frekvencija sata spusti za 10%, performanse idu na dolje za 10% , ali je smanjenje potrošnje energije daleko veće, kao što

VxWORKS i VIŠEJEZGRENI PROCESORI

se vidi na bar prikazima na slijedećoj slici.



Efekti smanjivanja frekvencije na smanjenje potrošnje električne energije

VxWORKS i VIŠEJEZGRENI PROCESORI

Malo žrtvovanje performansi čuva dosta energije. Sada, ako dva ovakva procesora mogu biti stavljena na isti čip sa malim smanjenjem frekvencije sata, u teoriji, imali bi situaciju prikazanu kao na prethodnoj slici : procesor sa istom potrošnjom energije ali sa teoretski dosta više procesorske moći.

Smanjivanje dimenzija i povećanje broja tranzistora čini ekonomičnijim stavljanje više procesorskih jezgara koje rade na nižim frekvencijama. Ovo je trend koji je rezultirao višejezgrenim procesorima. Ali za softverske inženjere iskorištavanje povećane procesorske moći isporučene na ovakav način je bitno drugačije nego iskorištavanje povećanja procesorske moći povećanjem frekvencije sata.

VxWORKS i VIŠEJEZGRENI PROCESORI

Višejezgrena tehnologija i tržište softvera

Podaci ankete koju je provela Venture Development Corporation (VDC), nezavisna konsultantska firma za istraživanje tržišta i strategiju, pokazuju jako zanimanje i povećanu upotrebu višejezgrene tehnologije na tržištu softvera. Mnogi proivođači uređaja već koriste višejezgrene procesore i multiprocesiranje za sisteme visokih performansi. U prostoru serverskih sistema, višejezgreni procesori su već dominantni: devet od 10 Intelovih procesora isporučenih za servere su već višejezgreni. Dakle, tržište se već kreče prema višejezgrenim procesorima.

Tržište softvera se ne kreće tako brzo. Ali kao trend, višejezgrenost se nastavlja dalje razvijati. Proizvođači poluvodića, iz razloga što su u trendu pravljenja računarske moći u malim pakovanjima uz niske troškove, svi proizvode svoje vlastite višejezgrene procesore. Neki od njih, kao što je

VxWORKS i VIŠEJEZGRENI PROCESORI

Intel, su usmjerili kompletnu proizvodnu liniju prema višejezgrenoj tehnologiji; dok ostali, kao što je Freescale

održava balansiraniji protfolio jednojezgrenih i visejezgrenih procesora, u zavisnosti od potreba tržišta.

Sve više i više uređaja će zahtijevati veće performanse procesora, i softverski arhitekti i developeri neće imati drugog izbora nego da se prilagode višejezgrenim procesorima kako bi postigli veće performanse u okviru ograničenja vezanih za potrošnju energije.

VxWORKS i VIŠEJEZGRENI PROCESORI

Razlike u višejezgrenoj tehnologiji

Tri aspekta čine razvoj aplikacija za višejezgrena okruženja bitno drugačijim nego za jedno jezgrena okruženja:

1. Nema garancije da će aplikacija dobiti na performansama iz multiprocesiranja. Ne mogu se riješiti svi problemi sa multiprocesiranjem. Potrebno je pronaći bitno različit algoritam kako bi se obavio posao. Postoječi algoritmi možda rade u jedno procesorskim okruženjima ali se ne skaliraju na više od jednog procesora.

2. Dizajn sistema je kompliciraniji nego kod jednoprocesorskog sistema, uz pretpostavku da imamo algoritam koji se može izvršavati konkurentno. Postoji jaka interakcija između načina na koji hardver podržava multiprocesiranje, načina na koji operativni sistem i sistemski softver podržavaju multiprocesiranje i načina na koji je aplikacija razdijeljena kako bi iskoristila prednosti više procesora. Korisnici moraju

VxWORKS i VIŠEJEZGRENI PROCESORI

odabrati pravu kombinaciju hardvera, OS-a, i dizajna aplikacije kako bi ostvarili sistem koji isporučuje veće performanse u odnosu na jednoprocesorski sistem. Često je potreban veliki broj pokušaja i grešaka kako bi se dobila prava kombinacija koja optimizira performanse.

3. Postoji mnogo složenosti u razvoju softverskog okruženja. Programeri godinama uče koncepte sekvencijalnog programiranja. Na primjer, kada više niti jedne aplikacije rade konkurentno, to aplikaciju više izlaže mogućnosti pojavljivanja grešaka. To krši neke osnovne pretpostavke nasuprot kojih je softver originalno napisan: za konkurentno izvršavanje dijelova koda nikada ne možemo očekivati da će se izvršiti konkurentno. Programeri se moraju boriti sa pitanjima sinhronizacije (npr. dodavanje sinhronizacije nad dijeljenim podacima što nije bio slučaj ranije) i pitanjem utrke nad podacima (data race). Ponašanje softvera može postati manje

VxWORKS i VIŠEJEZGRENI PROCESORI

determinističko zbog interakcije između niti, aktivnosti interapta u pozadini i načina na koji operativni sistem raspoređuje niti.

Prikladnost aplikacije za multiprocesiranje

Ako ima neki posao koji treba da se uradi i imamo na raspolaganju dva čovjeka, taj posao se može uraditi brže nego sa jednom osobom. Jezgre u višejezgrenom procesoru su kao radnici koje možemo da rasporedimo na posao. Očigledno, potrebno je imati dovoljno posla kako bi svi radnici bili uposleni. Zatim trebamo biti u mogućnosti da podijelimo posao na dovoljno dijelova kako bi svaki radnik imao nešto da radi. Ali ponekad više radnika ne riješava problem brzo. Radnici moraju raditi zajedno ili će doći do pada efikasnosti.

Proizvodna linija je primjer multiprocesiranja. Dosta posla može biti obavljeno pažljivom koordinacijom više aktivnosti tako da se mogu što je moguće više obavljati paralelno.

VxWORKS i VIŠEJEZGRENI PROCESORI

Međutim, postoje zavisnosti između trenutnih faza kroz koje posao prolazi. Ako zavisnosti nisu dobre ili posao nije urađen kako treba, naredni koraci čekaju, tako da izlaz trpi.

Neki zadaci se jednostavno ne mogu ubrzati dodavanjem više ljudi ili resursa, kao što je npr. čitanje knjige. To je striktno serijska operacija. Super računari i masivni paralelni računari se koriste samo za određene vrste aplikacija kao što je vremenska prognoza, finansijsko modeliranje itd., što nije neophodno za svaki računarski problem.

VxWORKS i VIŠEJEZGRENI PROCESORI

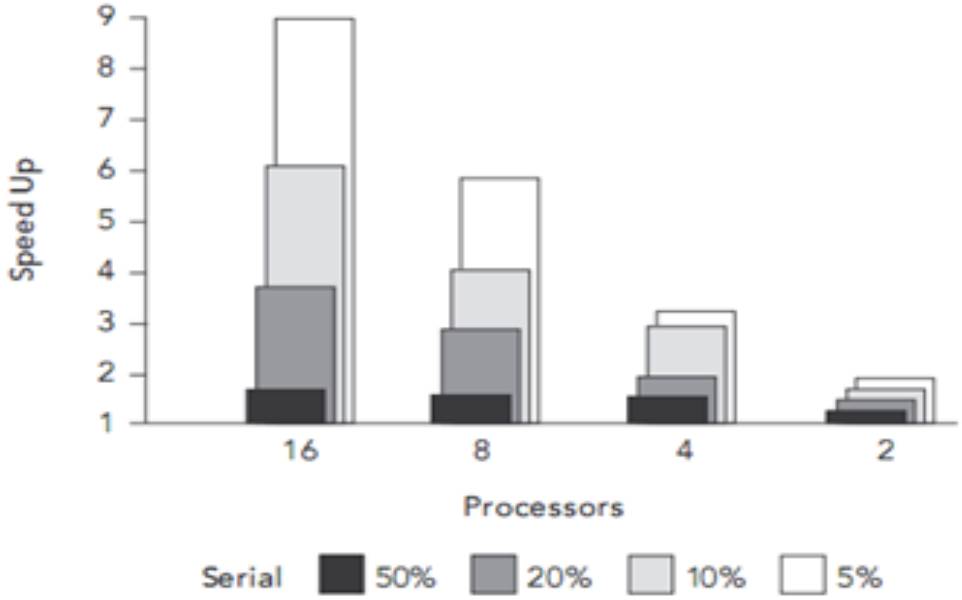
Amdalov zakon

Amdalov zakon je još jedno ograničenje koje se mora razumijeti. Svaki problem ima procenat posla koji se može paralelizovati i procenat posla koji mora biti urađen po nekom redoslijedu (npr. serijski). U zavisnosti od toga koliko posla može biti urađeno paralelno, postoji ograničenje koliko brzo posao može biti urađen bez obzira koliko je procesora dodijeljeno problemu. Amdalov zakon izgleda:

$$\text{Paralelno ubrzanje} = \frac{1}{Serijski\ \% + \frac{(1 - Serijski\ \%)}{Broj\ procesora}}$$

gdje Serijski % predstavlja procenat posla (koda) koji mora biti urađen serijski, i (1 – serijski %) je procenat posla (koda) koji se može paralelizovati. Naredna slika prikazuje efekte Amdalovog zakona kod aplikacije sa različitim stepenima serijalizacije i broja dostupnih procesora.

VxWORKS i VIŠEJEZGRENI PROCESORI



Efekti Amdalovog zakona

Sa 5% serijalizacije, ubrzanje performansi je relativno linearno kako se povećava broj procesora. Ali sa 16 procesora, dostiže se maksimum ubrzanja od devet puta. Serijalizacija od 5% ograničava maksimalno moguće ubrzanje na 9 puta za 16 procesorski sistem. Kako se procenat serijalizacije povećava maksimalno ubrzanje, kao i postignute performanse se značajno

VxWORKS i VIŠEJEZGRENI PROCESORI

smanjuje. Sa 50% serijalizacije, performanse se izravnavaju sa performansama postignutim sa četiri procesora. Uređaj sa 10% serijalizacijom neće imati bolje performanse sa osam procesorskim sistemom koji je samo četiri puta brži od uređaja pogonjenog procesorom sa jednim jezgrom.

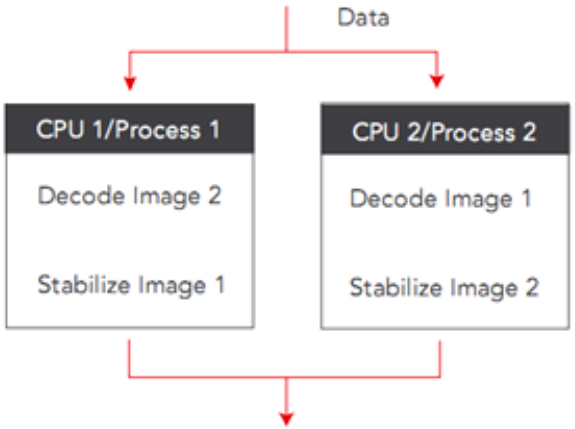
Čak i mala povećanja u obimu posla koji treba biti urađen serijski može uveliko umanjiti maksimalne performanse. Jedan primjer je umrežavanje gdje sistem rukuje sa višestrukim nezavisnim tokovima dolazečih paketa. Svaki tok je nezavisan u odnosu na druge; što se više procesora može angažovati na što više tokova podataka. Višejezgreni procesori pomažu ovakvim sistemima iz razloga što više procesora na čipu pomaže sistemu koji je manji i manje troši energije uz više urađenog posla u poređenju sa jednoprocesorskim sistemom.

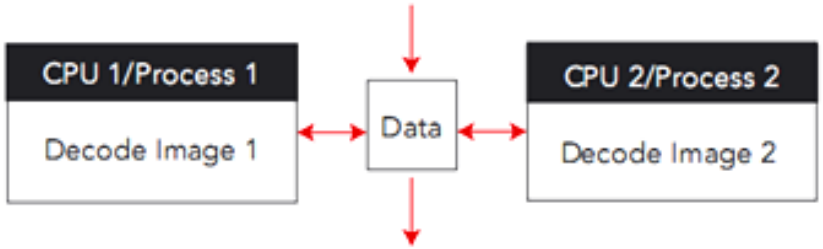
VxWORKS i VIŠEJEZGRENI PROCESORI

Multiprocesiranje i višejezgrenost imaju jak uticaj na arhitekturu hardvera, što pak određuje arhitekturu sistemskog softvera i ima uticaj na arhitekturu aplikacije.

Postoji dosta načina na koji posao u datoj aplikaciji može biti podijeljen zbog paralelizma – paralelizovan. Paralelizacija aplikacije je jedan od bitnijih aspekata na koje proizvođači uređaja moraju da vode računa prilikom dizajna višeprocesorskih sistema.

VxWORKS i VIŠEJEZGRENI PROCESORI





Svi oblici podjele aplikacija prikazani na gornjim slikama su validni i odgovaraju različitim problemima. Zadatak dizajnera sistema je da pronađe najbolji način podjele aplikacija za odgovarajući problem i dizajn hardvera. Na vrhu

VxWORKS i VIŠEJEZGRENI PROCESORI

je cijevni (pipe) dizajn, dosta sličan proizvodnoj liniji.

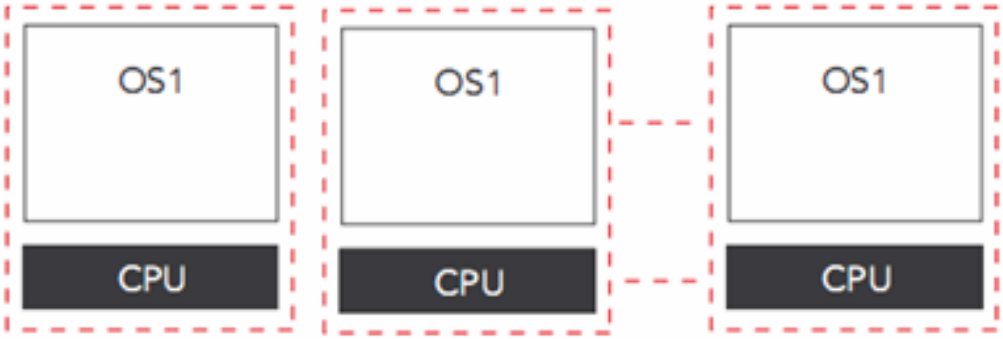
Na dnu je distribucija nezavisnog posla na više od jednog radnika.

Najkompikovaniji dizajn podjele aplikacije je prikazana na posljednjoj slici u nizu. Ovaj dizajn radi sa fino-granuliranim particioniranjem. Postoji jedan skup podataka, i da bi se postigle zahtijevane performanse, više operacija mora biti obavljeno nad tim podacima; ne postoji striktan raspored prema kojem bi ove operacije trebale biti obavljene. Maksimiziranje performansi zahtijeva više od jednog radnika koji će da istovremeno riješava isti problem. Ovo je kao da stavljamo više radnika da rade posao pod uslovom da posao mora biti urađen na način da svi rade efikasno i brzo. Svaki radnik uzima manji dio većeg posla i radi na njemu nezavisno u odnosu na druge radnike.

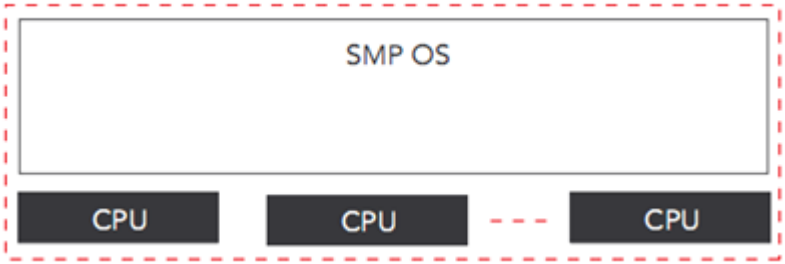
VxWORKS i VIŠEJEZGRENI PROCESORI

Podešavanja operativnog sistema za multiprocesiranje

Postoje dvije vrste podešavanja operativnih sistema koja su podesna za višeprocesorske sisteme: simetrično multiprocesiranje (SMP) i nesimetrično multiprocesiranje (AMP).



AMP konfiguracija



SMP konfiguracija

VxWORKS i VIŠEJEZGRENI PROCESORI

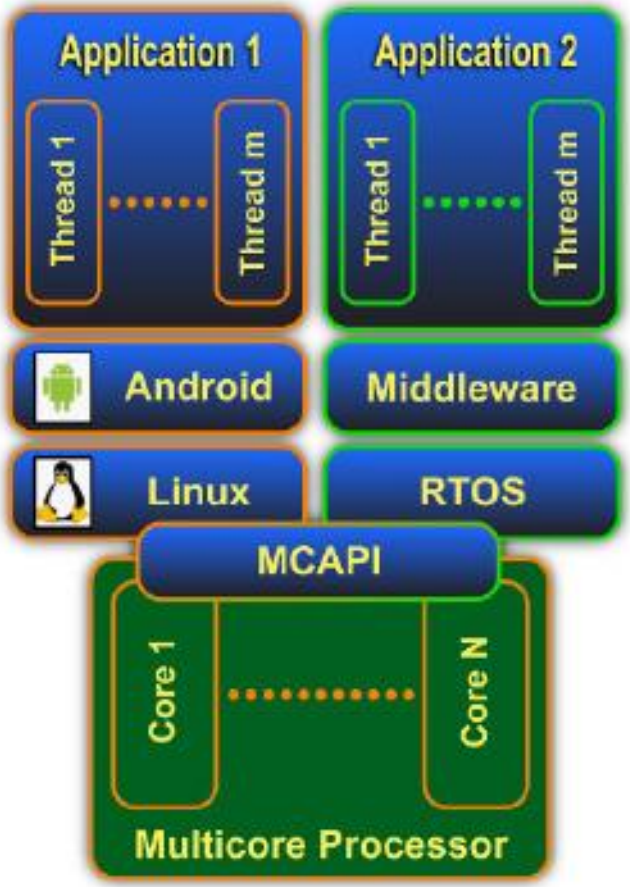
Kod AMP konfiguracije operativni sistem se izvršava na svakom procesoru ili jezgru multiprocesirajučeg sistema. Svaka kombinacija procesor/OS je jedan jedno-procesorski računar. Sistem se sam po sebi sastoji od više ovakvih računara (također poznati kao čvorovi) koji treba da urade posao koji im se rasporedi. Ono šta povezuje ove nezavisne čvorove u jedan veliki sistem je mehanizam međukonekcije kojeg koriste procesori kako bi međusobno komunicirali. Ovaj mehanizam je obično dio dijeljene memorije između procesora ali ponekad može biti mrežna konekcija ili druga periferalna sabirnica. Kod AMP sistema, posao koji obavlja svaki pojedinačni čvor mora biti definisan prilikom dizajna sistema. Posao cijelog sistema je tako particionisan na dijelove koji se zatim dodijeljuju pojedinačnim čvorovima u sistemu.

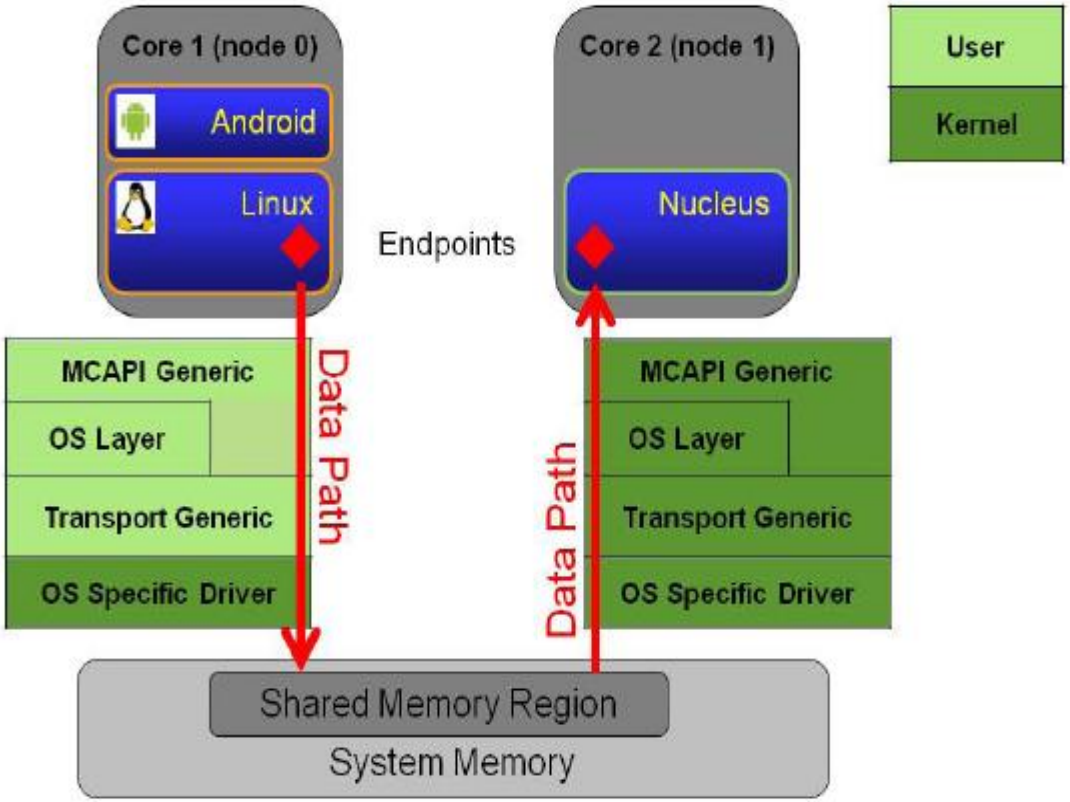
VxWORKS i VIŠEJEZGRENI PROCESORI

AMP sistemi mogu doći u različitim kombinacijama. Moguće je odabrati različite procesore u sistemu. Također je moguće odabrati različite operativne sisteme u AMP sistemu, sve u zavisnosti od potreba. Na primjer, VxWorks može upravljati osjetljivim real-time taskovima dok Wind River Linux može upravljati nekim od interfejsa – npr. media interface. Ponekad čvorovi mogu da obavljaju veoma ograničene funkcije koje ne zahtijevaju da imamo kompletan operativni sistem već samo jednostavno izvršavanje direktno na hardveru. Slijedeca slika

prikazuje takvu konfiguraciju gdje je za medjujezgrenu komunikaciju koristen MCAPI (multicore communication API) standard medjujezgrene komunikacije koji je u toku faze definisanja standardnog seta API poziva kojima se ona implementira. AMP sistemi obično trebaju više memorije nego SMP sistemi kako bi držali različite operativne sisteme i njihove aplikacije.

VxWORKS i VIŠEJEZGRENI PROCESORI





Po svojoj prirodi AMP sistemi su veoma zavisni od preciznog dizajna hardvera. Prebacivanje na drugi hardver sa nekoliko ili više procesora ili različiti mehanizam interkonekcije, uzrokuje da moramo prepraviti aplikaciju kako bi radila na novoj hardverskoj konfiguraciji.

VxWORKS i VIŠEJEZGRENI PROCESORI

SMP konfiguracija je konfiguracija kod koje jedan operativni sistem kontroliše više od jednog indentičnog procesora (ili jezgra u višejezgrenom sistemu). Aplikacije „vide“ i ostvaruju interakciju sa samo jednim operativnim sistemom, baš kao i kod jedno-procesorskog sistema. Činjenica da postoji nekoliko procesora u sistemu je detalj koji OS krije od korisnika. U tom smislu, SMP operativni sistem sažima detalje o hardveru. SMP operativni sistem je također simetričan; on treba da radi sa više identičnih procesora, pri čemu svaki može pristupiti cijeloj memorji i uređajima u sistemu na uniforman način. Dakle, bilo koji procesor (ili jezgro) u SMP sistemu je u mogućnosti da izvrši bilo koji task baš kao i svaki drugi procesor u sistemu. Ova simetrija u hardveru dopušta SMP operativnom sistemu da rasporedi bilo koji posao na bilo koji procesor u sistemu. SMP operativni sistem pokušava da drži sve procesore uposlenim izvršavanjem aplikacijskih niti sa

VxWORKS i VIŠEJEZGRENI PROCESORI

ciljem raspodjele opterećenja u sistemu.

Kao rezultat apstrakcije hardvera i raspodjele opterećenja u sistemu, SMP operativni sistem pojednostavljuje razvoj softvera kako bi se izvršavao na SMP hardveru. Iz perspektive programera, postoji samo jedan OS za pisanje aplikacije, koja automatski raspoređuje posao na sve dostupne procesore. SMP operativni sistem pruža istu semantiku programiranja kao i jednoprocesorski sistem.

Svi procesori, kod SMP sistema, dijele istu memoriju koja je pod kontrolom operativnog sistema. Memorija i lokalni keševi za svaki individualni procesor se drže sinhronizovanim. Stoga aplikacije u SMP sistemu mogu dijeliti podatke sa drugim procesorima na veoma jednostavan i efikasan način. Ovo čini SMP sisteme pogodnijim za aplikacije koje trebaju da dijele velike količine podataka uz nisku latenciju (kašnjenje).

VxWORKS i VIŠEJEZGRENI PROCESORI

Obzirom da SMP operativni sistem mora sinhronizovati i kontrolisati više od jednog procesora, biće potrebno obaviti više internog posla nego što je to slučaj kod operativnog sistema koji upravlja samo jednim procesorom. U poređenju sa jednoprocesorskim sistemom, SMP sistem je sporiji, ali SMP nadmašuje jednoprocesorske sisteme tamo gdje postoji visoki stepen paralelizma u aplikaciji koji dopušta sistemu da uradi više posla u datom vremenu u odnosu na jednoprocesorski sistem.

,

VxWORKS i VIŠEJEZGRENI PROCESORI

Izbor između SMP i AMP sistema

SMP operativni sistem je najbolji izbor kada se od operativnog sistema očekuje da efikasno rasporedi različite paralelne dijelove aplikacije. Jedno od bitnijih osobina hardvera za SMP je da obavlja hardverski pojačanu koherenciju keša. Ovo uzrokuje da hardver automatski ažurira kopije podataka na procesorskim keševima kada se izmjeni od strane drugog procesora. Za programe koji se izvršavaju, ažuriranje je transparentno tako da su izmjenjeni podaci odmah vidljivi u memorji i dostupni. Kada taskovi dijele mnogo podataka lociranih u memorji, pristupanje podacima je brzo i lagano kod SMP operativnog sistema. SMP je također pravi izbor kada aplikacija treba da bude portabilna između sistema sa različitim brojem jezgara. U svim ovim slučajevima, hardverska apstrakcija i balansiranje opterećenja obavljeno od strane SMP kernela će osloboditi programera od,

VxWORKS i VIŠEJEZGRENI PROCESORI

prilagođavanja aplikacije različitim hardverskim konfiguracijama.

AMP je bolji za sisteme koji su jasno podijeljeni na različite podsisteme i koji su uglavnom autonomni, ili imaju ograničene komunikacijske potrebe sa drugim čvorovima. Ovo sistemskim dizajnerima daje izbor dizajniranja hardvera sa mješavinom heterogenih procesora koji su idealni za operacije koje svaki čvor obavlja u sistemu. AMP je također najbolji izbor za redudantne sisteme kojima treba visoka dostupnost, obzirom da je svaki čvor u AMP sistemu samostalni jednoprocesorski računar.

VxWORKS i VIŠEJEZGRENI PROCESORI

VxWorks SMP

Wind River VxWorks SMP je dodatak svim Wind River VxWorks platformama koji pruža SMP mogućnosti VxWorks operativnom sistemu. VxWorks SMP omogućava višejezgrenim procesorima da dostignu konkurentno izvršavanje aplikacija, omogućavajući aplikacijama da unaprijede performanse kroz paralelizam.

Slijedeća tabela objašnjava prednosti i nedostatke svake OS konfiguracije.

VxWORKS i VIŠEJEZGRENI PROCESORI

AMP	SMP
Kada odabrati • Aplikaciju je lahko podijeliti na slabo uparene čvorove • Potrebna je redudantnost zbog pouzdanosti • Hardverska konfiguracija nije podesna za SMP • Neophodni su različiti procesori/OS • Dodjela aplikacija hardverskim resursima mora biti eksplicitna • Zahtijeva se minimum opterećenja kod raspoređivanja	**Kada odabrati** • Aplikacija podržava visoki stepen paralelizma kako bi se svi procesori što je moguće više opteretili • Taskovi/Niti treba da često dijele dosta podataka • Balansiranje opterećenja je bitno • Portabilnost aplikacije između različitog hardvera je esencijalno
Na šta obratiti pažnju • AMP dizajn može korstiti više memorije (više operativnih sistema) • AMP sistem mora biti podešen za hardversku konfiguraciju • Podešavanje komunikacije između jezgara i debagiranje je kompleksno	**Na šta obratiti pažnju** • SMP kerneli su više opterećeni neko jednoprocesorski kerneli • Aplikacije koje prave dosta OS poziva će se izvršavati sporije nego na jednoprocesorskom sistemu • Skrivene greške u softveru se javljaju pri prelasku sa jednoprocesorskog sistema

VxWORKS i VIŠEJEZGRENI PROCESORI

Uporedo sa pružanjem SMP mogućnosti, VxWorks SMP je dizajniran tako da bude kompatibilan sa jednoprocesorskom verzijom VxWorksa. Ovo dozvoljava VxWorks korisnicima da lakše migriraju svoje VxWorks aplikacije, drajvere i drugi softver sa jednoprocesorskog okruženja (VxWorks UP) na SMP okruženje.

VxWorks SMP se kupuje kao odvojeni dodatak bilo kojoj Wind River VxWorks 6.6 i kasnijim platformama (verzije 3.6 Wind River General Purpose Platform, Wind River Platform for Automotive Devices, Wind River Platform for Consumer Device, Wind River Platform for Industrial Devices ili Wind River Platform for Network Equipment). Nakon instalacije korisnici mogu po potrebi razvijati kako jednoprocesorske tako i SMP aplikacije.

VxWORKS i VIŠEJEZGRENI PROCESORI

VxWorks SMP osobine

VxWorks SMP omogućava simetrično multiprocesiranje VxWorks real-time operativnom sistemu. VxWorks SMP je izgrađen na istoj osnovi kao i VxWorks. Posebno, Vxworks SMP pruža slijedeće:

VxWorks pruža mogućnost VxWorks korisnicima da pokreću aplikacije u SMP okruženju na poznatim višejezgrenim procesorima. Ovo omogućava više funkcionalnosti, više performansi nego što je to slučaj sa jednoprocesorskim sistemima.

Kompatibilnost

VxWorks SMP je API kompatibilan sa jedno-procesorskom verzijom VxWorks operativnog sistema (uz par izuzetaka). Kompatibilnost API-ja promoviše iskorištavanje postojećeg VxWorks-baziranog softvera, dozvoljavajući kompanijama da unaprijede prethodne VxWorks investicije.

VxWORKS i VIŠEJEZGRENI PROCESORI

Glavne osobine kao što su real-time procesi, otkrivanje greške i izvještavanje su u potpunosti podržani čak i u SMP okruženju.

Real-time ponašanje

VxWorks dodaje SMP mogućnosti VxWorks operativnom sistemu bez ugrožavanja real-time karakteristika i ponašanja VxWorks operativnog sistema. Ovo omogućava VxWorks SMP-u da bude primjenjiv na ista ciljna tržišta i ciljne aplikacije kao i jednoprocesorski VxWorks sistemi. VxWorks SMP ima deterministički raspoređivač koji garantuje izvršavanje taska po prioritetu. Raspoređivač razmiješta N taskova najvećeg prioriteta koji su spremni na izvršavanje, gdje je N broj procesora u sistemu. Determinističko raspoređivanje zasnovano na prioritetu je također jedno od difirencirajučih osobina VxWorks operativnog sistema kod jednoprocesorskih sistema; to je ono što diferencira jedan real-time operativni

VxWORKS i VIŠEJEZGRENI PROCESORI

sistem od operativnih sistema opšte namjene. VxWorks SMP implementira paralelizam na nivou interapta, sposobnost više od jednog jezgra u sistemu da simultano rukuje različitim interaptima. Ovo povećava vrijeme reagovanja sistema. Mnoge kritične operacije unutar operativnog sistema su zaštičene spinlock čekanjem. Ove se odvijaju u determinističkom vremenu, što je funkcija vremena u kojem se drži spinlock. Spin-lock također štiti kritične podatke za vrijeme procesiranja interapta.

Podrška za najpoznatije višejezgrene procesore

VxWorks SMP se izvršava na svim poznatim procesorima koji su dostupni na tržištu. Ovo uključuje višejezgrene procesore od firmi ARM, Broadcom, Cavium, Freescale, Intel i Raza Microelectronics. Na ovaj način je korisnicima ponuđen širok izbor hardverskih platformi na kojima mogu izgraditi nove generacije svojih uređaja.

VxWORKS i VIŠEJEZGRENI PROCESORI

Portabilnost između različitih arhitektura

Ovo je zaštitni znak VxWorks operativnog sistema od njegovog početka. VxWorks je portabilan operativni sistem koji se izvršava na različitim procesorima sa uniformnim karakteristikama. SMP dakle proširuje VxWorks mogućnosti na polje višeprocesorskih sistema.

Dobro poznato razvojno okruženje

Wind River Workbench i njegovi alati rade i sa jednoprocesorskim i sa SMP verzijama VxWorksa. Također, alati imaju dodatne mogućnosti za VxWorks SMP koji su dizajnirani da vizualiziraju izvršavanje i učitavanje uzoraka koda na svim jezgrima

VxWorks SMP je pogodan za probleme koji trebaju SMP operativni sistem i karakteristike RTOS-a kao što je determinizam, niska latencija i malo zauzeće memorije.

VxWORKS i VIŠEJEZGRENI PROCESORI

U nastavku su prikazane razlike i jedinstvene mogućnosti jedne SMP konfiguracije. Sumarno, VxWorks SMP bi mogao biti pravo riješenje kada je jedno ili više slijedečih uslova ispunjeno:

1. Aplikacija je (ili će biti) dizajnirana u uslovima skupa paralelnih niti, dopuštajuči sistemu da obavi više posla simultano nego što bi obavio jednoprocesorski VxWorks sistem. Stoga mora biti dovoljno mnogo paralelizma u aplikaciji kako bi se postigle značajnije performanse korištenjem više procesora. Podsjetimo se Amdalovog zakona i kako on postavlja gornju granicu na to kakve se performanse mogu dobiti za dati nivo paralelizma.
2. Niti dijele veliku kolićinu podataka, ili imaju fino granulisan dizajn dijeljenja podataka.
3. Dizajner sistema treba da pusti operativnom sistemu da raspoređuje niti koje su spremne za izvršenje na sve dostupne

VxWORKS i VIŠEJEZGRENI PROCESORI

procesore (tzv. load balancing).

Na primjer, mrežni ruter može ostvariti bolju propusnost ako ima više jezgara koja paralelno obrađuju pakete iz više različitih tokova podataka (komunikacionih kanala koje preklapai rutira). Mrežna propusnost može također biti povećana tako što će se imati nekoliko jezgara koji će imati ulogu sigurnosnog procesiranja, gdje jedno jezgro može izvršavati glavnu logiku TCP/IP steka, dok su ostale koncentrisane na funkcije rutiranja i kriptovanja. Na primjeru mrežnog rutera, više jezgara mogu paralelno procesirati dodatne nezavisne tokove podataka, dok u drugom primjeru, dio mrežnog procesiranja se obavlja paralelno od strane drugih jezgara. Obadva ova primjera izlažu mogućnosti koje nam nudi mulitprocesiranje, obavljanjem više posla u datom kvantumu vremena nego što je to moguće uraditi sa samo jednim procesorom.

VxWORKS i VIŠEJEZGRENI PROCESORI

Nove mogućnosti u VxWorks SMP operativnom sistemu

Neke od novih osobina i API funkcija u VxWorks SMP-u su opisane u nastavku. Ovo nije iscrpan spisak osobina:

1. **Spinlocks** – Spinlock je lagani, brzi, sigurnosni mehanizam koji se koristi da sihnronizuje niti i rukovatelje (handlere) interpatima koji se izvršavaju na različitim procesorima u SMP sistemu. Spinlock mogu biti realizovni kao mutex semafori za više procesorska okruženja. Postoje dvije osnovne vrste spinlock mehanizama (na nivou taska i na nivou interapta) i mogu se koristiti za ostvarivanje sinhronizacije između različitih taskova i između taskova i interapt servisnih rutina.
2. **Atomične operacije** – Atomične operacije su klasa API poziva koji omogućavaju brze sigurne višeprocesorske operacije na jednostavnim memorijskim varijablama, na primjer, inkrement, dekrement, logičke operacije i jednostavne

VxWORKS i VIŠEJEZGRENI PROCESORI

aritmetičke operacije. Atomične operacije razvijene za VxWorks SMP su također dostupne i u jednoprocesorskoj varijanti VxWorks sistema.

3. **Reader-writer semafori** – Reader-writer semafor je nova vrsta semafora u VxWorks 6.6 operativnom sistemu. Reader-writer semafor dopušta da više niti koje čitaju ostvaruje brži pristup dijeljenoj strukturi podataka. U jednom trenutku je samo jednoj niti koja piše (writer thread) dopušteno da mijenja strukturu podataka. Ovi semafori su optimizirani za situacije sa višestrukim reader nitima i jednom writer niti. Njihova upotreba se susreće kod proizvođač-potrošač vrsta algoritama i posebno je pogodna za multiprocesirajuče sisteme. Kao i VxWorks mutex semafori, reader-writer semafori također podržavaju inverziju prioriteta kao opciju.

VxWORKS i VIŠEJEZGRENI PROCESORI

Wind River Workbench i VxWorks SMP

VxWorks UP i SMP dijele isto Wind River Workbench razvojno okruženje. Workbench je nadograđen sa ciljem da podrži debugging i analiziranje SMP okruženja. Workbench 3.2 ima mogućnost da debagira VxWorks SMP kernel i real-time procese (RTP) kako u task (user) tako i u sistemskom modu. Workbench pojednostavljuje otkrivanje utrke za podacima (data race), međublokada (deadlock), što su dva najčešća pitanja koja se pojavljuju prilikom razvoja programa za višejezgrena okruženja.

- Kod Workbencha, Vxworks SMP projekti se podešavaju na identičan način kao da se radi o jednoprocesorskoj verziji VxWorks sistema. Korisnici mogu odabrati da razvijaju UP i SMP projekte. Kako svaki projekt ili aplikacija nije podesna za SMP, korisnici mogu odabrati konfiguraciju OS-a koja najbolje odgovara njihovim potrebama. Workbench proširenja kao

VxWORKS i VIŠEJEZGRENI PROCESORI

Workbench Debugger, System Viewer, Workbench Performance Profiler, Workbench Code Coverage Analyzer, Workbench Memory Analyzer, Workbench data monitor i Workbench Function Tracer su nadograđeni tako da mogu biti upotrebljeni kako kod UP tako i kod SMP sistema.

Debagiranje SMP sistema

Greške i problemi sa kojima se programer susreće u SMP sistemu su ista vrsta problema koji se pojavljuju u UP višenitnom sistemu. Neke greške će se prije pojaviti u SMP sistemu nego u UP sistemu, posebno ako softver pravi određene pretpostavke po pitanju koherencije, ali to nisu striktno SMP pitanja. Debagiranje SMP sistema nije inherentno različito u odnosu na UP sisteme. Sljedeće je neka vrsta vodiča o problemima sa kojima se susreću programeri kod SMP sistema i kako treba da ih riješavaju:

VxWORKS i VIŠEJEZGRENI PROCESORI

- Problemi koji se javljaju iz nezaštičenih ili nedovoljno zaštičenih kritičnih sekcija, kao što je utrka za podacima i uzurpiranost memorije, se najčešće javljaju kada se niti izvršavaju konkurentno. Ovo se dešava iz razloga što softver koji se izvršava konkurentno je daleko skloniji greškama nego softver koji se izvršava sa malo ili nimalo konkurentnosti. Memory Analyzator i system Viewer pomažu u otkrivanju grešaka, na isti način kao i kod UP sistema.
- Inverzija prioriteta, na primjer, kada nit niskog prioriteta nasljeđuje prioritet preko niti višeg prioriteta koja čeka na resurs, može biti uzrokovana kada druge niti naprave određene pretpostavke po pitanju prioriteta niti. System Viewer može pomoći u vizualizaciji baš kao i kod UP sistema.
- Pitanja konkurentnosti interapta taska su rezultat rukovanja interpatima na jednoj jezgri dok druga jezgra izvršava task. VxWorks aplikacije mogu pretpostaviti da je procesiranje

VxWORKS i VIŠEJEZGRENI PROCESORI

taska i interapta uzajamno isključivo, što nije istina kod SMP sistema. System Viewer može pomoći da se ovo vizualizuje baš kao i kod UP sistema.

Generalno, sva uobičajena pitanja koja uz pomoć run-time alata možemo observirati i razumjeti rade jednako i sa UP i sa SMP sistemima. Jednako je lahko koristiti alate u višejezgrenom kao i u jednoprocesorskom okruženju.

Performance profiler

Preformance profiler nadgleda tok izvršavanja real-time programa na funcija-po-funkcija osnovi. On dozvoljava programerima sistemskih i ugrađenih aplikacija da tačno analiziraju kako ugrađeni programi iskorištavaju procesor. Preformance profiler je dizajniran tako da ostvari minimalan utjecaj na performanse real-time koda koji se analizira i ne zahtijeva se posebno kompajliranje. Ovaj alat se može iskoristiti u analizi svih procesa koji se simultano izvršavaju na

VxWORKS i VIŠEJEZGRENI PROCESORI

ugrađenom sistemu

Pregled osobina Performance Profilera

Performanse su kritični dio svakog ugrađenog real-time sistema. Uzroke siromašnih performansi je obično lahko riješiti, ali teško pronaći. Performanse Profiler nam omogućava da analiziramo gdje CPU troši cikluse. On pruža detaljnu funkcija-po-funkcija analizu, razbijanje pojedinačnih rutina u okviru procesa koji koriste procesor.

Performance Profiler generiše statistiku iz koje možemo saznati slijedeće

- Šta radi procesor
- Koje su rutine pozivane
- Koje rutine poziva svaka rutina

VxWORKS i VIŠEJEZGRENI PROCESORI

Ove informacije ukazuju na neefikasnosti koje nam dopuštaju da fino podesimo sistem za maksimalne performanse.

Performanse Profiler prikazuje izvršavanje u slijedečim modovima:

- Kao stablo sa punim prikazom steka poziva –System pogled
- Kao lista najviše korištenih rutina – Function pogled
- Lista profiliranih procesa – Process pogled

Performanse Profile može da prikaže statistiku slijedečih komponenti:

- Cijelog sistema
- Bilo kojeg skupa procesa
- Jednog procesa
- Jednog Linux korisnika
- Individualnog procesora na višeprocesorskom Linux sistemu

VxWORKS i VIŠEJEZGRENI PROCESORI

Obzirom da Performance Profiler ne zahtijeva posebno kompajliranje ili poseban hardver on također može analizirati kôd koji nismo mi napisali kao što su rutine operativnog sistema ili third-party biblioteke.

Stek poziva

Svaki proces ima svoj stek. Stek se koristi kao privremeni prostor za aplikacijske varijable i stanje programa. Svaki put kad procesor pokrene funkciju, on alocira novu sekciju u steku kako bi držao informacije koje su potrebne funkciji. Ova sekcija se zove stek okvir funkcije (function stack frame). Ona sadrži privremene varijable i nekoliko ključnih informacija za funkciju uključujući sačuvana stanja registara procesora i povratnu adresu parent (pozivajuče) rutine.

Za kôd koji je kompajliran sa opcijom koja uključuje pokazivače okvira (frame pointers – FP), specijalni registar u procesoru koji uvijek pokazuje na stek okvira

VxWORKS i VIŠEJEZGRENI PROCESORI

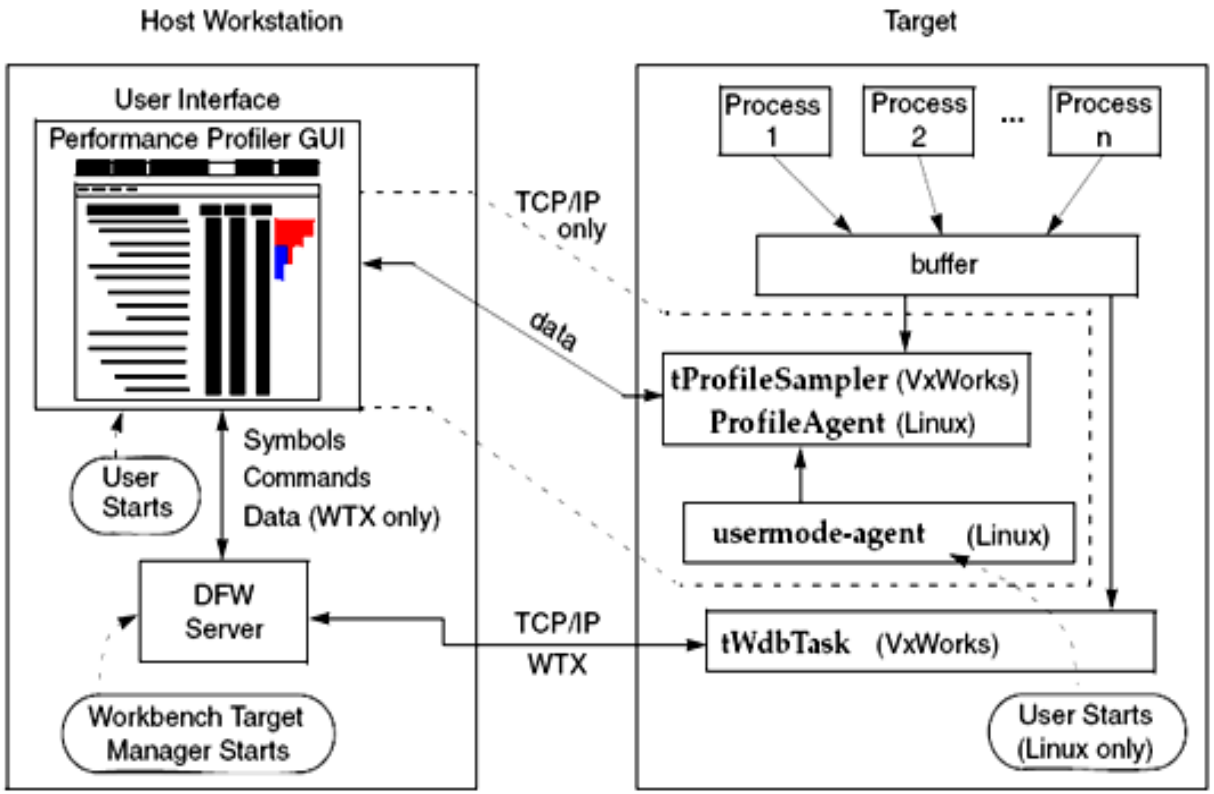
funkcije koja se trenutno izvršava.

- Kada se pozove rutina, stari FP se pohranjuje na stek, kreira se novi stek okvira i FP se ažurira. Dakle, u bilo kojem trenutku vremena, postoji cijelokupna historija poziva funkcija na steku tako da se može odrediti lanac pokazivača smještenih na stek. Ova historija poziva funkcija je poznata kao stanje izvršavanja (execution state).
- Performanse Profiler koristi tehniku nazvanu statističko profiliranje, koja radi tako što periodično uzima uzorke stanja izvršavanja i koristi ih u izračunavanju gdje procesor troši svoje vrijeme. Analiziranjem programskog brojača, Performanse Profiler može ustanoviti koja se funkcija u programu izvršavala u vremenu uzorka. Analiziranjem lanca stek okvira, Performance Profiler može ustanoviti sekvencu funkcija koja je rezultirala u trenutnom stanju izvršavanja.

VxWORKS i VIŠEJEZGRENI PROCESORI

Arhitektura Performance Profilera

Arhitektura Performance Profilera se sastoji od dvije glavne komponente; agenta za sakupljanje (collection agent) koji se izvršava u realnom vremenu na ciljnoj VxWorks ili Linux mašini i GUI-a koji se izvršava na hostu, kako je prikazano na slici



VxWORKS i VIŠEJEZGRENI PROCESORI

Postoje manje razlike u implementaciji ciljne arhitekture između VxWorks i Linuxa. Te razlike su objašnjene u nastavku.

VxWorks target arhitekture

Komponente komuniciraju kroz VxWorks Target Manager ukjlučujući DFW server i ako je dostupan TCP/IP link ili opciono WTX link.

Performance Profiler obavlja uzorkovanje na ciljnoj mašini korištenjem visoko-prioritetnog taska **tProfilerSampler**. Ovaj task smješta uzorke u red poruka za kasnije korištenje od strane taska za analizu koji se izvršava na hostu. Ovo omogućava efikasan mehanizam uzorkovanja koji ima minimalan uticaj na izvršavanje ciljnog sistema. Na prethodnoj slici je prikazano kako Performance Profiler prima ove uzorke iz Profile Agenta u TCP/IP režimu rada ili direktno putem DFW servera u WTX modu.

VxWORKS i VIŠEJEZGRENI PROCESORI

Linux ciljne arhitekture

Komponente na Linux ciljnoj arhitekturi komuniciraju sa GUI-jem samo preko TCP/IP linka. ProfileAgent je sampler na ciljnoj mašini pokrenut od strane Performance Profilera za Linux. Kada je pokrenut, ProfileAgent postavlja System Clock da periodično prekida mašinu. Upravljač prekidima, također postavljen od strane ProfileAgenta, uzorkuje proces koji se izvršava prije nego što se upravljač pozove. Upravljač prekidima sakuplja podatke od procesa i smješta ih u bafer. ProfileAgent periodično provjerava ovaj bafer i prebacuje sadržaj na host-bazirani GUI preko TCP/IP konekcije.

GUI na Hostu

GUI koristi izlaz statističke analize kako bi održavao dinamički zapis sistemskih aktivnosti. Ovaj izlaz se sastoji od niza brojeva za svaku rutinu pri čemu ti brojevi predstavljaju procenat vremena koje procesor potroši na tu rutinu.

VxWORKS i VIŠEJEZGRENI PROCESORI

Ovi brojevi su u rangu od 0.0 (nikada aktivan) do 100.0 (uvijek aktivan). Analiza računa i trenutnu (current) i kumulativnu (cumulative) statistiku. Kao dio uzorkovanih podataka, Performance Profiler uključuje i vrijednost programskog brojača (PC), i prolazi stekom do zapisa svih rutina kako bi mogao aktivirati trenutnu rutinu. PC također može pokazati gdje je svaka podrutina pozvana iz jedne funkcije (exploded mod) kako bi izračunao korištenje svake individualne podrutine. Korištenjem ove informacije, GUI gradi proširivo stablo koje prikazuje putanje do poziva svake podrutine.

GUI obavlja statističku analizu nad podacima svakog procesa i prikazuje analizirane informacije. Međutim, ako se ne ustanove svi simboli iz steka tada se umjesto naziva rutine stavlja hexadecimalna adresa simbola i cijeli primjer se samo baferuje i ne prikazuje se. Informacije obavljene analize se

periodično ažuriraju u grafičkom interfejsu.

Postavljanje korisničkih parametara

Performanse Performance Profilera mogu biti naštimane tako da se najefikasnije analizira izvorni kod i pomogne programeru da brzo identifikuje probleme. Slijedeći podesivi parametri se mogu podesiti u Performance Profileru. Ti parametri su:

- Stopa uzorkovanja (sample rate)
- Period analize
- GUI period ažuriranja
- Modovi analize
- Digitalni filteri podataka

VxWORKS i VIŠEJEZGRENI PROCESORI

Stopa uzorkovanja

Stopa uzorkovanja je frekvencija pri kojoj Performance Profiler uzima uzorke stanja izvršavanja taska. Standardna stopa uzorkovanja je 113 Hz. Stope prostih brojeva su najbolje iz razloga što maksimiziraju slučajnost uzoraka i minimiziraju sinhronizaciju sa raspoređivačem. Međutim, u zavisnosti od dostupnog hardvera, ova stopa možda neće biti dostižna; u ovim slučajevima, Performance Profiler odabire najbližu stopu.

Period analize

Period analize je vrijeme koje Performance Profiler čeka prije obavljanja druge statističke analize uzoraka. Uobićajeni period analize je 2.5 sekunde. Sa uobićajenom stopom uzorkovanja i uobićajenim periodom analize, svaki period analize obićno procesira približno 282 uzorka.

VxWORKS i VIŠEJEZGRENI PROCESORI

Period ažuriranja grafičkog prikaza

Predstavlja količinu vremena koje Performance Profiler čeka između dva osvježavanja prikaza. Standardno vrijeme osvježavanja je 10 sekundi. Manji periodi pružaju ažurnije podatke, ali zato troše više procesorskog vremena na hostu. Period osvježavanja mora biti proizvod perioda analize. Ne može biti manji od 2.5 sekunde.

Modovi analize

Performance Profiler može nadgledati aktivnost cijelog sistema ili jednog procesa. Ova dva moda pružaju različite poglede na performanse sistema. Na primjer, razmotrimo jedan proces koji je blokiran većinu vremena.

U sistemskom modu, Performance Profiler nadgleda aktivnosti svih procesora na ciljnom sistemu. Vrijeme iskorišteno od strane funkcija procesa se prikazuje kao procenat ukupnog vremena.

VxWORKS i VIŠEJEZGRENI PROCESORI

Dakle, proces koji je blokiran većinu vremena će zabilježiti mali procenat iskorištenosti procesora.

- U jednoprocesnom modu, Performance Profiler nadgleda samo aktivnost jednog odabranog procesa, prikazujuči nam gdje proces troši svoje vrijeme. Vrijeme korišteno od strane funkcija tog procesa se prikazuje kao procenat ukupnog vremena korištenog od strane tog procesa. Dakle, funkcije kod kojih je proces blokiran se prikazuju kao da uzimaju više procenata vremena. Suma korištenja procesora za sve funkcije u procesu je 100%.

Digitalni filteri podataka

Statistike za svaki profil su digitalno filtrirane kako bi se eliminisao šum u podacima. Stoga, Current statistike svake funkcije (ili niti) su prosječne vrijednosti većine nedavnih analiza i prošlih rezultata. Iako standardni parametri filtera rade dobro za većinu aplikacija, Performance Profiler dopušta

da se podese parametri kako bi se kontrolisao odgovor trenutne statistike.

Sugestije za efikasnije korištenje Performance Profilera

U nastavku ćemo dati neke sugestije vezane za odabir kolone koju želimo prikazati u Performance Profileru u zavisnosti od toga šta želimo dalje da uradimo po pitanju optimizacije.

Da bi optimizirali kod minimiziranjem poziva „skupih“ funkcija treba da uradimo slijedeće:

- Odaberemo **Cumulative Indirect %**, **Current Indirect %** i **Max Indirect %** kolone, a ostale isključimo.
- Lociramo funkciju koja najviše iskorištava procesor.
- Proširimo funkcije kako bi utvrdili gdje se pojavljuju „skupi“ pozivi funkcija.

VxWORKS i VIŠEJEZGRENI PROCESORI

- Pregledom izvornog koda ustanoviti da li su ti pozivi neophodni.

  Da bi ustanovili dugoročno prosječno korištenje procesora u sistemu trebamo:

- Koristiti **Cumulative Indirect %** i **Cumulative Direct %** kolone. Ostale isključiti.

  Ovaj pogled je pogodan za sisteme koje imaju stalno jednaku aktivnost. Za ovakve sisteme, trenutna statistika bi trebala biti slična Kumulativnoj tako da nema potrebe za prikazivanjem trenutne statistike.

  Da bi ustanovili kratkoročno prosječno iskorištenje sistema treba:

- Koristiti **Current Indirect %** i **Current Direct %** kolone, a ostale isključiti.
- Uzorkovati događaj u sistemu koji inicira aktivnosti koje želimo.

VxWORKS i VIŠEJEZGRENI PROCESORI

- Nakon dobijanja statistike, odabrati Stop na alatnoj letvici kako bi zaustavili naredna ažuriranja.
- Pregledati rezultate.
- Možda će biti potrebno primjeniti filtere procesa kako bi izbacili taskove koje ne želimo da posmatramo.

Statistička polja

Sva satistička polja (kolone) za Function pogled su Direct statistike. Polja u Function pogledu koja se mogu odabrati korištenjem Columns pogleda su:

- **System cumulative %**

Predstavlja dugoročnu prosječnu upotrebu procesora od strane rutine, ne uključujući bilo koje njene podrutine ili direktno korištenje procesora u procentima od ukupnog procesorskog vremena.

VxWORKS i VIŠEJEZGRENI PROCESORI

- **System Current %**

  Predstavlja kratkoročnu prosječnu upotrebu procesora od strane rutine, ne uključujući bilo koje njene podrutine, ili direktnu upotrebu procesora izraženu u procentima od ukupnog procesorskog vremena.

- **System Max %**

  Predstavlja najveću vrijednost kratkoročnih prosjeka upotrebe procesora od strane rutine, ne uključujući bilo koje pozive podrutina, ili direktno korištenje procesora izraženo u procentima od ukupnog procesorskog vremena.

- **System Cumulative Count**

  Ovaj broj se koristi za izračunavanje procenta za System Cumulative Direct. Resetuje se na nulu kada se pokrene nova sesija, ili kada se izmijeni mod analize. Predstavlja brojač koliko puta se neki uzorak pojavio kada je ciljna mašina izvršila neku rutinu, ali ne i bilo koju od njenih podrutina.

VxWORKS i VIŠEJEZGRENI PROCESORI

- **System Current Count**

  Brojač koliko se puta pojavio neki uzorak kada ciljna mašina izvršava rutinu, ali ne i njene podrutine. Ova vrijednost se koristi da se izračuna procenat za System Current Direct. Vrijednost brojača se resetuje na nulu svaki put kada Performace Profiler završi sa izračunom skupa statistika (svaki period analize).

  Na Linuxu baziranoj ciljnoj višeprocesorskoj mašini je moguće vizualizirati bilo koji od ovih statističkih polja za jedan procesor ili za cijeli sistem.

  Interpretacija podataka

  Kako bi se Performanse Profiler koristio što je moguće efikasnije, u nastavku ćemo se osvrnuti na način kako da se interpretiraju podaci prikupljeni sa Performance Profilerom.

VxWORKS i VIŠEJEZGRENI PROCESORI

Statistička tačnost u odnosu na broj uzoraka

Tačnost statističkih podataka je određena brojem uzoraka koje prikupi Performance Profiler. Jako je bitno razumijeti koliko dugo profiler treba da sakuplja podatke kako bi obezbijedio potrebnu tačnost. Dobro poznata formula koja se koristi kod Performance Profilera je da izračuna broj uzoraka potrebnih za postizanje tačnosti sa određenim nivoom povjerenja je:

$$N = \frac{pqZ^2\left(\frac{(1+g)}{2}\right)}{e^2}$$

gdje je:

N je broj uzoraka potrebnih da bi se dostigla specificirana tačnost.

p je vjerovatnoča da se desi neki događaj. Vrijednost 0.01 znači da smo zainteresovani za rutinu sa otprilike 1% iskorištenja procesora.

VxWORKS i VIŠEJEZGRENI PROCESORI

q = 1 – p (je vjerovatnoča da se događaj ne dogodi)

Z- je inverzna vrijednost integrala funkcije Normalne distribucije

g - je koeficijent pouzdanosti sa vrijednošću između 0 i 1. Vrijednost od 0.95 znači da želimo biti 95% sigurni u rezultat.

e je interval pouzdanosti, na primjer, tolerancija greške rezultata. Vrijednost od 0.005 znači da želimo da greška bude u granicama od +/-0.25%.

Primjer: Tipična tačnost

Da bi tačno odredili uska grla u kodu, obično je dovoljno da prikupimo statističke podatke od Performance Profilera sa 95% tačnošću do najbliže pola procenta iskorištenosti procesora. Ovo znači da smo 95% sigurni da su prikupljeni podaci u okviru plus ili minus 0.25% njihovih stvarnih vrijednosti. Ovaj nivo tačnosti je dostupan skoro pa trenutno, kao što se može vidjeti za rutinu koja koristi 1% procesora:

VxWORKS i VIŠEJEZGRENI PROCESORI

$p = 0.01$

$q = 0.99$

$g = 0.95$

$e = 0.005$

$Z[...] = 1.96$

Korištenjem ovih vrijednosti u formuli daje rezultat od približno 1500 uzoraka ili oko 13 sekundi prikupljanja podataka sa Performance Profilerom uz sample rate od 113Hz.

Primjer: Visoka tačnost

Da bi dostigli tačnost od 95% pouzdanosti za istu rutinu koja koristi 1% procesora, vrijednosti koji se koriste u formuli su:

VxWORKS i VIŠEJEZGRENI PROCESORI

$p = 0.01$

$q = 0.99$

$g = 0.95$

$e = 0.001$

$Z[...] = 1.96$

Ovaj rezultat je za približno 37,000 uzoraka, ili oko 5.5 minuta prikupljanja podataka sa Performance Profilerom uz sample rate od 113Hz. Povećanjem brzine uzorkovanja na 500Hz vrijeme se skračuje na jednu minutu.

Performance Profiler računa kratkoročnu i dugoročnu statistiku iskorištenja procesora. Kartakoročna statistika je poznata kao Current prosjeci. To su vrijednosti koje pružaju dinamičniji pogled na sistem.

VxWORKS i VIŠEJEZGRENI PROCESORI

Dugoročna statistika je poznata kao “Cumulative” prosjeci. Izračunava se korištenjem uzoraka sakupljenih za cjeokupnu sesiju. Maksimalne vrijednosti prikazuju najveću iskorištenost procesora od “Current” prosjeka za svaku rutinu ili proces za vrijeme trenutne sesije.

Kada analiziramo aplikacije, korisno je da se istraže slijedeći parametri:

- **Indirektna statistika** – Ukupno vrijeme koje procesor potroši na funkciju, koje se sastoji od procenta vremena koje procesor potroši na tu funkciju, plus procenat vremena koje se utroši na ostale funkcije koje poziva ta funkcija.
- **Direktna statistika** – Procenat vremena koje procesor potroši samo na funkciju (na primjer, isključujući procesorsko vrijeme utrošeno na ostale funkcije koje poziva ta funkcija).

VxWORKS i VIŠEJEZGRENI PROCESORI

Indirektnu i direktnu statistiku treba koristiti kako bi se odredila iskorištenost procesora od strane koda funkcije ili od strane podrutina te funkcije.

Na primjer, ako je indirektna iskorištenost funkcije 10.0%, i Direktna iskorištenost 9.9%, tada je procesor iskorišten samo od strane funkcije bez bilo kojeg poziva podrutine. Na taj način se možemo skoncentrisati samo na tu funkciju u pokušaju da je optimiziramo.

VxWORKS i VIŠEJEZGRENI PROCESORI

System Viewer

System Viewer pruža mogućnost obavljanja analize kako bi pomogao programerima koji razvijaju SMP kernel i aplikacije. Za SMP, System Viewer grafik događaja je popunjen informacijama koje pomažu programerima da identifikuju i izoliraju utrkivanje za podacima, međublokade i izgladnjivanja, kako kod UP tako i kod SMP sistema. Ovo može biti korisno pri minimizaciji gubitka performansi. Korištenjem System Viewera korisnici mogu vidjeti odvijanje paralelnih aktivnosti u sistemu: paralelizam na nivou taska, interapt-interapt paralelizam. Ovo može biti korisno pri vizualizaciji izvršavanja aplikacije i pomoći kod dijagnosticiranja grešaka u softveru koje proizilaze iz konkurentnog izvršavanja. System Viewer prikazuje nivo opterečenosti procesora za svaku njegovu jezgru što je korisno kod podešavanja performansi i raspoređivanja.

VxWORKS i VIŠEJEZGRENI PROCESORI

opterećenja na svim jezgrima u sistemu.

Wind River System Viewer pruža slijedeće mogućnosti:

- Detektuje utrke za podacima, međublokade, neiskorištenost procesora, i druge probleme koji se odnose na taskove.
- Određuje brzinu aplikacije i njene performanse.
- Posmatra cikličke uzorke u ponašanju aplikacije
- Sprema podatke za kasniju analizu.

System Viewer alati

Glavni System Viewer alati su:

- System Viewer Configuration – Ovaj alat se koristi za konfiguraciju šta želimo da upisujemo, kako da prebacimo log fajl sa target na host i da gledamo analizirane podatke.

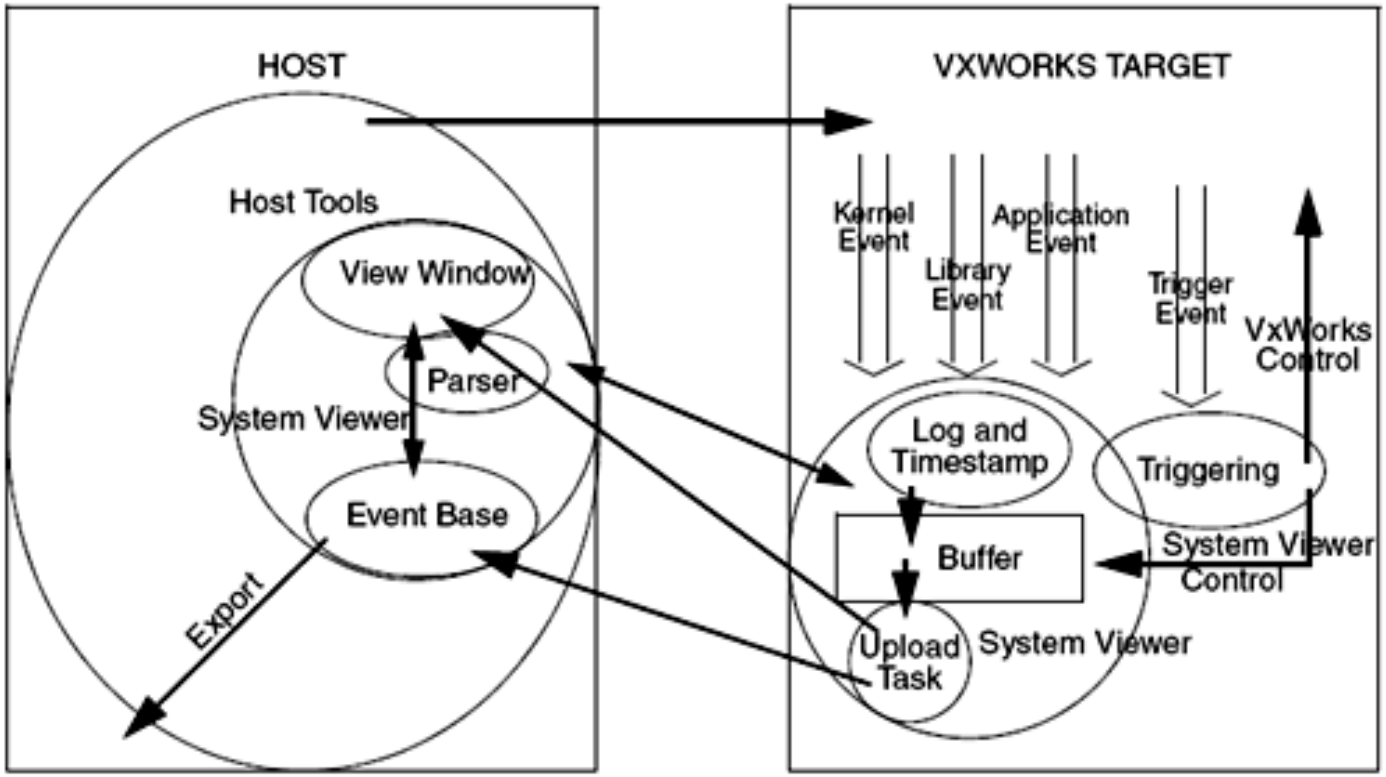
VxWORKS i VIŠEJEZGRENI PROCESORI

- Triggering - koristi se da precizno odredimo kada (sa kojim događajem) da počnemo i zaustavimo upis podataka u log fajl. Većina događaja koji se upisuju također mogu biti korišteni kao trigger.
- Event Recieve – Socket listener za prikupljanje log podataka prebačenih od strane ciljne mašine.
- Log Viewer – Koristi se za analizu prikupljenih podataka. Prikupljeni podaci se mogu sačuvati sa ili bez trenutnih postavki vezanih za način prikazivanja, u System Viewer formatu ili u tekstualnom formatu.
- Analysis Suite Viewers – Koristi se za posmatranje iskorištenosti procesora i podataka vezanih za događaje iz System Viewer log fajlova. Ovi alati, dostupni iz System Viewer Configuration editora, su korisni u kontektsu višestrukih procesora.

VxWORKS i VIŠEJEZGRENI PROCESORI

Arhitektura System Viewer –a

Kao što se može vidjeti na slici, System Viewer pruža i target i host funkcionalnosti, pri čemu se većina procesiranja obavlja što je moguće više na hostu kako bi se minimizirao efekat upisa podataka u log fajl na ostale aktivnosti:



VxWORKS i VIŠEJEZGRENI PROCESORI

Komunikacija između hosta i targeta se vrši preko WDB protokola i to preko serijske linije ili mrežne konekcije. Ova putanja se dijeli od strane svih Wind River Workbench alata kako bi se ostvarila komunikacija sa target mašinom.

VxWorks Simulator

VxWorks Simulator može simulirati SMP aplikacije na jednom ili više procesora. Korištenjem simulatora korisnici mogu obavljati poslove od osnovnih do složenih izmjena aplikacija, simulacija i podešavanja prije prebacivanja aplikacije na stvarni višejezgreni hardver. System Viewer je također u mogućnosti da na host mašini prikuplja i vizualizuje SMP podatke čak i kad se koristi Simulator. Treba napomenuti da Simulator ne može biti korišten u određivanju performansi SMP aplikacije iz razloga što će hardver na kojem se odvija simulacija biti bitno drugačiji od željenog ciljnog hardvera.

VxWORKS i VIŠEJEZGRENI PROCESORI

Rad sa VxWorks Simulatorom

U nastavku će biti opisana upotreba Wind River VxWorks Simulatora i Wind River Workbench alata u razvoju jedne SMP aplikacije.

Kreiranje Imagea sa podrškom za SMP za VxWorks Simulator

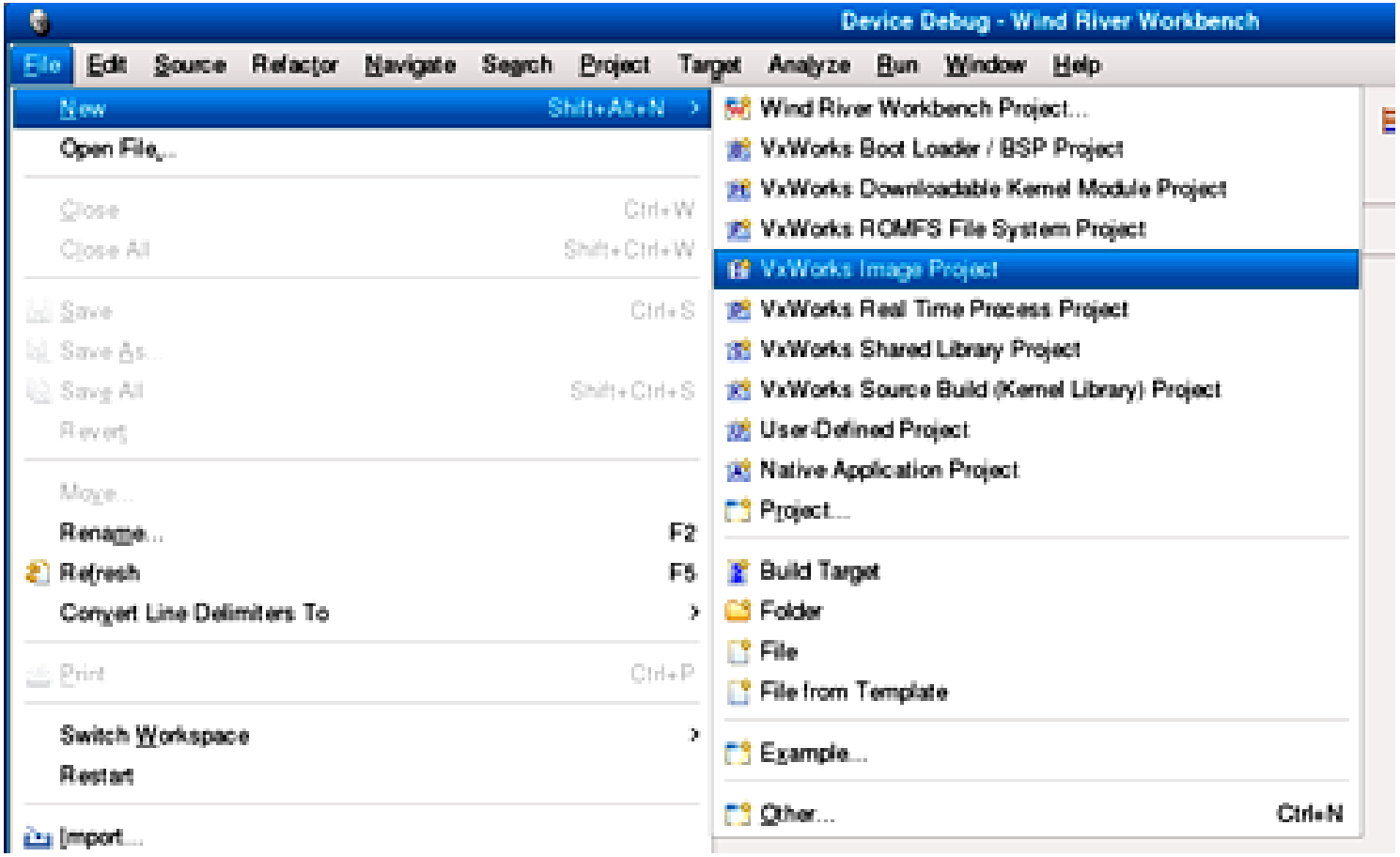
Najprije je potrebno kreirati funkcionalni VxWorks simulator koji može simulirati do osam jezgara. Simulator se može koristiti tokom faze razvoja ili kada je hardver preskup ili nije dostupan. Svi Wind River alati su u potpunosti funkcionalni sa VxWorks simulatorom i ostavljaju utisak kao da smo spojeni na stvarnu target ploču. Da bi kreirali image treba da uradimo slijedeće:

.

VxWORKS i VIŠEJEZGRENI PROCESORI

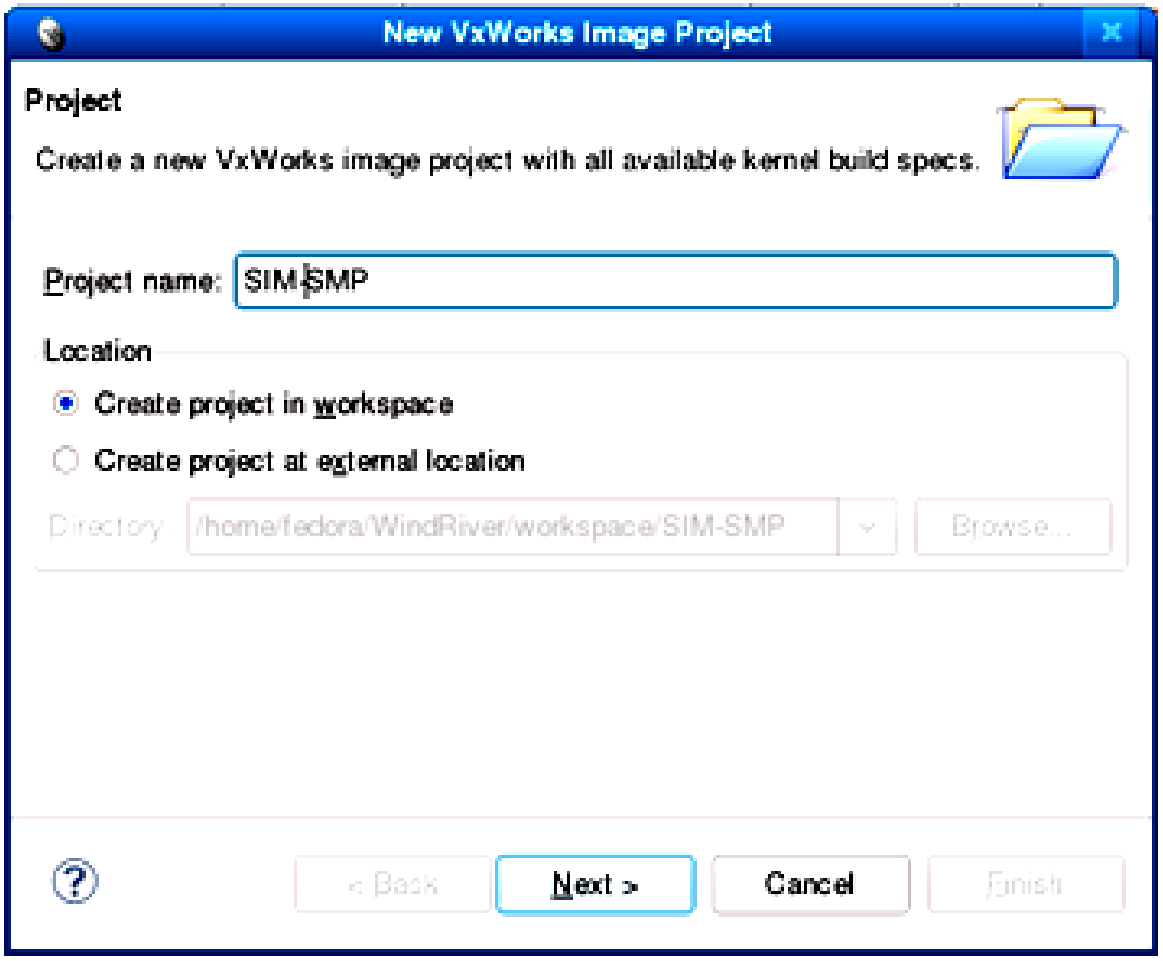
Korak 1 – Kreiranje novog VxWorks Image projekta: **File –>New->VxWorks Image Project**

.



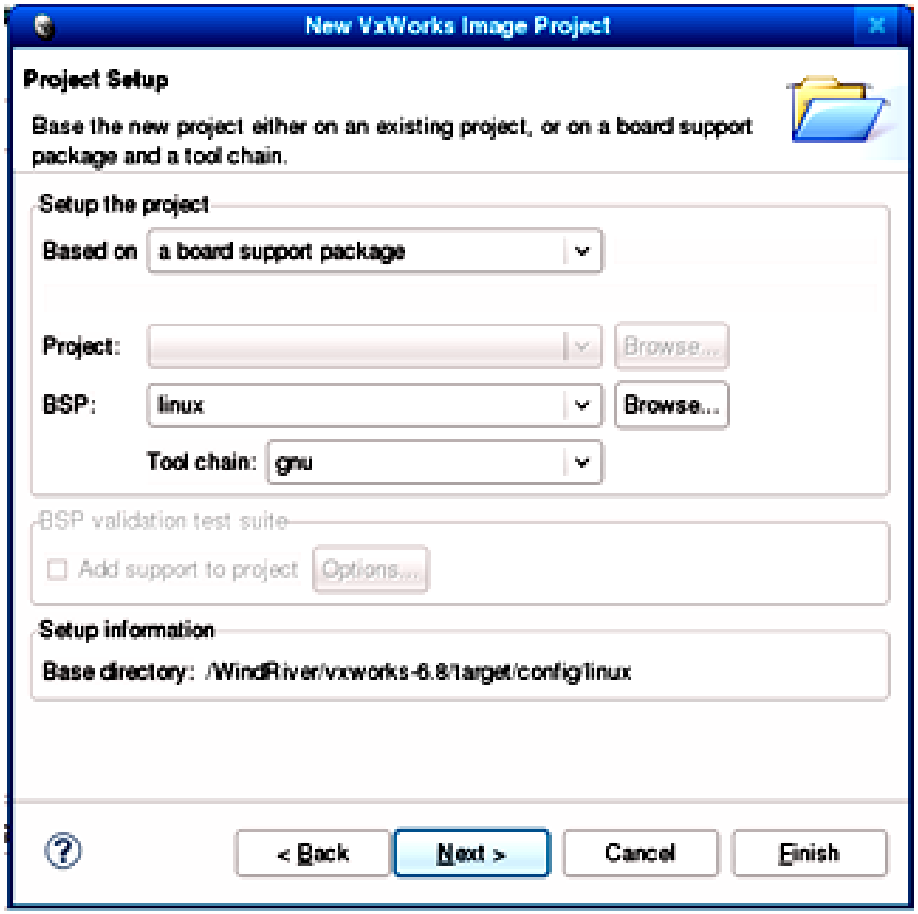
VxWORKS i VIŠEJEZGRENI PROCESORI

Korak 2 – Dodijeljivanje imena SIM-SMP u VxWorks Image Project dijalogu.



VxWORKS i VIŠEJEZGRENI PROCESORI

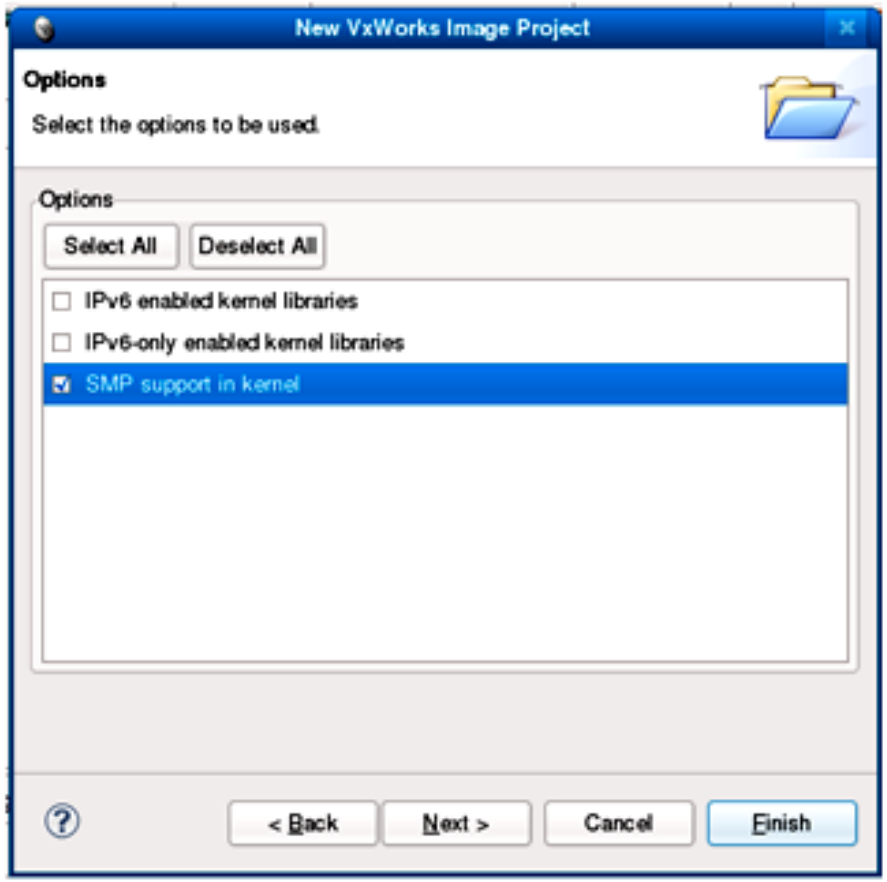
Korak 3 - U New VxWorks Image Project – Project Setup dijalogu ostaviti sve parametre na njihove defaultne vrijednosti. Obzirom da će se simulator izvršavati na Linuxu, treba da koristimo Linux kao osnovu za projekt.



VxWORKS i VIŠEJEZGRENI PROCESORI

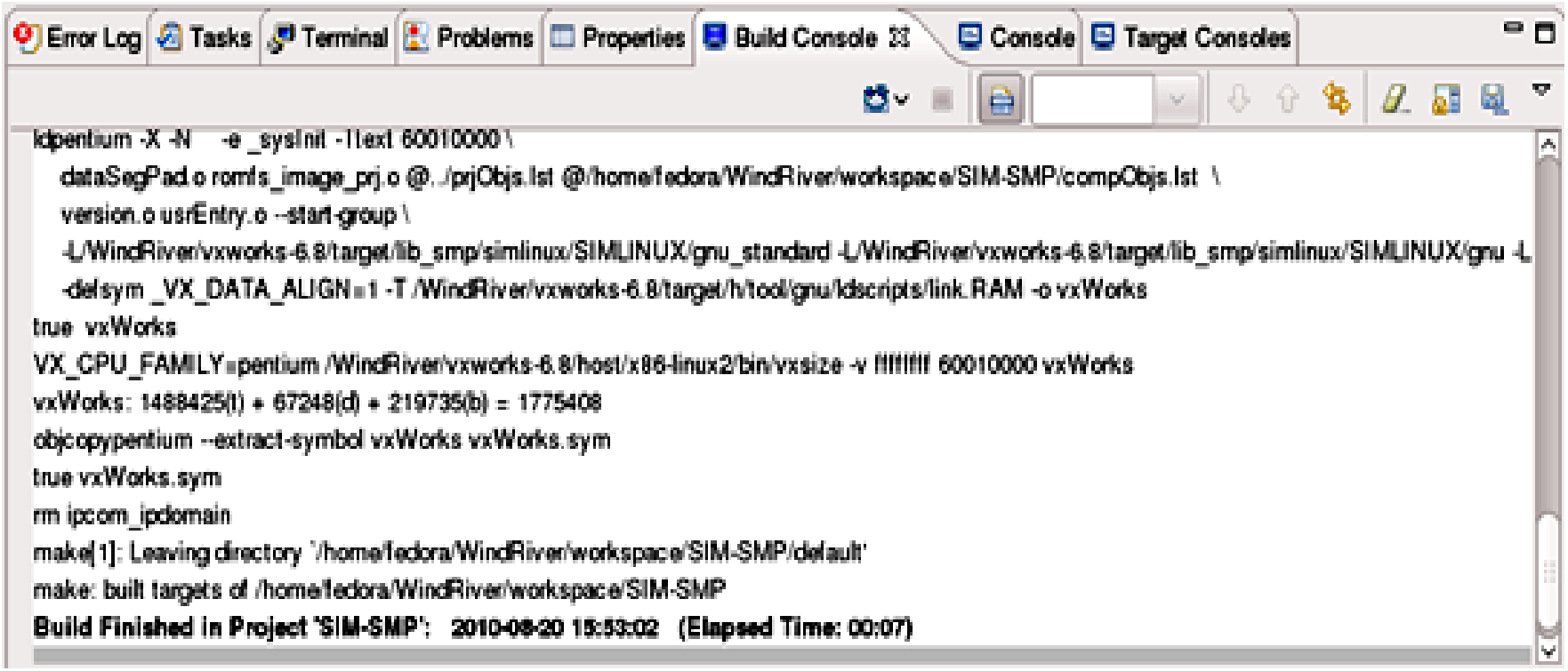
Korak 4 - Provjeriti da je Tool chain postavljen na **gnu**.

Korak 5 - U New VxWorks Image Project – Options dijalogu, odabrati **SMP support in kernel** opciju kako bi uključili sve neophodne module koji su potrebni za SMP VxWorks kernel.



VxWORKS i VIŠEJEZGRENI PROCESORI

Ovo su sve opcije koje moraju biti konfigurisane kako bi kreirali novi Image projekt. Kreirani Image projekt možemo pogledati u Project Exploreru. Da bi izgradili novi VxWorks image, moramo odabrati SMP-SIM projekt u Project Exploreru a zatim odabrati **Rebuild Project**. Kada je image izgrađen, u konzolnom prozoru možemo vidjeti indikaciju da je proces kompletiran .

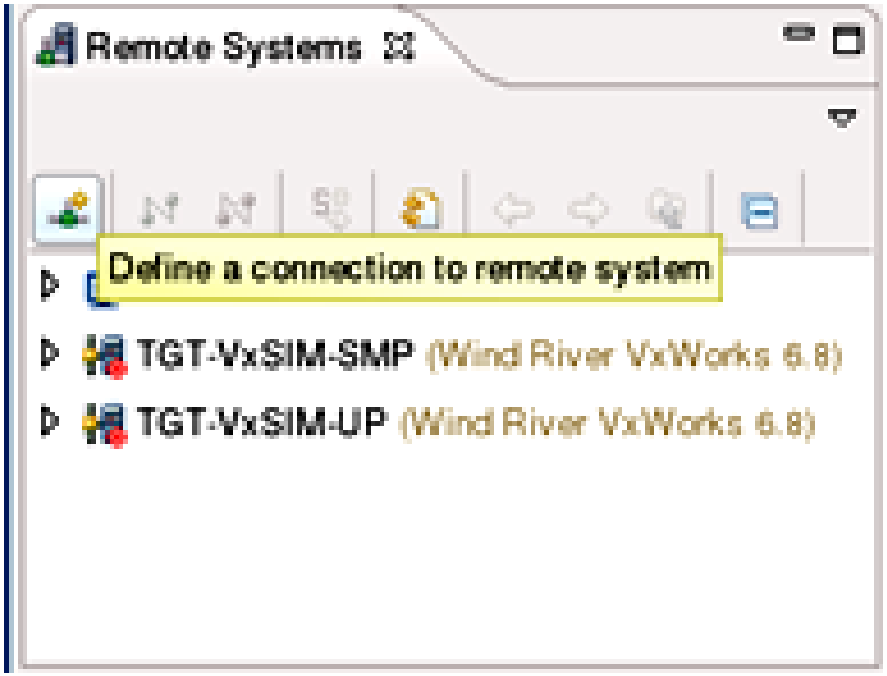


VxWORKS i VIŠEJEZGRENI PROCESORI

Korištenje VxWorks Simulatora

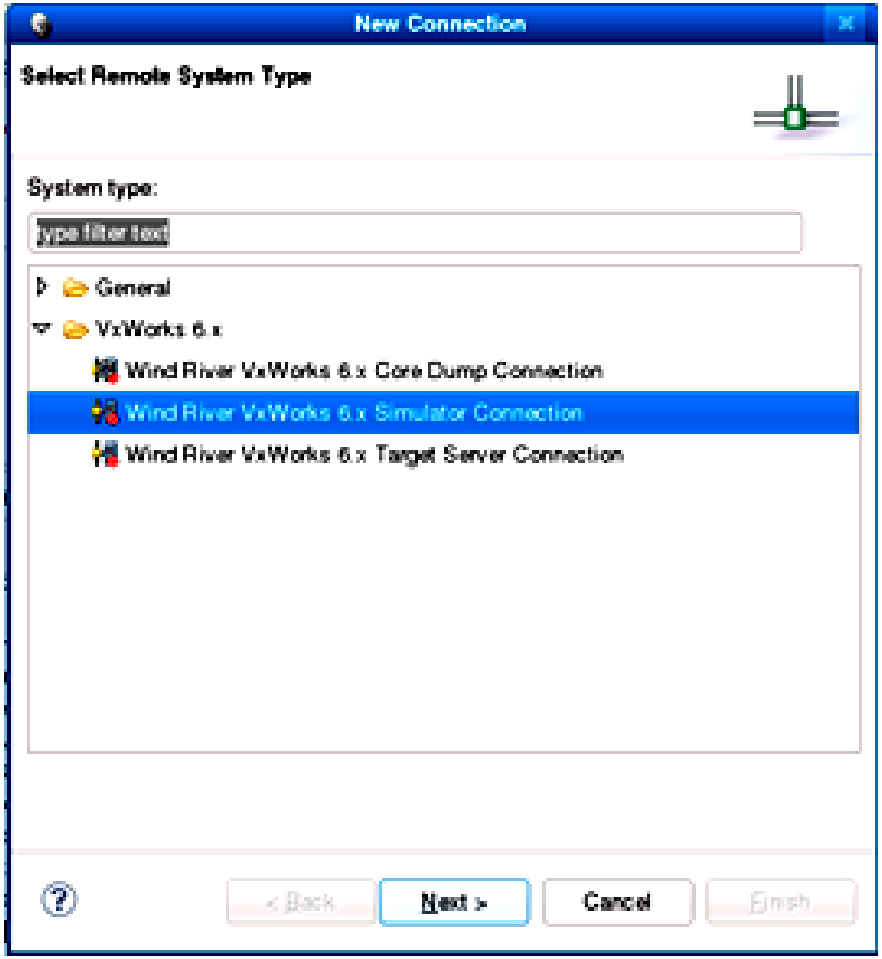
Najprije treba kreirati novi VxWorks simulator i vezu, a zatim povezati simulator sa vezom ka VxWorks image-u koji smo ranije kreirali.

1. U Remote System pogledu, odabrati **Define a connection to remote system**.



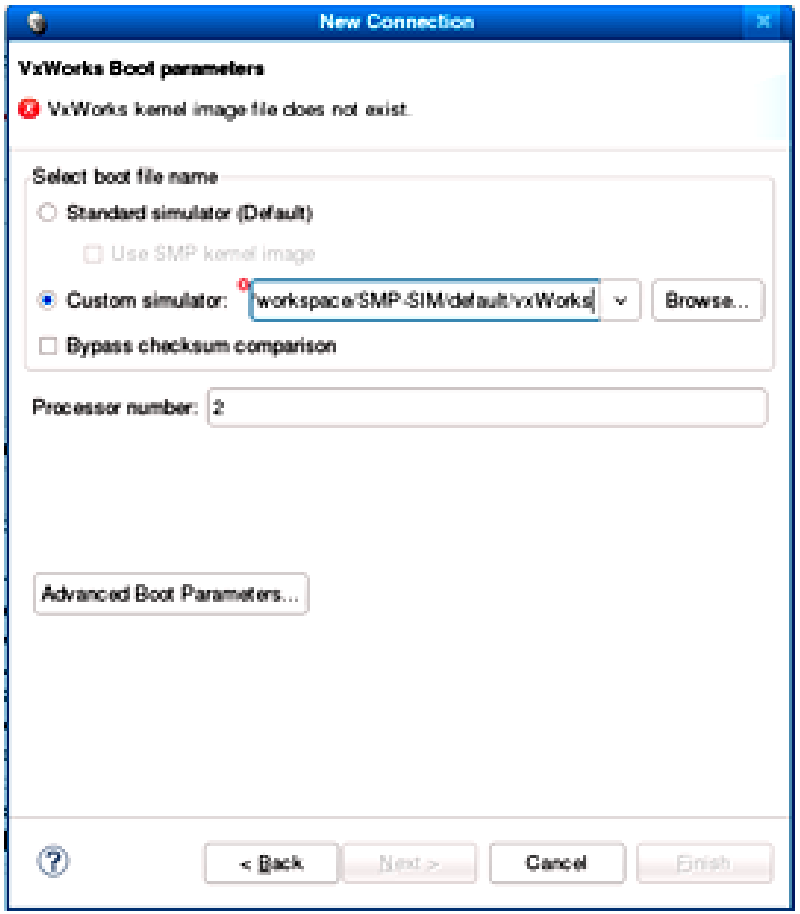
VxWORKS i VIŠEJEZGRENI PROCESORI

2. U New Connection dijalogu, pod VxWorks 6.x čvorom odabrati **Wind River VxWorks 6.x Simulator Connection**



VxWORKS i VIŠEJEZGRENI PROCESORI

3. U New Connection – VxWorks Boot Parameters dijalogu odabrati **Custom simulator**, a zatim odabrati slijedeći fajl: **/home/fedora/WindRiver/workspace/SMP-SIM/default /vxWorks**



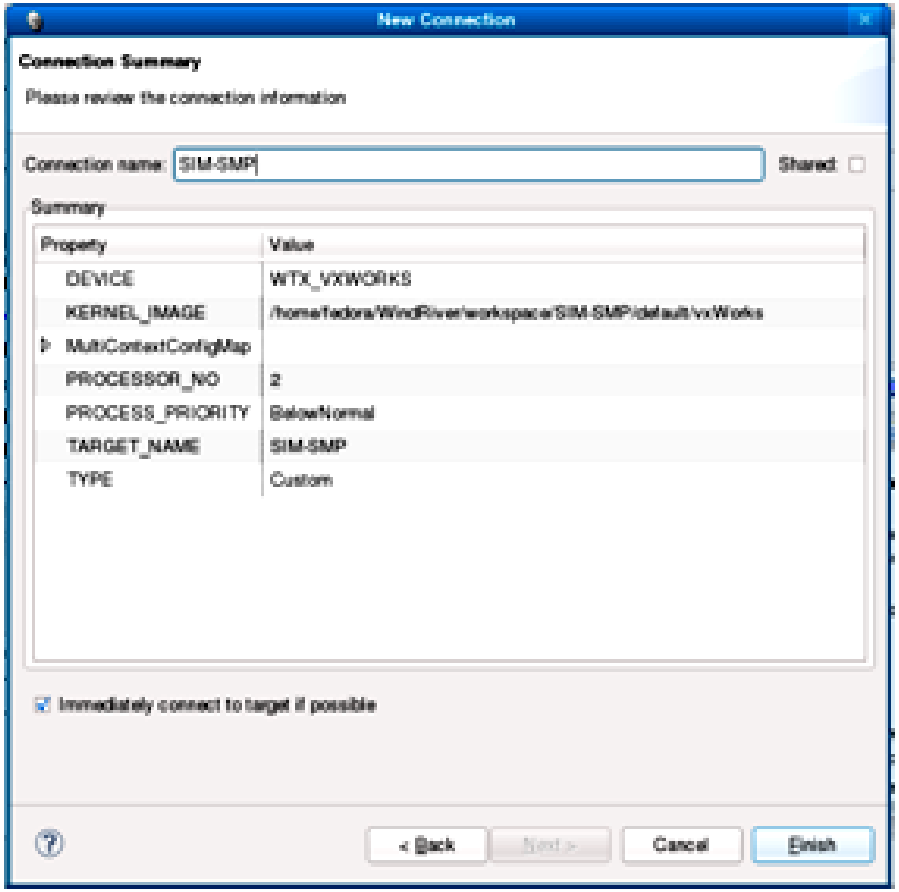
VxWORKS i VIŠEJEZGRENI PROCESORI

4. Nakon što je odabran VxWorks image fajl treba odabrati **Advanced Boot Parameters** kako bi se odabrao Boot device (promijeniti vrijednost sa **passDev** na **simnet**). Ovo predstavlja metod koji Workbench koristi kako bi povezao target ploču – simulator.

5. Unjeti **SMP-SIM** u Target Name polje

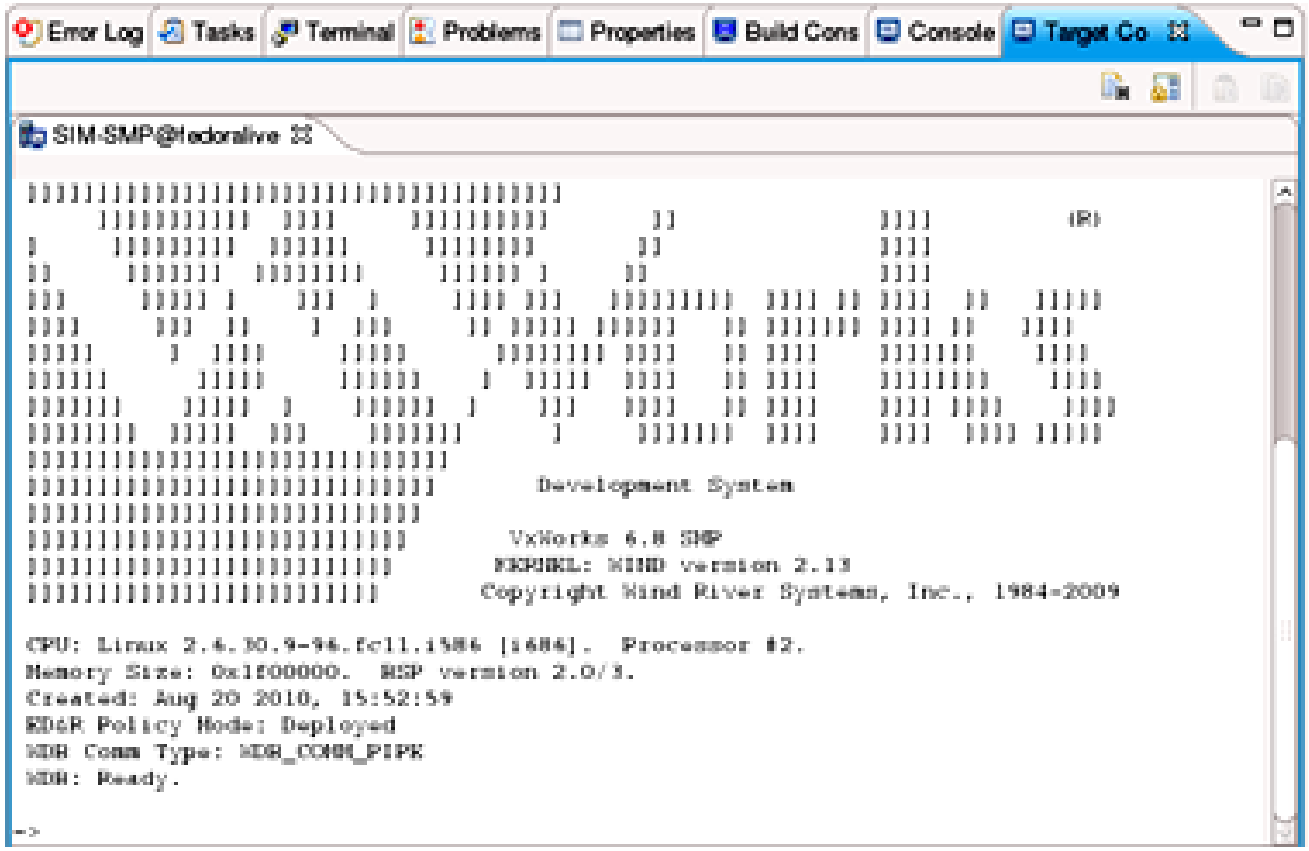
VxWORKS i VIŠEJEZGRENI PROCESORI

6. U New Connection – Connection Summary dijalogu unijeti SMP-SIM za **Connection name**. Na ovaj način možemo da identifikujemo sesiju u slučaju većeg broja sesija.



VxWORKS i VIŠEJEZGRENI PROCESORI

7. Nakon ovoga pojavljuje se nova veza u Remote Systems pogledu.
8. Pojavljuje se novi prozor sa VxWorks pozdravnom porukom. To predstavlja našu simulacijsku sesiju koja pokreće VxWorks SMP Image. Možemo preuzeti krenel module (DKM) i Real-Time Aplikacije (RTP) u slučaju da želimo da radimo sa target pločom.



VxWORKS i VIŠEJEZGRENI PROCESORI

Verificiranje simuliranog koda

Simulator je, standardno, konfigurisan da simulira dvije jezgre. Unošenjem komande **vxCpuConfiguredGet** možemo vidjeti koliko jezgri je podešeno.

```
WDB: Ready.

-> vxCpuConfiguredGet
value = 2 = 0x2
-> 
```

Sa komandom **i** izlistavaju se svi taskovi u kernelu. Unošenjem i komande možemo vidjeti da postoje dva idle taska (tidleTask0 i tIdleTask1) jer svako jezgro u SMP sistemu ima svoj vlastiti idle task.

```
  NAME         ENTRY       TID     PRI   STATUS      PC       SP     ERRNO  CPU #
----------  ------------ -------- --- ---------- -------- -------- ------- -----
tJobTask    6007bf30     603de010   0 PEND       60120185 6044df38       0     -
tExcTask    6007b220     601beec0   0 PEND       60120185 601bedc8       0     -
tLogTask    logTask      603e7c38   0 PEND       6011e615 6045bed8       0     -
tNbioLog    6007cdd0     603e2010   0 PEND       60120185 60469ef8       0     -
tShell0     shellTask    60517ab0   1 READY      60129d80 6052c058       0     0
tWdbTask    wdbTask      603d3840   3 PEND       60120185 6050dee8       0     -
ipcom_tick> 6013b550     604956e8  20 DELAY      60127a3b 604bff88       0     -
tAioIoTask> aioIoTask    60471830  50 PEND       60120c9c 6048fd28       0     -
tAioIoTask> aioIoTask    60471bc8  50 PEND       60120c9c 604a1d28       0     -
tNet0       ipcomNetTask 60483af0  50 PEND       6011ff5a 604afef8       0     -
ipcom_sysl> 600462f0     60453948  50 PEND       60120c9c 604dde58       0     -
tAioWait    aioWaitTask  60441c00  51 PEND       60120185 6047deb8       0     -
tIdleTask0  idleTaskEntr 60399f40 287 READY      60023536 60399da8       0     -
tIdleTask1  idleTaskEntr 603b2000 287 READY      60023536 603b1e68       0     1
value = 0 = 0x0
-> 
```

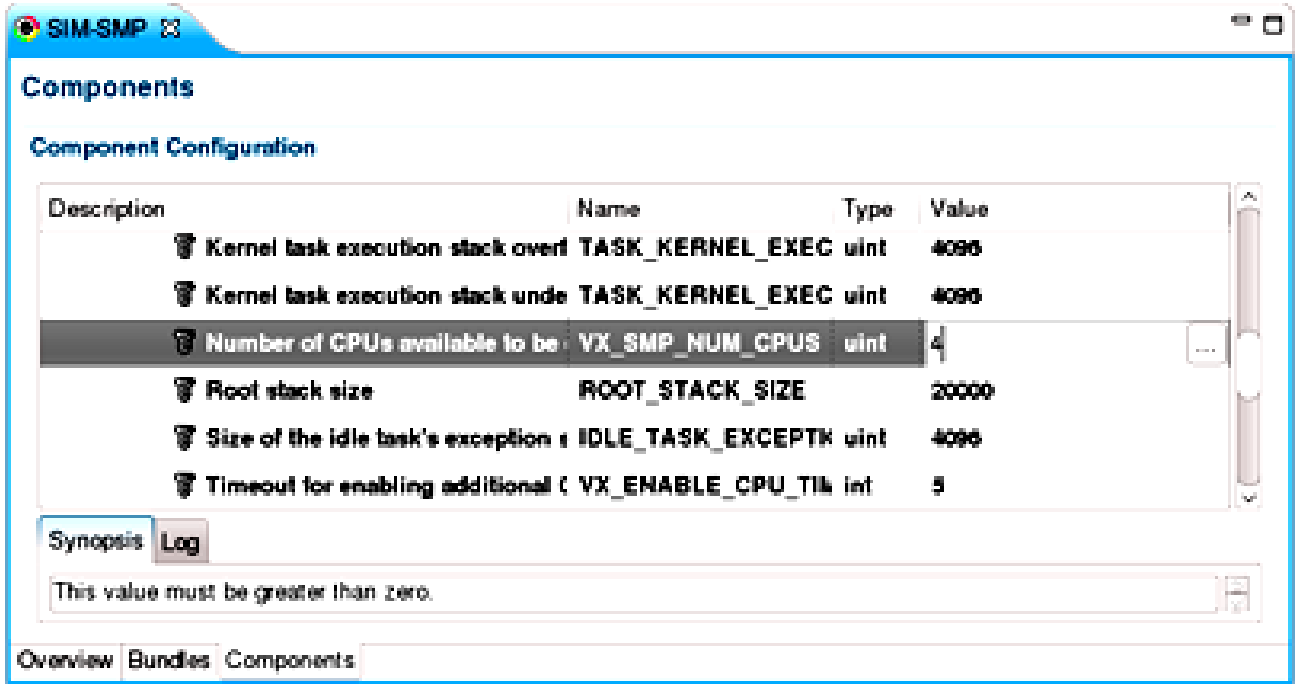
VxWORKS i VIŠEJEZGRENI PROCESORI

Dodavanje više jezgara simulatoru

VxWorks simulator ima mogućnost dodavanja do 32 jezgre. Dodatne jezgre možemo dodati na slijedeći način:

1. U Project Explorer pogledu duplim klikom odabrati **Kernel Configuration**.
2. U Components pogledu pronaći **operating system components -> kernel components -> kernel**.
3. Promijeniti vrijednost **Number of CPUs to be enabled for VxWorks SMP** sa 2 na 4.

VxWORKS i VIŠEJEZGRENI PROCESORI



4. Da bi se sačuvale promijene VxWorks Image mora biti ponovo izgrađen – rebuild.
5. Potrebno se otkopčati sa simulatora, ako je isti pokrenut, a zatim se ponovo prikopčati sa nanovo ažuriranom konfiguracijom.

VxWORKS i VIŠEJEZGRENI PROCESORI

6. Da bi verifikovali napravljene izmjene, ponovo unesemo **vxCpuConfigurationGet** komandu u simulacijsku sesiju.

```
-> vxCpuConfiguredGet
value = 4 = 0x4
->
```

7. Također, ponovo unesemo komandu **i** kako bi se uvjerili da imamo četiri idle taska, svaki za po jedno jezgro.

VxWORKS i VIŠEJEZGRENI PROCESORI

NAME	ENTRY	TID	PRI	STATUS	PC	SP	ERRNO	CPU #
tJobTask	6007bf30	6040e010	0	PEND	60120185	6047df38	0	-
tExcTask	6007b220	601c0120	0	PEND	60120185	601c0028	0	-
tLogTask	logTask	60417c38	0	PEND	6011e615	6048bed8	0	-
tNbioLog	6007cdd0	60412010	0	PEND	60120185	60499ef8	0	-
tShell0	shellTask	60547ae8	1	READY	60129d80	6055c058	0	0
tWdbTask	wdbTask	60403840	3	PEND	60120185	6053dee8	0	-
ipcom_tick>	6013b550	604c56e8	20	DELAY	60127a3b	604eff88	0	-
tAioIoTask>	aioIoTask	604a1830	50	PEND	60120c9c	604bfd28	0	-
tAioIoTask>	aioIoTask	604a1bc8	50	PEND	60120c9c	604d1d28	0	-
tNet0	ipcomNetTask	604b3af0	50	PEND	6011ff5a	604dfef8	0	-
ipcom_sysl>	600462f0	60483948	50	PEND	60120c9c	6050de58	0	-
tAioWait	aioWaitTask	60471c00	51	PEND	60120185	604adeb8	0	-
tIdleTask0	idleTaskEntr	6039b098	287	READY	60023536	6039af00	0	-
tIdleTask1	idleTaskEntr	603b2000	287	READY	60023536	603b1e68	0	1
tIdleTask2	idleTaskEntr	603ca000	287	READY	60023536	603c9e68	0	2
tIdleTask3	idleTaskEntr	603e2000	287	READY	60025900	603e1fd8	0	3

value = 0 = 0x0

Do sada smo izgradili VxWorks SMP image za simulator. Početni broj jezgara je dva, ali se može lahko promijeniti korištenjem kernel configuration alata. VxWorks simulira zadati broj jezgara i kao takav može se koristiti kao izolirano okruženje za razvoj i testiranje projekta. Wind River alati kao što je System Viewer (nadgledanje događaja i taskova), Performance Profiler (nadgledanje performansi) i SMP debugger omogućavaju obavljanje analize aplikacije i na

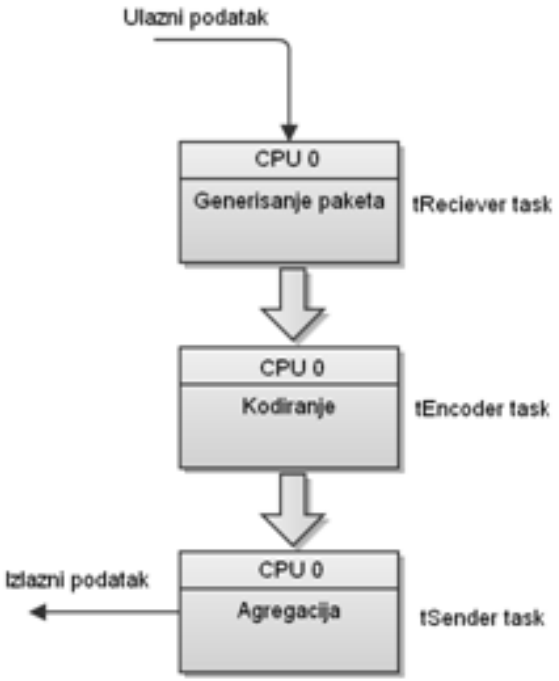
simulatoru. Ovdje treba uzeti u obzir da se simulator ne može iskoristiti za mjerenje performansi iz razloga što su performanse simulatora zavisne od hardvera na kojem je simulator pokrenut (vrsta i brzina procesora, dostupna količina RAM-a itd.). Simulator je dobar alat za planiranje i dizajn faze migracije sa UP na SMP s tim da ne treba obračati previše pažnje na performanse jer se radi o simuliranom okruženju.

Migracija aplikacije sa jednoprocesorskog (UP) na višeprocesorsko (SMP) okruženje

Razmotrimo proces migracije aplikacije napisane za jednoprocesorske sisteme (UP) u aplikaciju koja može da iskoristi prednosti višeprocesorskog sistema (SMP). Migracija će biti objašnjena na aplikaciji za mrežni koder. Aplikacija prima i baferuje dolazne podatke, kodira podatke i konačno šalje kodirane podatke. Naredna slika prikazuje gradivne

VxWORKS i VIŠEJEZGRENI PROCESORI

blokove aplikacije.



Task koji prima podatke, baferuje ih, dijeli ih na pakete jednake veličine, označava svaki paket i na kraju gradi stablo iz strukture podataka. Kodirajuči task prima pakete iz drveta i kodira podatke. Na kraju task šalje podatke i uklanja sve interne oznake.

VxWORKS i VIŠEJEZGRENI PROCESORI

Migracija sa UP na SMP

Naredni koraci se preporučuju kada migriramo aplikaciju sa UP sistema na SMP sistem. Prva tri migracijska koraka će biti obavljena dok koristimo samo jedno jezgro (jednojezgreni image). Ovo će osigurati da SMP dizajn radi na UP okruženju.

Korak 1: Analiza UP aplikacije

U ovom koraku treba analizirati i identifikovati područje koje će donjeti korist iz SMP okruženja. Ovo znači da treba identifikovati područja u aplikaciji koje bi se trebala izvršavati paralelno kada se ista izvršava na SMP sistemu.

Korak 2: Redizajniranje aplikacije za SMP

U ovom koraku potrebno je redizajnirati aplikaciju kako bi se izvukla korist iz SMP okruženja tako što će se kôd koji se treba izvršavati paralelno izvršavati od strane više niti u paraleli na SMP sistemu.

VxWORKS i VIŠEJEZGRENI PROCESORI

Korak 3: Testiranje redizajnirane aplikacije (višenitne) u UP modu

Ovdje je potrebno osigurati da dio aplikacije koji je redizajniran da se izvršava paralelno se pravilno izvršava u višenitnom okruženju.

Korak 4: Testiranje redizajnirane aplikacije u SMP modu

Potvrđivanje da aplikacija radi u SMP sistemu.

Prva tri migracijska koraka će biti urađena dok koristimo samo jedno jezgro (single-core image). Ovo će osigurati da SMP dizajn radi na UP okruženju.

Analiza UP aplikacije

Prvi korak kod redizajniranja UP aplikacije za SMP sistem je utvrđivanje kako SMP može unaprijediti aplikaciju. Kako bi se ovo uradilo, potrebno je identifikovati koji taskovi konzumiraju najviše CPU ciklusa, i kako se ti taskovi mogu razbiti na male

dijelove koji bi se mogli izvršavati simultano. Kako bi obavili prvi korak potrebno je koristiti Wind River Workbench Performance Profiler.

Učitavanje i pokretanje aplikacije

Prije nego što krenemo sa korištenjem Performance Profilera najprije moramo učitati aplikaciju na ciljnu mašinu (u našem slučaju simulator) i pokrenuti je. Naredni koraci opisuju kako da se obavi ovaj zadatak.

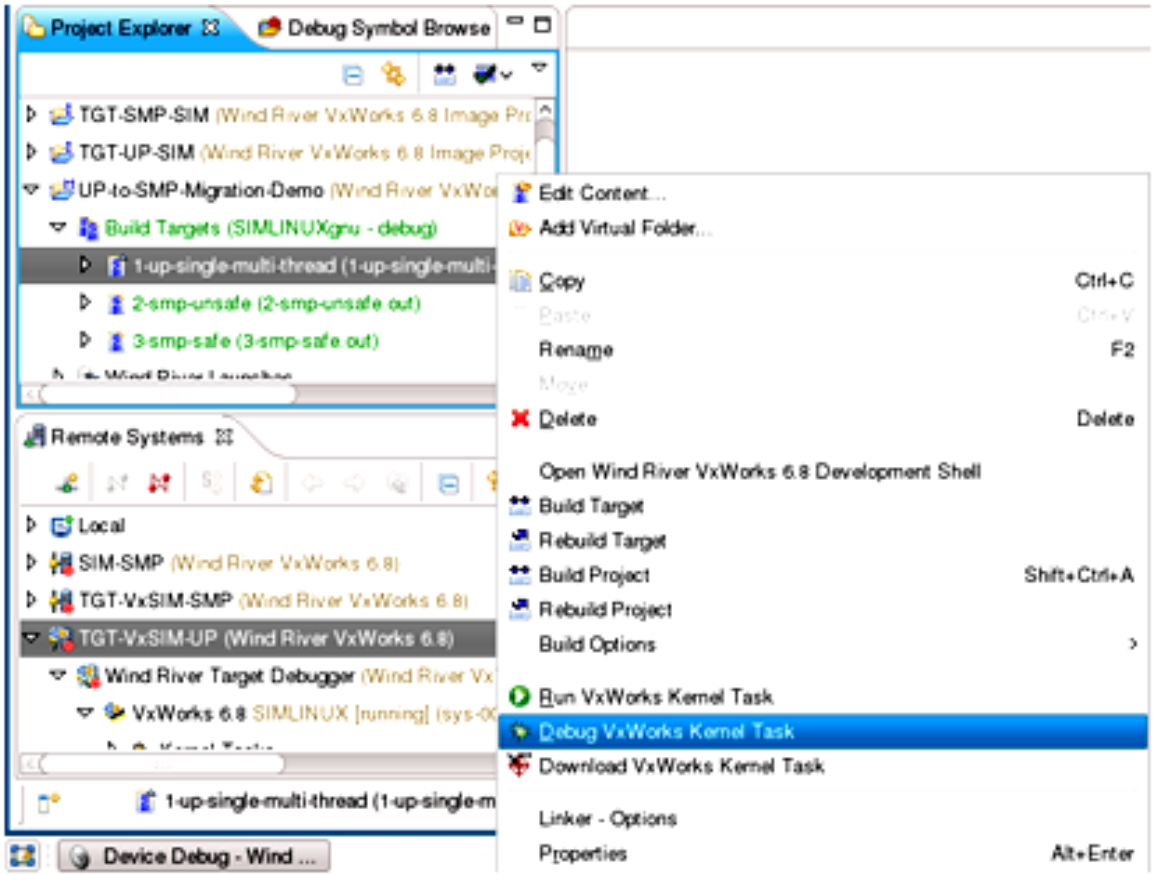
UP aplikacija koju treba analizirati je uključena kao *downloadable kernel module* (DKM) pod nazivom **UP-To-SMP-Migration**.

1. Koristiti UP simulator, **TGT-VxSIM-UP**, u Remote System pogledu.
2. Klikom na **Connect** pokrenuti simulator.

VxWORKS i VIŠEJEZGRENI PROCESORI

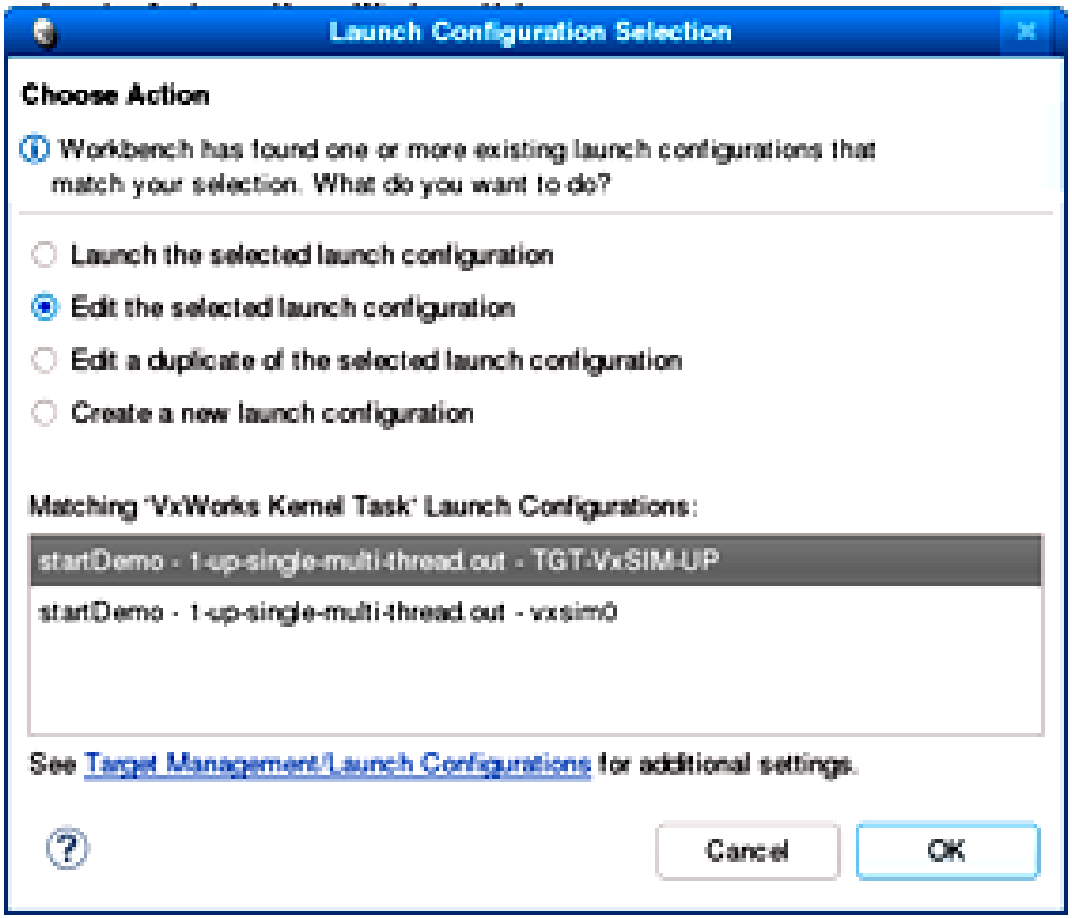
3. U Project Explorer pogledu, raširiti **UP-To-SMP-Migration-Demo** projekt i **Built Target** stavku u sklopu nje

4. Odabrati **1-up-single-multi-thread** a zatim iz kontekstnog menija odabrati **Debug VxWorks Kernel Task**.



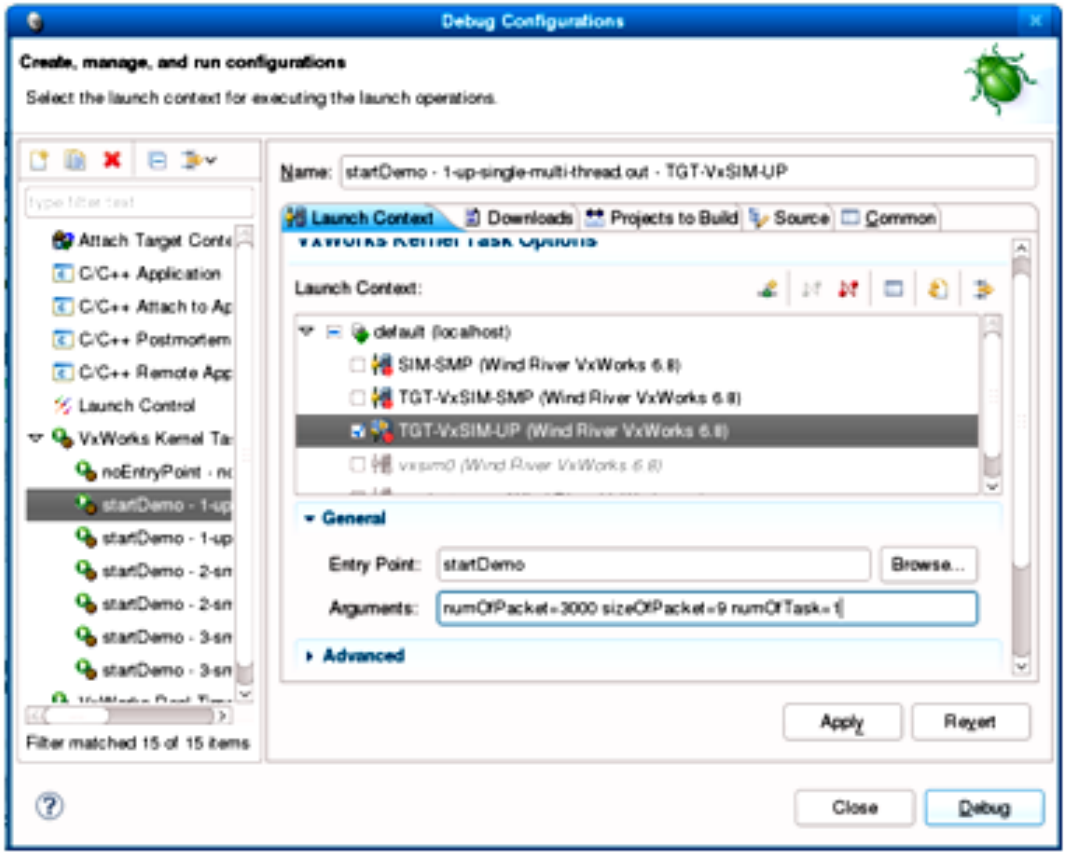
VxWORKS i VIŠEJEZGRENI PROCESORI

5. U dijalogu **Launch Configuration Selection** odabrati **Edit selected launch configuration** i kliknuti na OK.



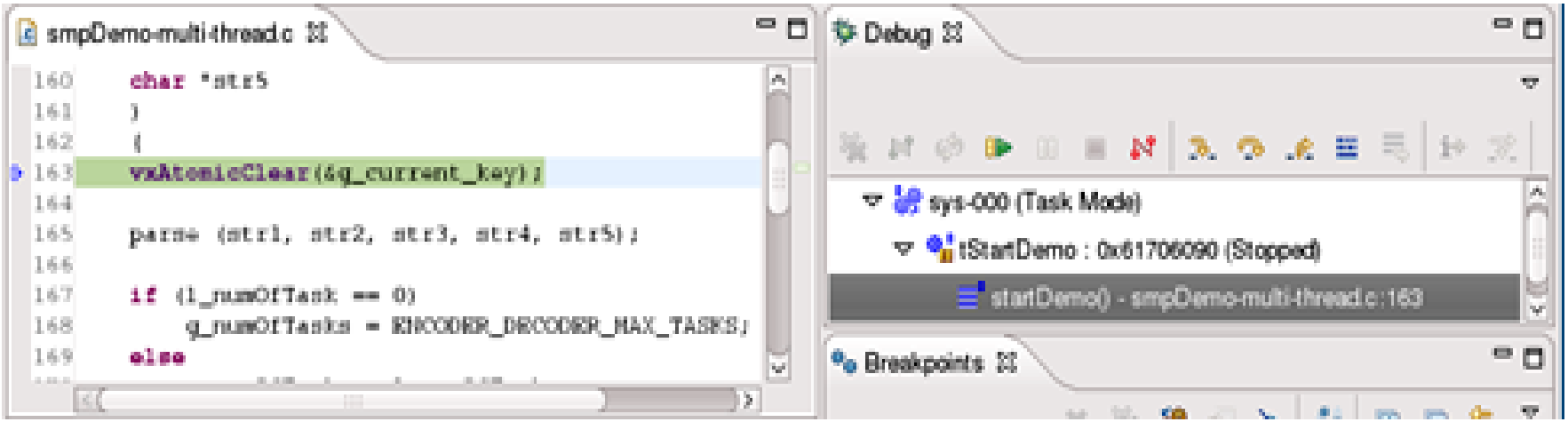
VxWORKS i VIŠEJEZGRENI PROCESORI

6. U Debug Configurations dijalogu verifikovati da je **Arguments** zadato sa **numOfPacket=30000 sizeOfPacket=9 numOfTask=1**.



VxWORKS i VIŠEJEZGRENI PROCESORI

7. Klikom na dugme Debug pokrećemo aplikaciju. U Workbench editoru bi trebali vidjeti fajl sa izvornim kodom s tim da je aplikacija stopirana u Debug pogledu.



.

VxWORKS i VIŠEJEZGRENI PROCESORI

Korištenje Wind River Workbench Performance Profiler alata za analizu aplikacije

Wind River Workbench Performance Profiler je program za dinamičku analizu koji pruža detaljnu *funkcija-po-funkcija* analizu performansi uz prikazivanje individualnih rutina u sklopu programa koje troše CPU cukluse. Performance Profiler kreira direktnu mapu onoga šta procesor radi, koje se rutine pozivaju i koje rutine su one pozvale. On ukazuje na neefikasnosti u kodu i to je informacija koju trebamo kako bi sistem podesili za optimalne performanse.

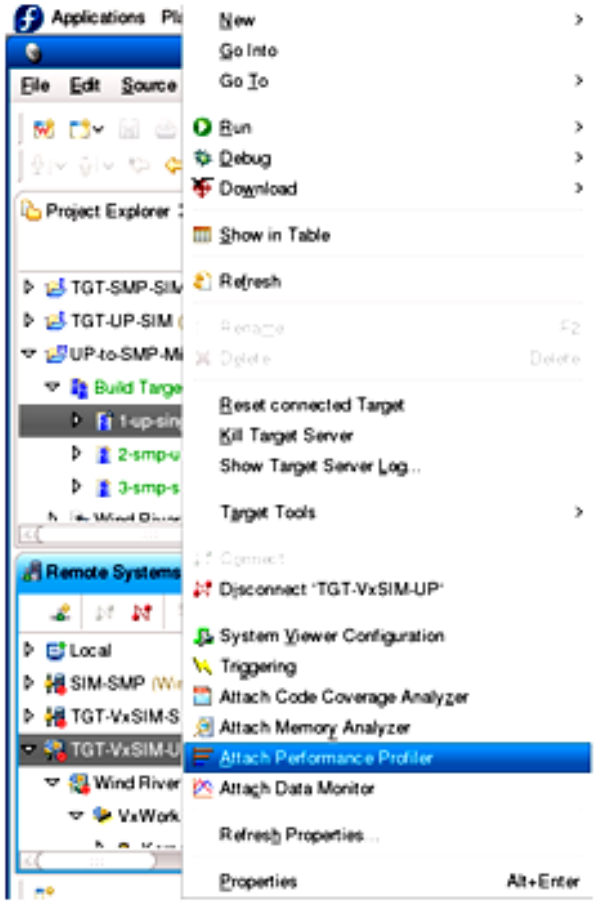
Pokretanje Performance Profilera

Da bi pokrenuli Performance Profiler treba da uradimo slijedeće:

1. Odabrati **TGT-VxSIM-UP** u Remote Systems pogledu

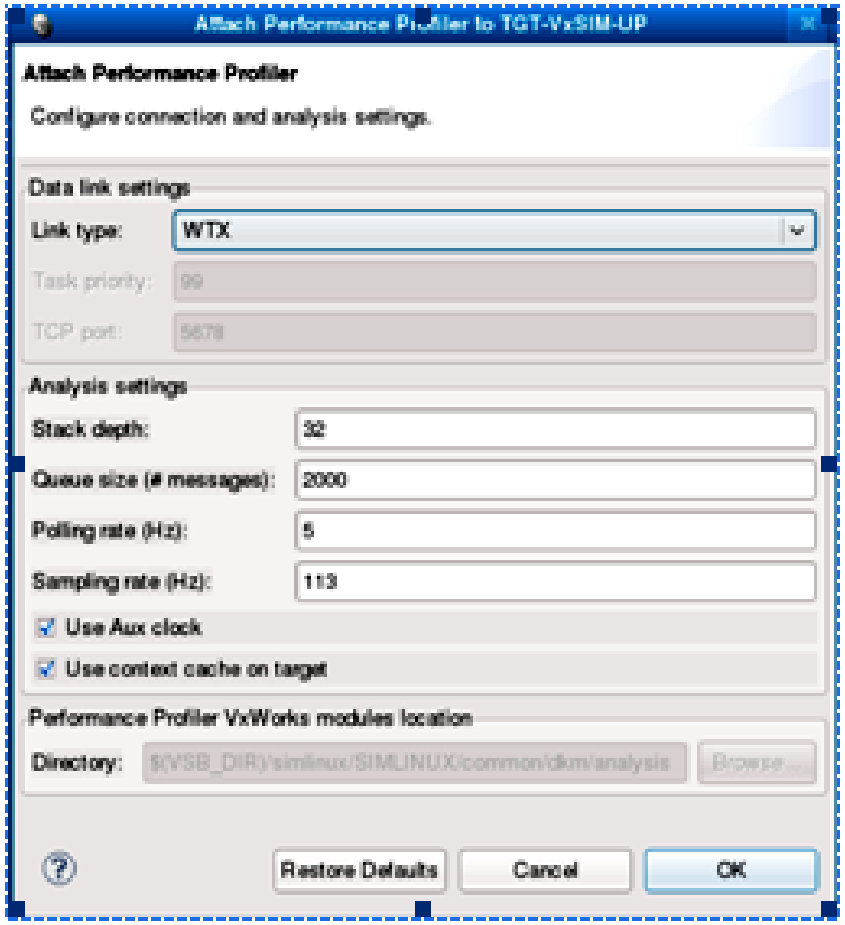
VxWORKS i VIŠEJEZGRENI PROCESORI

2. Iz kontekstnog menija odabrati **Attach Performance Profiler**



VxWORKS i VIŠEJEZGRENI PROCESORI

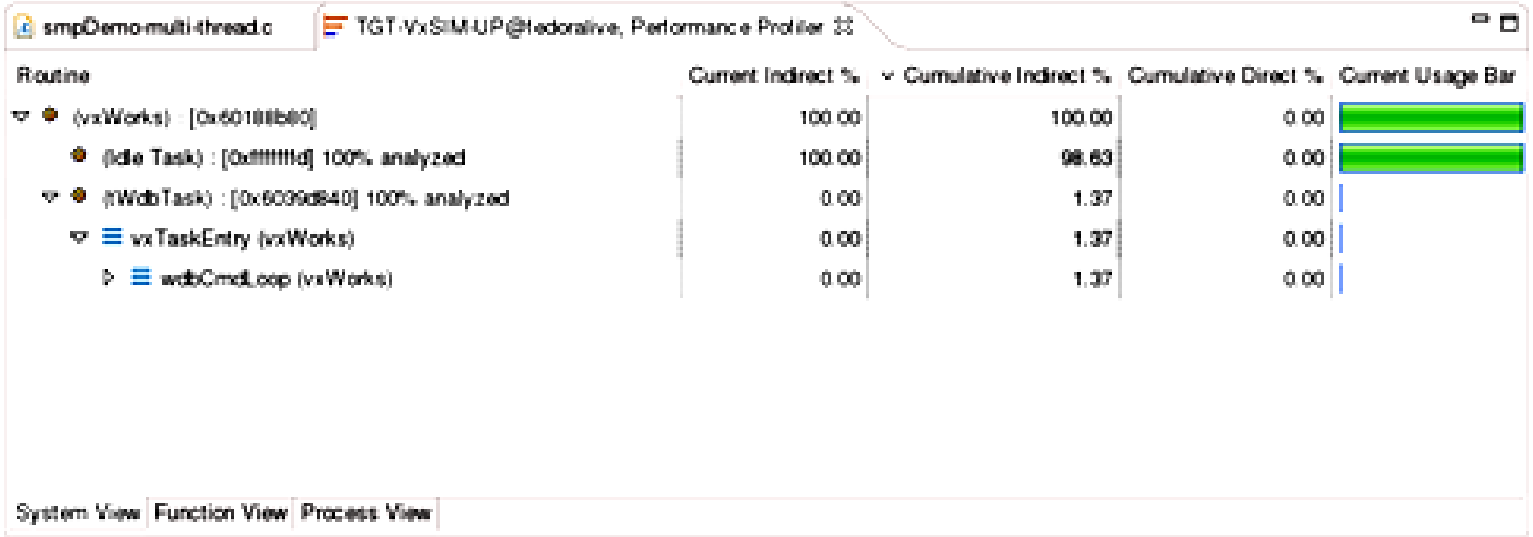
3. U Attach Performance Profiler dijalogu, promijeniti **Connection Type** sa TCP/IP na WTX



VxWORKS i VIŠEJEZGRENI PROCESORI

TCP/IP se koristi za spajanje Performance Profilera na target mašinu preko Etherneta kada se koristi prava target mašina, dok se WTX koristi kada radimo sa VxWorks simulatorom.

4. Performance Profiler pogled treba da se prikaže u Workbenchu.

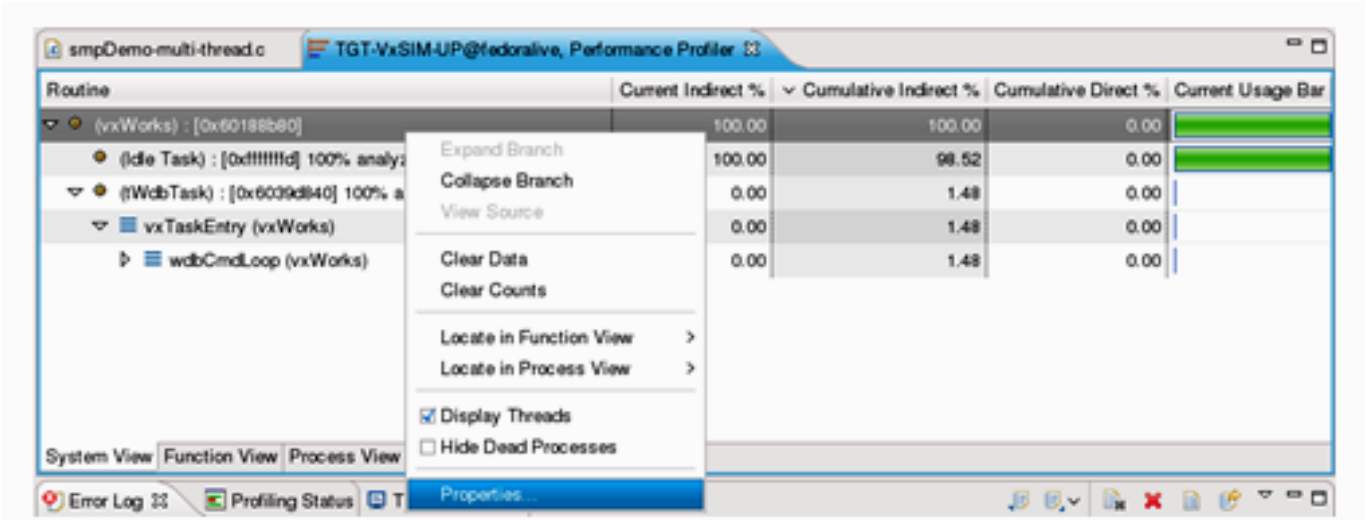


VxWORKS i VIŠEJEZGRENI PROCESORI

Podešavanje Performance Profiler pogleda

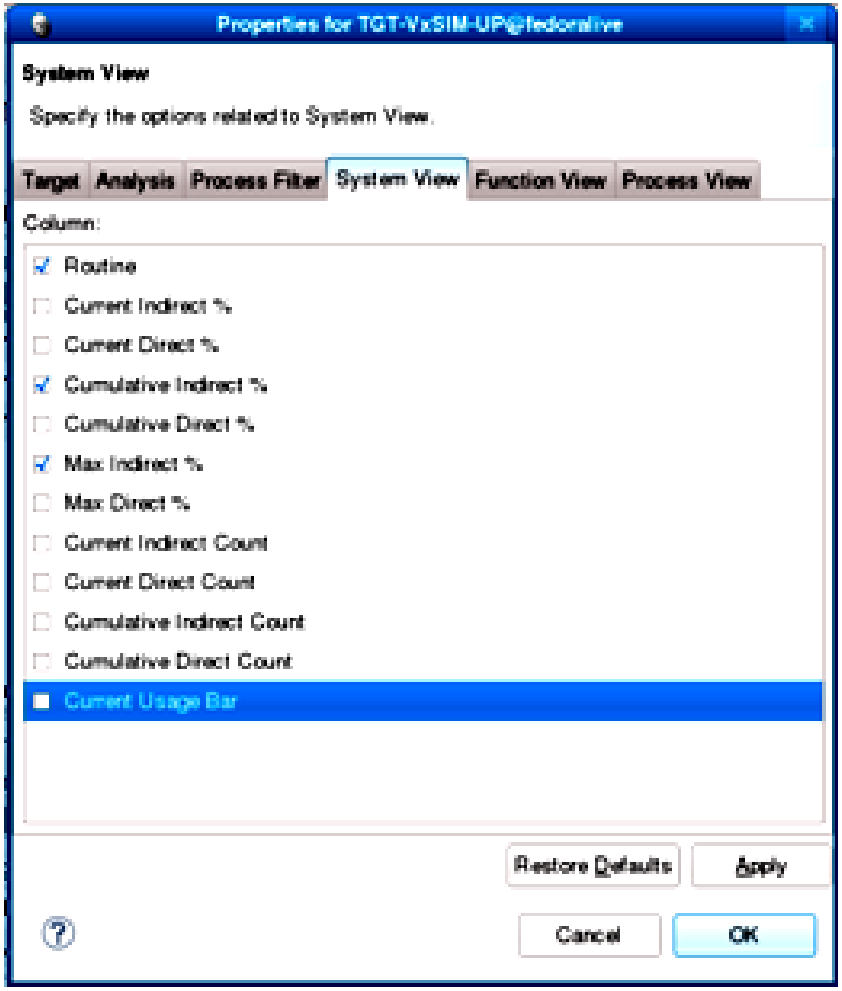
Kao što je ranije rečeno Performance Profiler može da prikaže dosta informacija vezanih za iskorištenje procesora. Kod aplikacija uglavnom nas najviše zanimaju taskovi i funkcije koje konzumiraju većinu procesorskog vremena. Kako bi podesili Performance Profiler da prikazuje te vrijednosti, potrebno je uraditi slijedeće:

1. Desni klik nad Performance Profiler pogledom.
2. Iz kontekstnog menija odabrati **Properties**.



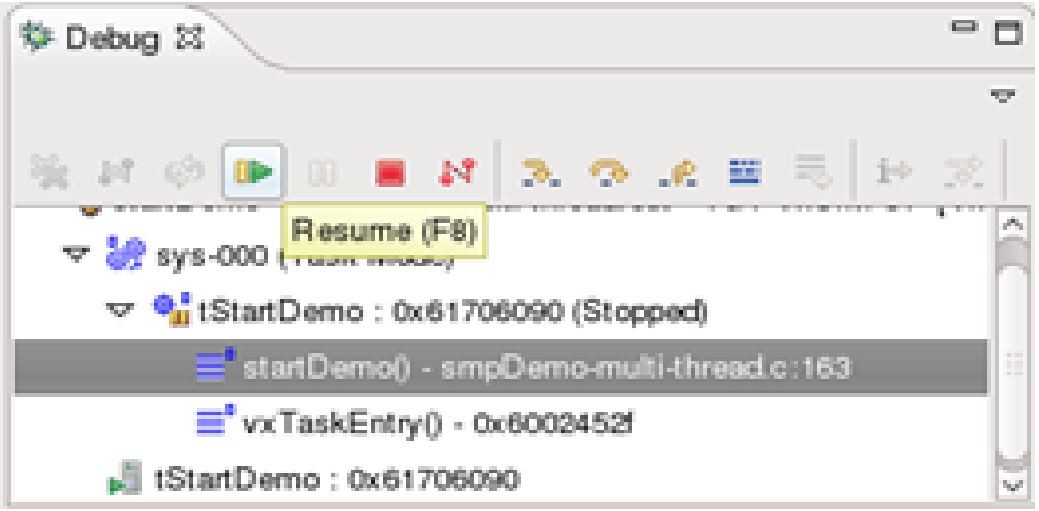
VxWORKS i VIŠEJEZGRENI PROCESORI

3. Na System View kartici odabrati **Routine**, **Cumulative Indirect** i **Max Indirect**, a zatim kliknuti na OK



VxWORKS i VIŠEJEZGRENI PROCESORI

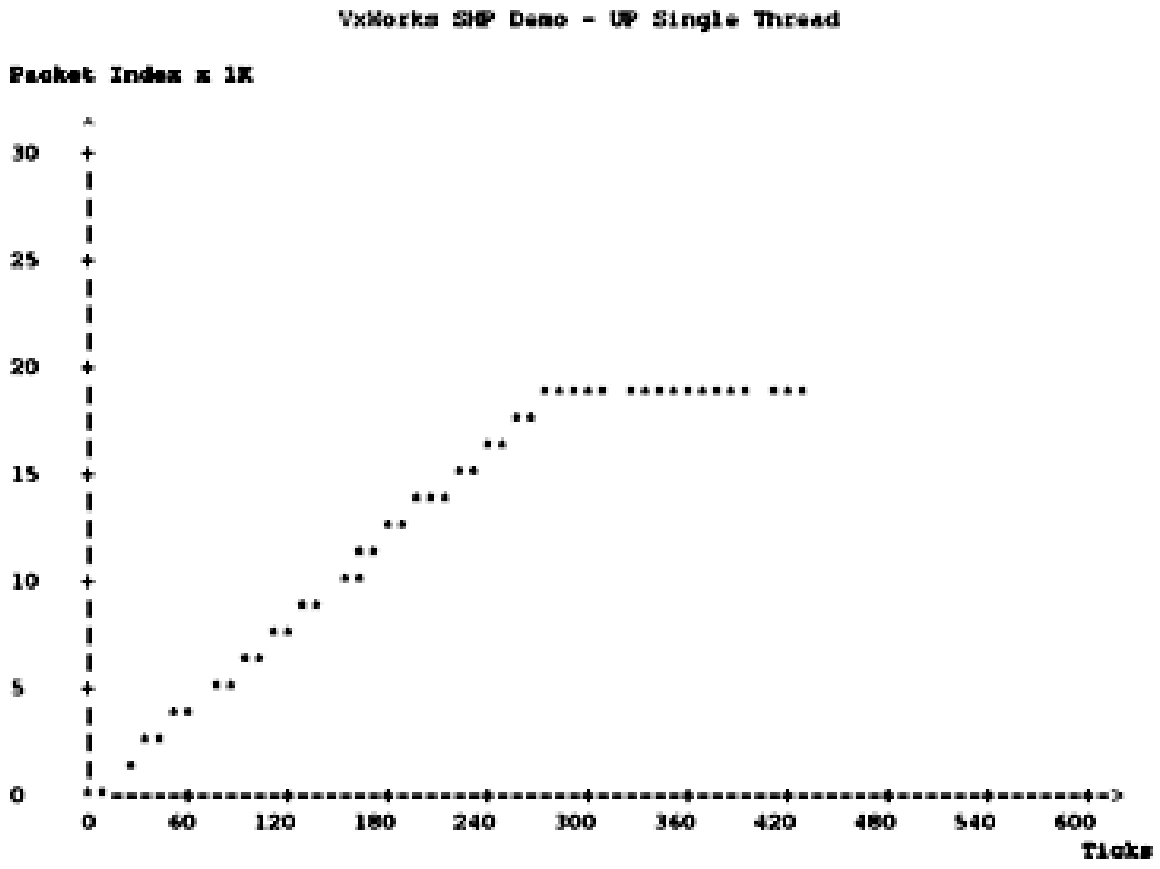
4. Aplikacija je sada spremna za izvršavanje. Aplikaciju pokrećemo klikom na Resume dugme u Debug pogledu.



5. Kada aplikacija završi sa izvršavanjem, trebali bi da vidimo slijedeći izlaz na simulatoru (vidjeti slijedeću sliku).

Rezultati mogu varirati u zavisnosti od vrste i brzine procesora, broja jezgara, memorije i opterećenja sistema na mašini na kojoj je pokrenut simulator.

VxWORKS i VIŠEJEZGRENI PROCESORI



TX buffer is same

Computation time = 441 ticks = 8.82 sec, 9 byte size, 30612 bytes/sec

VxWORKS i VIŠEJEZGRENI PROCESORI

6. Zaustaviti prikupljanje podataka odabirom Performance pogleda u editoru.

7. Iz glavnog menija odabrati **Performanse Profiler -> Pause**.

VxWORKS i VIŠEJEZGRENI PROCESORI

Analiza prikupljenih podataka

Nakon što raširimo vxWorks, Performance Profiler bi trebao da izgleda kao na narednoj slici.

smpDemo-smp-safe.c | smpDemo-smp-non-safe.c | smpDemo-multi-thread.c | TGT-VxSIM-UP@fedoralive, Performanc

Routine	Current Indirect %	Cumulative Indirect %	Max Indirect %
(vxWorks) : [0x60188b80]	100.00	100.00	100.00
(tWdbTask) : [0x6039d840] 100% analyzed	0.00	1.81	5.65
vxTaskEntry (vxWorks)	0.00	1.81	5.65
wdbCmdLoop (vxWorks)	0.00	1.81	5.65
(tStartDemo) : [0x61706090] 100% analyzed	0.00	0.04	0.40
(tReceiver) : [0x60739020] 100% analyzed	0.00	0.41	4.03
(tEncoder) : [0x6071d9a8] 100% analyzed	0.00	15.66	97.10
(tIdle Task) : [0xffffffd] 100% analyzed	100.00	82.08	100.00

System View | Function View | Process View

VxWORKS i VIŠEJEZGRENI PROCESORI

Sa slike možemo vidjeti koliko se procesorskog vremena (u procentima) troši na svaki task u sistemu. Praktično, **Max Indirect %** kolona nam prikazuje **maksimalnu vrijednost korištenja procesora** za vrijeme uzorkovanja, tako da se ne treba iznenaditi ako suma ukupnih vrijednosti u ovoj koloni premaši 100%. Ovaj pogled koristimo pri istraživanju jer nam daje jasan pogled gdje se alociraju procesorski resursi za vrijeme trajanja uzorkovanja. Sa slike možemo zaključiti da je task koji najviše koristi procesor **tEncoder**.

Također, moguće je raširiti svaki task klikom na trougao lijevo od taska kako bi vidjeli koje funkcije u tasku obavljaju najviše posla.

VxWORKS i VIŠEJEZGRENI PROCESORI

smpDemo-smp-safe.c | smpDemo-smp-non-safe.c | smpDemo-multi-thread.c | TGT-VxSIM-UP@kodoralive, Performanc

Routine	Current Indirect %	Cumulative Indirect %	Max Indirect %
(vxWorks) : [0x60188b80]	100.00	100.00	100.00
(tWdbTask) : [0x6039d840] 100% analyzed	0.00	1.81	5.65
vxTaskEntry (vxWorks)	0.00	1.81	5.65
wdbCmdLoop (vxWorks)	0.00	1.81	5.65
(tStartDemo) : [0x61706090] 100% analyzed	0.00	0.04	0.40
(tReceiver) : [0x60739020] 100% analyzed	0.00	0.41	4.03
(tEncoder) : [0x6071d9a8] 100% analyzed	0.00	15.66	97.10
vxTaskEntry (vxWorks)	0.00	15.66	97.10
encoderTask (1-up-single-multi-thread.out)	0.00	15.66	97.10
getReceiverData (1-up-single-multi-thread.out)	0.00	0.07	0.41
doEncoding (1-up-single-multi-thread.out)	0.00	15.59	97.10
(Idle Task) : [0xfffffffd] 100% analyzed	100.00	82.08	100.00

System View | Function View | Process View

.

Treba primjetiti da je **doEncoding()** rutina najskuplja u **tEncoder** tasku. Ovo ima smisla zbog toga što procesor treba da pokrene algoritam za kodiranje, dok taskovi koji primaju i šalju podatke samo rade sa pokazivačima.

VxWORKS i VIŠEJEZGRENI PROCESORI

```
/*******************************************************************************
* encoderTask - polazna tačka taska za kodiranje
*
*Ovaj task dobija pakete pripremljene od strane Reciever taska, pokreće odredjeni algoritam
*koji po slucajnom izboru trosi vrijeme nad primljenim podacima (kao sto je kodiranje-
*dekodiranje, racunske operacije, itd.) i kada su svi pakati spremni, salje ih taske
*posiljaocu.
* RETURNS: N/A
*/
void encoderTask (int arg)
        {
        /* task threadNum - prvi encoder task je TRUE, a drugi je FALSE*/
        int threadNum = (int)arg;
        atomic_t        selfCount;          /* task specific atomic counter */

        /* jedan komad dolaznog bafera iz prijemnika */
        char indexed_data_element[MAX_RX_TX_BUFFER_LENGTH];
        /* jedan komad odlaznog bafera ka tasku posiljaocu */
        char encoded_data_element[MAX_RX_TX_BUFFER_LENGTH];

        /* struktura za prikupljanje podataka za kasniju analizu*/
        encoder_statistics_t encoder_statistics;

        /* inicijalizacija taska */
        encoderInit (&selfCount, &encoder_statistics, indexed_data_element);

        /* cekaj zauvijek sve dok signal iz primaoca ne naznaci da je
         * podatak u AVL drvetu spreman*/
        eventReceive (ENCODER_DATA_READY, EVENTS_WAIT_ALL, WAIT_FOREVER, NULL);
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
#if DEBUG
                                testEncoderInputAvlTree();
#endif
        while(1)
                {
                /* kada ovaj task završi sa procesiranjem zadnjeg paketa, samo zaustavi task*/
                if (g_current_key == g_numOfPackets)
                        {
                        /*obavijesti task pošiljaoca da su svi čvorovi u  AVL drvetu*/
                        eventSend (g_SenderTaskId, SENDER_ACK);
#if DEBUG
                        testEncoderOutputMsgQueue (encoded_data_element);
#endif
                        break;
                        }

                        /* uzmi paket pripremljen u tasku primaocu*/
                        getReceiverData (selfCount, threadNum, sencoder_statistics,
indexed_data_element);

                        /* Kodiraj podatke u indexed_data_element i proizvedi
indexed_encoded_data
                         * a zatim ih pošalji taksu pošiljaocu*/
                doEncoding(threadNum, sencoder_statistics, indexed_data_element,
encoded_data_element);
                /* sakupi statistiku */
                incEncoderOutputBufferCounter(sencoder_statistics);
                }

        }

/*****************************************************************************


/*******************************************************************************
* doEncoding - kodiranje podataka
*
* Ova rutina kodira podatke iz indexed_data_element i proizvodi
* indexed_encoded_data a zatim ih šalje tasku pošiljaocu
*
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
* RETURNS: N/A
*/
void doEncoding
        (
        int threadNum,
        encoder_statistics_t *encoder_statistics,
        char *indexed_data_element,
        char *encoded_data_element
        )
        {
        int size, index, i ;
        /* dohvati indeks paketa iz bafera primaoca */
        index = getIndex(indexed_data_element);
        /* dohvati veličinu paketa iz bafera primaoca */
        size = getSize(indexed_data_element);
        /* ukloni indeks i informaciju o veličini paketa iz bafera primaoca */
        indexed_data_element = removeHeaderParams(indexed_data_element);

        /* ako je g_primeWork uključen, učitaj svaki task različito u odnosu na'threadNum' i
g_primeWork varijable */
        if (g_primeWork)
                {
                if (threadNum)
                        for (i = 0; i < size; i++)
                        {
                                prime(indexed_data_element[i] * g_primeWork);
                        }
                else
                        for (i = 0; i < size; i++)
                        {
                                /* izračunaj sve prosta brojeve nad svakim bajtom primljenog
                                 * paketa kako bi se povećala opterećenost
                                 * */
                                /* for (j = 0; j < 5; j++) */
                                  prime(indexed_data_element[i] * g_primeWork);
                        }
                }
        /* dodaj indeks i veličinu u bafer*/
        insertHeaderParams(encoded_data_element, indexed_data_element, index, size);
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
#if DEBUG
        for (i=0;i<MAX_RX_TX_BUFFER_LENGTH;i++) printf("%c", encoded_data_element[i]);
#endif
        /* posalji bafer u red posiljaoca koristenjem standardnih redova poruka */
        msgQSend(g_decoder2sender_Queue, (char *)(encoded_data_element),
MAX_RX_TX_BUFFER_LENGTH, WAIT_FOREVER, MSG_PRI_NORMAL);

        /* za statistiku */
        setEncoderOutputBufferSize(encoder_statistics, MAX_RX_TX_BUFFER_LENGTH);

#if DEBUG
    printf("\ntaskId - %#x-%s task: output chunk size to the decoder task %#x is %u, sent
buffer index %d\n", taskIdSelf(), getEncoderTaskName(encoder_statistics),
getPeerTaskId(taskIdSelf()), getEncoderOutputBufferSize(encoder_statistics),
getIndex(encoded_data_element));
#endif
        }

/*****************************************************************************
```

Listing 1. doEncoding rutina koja najviše iskorištava procesor.

VxWORKS i VIŠEJEZGRENI PROCESORI

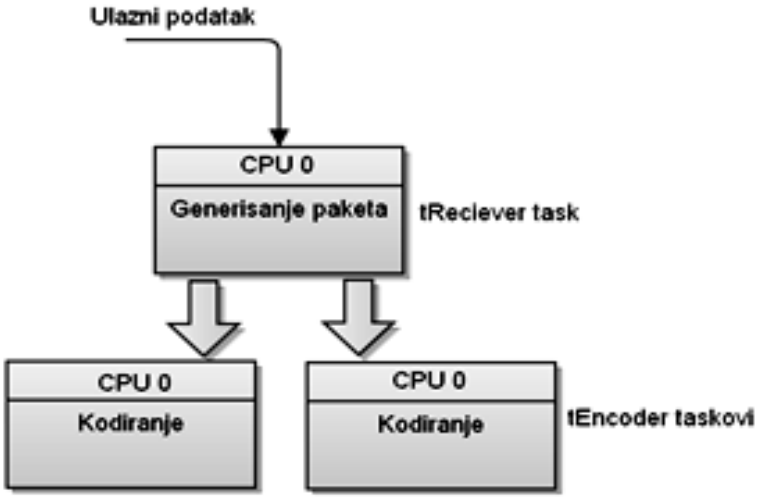
Priprema aplikacije za SMP

Nakon što identifikujemo koje dijelovi UP aplikacije treba paralelizovati, potrebno je izmijeniti te dijelove i promijeniti ih kako bi bili u mogućnosti da iskoristimo prednosti SMP sistema.

Razmatranja vezana za SMP dizajn

Kada smo identifikovali task koji najintenzivnije koristi procesor, spremni smo da dizajniramo SMP aplikaciju. Kako je **tEncoder** task taj koji najviše koristi procesor, imalo bi smisla da se on paralelizira i multiplicira. Na ovaj način će se povećati performanse ako se ovi taskovi mogu konkurentno raspoređivati na više jezgara. Identifikacija skupih-zauzetih taskova u kodu je ključno za SMP dizajn. Ovo su područja gdje bi se trebao iskoristiti paralelizam kako bi se unaprijedile performanse.

VxWORKS i VIŠEJEZGRENI PROCESORI



Slika prikazuje SMP dizajn aplikacije.

Međutim, ovaj redizajn uvodi novi problem: sada imamo dva **tEncoder** taska koji se takmiče za iste podatke. U jednojezgrenom svijetu, tipično rješenje je da se onemogući priempcija taska kada task pristupi dijeljenom resursu. Dakle, kada task za kodiranje pristupi podacima iz taska za primanje podataka, task isključuje raspoređivač kako bi ušao u kritičnu sekciju. Kada task završi sa primanjem podataka iz taska za prihvatanje podataka, on ponovo uključuje preempciju

VxWORKS i VIŠEJEZGRENI PROCESORI

i to predstavlja kraj kritične sekcije. Ova vrsta uzajamnog isključivanja podržana je u VxWorks UP aplikaciji sa **taskLock()** i **taskUnlock().**

```
/*******************************************************************************
* getReceiverData - primanje podataka
*
* Ova rutina prima bafer iz AVL drveta,
* ukljanja ga iz AVL drveta a zatim iscrtava
* primljeni bafer na grafikonu
*
* RETURNS: N/A
*/
void getReceiverData
        (
        atomic_t *selfCount,
        int threadNum,
        encoder_statistics_t *encoder_statistics,
        char *indexed_data_element
        )
        {
    int ret;
    char *received_data;
    char eventlBuff[EVENT_1_BUFSIZE];

#if DEBUG
    printf("threadNum - %d, Number Of Nodes: %d\n", threadNum, getNumberOfNodes());
#endif

    /* U multi-thread aplikaciji postoje dva tEncoder taska koji se izvršavaju
     * u UP sistemu. Raspoređivač taskova sprečava drugi tEncoder task da
     * pristupi AVL stablu jer je pristup drvetu zaštićen
     * sa taskLock () rutinom koja onemogućava task context switching.
     * Kako je AVL stablo izbalansirano,
     * nesinhroniziran pristup drvetu može koruptirati primljeni
     * čvor. Korištenje taskLock () rutine pristup AVL drvetu
     * je zaštićen sve do ponovnog uključivanja context task switchinga sa
     * taskUnlock () rutinom.
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
     */
    taskLock();

        /* sve dok ne primimo ocekivane pakete, continue nastavi sa processiranjem*/
    if (vxAtomicGet(sg_current_key) < g_numOfPackets)
        {
                bzero(eventlBuff, EVENT_1_BUFSIZE);
         sprintf(eventlBuff, "task-%d Event %d buffer# %d", threadNum, EVENT_1_ID,
(int)g_current_key);

         /* ucitaj EVENT_1_ID i g_current_key vrijednosti u eventlBuff,
         * upisi ih u system viewer koristenjem wvEvent */
         wvEvent (EVENT_1_ID, eventlBuff, EVENT_1_BUFSIZE);

         /* dohvati bafer sa indeksom - g_current_key iz AVL drveta*/
        received_data = avlEntryGet( EXACT_VALUE, sg_current_key);
#if DEBUG
        if (received_data == NULL)
                {
                printf("received_data from AVL tree is NULL, count = %d\n", g_current_key);
                }
#endif
        /* pomjeri AVL primljeni bafer u interni bafer taskova */
        memcpy(indexed_data_element, received_data, MAX_RX_TX_BUFFER_LENGTH);
        /* ukloni kljuc primljenih AVL bafera - g_current_key iz drveta */
                ret = avlRemove(g_current_key);

        /* g_ioPrint argument oznacava that da korisnik moze preuzeti kontrolu nad I/O.
         * Ako je ioPrint jednako 0, nece biti real-time izvjestaja na grafiku
         * sve dok rade tEncoder taskovi.*/
        if (g_ioPrint)
                setGraphPoints(indexed_data_element, threadNum);
        /* povecaj AVL key na slijedeci */
        vxAtomicInc (sg_current_key );
        }
        /* otpusti tasklock */
    taskUnlock();
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
        /* otpusti tasklock */
    taskUnlock();

        /* sakupi statistiku */
        setEncoderInputBufferSize(encoder_statistics, MAX_RX_TX_BUFFER_LENGTH);
        /* sakupi statistiku */
        incEncoderInputBufferCounter(encoder_statistics);

        /* Atomicno povecaj lokalni brojac. Ovaj brojac treba
         *za kasniju upotrebu za trenutni brojac.
         * Ovaj brojac se mora poslati na primjer u
         *klijent-server aplikaciji klijentu.
         *Klijentska aplikacija mora predstavljati aplikacijsku
         *statistiku na neki drugaciji nacin.
         * can present application statistics in different way*/

     vxAtomicInc ((atomic_t *) selfCount );

    /* postavi statistiku */
    setEncoderIterationCounter(encoder_statistics, (atomic_t *)selfCount);

    /* Atiomicno povecaj globalni brojac */
    vxAtomicInc ((atomic_t *) sg_general_statistics.count );

#if DEBUG
    printf("\ntaskId - %#x-%s task: # received chunks %u input buffer size %u buffer index
%d\n", taskIdSelf(), getEncoderTaskName(encoder_statistics),
getEncoderInputBufferCounter(encoder_statistics),
getEncoderInputBufferSize(encoder_statistics), getIndex(indexed_data_element));
#endif

}

/*****************************************************************************
```

Listing 2. Upotreba taskLock() rutine sa ciljem onemogučavanja task context switchinga.

VxWORKS i VIŠEJEZGRENI PROCESORI

API izmjene pri prelasku sa UP na SMP

Kod VxWorks UP dizajna, rutine za zaključavanje taska se koriste od strane taska kako bi se spriječilo raspoređivanje bilo kojeg drugog taska u sistemu, sve dok se ne pozove odgovarajuća rutina za otključavanje. Upotreba ovih rutina garantuje pristup kritičnoj sekciji koda uzajamnim isključivanjem.

Ovaj mehanizam može biti neprikladan za višeprocesorski sistem, što je razlog zašto rutine kao što **taskLock()** i **taskUnlock()** nisu podržane kod VxWorks SMP sistema. Kada se pozove API koji nije podržan u SMP kernelu, VxWorks dynamic linker javi grešku.

Za zaključavanje taskova, VxWorks SMP pruža alternativne **taskCpuLock()** rutine za situacije gdje svi taskovi koji učestvuju u scenariju zaključavanja taskova imaju jednak afinitet prema procesoru.

VxWORKS i VIŠEJEZGRENI PROCESORI

```
/*******************************************************************************
* getReceiverData - primanje podataka
*
* Ova rutina prima bafer iz AVL drveta,
* uklanja ga iz AVL drveta a zatim prikazuje indeks
* primljenog bafera na grafiku
*
* RETURNS: N/A
*/
void getReceiverData
        (
        atomic_t *selfCount,
        int threadNum,
        encoder_statistics_t *encoder_statistics,
        char *indexed_data_element
        )
        {
    int ret;
    char *received_data;
    char eventlBuff1[EVENT_1_BUFSIZE];
    char eventlBuff2[EVENT_2_BUFSIZE];

#if DEBUG
    printf("threadNum - %d, Number Of Nodes: %d\n", threadNum, getNumberOfNodes());
#endif

    /* U 4-smp-non-safe application, postoje dva tEncoder taska
     * koji se izvršavaju u SMP sistemu koji treba da isključuju jedan drugog. taskLock ()
     * zamijenjuje taskCpuLock () u SMP sistemu, i ne pruža
     * mehanizam isključivanja opisan u multi-thread aplikaciji.
     * Na visoprocesorskom sistemu kao što je SMP, ako isključimo preempciju
     * sa taskCpuLock, preempcija je isključena za prvi tEncoder
     * task koji se izvršavan a prvoj jezgri. Isključivanje preempcije na prvoj jezgri
     * ne znači da drugi tEncoder task, koji se izvršava na drugoj jezgri,
     * ne može pristupi AVL drvetu. Kako je AVL drvo izbalansirano
     * nesinhronizovan pristup drvetu tokom procesa balansiranja
     * od strane drugog jezgra može korumpirati zaprimljeni čvor. Korištenjem
     * taskCpuLock () na SMP sistemu ne osigurava da će pristup
     * AVL drvetu biti zaticen. Na SMP sistemu, čak i ako je preempcija
     * zaticena od strane jednog taska, ostali taskovi koji se izvršavaju na drugom procesoru
     * još uvijek mogu pristupiti kriticnoj sekciji; tako da nismo sigurni.
     * */
    taskCpuLock();
/*    taskLock();*/
```

VxWORKS i VIŠEJEZGRENI PROCESORI

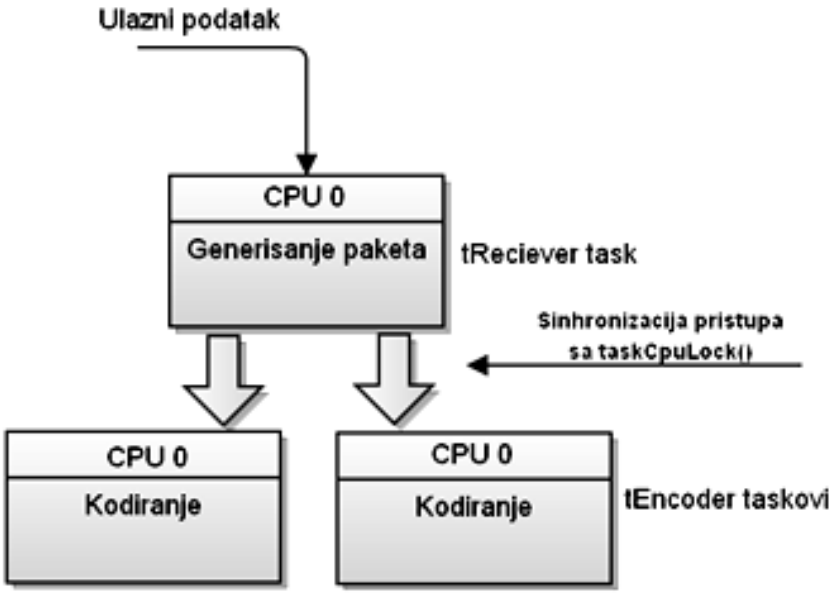
```
     * */
    taskCpuLock();
/*    taskLock();*/

#define NON_SAFE_RUN
#ifdef NON_SAFE_RUN
    if (vxAtomicGet(&g_current_key) > (g_numOfPackets-2))
    {
        vxAtomicInc (&g_current_key );
        return;
    }
#endif

    if (vxAtomicGet(&g_current_key) < g_numOfPackets)
        {

        bzero(eventlBuff1, EVENT_1_BUFSIZE);
        sprintf(eventlBuff1, "core-%d Event %d buffer# %d", threadNum, EVENT_1_ID,
vxAtomicGet(&g_current_key));

        bzero(eventlBuff2, EVENT_2_BUFSIZE);
        sprintf(eventlBuff2, "core-%d Event %d buffer# %d", threadNum, EVENT_2_ID,
g_current_key);

         wvEvent (EVENT_1_ID, eventlBuff1, EVENT_1_BUFSIZE);

        received_data = avlEntryGet( EXACT_VALUE, &g_current_key);

        if (received_data == NULL)
                {
                /*printf("received_data from AVL tree is NULL, count = %d\n",
vxAtomicGet(&g_current_key));*/
                goto error;
                }

                memcpy(indexed_data_element, received_data, MAX_RX_TX_BUFFER_LENGTH);
                if (vxAtomicGet(&g_current_key) < g_numOfPackets)
                        ret = avlRemove(g_current_key);
        if (g_isPrint)
                setGraphPoints(indexed_data_element, threadNum);
        vxAtomicInc (&g_current_key );

         wvEvent (EVENT_2_ID, eventlBuff2, EVENT_2_BUFSIZE);
        }
    taskCpuUnlock();
/*    taskUnlock();*/
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
    taskCpuUnlock();
/*    taskUnlock();*/

        setEncoderInputBufferSize(encoder_statistics, MAX_RX_TX_BUFFER_LENGTH);
        incEncoderInputBufferCounter(encoder_statistics);

    vxAtomicInc ((atomic_t *) selfCount );

    setEncoderIterationCounter(encoder_statistics, (atomic_t *)selfCount);

    vxAtomicInc ((atomic_t *) sg_general_statistics.count );

#if DEBUG
    printf("\ntaskId - %#x-%s task: # received chunks %u input buffer size %u buffer index
%d\n", taskIdSelf(), getEncoderTaskName(encoder_statistics),
getEncoderInputBufferCounter(encoder_statistics),
getEncoderInputBufferSize(encoder_statistics), getIndex(indexed_data_element));
#endif

}

/*****************************************************************************
```

Listing 3. Upotreba taskCpuLock() rutine u SMP sistemu.

VxWORKS i VIŠEJEZGRENI PROCESORI



Testiranje izmjenjene aplikacije u UP režimu rada

U prethodnom koraku smo identifikovali koji dijelovi UP aplikacije će iskoristiti prednost SMP dizajna. Sada je potrebno redizajnirati te dijelove aplikacije i izmijeniti ih kako bi bili u mogućnosti da iskoristimo prednost SMP dizajna. Najprije je potrebno testirati redizajniranu aplikaciju kako bi se

VxWORKS i VIŠEJEZGRENI PROCESORI

ustanovilo da radi u UP režimu. Ovo je bitan korak kod UP-SMP migracije, obzirom da je prvi korak u postizanju paralelizma višenitnost (multithreading) dio koji bi se trebao izvršavati paralelno i najlakši način da se potvrdi da višenitnost radi je da se aplikacija testira u UP režimu.

Da bi ovo postigli treba da uradimo slijedeće:

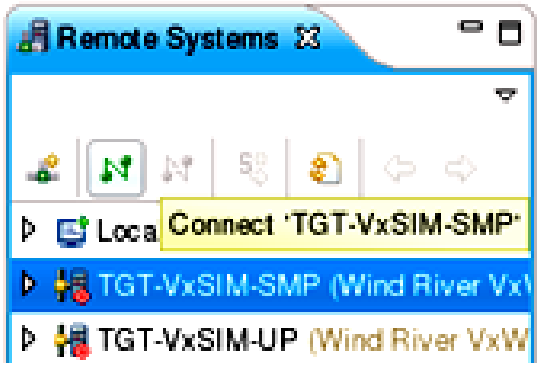
- Pokretanje redizajnirane verzije demo aplikacije koja implementira SMP dizajn (višenitnost) u UP režimu.
- Korištenje Wind River System Viewera kako bi ustanovili da li redizajnirana aplikacija radi kako očekujemo.

VxWORKS i VIŠEJEZGRENI PROCESORI

Pokretanje redizajnirane aplikacije

U sklopu VxWorks SMP razvojnog okruženja dolazi unaprijed modificirana verzija demo aplikacije koja implementira SMP dizajn korištenjem dva taska za kodiranje. Da bi pokrenuli redizajniranu aplikaciju (sa dva Encoder taska) u UP režimu potrebno je uraditi slijedeće:

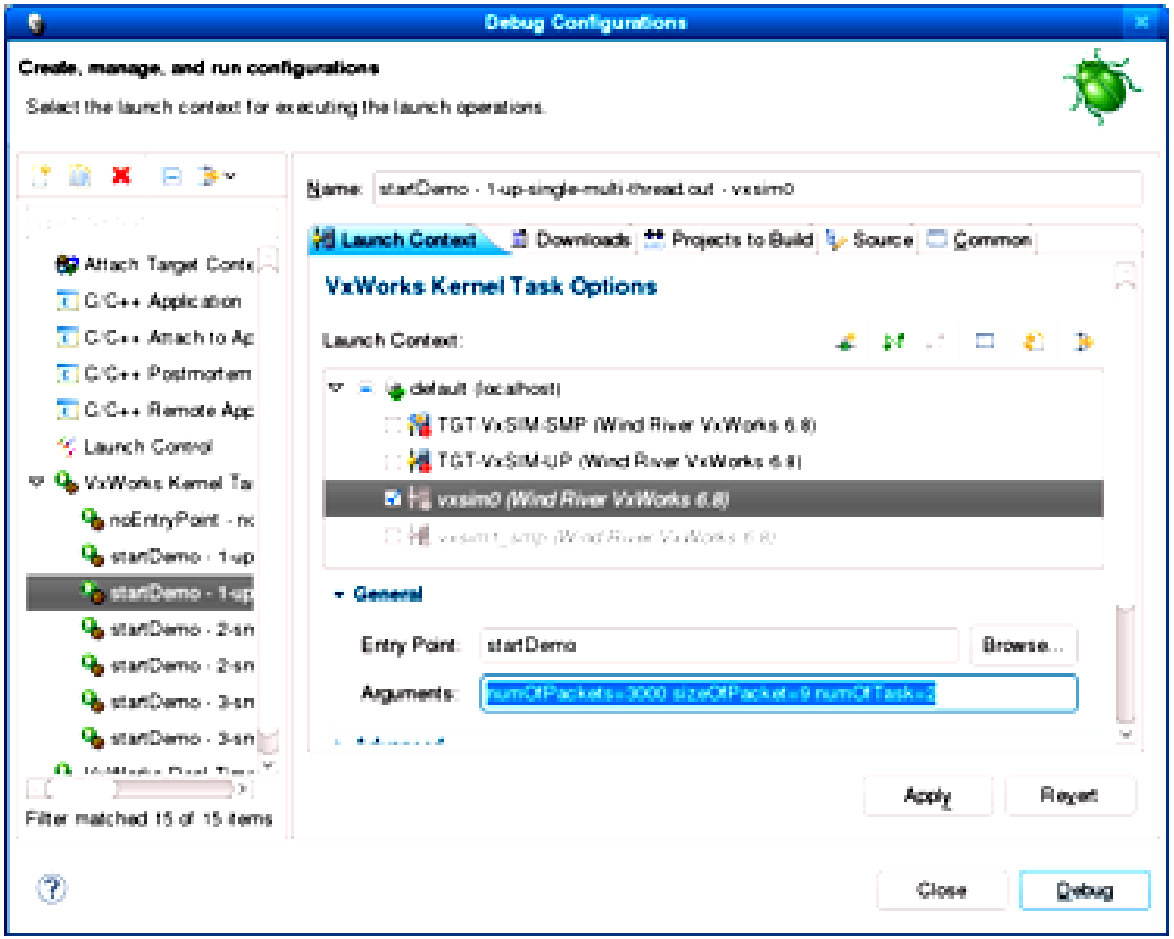
1. Pokrenuti simulator izborom **TGT-VxSMP-UP** simulatora u Remote Systems pogledu, a zatim kliknuti na dugme **Connect**.



VxWORKS i VIŠEJEZGRENI PROCESORI

2. Iz Project Explorer pogleda odabrati **UP-To-SMP-Migration-Demo** projekt i **Build Targets** stavku u sklopu toga.
3. Desni klik na prvi projekt, **1-up-single-multi-thread**.
4. Odabrati **Debug VxWorks Kernel Task** iz kontekstnog menija.
5. U **Launch Configuration Selection** dijalogu, odabrati **Edit the selected launch configuration**, a zatim kliknuti na OK.
6. U Debug Configuration dijalogu, potvrditi da je u **Arguments** upisano **numOfPacket=30000 sizeOfPacket=9 numOfTask=2**.

VxWORKS i VIŠEJEZGRENI PROCESORI



7. Odabrati Debug. Ovo pokreće aplikaciju. Sada bi trebali da vidimo izvorni kod u Workbench editoru kao i to da je aplikacija zaustavljena u Debug pogledu.

VxWORKS i VIŠEJEZGRENI PROCESORI

Korištenje Wind River System Viewera

Wind River System Viewer je run-time alat za analizu namijenjen programerima koji razvijaju softver za uređaje i koji treba da istraže dinamičko ponašanje sistema sa ciljem da se ustanove run-time problemi i poboljšaju performanse sistema.

System Viewer zapisuje sistemske događaje koji se dese za vrijeme izvršavanja u realnom vremenu a zatim ih prikazuje kao i stanja taska na grafičkom displeju koji je intuitivan i informativan. Postoje dvije vrste događaja koji se mogu prikazati u System Vieweru:

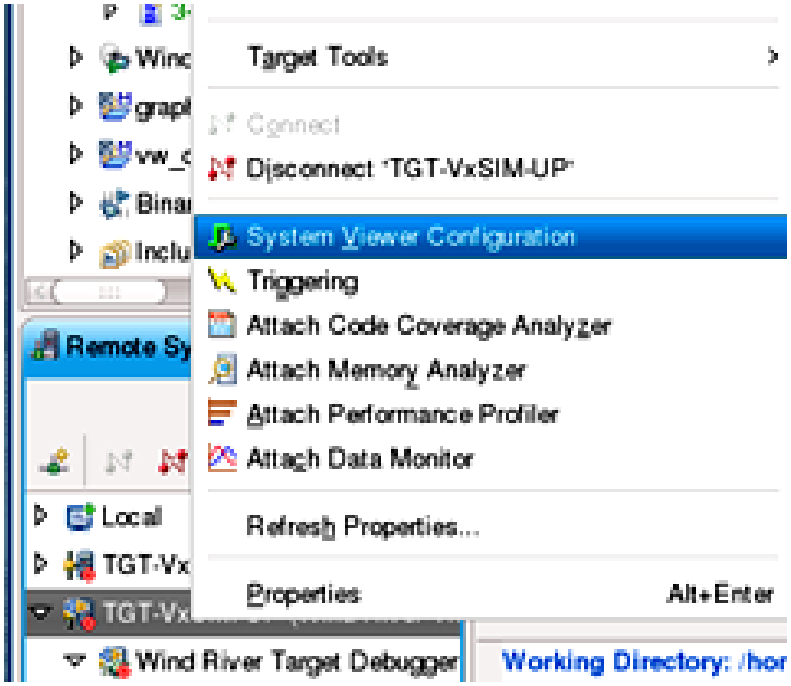
- Sistemski događaji – kao što su akvizicije semafora, ili promjena stanja taska iz **running** u **pending**.
- Korisnički događaji – Ovi događaji su podešeni od strane korisnika i mogu sadržavati podesive stringove.

VxWORKS i VIŠEJEZGRENI PROCESORI

Pokretanje System Viewer alata

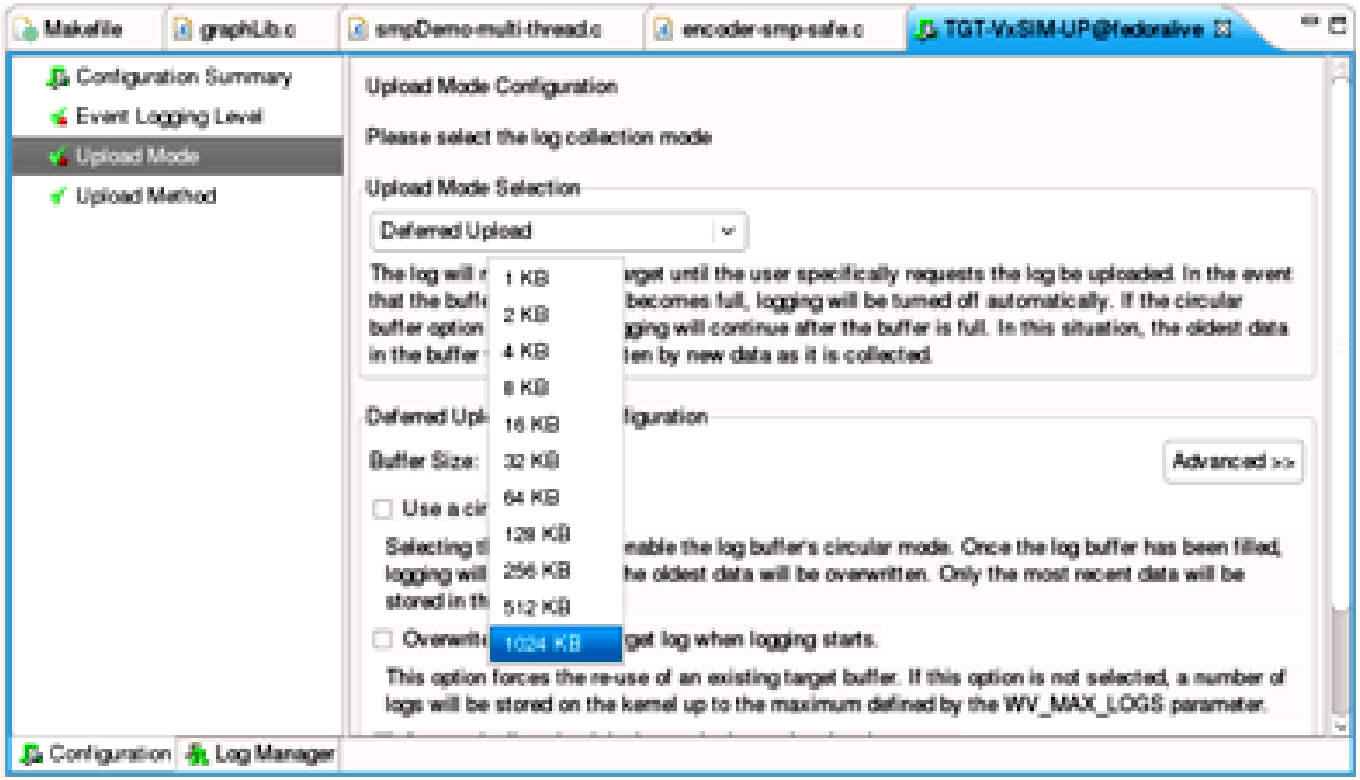
Da bi pokrenuli System Viewer potrebno je uraditi slijedeće:

1. Desni klik na **TGT-VxSIM-UP** u Remote Systems pogledu.
2. Iz kontekstnog menija odabrati **System Viewer Configuration**.



VxWORKS i VIŠEJEZGRENI PROCESORI

3. Pojavljuje se slijedeći pogled u Editoru
4. Standardna velićina bafera u koji će System Viewer smještati događaje je 32KB. Da bi obezbijedili da se upišu svi podaci, moramo povećati bafer tako što ćemo odabrati **Upload Mode**.
5. Koristiti **Buffer Size** kontrolu kako bi podesili velićinu na 1024KB.



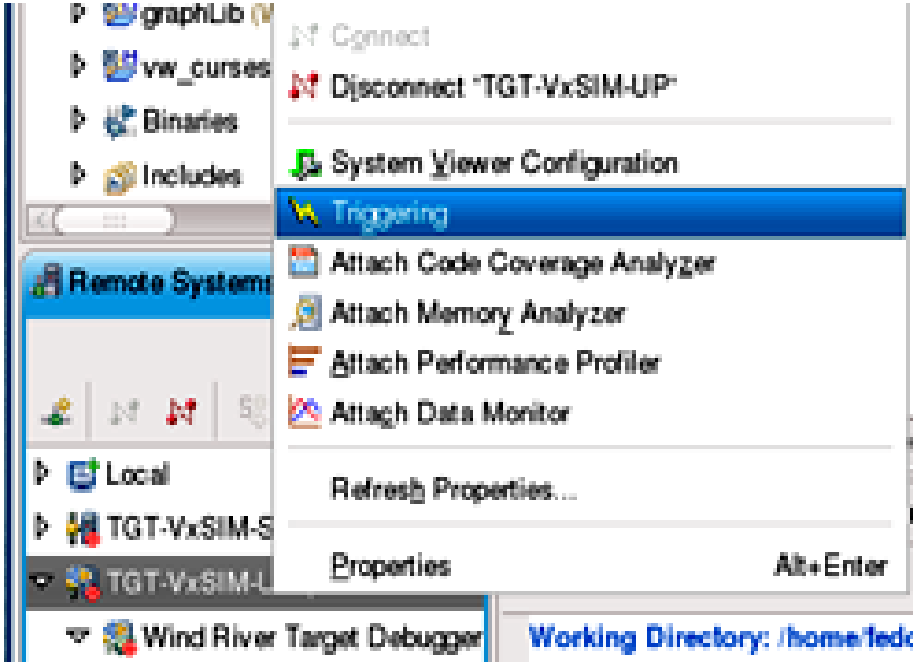
VxWORKS i VIŠEJEZGRENI PROCESORI

Korištenje System Viewer Triggering opcije

Okidanje (triggering) počinje i završava dijagnostičko sakupljanje podataka po pozivanju. Na primjer, pretpostavimo da je aplikacija veoma duga sa stotinama događaja; međutim, zainteresovani smo samo za mali dijelić izvršavanja aplikacije. Uz pomoć okidanja možemo podesiti alat da počne sa prikupljanjem podataka na malom dijelu aplikacije. Da bi maksimalno iskoristili System Viewer alat, treba da koristimo unaprijed pripremljeni fajl za okidanje. Ovo će usredotočiti System Viewer na interesantno područje izvršavanja. Interesantni dio aplikacije je task za kodiranje. Potrebno je slijediti slijedeće korake kako bi se pokrenuo fajl za okidanje:

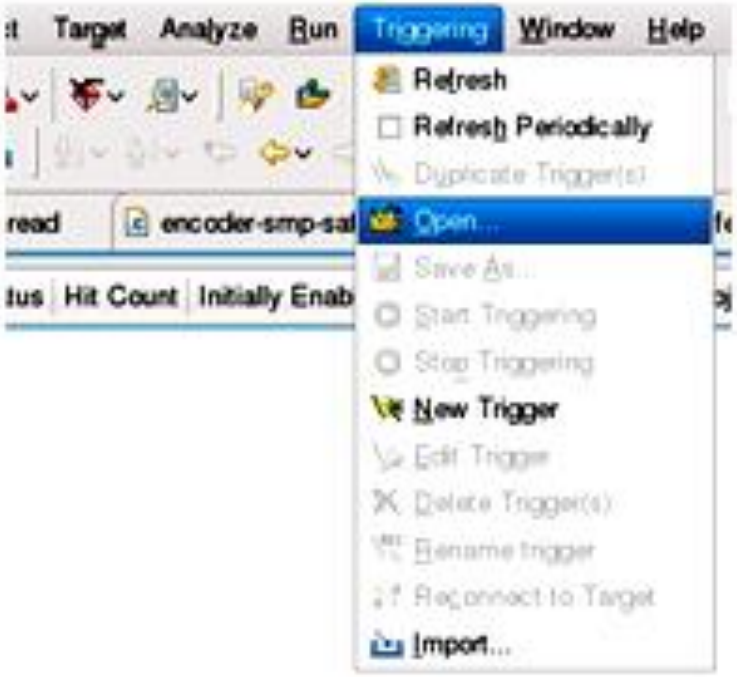
1. Desni klik na **TGT-VxSIM-UP** u Remote Systems pogledu.
2. Iz kontekstnog menija odabrati **Triggering**.

VxWORKS i VIŠEJEZGRENI PROCESORI



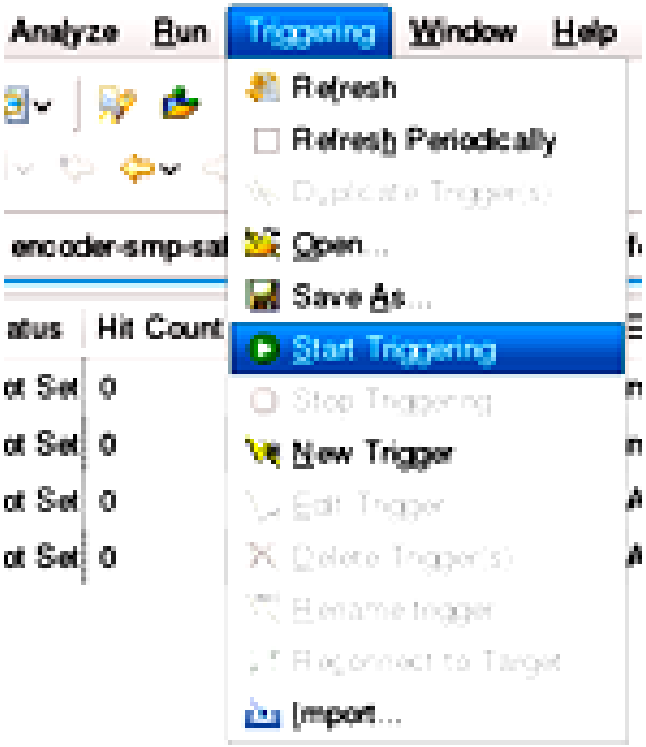
3. Novi Triggering pristup je dodat Workbenchu.
4. Kliknuti na novi Triggering pristup i odabrati Open.

VxWORKS i VIŠEJEZGRENI PROCESORI



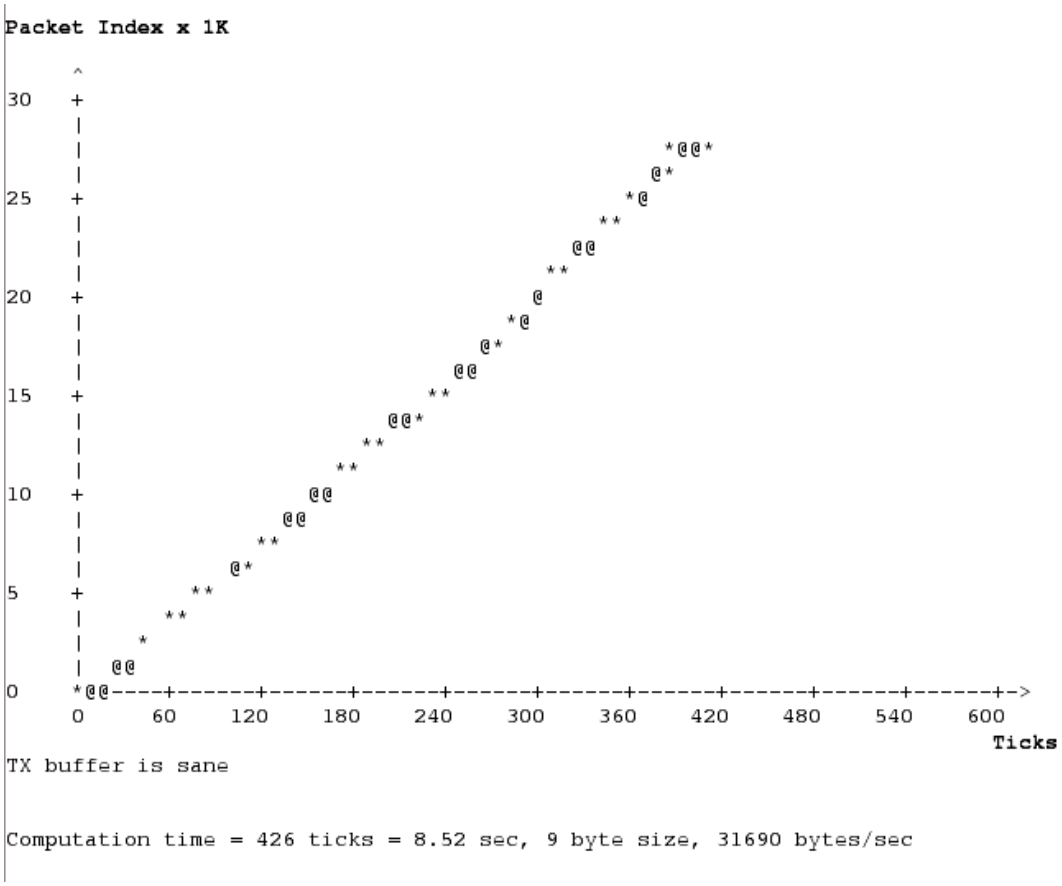
5. U Open Trigger File dijalogu, **pronaći /home/fedora/WindRiver/workspace/smp-demo-source**.
6. Odabrati **startTrig.trig** fajl.
7. Kliknuti na OK.

VxWORKS i VIŠEJEZGRENI PROCESORI



10. Kada aplikacija završi sa izvršavanjem, bićemo u mogućnosti pokrenuti aplikaciju klikom na **Resume** dugme u Debug prozoru.
11. Vidjet ćemo slijedeći izlaz na simulatoru:

VxWORKS i VIŠEJEZGRENI PROCESORI



Zvjezdica (*) predstavlja podatke procesirane od strane prvog taska za kodiranje, dok znak @ predstavlja podatke procesirane od strane drugog taska za kodiranje. Rezultati mogu varirati u zavisnosti od računara na kojem je pokrenut simulator, vrste i brzine procesora, broja jezgara, velićine

VxWORKS i VIŠEJEZGRENI PROCESORI

i opterećenja sistema

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE triggerList [<!ELEMENT triggerList (trigger*)>
<!ATTLIST triggerList TriggerClass CDATA #REQUIRED >
<!ELEMENT trigger (extensions*)>
<!ATTLIST trigger TriggerName CDATA #REQUIRED InitialState CDATA #REQUIRED Event CDATA
#REQUIRED Context CDATA #REQUIRED Object CDATA #REQUIRED Conditional CDATA #REQUIRED
Expression CDATA #REQUIRED ConstantExpression CDATA #REQUIRED Operator CDATA #REQUIRED Action
CDATA #REQUIRED ActionFunction CDATA #REQUIRED ActionArgument CDATA #REQUIRED Chain CDATA
#REQUIRED ActionDeferred CDATA #REQUIRED DisableAfterFire CDATA #REQUIRED Rtp CDATA #REQUIRED
>
<!ELEMENT extensions EMPTY>
<!ATTLIST extensions ExtensionName CDATA #REQUIRED ExtensionValue CDATA #REQUIRED >]>
<triggerList TriggerClass = "VxWorks6x">
        <trigger TriggerName="stopTrig" InitialState="false" Event="msgQDelete" Context="Any
Task" Object="Any Object" Conditional="Unconditional" Expression="identifier"
ConstantExpression="constant" Operator="TRIGGER_EQ" Chain="svInitUpload"
DisableAfterFire="true" Action="Call Function" ActionFunction="wvEvtLogStop"
ActionArgument="0" ActionDeferred="false" Rtp="Default (VxKernel)" >
        </trigger>
        <trigger TriggerName="startTrig" InitialState="true" Event="msgQCreate" Context="Any
Task" Object="Any Object" Conditional="Unconditional" Expression="identifier"
ConstantExpression="constant" Operator="TRIGGER_EQ" Chain="stopTrig" DisableAfterFire="true"
Action="Call Function" ActionFunction="wvEvtLogStart" ActionArgument="0"
ActionDeferred="false" Rtp="Default (VxKernel)" >
        </trigger>
        <trigger TriggerName="svSetup" InitialState="false" Event="Any Event" Context="Any"
Object="Any Object" Conditional="Unconditional" Expression="identifier"
ConstantExpression="constant" Operator="TRIGGER_EQ" Chain="None" DisableAfterFire="true"
Action="Start System Viewer Logging" ActionFunction="wvEvtLogStart" ActionArgument="argument"
ActionDeferred="true" Rtp="Default (VxKernel)" >
        </trigger>
        <trigger TriggerName="svInitUpload" InitialState="false" Event="Any Event"
Context="Any" Object="Any Object" Conditional="Unconditional" Expression="identifier"
ConstantExpression="constant" Operator="TRIGGER_EQ" Chain="None" DisableAfterFire="true"
Action="Stop System Viewer Logging" ActionFunction="wvEvtLogStop" ActionArgument="argument"
ActionDeferred="true" Rtp="Default (VxKernel)" >
        </trigger>
</triggerList
>
```

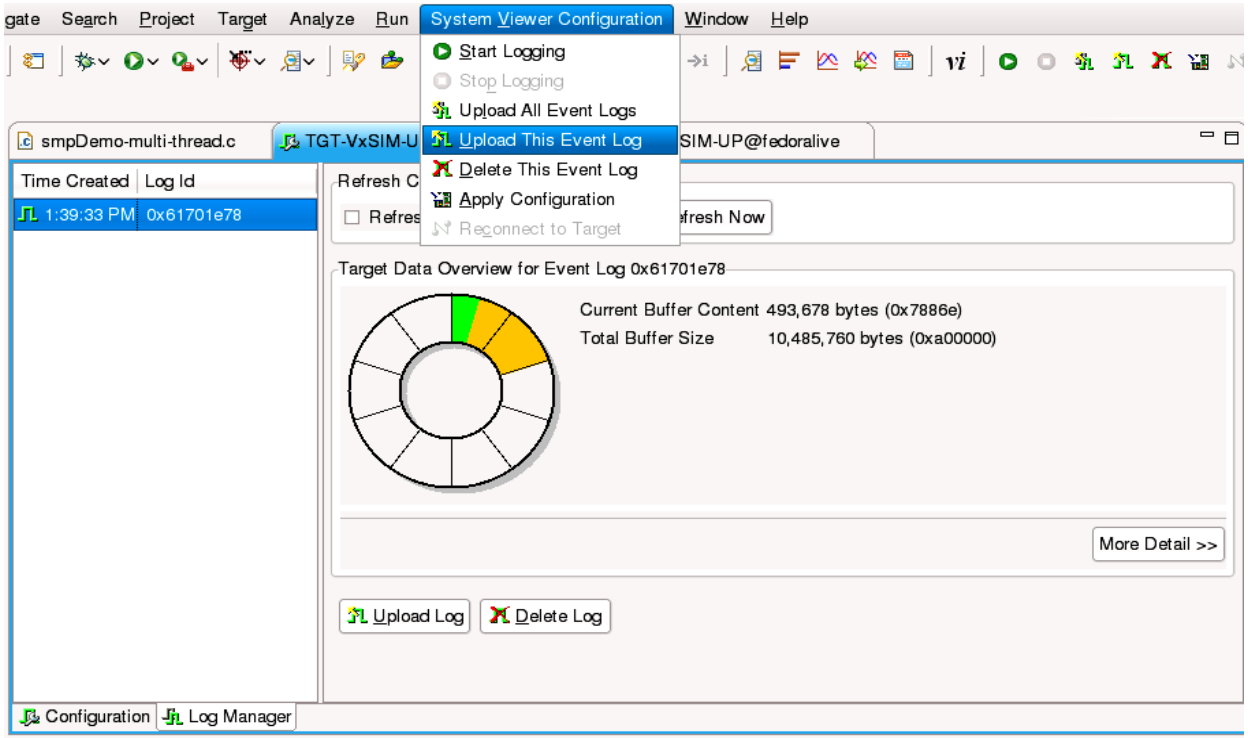
Sadržaj startTrig.trig fajla

VxWORKS i VIŠEJEZGRENI PROCESORI

Prikaz rezultata u System Viewer alatu

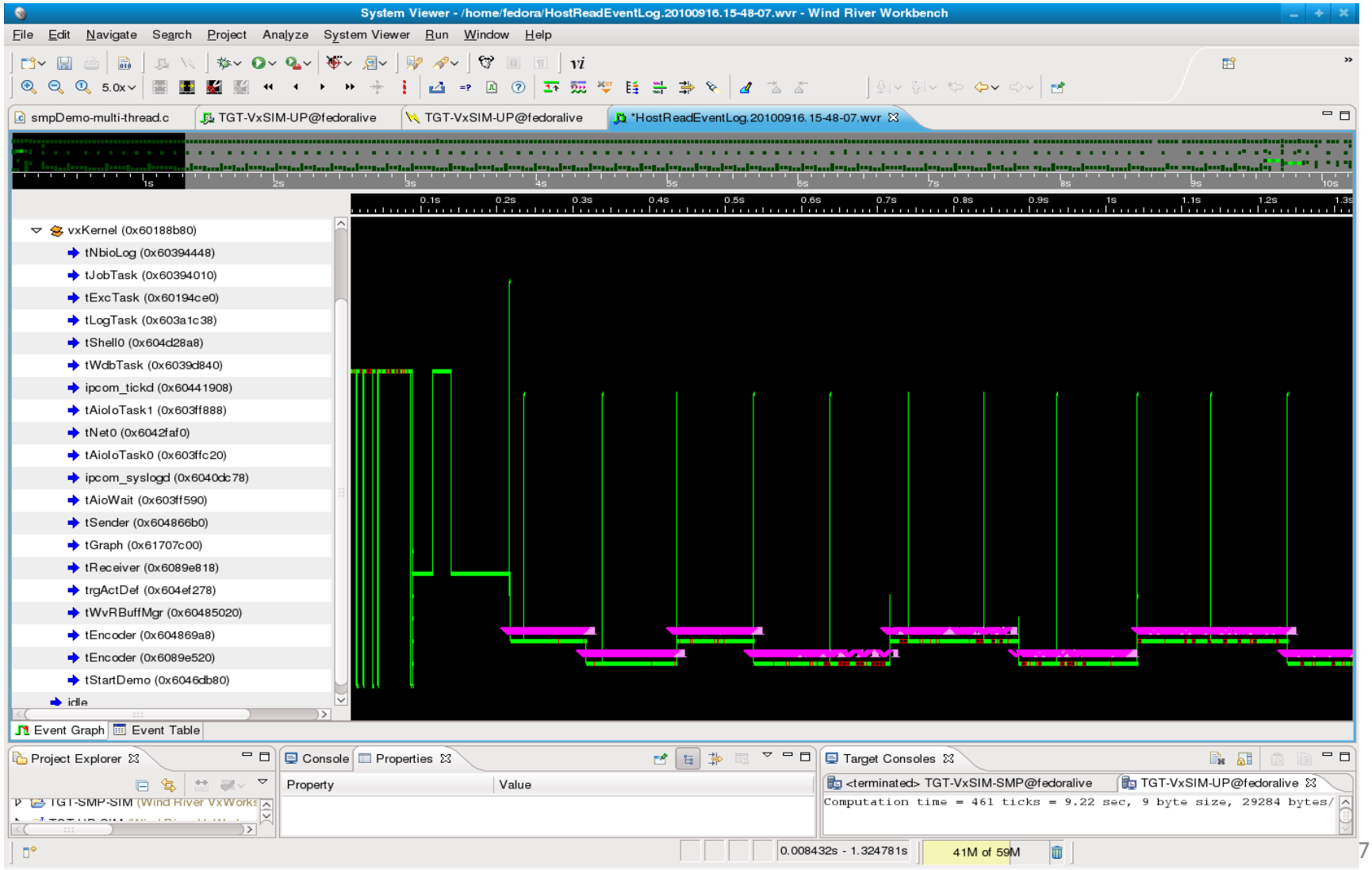
Da bi pregledali događaje koji su se desili prilikom izvršavanja potrebno je slijediti slijedeće korake:

1. Kliknuti na **System Viewer Configuration** traku.
2. Pod System Viewer Configuration Menu stavkom, odabrati **Upload This Event Log** kako bi prebacili podatke sa target mašine u cilju analize performansi.



VxWORKS i VIŠEJEZGRENI PROCESORI

3. Kada se prebace podaci o događajima, Performance Profiler kreira grafički displej aktivnosti na ciljnoj mašini.

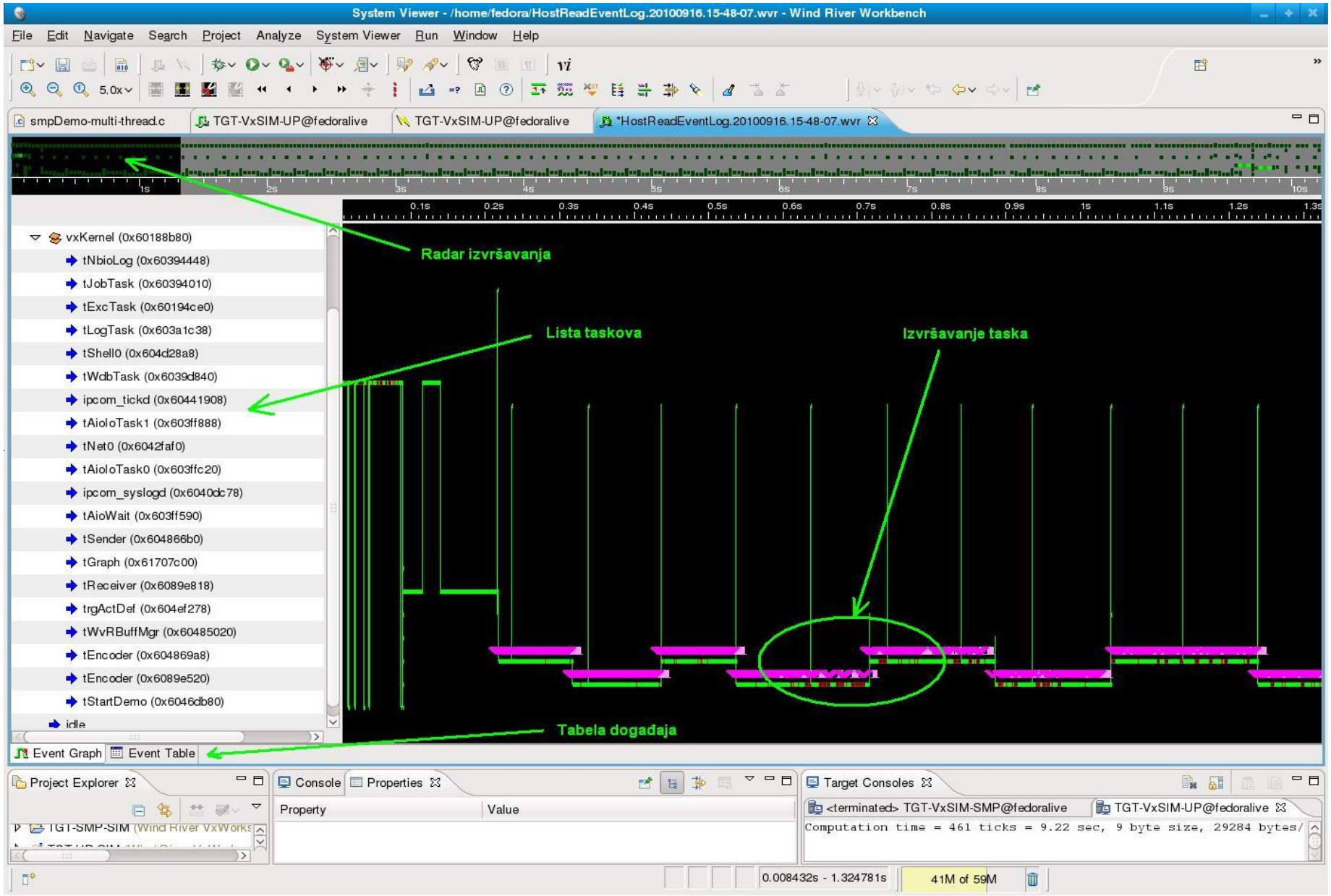


VxWORKS i VIŠEJEZGRENI PROCESORI

Korištenje System Viewer alata kako bi se razumjelo izvršavanje izmjenjene aplikacije

Taskovi su prikazani na lijevoj strani. Puna linija predstavlja izvršavanje. U mogućnosti smo da pratimo svaku milisekundu izvršavanja. Radar izvršavanja je lociran na vrhu grafika i naglašava trenutno područje prikazano na grafikonu. Ako želimo da proširimo područje pogleda potrebno je da proširimo radar.

VxWORKS i VIŠEJEZGRENI PROCESORI

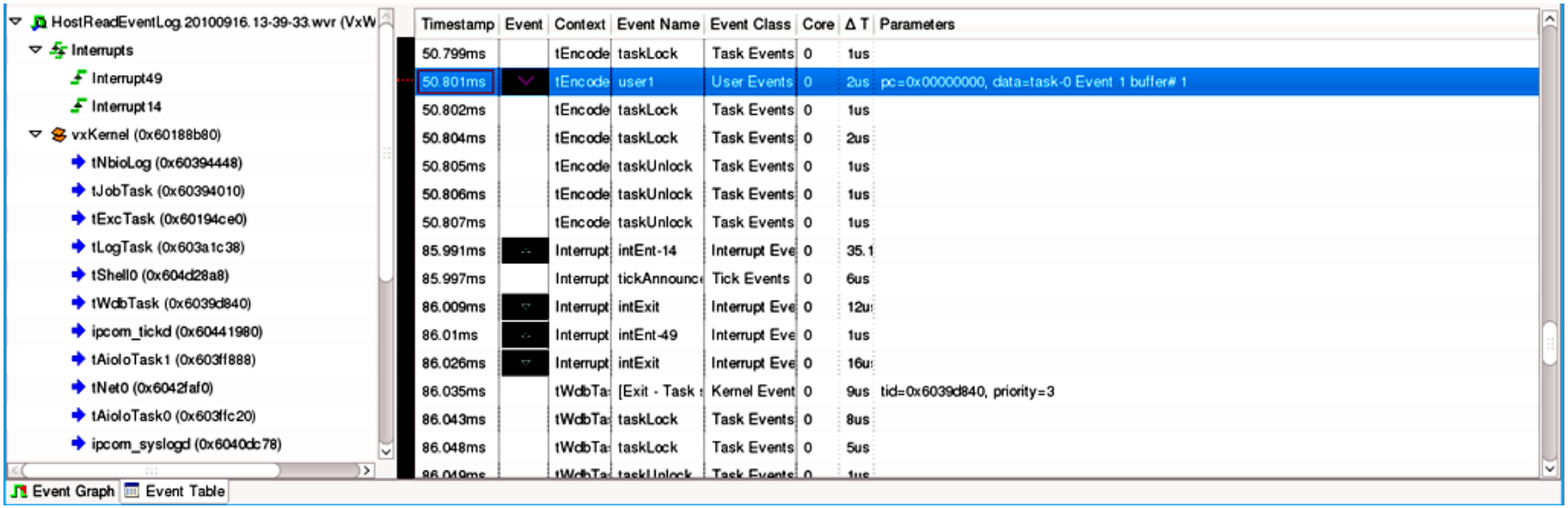


VxWORKS i VIŠEJEZGRENI PROCESORI

Identifikacija System Viewer događaja

Treba primjetiti da postoje dva taska za kodiranje. Brojevi u zagradama su identifikacijski brojevi taskova. Taskovi za kodiranje su zauzeti većinu vremena, i procesor se prebacuje između izvršavanja ovih taskova. Kritična sekcija se povećala u novom SMP dizajnu iz razloga što se ista struktura podataka dijeli između dva taska za kodiranje. Kôd aplikacije generiše korisnički događaj na mjestu ulaska u kritičnu sekciju. Ovaj događaj je zabilježen od strane System Viewera i označen na grafiku sa V1. Svaki događaj je predstavljen kao ikona u System Viewer grafikonu; dodatno, svi događaji su prikazani u tabeli događaja.

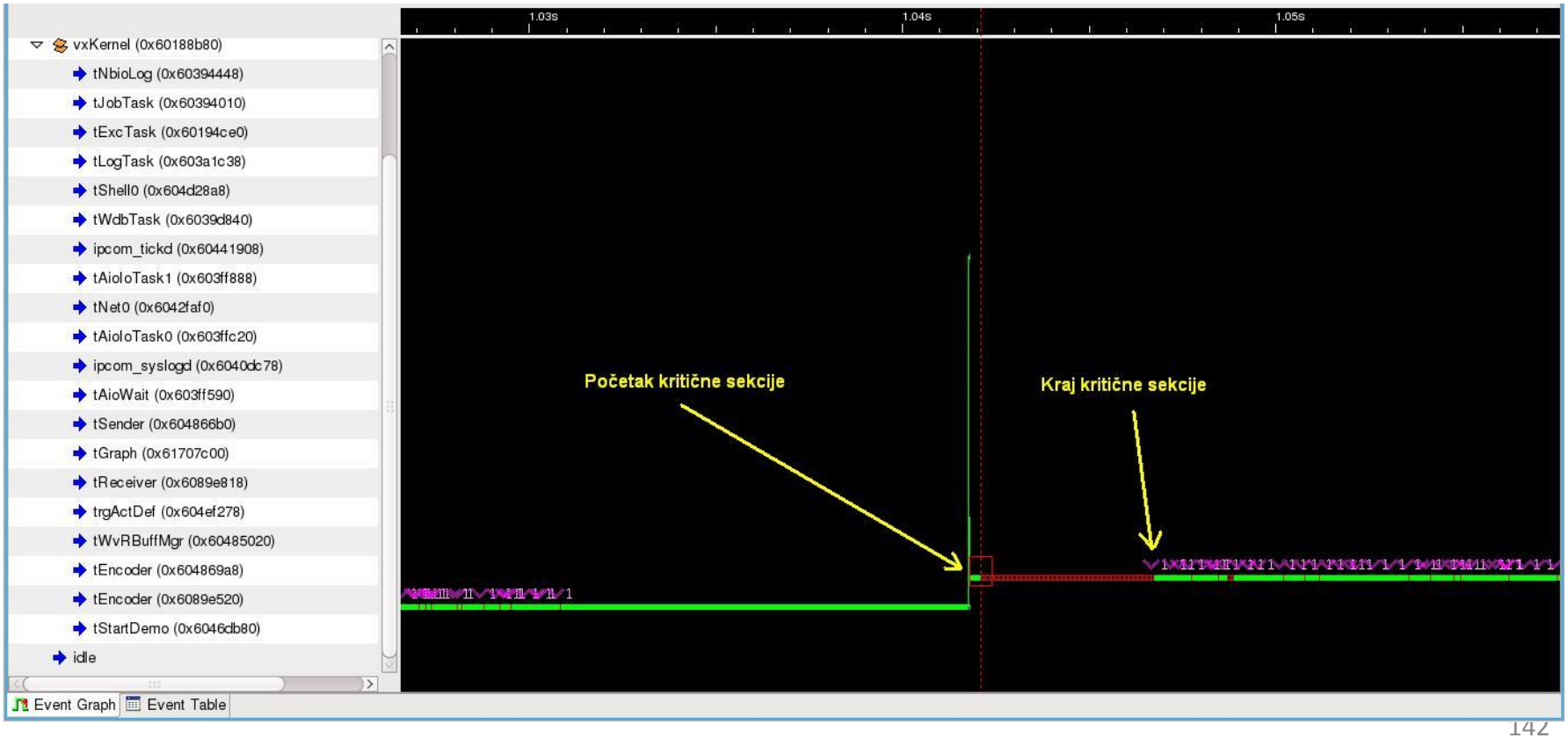
VxWORKS i VIŠEJEZGRENI PROCESORI



Sa ciljem da uspostavimo vezu između specifičnog događaja sa ikonom tog događaja u grafikonu izvršavanja potrebno je ostvariti dvostruki klik na događaj kako bi ga označili. Treba primjetiti da je dodata crvena strelica u System Viewer radar. Klikom na Event Graph karticu i Zoom dugme, zumiramo grafik kako bi mogli detaljnije sagledati područje izvršavanja. Treba primjetiti da se na mjestu gdje task ulazi u kritičnu sekciju boja taska mijenja iz zelene u crvenu boju.

VxWORKS i VIŠEJEZGRENI PROCESORI

Na narednoj slici, strelica pokazuje na kritičnu sekciju. Kritična sekcija u ovoj aplikaciji počinje kada jedan od taskova za kodiranje primi podatke iz Reciever taska. Kako je status taska promijenjen iz **running** u **running+locked**, boja linije izvršavanja taska se mijenja iz pune zelene u tačkasto crvenu



VxWORKS i VIŠEJEZGRENI PROCESORI

Testiranje i izmjena aplikacije u SMP režimu

U ovom koraku ćemo umogućiti višejezgrenost i pokrenuti aplikaciju. Ovo znači da se dva ili više taskova mogu pokrenuti konkurentno. Dakle, ako aplikacija zavisi od implicitne sinhronizacije – na primjer ako je sinhronizacija zasnovana na prioritetu taska – tada će postojati utrka za podacima, što može dovesti do nepredvidivog ponašanja.

Pokretanje izmjenjene aplikacije u SMP režimu

Da bi pokrenuli novi SMP dizajn aplikacije (sa dva Encoder taska) potrebno je uraditi slijedeće:

1. Kreirati i pokrenuti **TGT-VxSIM-SMP** simulator u Remote Systems pogledu, a zatim kliknuti na **Connect** dugme.
2. U Project Explorer pogledu, raširiti **UP-To-SMP-Migration-Demo** i **Build Targets** u sklopu njega.
3. Desni klik na drugi projekt, **2-SMP-unsafe**.

4. U kontekstnom meniju odabrati **Debug VxWorks Kernel Task**.
5. U Launch Configuration Selection dijalogu odabrati **Edit the selected launch configuration** a zatim klinuti na OK.
6. U Debug Configurations dijalogu, promijeniti **Arguments** u: **numOfPacket=30000 sizeOfPacket=9 numOfTask=2**.
7. Kliknuti na **Debug**. Ovo pokreće aplikaciju. Trebalo bi da se pojavi izvorni fajl u Workbench editoru s tim da je aplikacija zaustavljena u Debug pogledu.

Korištenje Wind River System Viewer alata

Potrebno je ponoviti istu proceduru koja je već ranije opisana za slučaj **TGT-VxSIM-UP** samo što je sada potrebno odabrati **TGT-VxSIM-SMP**. Također je potrebno koristiti triggering opciju na ranije opisan način.

VxWORKS i VIŠEJEZGRENI PROCESORI

U primjeru je dat nesinhronizovani scenario u kojem dva taska pristupaju istom resursu istovremeno. Kao rezultat toga, izvršavanje može pasti zbog greške segmentacije memorije. Ako simulator bude blokiran više od 30 sekundi, potrebno se isključiti sa istog i ponovo krenuti od prvog koraka. Možemo primjetiti da aplikacija proizvodi neke greške u obliku „index X, packet is missed“. Kada aplikacija završi sa izvršavanjem, vidjećemo da se aplikacija nije izvršavala.

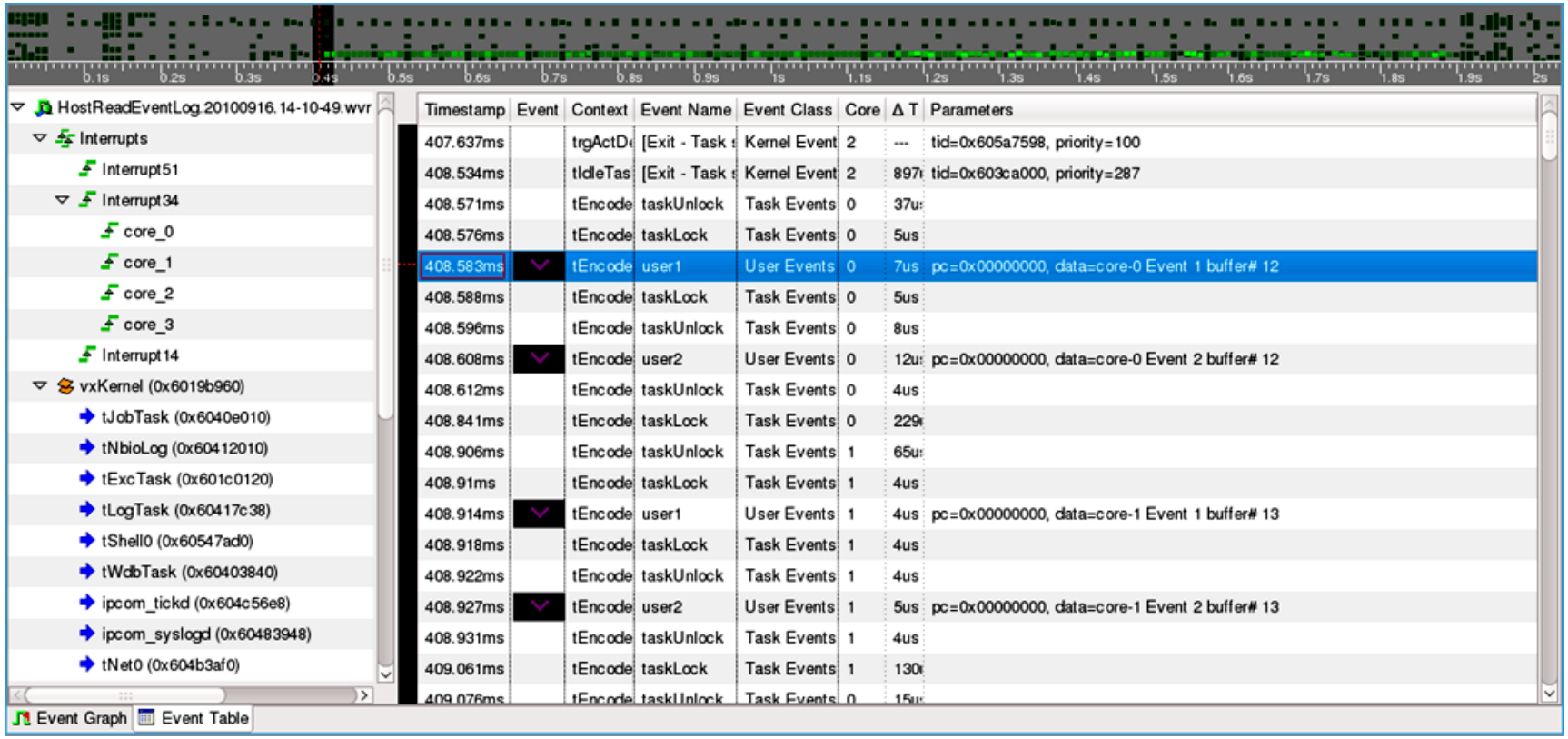
VxWORKS i VIŠEJEZGRENI PROCESORI

Analiza novih SMP grešaka sa System Viewer alatom

Treba primjetiti da aplikacija prijavljuje da jedan ili više bafera nedostaje. Da bi ustanovili u čemu je problem potrebno je uraditi slijedeće:

1. Pokrenuti System Viewer odabirom System Viewer Configuration u glavnom meniju i izabrati *Upload This Event Log* iz podmenija.
2. Kliknuti na **Event Table** karticu. Ovo će nam pomoći da se fokusiramo na određeni događaj, obzirom da ima na stotine događaja koji su se desili u toku izvršavanja.
3. Pronači događaj gdje se desila greška. Na primjer, ako jedna greška prikazuje da bafer 308 nedostaje, potrebno je pogledati u događaj 307 i njegove parametre.

VxWORKS i VIŠEJEZGRENI PROCESORI



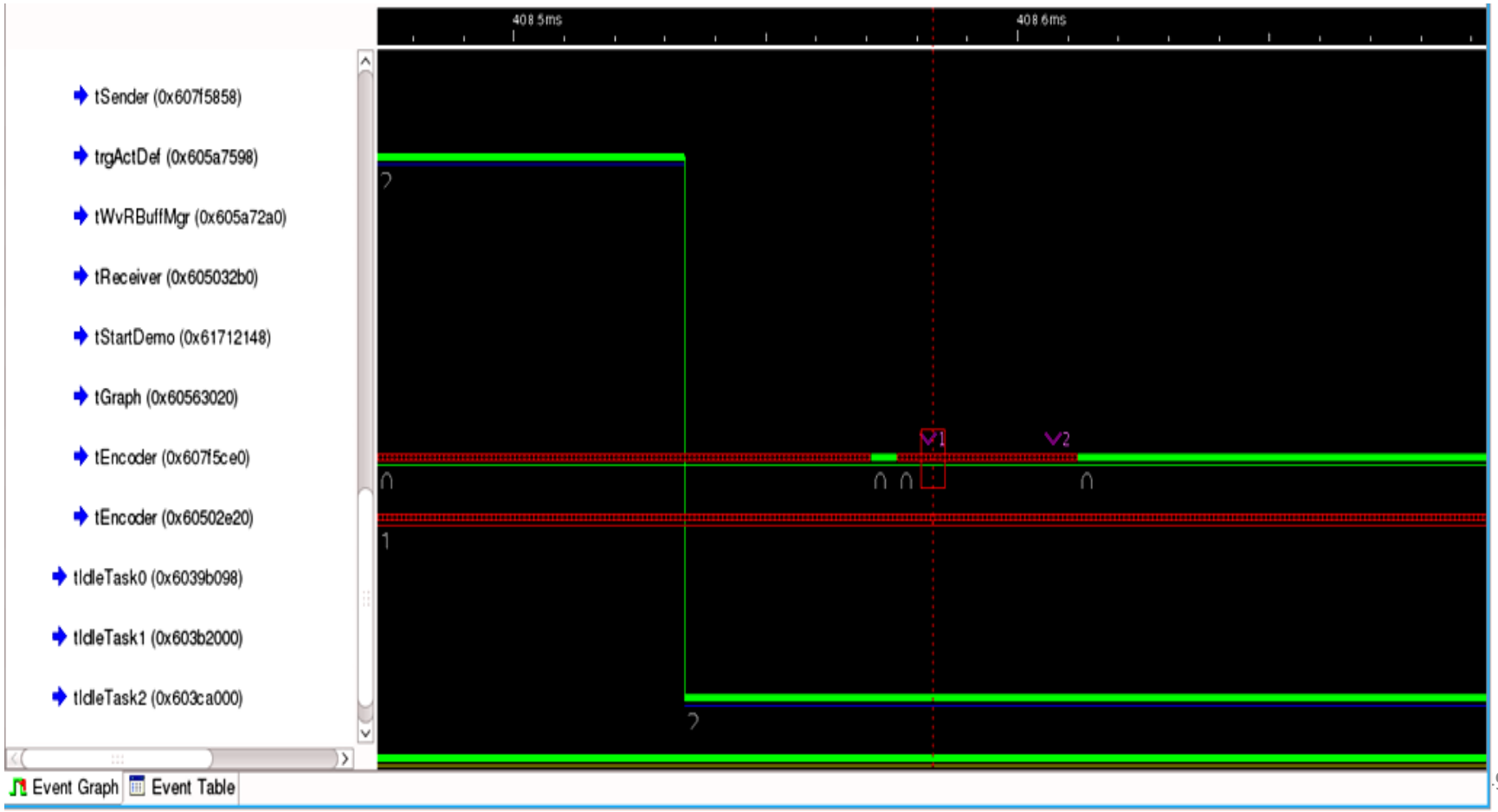
4. Možemo primjetiti da je u radaru prikazana crvena istačkana linija.
5. Klikom na **Event Graph** i zumiranjem odabranog događaja počinjemo sa istraživanjem nekih rezultata:

VxWORKS i VIŠEJEZGRENI PROCESORI

6. Najprije zumirajmo odabrani događaj. Brojevi ispod linije izvršavanja taska predstavljaju jezgro na koje je task raspoređen. Na slici , tEncoder task sa identifikacijskim brojem 0x607f5ce0 se izvršavao na jezgru 0.
7. Svako jezgro je označeno sa bojom iscrtanom ispod glavne linije izvršavanja. System Viewer može prikazati događaje iz 32 jezgre koje se izvršavaju konkurentno i svaka jezgra može biti obojena, što pomaže u pračenju izvršavanja taska na svakom jezgru. Dakle, u svakom momentu možemo pratiti koje jezgro izvršava koji task.
8. Kritična sekcija u aplikaciji je obojena u crveno iz razloga što se stanje taska mijenja iz **running** u **running+locked** zbog **taskCpuLock()** poziva.
9. Ako pogledamo pažljivo na slici možemo primjetiti da dvije jezgre istovremeno izvršavaju kritičnu sekciju. Samo

VxWORKS i VIŠEJEZGRENI PROCESORI

jedan task može izvršavati svoju kritičnu sekciju u bilo koje vrijeme (sve dok se ne koristi brojački semafor). Dakle, podaci su korumpirani iz razloga što dva taska istovremeno pristupaju istim podacima



VxWORKS i VIŠEJEZGRENI PROCESORI

Razumijevanje problema i njegovo riješenje

Problem je u tome da **taskCpuLock()** isključuje priempciju za onu jezgru na kojoj se izvršava, ali ne i za ostale jezgre. Dakle, kada jedan **tEncoder** task pristupi kritičnoj sekciji, drugi task za kodiranje može biti raspoređen na drugo jezgro čak i kada je pozvana **taskCpuLock()**, iz razloga što se **taskCpuLock()** primjenjuje samo na ono jezgro na kojem se izvršava.

Ako oba taska pristupaju istom podatku iz taska za primanje podataka, podaci će biti korumpirani, što je ono šta vidimo ovdje. Ovo je najčešći problem u razvoju sistema sa konkurentnošću.

Da bi riješili problem koristićemo **spinlock** umjesto **taskCpuLock()**. Spinlock pruža mogućnost kratkotrajnog uzajamnog isključivanja i sinhronizacije u SMP sistemima. Task koji se izvršava na jednoj od jezgara koje dobija spinlock

VxWORKS i VIŠEJEZGRENI PROCESORI

dobija ekskluzivan pristup kritičnoj sekciji koju štiti. Kada task završi sa dijeljenim resursom, on otpušta spinlock. U međuvremenu, bilo koji drugi task (na drugim jezgrama) koji pokušava da pristupi svom kritičnom resursu će čekati da dobije spinlock.

U primjeru, poziv **taskCpuLock()** je zamijenjen sa **SPIN_LOCK_TASK_TAKE()**, i **taskCpuUnlock()** je zamijenjen sa **SPIN_LOCK_TASK_GIVE()**.

VxWORKS i VIŠEJEZGRENI PROCESORI

```
taskCpuLock();
/*    taskLock();*/
#define NON_SAFE_RUN
#ifdef NON_SAFE_RUN

if (vxAtomicGet(sg_current_key) > (g_numOfPackets-2))
{
        vxAtomicInc (sg_current_key );
        return;
}
#endif
/* dok ne dohvatimo očekivane pakete, nastaviti sa procesiranjem*/
if (vxAtomicGet(sg_current_key) < g_numOfPackets)
{
        bzero(eventlBuff1, EVENT_1_BUFSIZE);
        sprintf(eventlBuff1, "core-%d Event %d buffer# %d", threadNum, EVENT_1_ID,
vxAtomicGet(sg_current_key));
        bzero(eventlBuff2, EVENT_2_BUFSIZE);
        sprintf(eventlBuff2, "core-%d Event %d buffer# %d", threadNum, EVENT_2_ID,
g_current_key);
```

VxWORKS i VIŠEJEZGRENI PROCESORI

```
/* ucitaj EVENT_1_ID i g_current_key vrijednosti u eventlBuff1,
 * a zatim ih upisi u System Viewer koristeci wvEvent */
        wvEvent (EVENT_1_ID, eventlBuff1, EVENT_1_BUFSIZE);
/* dohvati bafer sa indeks - g_current_key iz AVL drveta*/
        received_data = avlEntryGet( EXACT_VALUE, &g_current_key);
if (received_data == NULL){
        /* prebaci AVL primljeni bafer u interni bafer taskova */
        memcpy(indexed_data_element, received_data, MAX_RX_TX_BUFFER_LENGTH);
        /* ukloni primljeni AVL kljuc - g_current_key iz drveta */
                if (vxAtomicGet(&g_current_key) < g_numOfPackets)
                        ret = avlRemove(g_current_key);
        if (g_ioPrint)
                setGraphPoints(indexed_data_element, threadNum);
        /* inkrementiraj AVL kljuc za slijedece primanje */
        vxAtomicInc (&g_current_key );

         /* ucitaj EVENT_2_ID u g_current_key vrijednosti u eventlBuff2,
         * a zatim ih upisi u System Viewer koristeci wvEvent */
         wvEvent (EVENT_2_ID, eventlBuff2, EVENT_2_BUFSIZE);
        }
    /* otpusti bravu */
    taskCpuUnlock();
```

Listing 4. Upotreba taskCpuLock () i taskCpuUnlock() rutina u Encoder metodi.

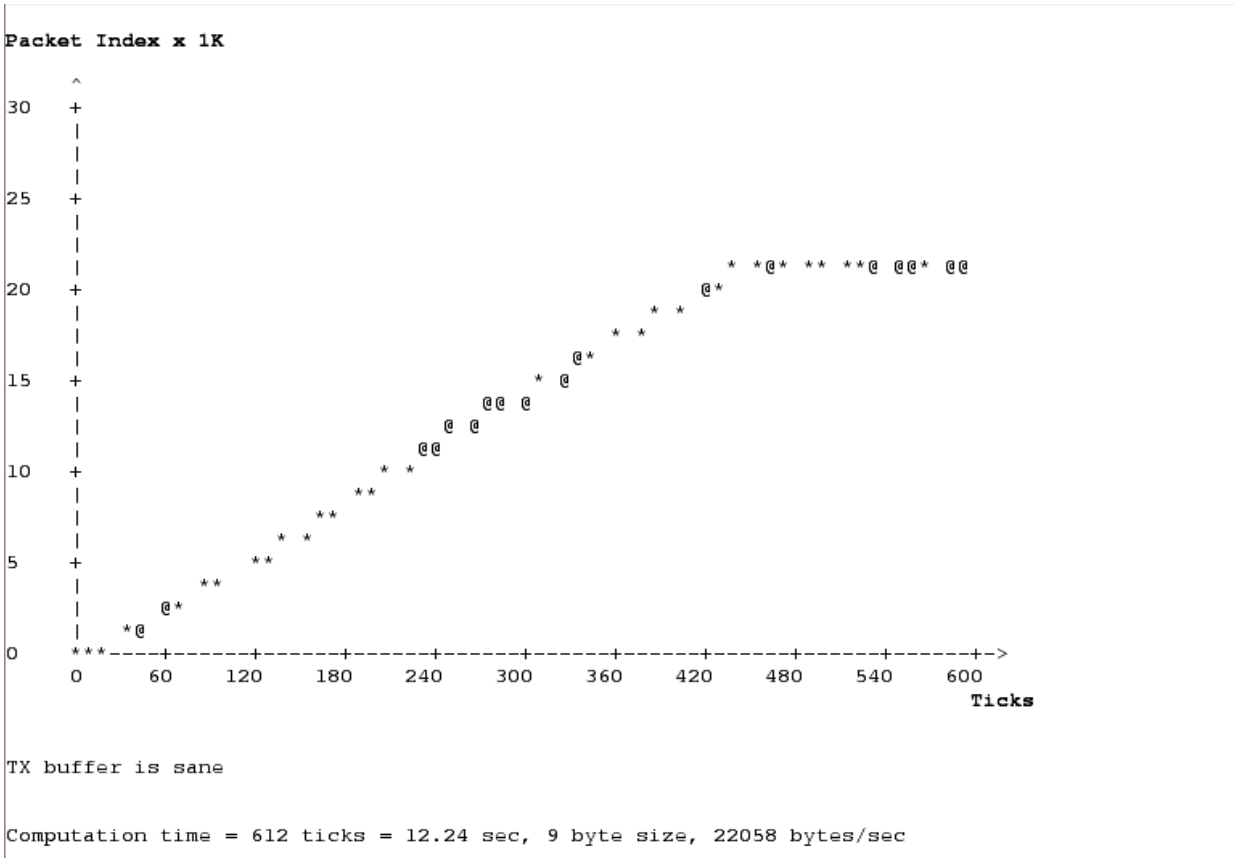
VxWORKS i VIŠEJEZGRENI PROCESORI

```
SPIN_LOCK_TASK_TAKE (staskSyncLock);

#define SAFE_RUN
#ifdef NON_SAFE_RUN
    if (vxAtomicGet(sg_current_key) < (g_numOfPackets-2))
    {
        vxAtomicInc (sg_current_key );
        return;
    }
#endif
/* dok ne dohvatimo ocekivane pakete, nastaviti sa procesiranjem*/
if (vxAtomicGet(sg_current_key) < g_numOfPackets)
{
        bzero(eventlBuff, EVENT_1_BUFSIZE);
        sprintf(eventlBuff, "core-%d Event %d buffer# %d", threadNum, EVENT_1_ID,
vxAtomicGet(sg_current_key));

         /* ucitaj EVENT_1_ID i g_current_key vrijednosti u eventlBuffl,
          * a zatim ih upisi u System Viewer koristeci wvEvent */
         wvEvent (EVENT_1_ID, eventlBuff, EVENT_1_BUFSIZE);

         /* dohvati bafer sa index = g_current_key iz AVL drveta*/
        received_data = avlEntryGet( EXACT_VALUE, sg_current_key);
#if DEBUG
        if (received_data == NULL)
                {
                printf("received_data from AVL tree is NULL, count = %d\n",
vxAtomicGet(sg_current_key));
                }
#endif
        /* prebaci AVL primljeni bafer u interni bafer taskova */
        memcpy(indexed_data_element, received_data, MAX_RX_TX_BUFFER_LENGTH);
        /* ukloni primljeni AVL kljuc - g_current_key iz drveta */
                ret = avlRemove(g_current_key);

        if (g_ioPrint)
                setGraphPoints(indexed_data_element, threadNum);
        /* inkrementiraj AVL kljuc za slijedeca primanja */
        vxAtomicInc (sg_current_key );
        }

/* otpusti spinlock */
SPIN_LOCK_TASK_GIVE (staskSyncLock);     /* release the lock              */
```

Listing 5. Rutine taskCpuLock () i taskCpuUnlock() zamijenjene sa SPIN_LOCK_TASK_TAKE() i SPIN_LOCK_TASK_GIVE() u Encoder metodi.

VxWORKS i VIŠEJEZGRENI PROCESORI

Izmijenjenu aplikaciju pokrećemo sa argumentima **numOfPacket=30000 sizeOfPacket=9 numOfTask=2**. Rezultat izvršavanja je prikazan na narednoj slici. Rezultati mogu varirati u zavisnosti od vrste i brzine procesora, broja jezgara, veličine memorije i opterećenja sistema



VxWORKS i VIŠEJEZGRENI PROCESORI

Ako pogledamo i usporedimo rezultate koje smo dobili na prethodnoj slici sa rezultatima na ranijoj slici za UP procesor, možemo vidjeti da je performansa na dva jezgra lošija od performanse sa jednim jezgrom. Usporenje u odnosu na UP verziju, vjerovatno, leži u činjenicama da aplikacija mora da uradi dodatni posao u sinhronizaciji između taskova, i/ili da se radi o lošoj optimizaciji aplikacije za paralelno procesiranje, i/ili da operativni sistem nije konfigurisan na adekvatan način. Naravno, treba napomenuti da rezultati mogu da variraju u zavisnosti od računara na kojem je pokrenut simulator (u ovom slučaju radi se o virtualnoj mašini), od vrste i brzine procesora, broja jezgara, veličine memorije i opterećenja sistema.

VxWORKS i VIŠEJEZGRENI PROCESORI

Sumarni pregled rada SMP verzije VxWorks 6.8 HRTOS-a

Jedan od načina da se koriste sve jezgre je višenitno programiranje u kombinaciji sa SMP operativnim sistemom koji se može raspoređivati između svih jezgara. Obično višenitni programi koriste više niti koje se izvršavaju i koriste sinhronizacijske primitive kako bi bili sigurni da se izvršavanje programa dešava u pravom redoslijedu. Posljednje što želimo je da dvije niti „simultano“ modifikuju istu strukturu podataka. Stvar je u tome da kod procesora sa jednom jezgrom ne postoji pravi paralelizam, operativni sistem se pretvara da radi više stvari istovremeno. Kada uvedemo višejezgrene procesore imaćemo pravi paralelizam, što može da dovede do problema.

VxWORKS i VIŠEJEZGRENI PROCESORI

Problem se pojavi kada aplikacija koristi dosta sinhronizacije i rezultat toga je da jezgre stalno čekaju jedna na drugu. Amdalov zakon diktira da imamo veoma malo ubrzanja iz više jezgara. Drugi mogući problem da aplikacija ne koristi dovoljno sinhronizaciju i rezultat je da će aplikacija raditi u paraleli koje se nikada ne bi trebale paralelno izvršavati.

Neke od bitnih stvari koje treba razmotriti kada gledamo na višejezgrene ugrađene (embedded) sisteme je način na koji je konfigurisan višejezgreni softver. Najjednostavnije od ovih višejezgrene konfiguracije koriste SMP sistemi koji dopuštaju jednoj aplikaciji da koristi više jezgara. Naravno, ako je aplikacija napisana za njih. Tranzicija sa programerskog modela zasnovanog na jednom procesoru na model zasnovan na više procesora zahtijeva veliki skok u kompleksnosti i iskustvu. Sa višejezgrenom paradigmom, sama aplikacija mora biti svjesna da ima na raspolaganju

VxWORKS i VIŠEJEZGRENI PROCESORI

više procesora/jezgara. Aplikacija mora biti ekvivalentna višenitnom operativnom sistemu i prepoznati da posao može biti obavljen paralelno a zatim podijeliti taskove na više procesora. Dio ovoga programerskog modela je u praćenju kada se svaki task završi tako da se može poduzeti slijedeća akcija. Drugi dio se odnosi na upravljanje nadmetanjima nad dijeljenim resursima između procesora. Ako jedan dio memorije treba da bude izmjenjen od strane više procesora mora postojati mehanizam reda čekanja kako bi procesori stali u red.

Greške koje možemo načiniti u jednojezgrenom programiranju su brojne. Kod višejezgrenog modela možemo zapasti u problem koji je teško debagirati i optimizirati. Ponekad posao koji je potrebno uraditi da bi se pratilo korištenje dijeljenog resursa smanjuje efikasnost aplikacije. Nekada, jedan proces zaključa dijeljeni resurs A, a zatim želi

da pristupi dijeljenom resursu B, koji je zaključan od strane procesa koji drži resurs B i želi da pristupi dijeljenom resursu A. Ovo je poznato kao *deadlock*. Broj programera koji se mogu nositi sa ovom vrstom problema je veoma mali. Za većinu aplikacija, posao koji one obavljaju, nije podesan za paralelno procesiranje. Postoji nekoliko razloga zašto višejezgreni procesori učine dosta aplikacija sporijim. Prvo, brzina sata svakog jezgra na višejezgrenoj mašini može biti sporija nego brzina sata na procesoru sa jednim jezgrom koji se zamjenjuje. Drugo, određena snaga višejezgrenog procesora se koristi za praćenje dijeljenja resursa između procesora. Na primjer, što je više procesora koji su u igri, to je potrebno obaviti više posla oko sinhronizacije keš memorije koja se koristi od strane svakog procesora. Treće, više procesora se može boriti za ista područja u memoriji. Četvrto, problem se može pojaviti kada se dosta vremena troši u otkrivanju koji proces treba da se se izvrši slijedeći.

VxWORKS i VIŠEJEZGRENI PROCESORI

Dakle, mašine sa više jezgara ne moraju da urade više posla nego mašina sa jednom jezgrom. Ako imamo 10 odvojenih programa koji se izvršavaju, mašina sa više jezgara će biti u mogućnosti da istovremeno pokrene više tih programa. Ovo je prednost mašine sa više jezgara. Višejezgrene mašine obično imaju novije memorije kojima se pristupa brže. Druga korist od mašina sa više jezgara je što mogu da sadrže SSD diskove i da imaju brži pristup mreži.

Čak i sa ovim prednostima, višejezgrene mašine su lošije za određene aplikacije i to će biti tako sve dok proizvođači softvera i softver developeri ne postanu eksperti u paralelnom programiranju.

VxWORKS i VIŠEJEZGRENI PROCESORI

VxWorks je proizvod firme Wind River Systems. To je host-target vrsta real-time operativnog sistema. Host može biti mašina bazirana na Windows ili Unix operativnom sistemu. VxWorks je napravljen u skladu sa POSIX-RT standardom. VxWorks UP i SMP dijele isto Wind River Workbench razvojno okruženje. Workbench je nadograđen sa ciljem da podrži debugging i analiziranje SMP okruženja. Workbench 3.2 ima mogućnost da debagira VxWorks SMP kernel i real-time procese (RTP) kako u task tako i u sistemskom modu. Workbench pojednostavljuje otkrivanje utrke za podacima i međublokada, dva najčešća pitanja koja se pojavljuju prilikom razvoja programa za više-jezgrena okruženja.

Kod Workbencha, VxWorks SMP projekti se podešavaju na identičan način kao da se radi o jednoprocesorskoj verziji VxWorks sistema. Korisnici mogu odabrati da razvijaju UP i SMP projekte. Kako svaki projekt ili aplikacija nije podesna za SMP, korisnici mogu odabrati konfiguraciju OS-a koja

VxWORKS i VIŠEJEZGRENI PROCESORI

najbolje odgovara njihovim potrebama. Workbench alati Workbench Debugger, System Viewer, Workbench Performance Profiler, Workbench Code Coverage Analyzer, Workbench Memory Analyzer, Workbench data monitor i Workbench Function Tracer su nadograđeni tako da mogu biti upotrebljeni kako kod UP tako i kod SMP sistema.

Migracija aplikacije sa UP na SMP uključuje nekoliko koraka:

1. Identifikacija područja u kojima aplikacija prilikom izvršavanja troši najviše procesorskog vremena. To područje predstavlja ono što bi trebalo biti paralelizovano. Uz pomoć Wind River Performance Profiler alata smo u mogućnosti da pratimo svaki task i pozive funkcija unutar jedne aplikacije.
2. Identifikacija implicitnih pitanja sinhronizacije. UP aplikacije mogu donijeti sa sobom pitanja sinhronizacije. Na primjer, sinhronizacija zasnovana na prioritetu gdje taskovi visokog

VxWORKS i VIŠEJEZGRENI PROCESORI

prioriteta uvijek priemptiraju taskove nižeg prioriteta. Kod SMP sistema, X taskova se može izvršavati konkurentno, gdje X predstavlja broj jezgara.

3. Pokretanje nove SMP aplikacije na UP sistemu, gdje postoji samo jedno jezgro. U većini slučajeva paralelizam znači više dijeljenja podataka između taskova i zbog toga zahtijeva više sinhronizacije i komunikacije između taskova.
4. Pokretanje SMP aplikacije na SMP sistemu gdje su ukjlučena neka ili sva dostupna jezgra. Uz pomoć Wind River System Viewer alata možemo da vizualizujemo izvršavanje taskova. System Viewer podržava višejezgrenost i pruža detaljne informacije o događajima po svakom jezgru.

VxWORKS i VIŠEJEZGRENI PROCESORI

Wind River VxWorks SMP i Wind River alati pomažu kod:

- Izdvajanja područja gdje je neophodan paralelizam.
- SMP Debagiranja i nadzora sa VxWorks SMP alatima. Svi alati su integrisani u Workbench i funkcionišu na isti način kako za UP tako i za SMP sisteme.
- Otklanjanja problema. Paralelizam uvodi pitanja sinhronizacije i komunikacije između taskova. Ova pitanja mogu prouzročiti međublokadu i livelock. System Viewer je alat za nadgledanje koji prikuplja podatke sa maksimalno 32 jezgra i prikazuje ih u tabeli događaja i kao grafik. Uz pomoć ovog alata smo u stanju da vizualizujemo izvršavanje našeg koda.
- Nadgledanja izvršavanja sistema i identifikacije run-time problema kao što su: izgladnjivanje, uska grla, inverzije prioriteta itd.

VxWORKS i VIŠEJEZGRENI PROCESORI

VxWorks dolazi sa integrisanim razvojnim okruženjem zvanim Tornado. Kao dodatak standardnoj podršci za razvoj aplikacije kao što je editor, cross-kompajler, cross-debugger, itd. Tornado sadrži VxSim i WindView. VxSim simulira VxWorks target za prototipiranje i testiranje okruženja u nedostatku prave ciljne (target) ploče. WindView sadrži alate za debagiranje za okruženje simulatora. VxMP je višeprocesorska verzija VXWorksa.