ELEKTRONIK TIDNINGEN

Jakob Engblom
Teknisk marknadsförare för Simics Wind RIver

Analys av analysmaskinen Analyzer

Version 4.4 av systemsimulatorn Simics har blivit bättre på att presentera programsimuleringar på en hög beskrivningsnivå, i termer av trådar och processer. Den har en högre så kallad ”operativsystemsmedvetenhet” vilket bland annat underlättar felsökning i system med multipla operativsystem. Jakob Engblom berättar hur Simics Analyzer fungerar och används.

Redaktör
Jan Tångring
jan@etn.se
0734-17 13 09

EMBEDDED EXPERT

Analys av analysmaskinen Analyzer

Så fungerar Simics Analyzer och dess tidslinjevy.

Av Jakob Engblom, Wind River

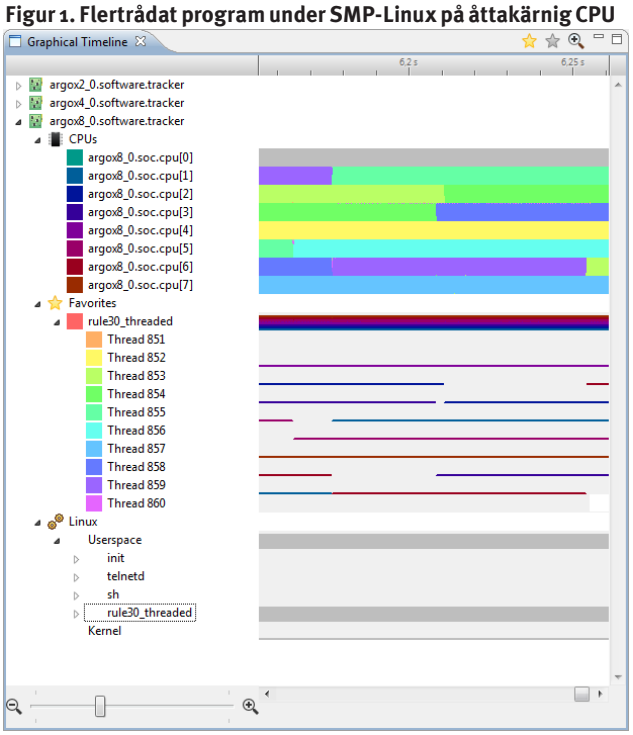
Jakob Engblom har jobbat med Simics sedan 2002 och kom till Wind River efter uppköpet av Virtutech i början av 2010.

Den stora nyheten i Wind River Simics 4.4 är **Simics Analyzer**, en uppsättning verktyg för att få insikt i hur mjukvaran i ett system beter sig.

Analyzer innehåller flera olika tekniker, men det mest uppenbara är den nya tidslinjevyn. Tidslinjen visar hur programvaran på målsystemet arbetar över en längre eller kortare tid, vilket gör att det är lätt att se vad som har körts, när och var.

Vi börjar med ett enkelt exempel på tidslinjevyn. **Figur 1** visar hur ett flertrådat program kör på en maskin med åtta processorkärnor (ovanpå en SMP Linux som använder alla åtta kärnorna). Vi kan se att vi inte riktigt lyckats lasta alla kärnorna, eftersom kärna noll mestadels lämnas oanvänd (troligen på grund av att operativsystemet vill ha den för sig själv, enligt vad vi har lyckats se). Vi ser också att operativsystemets schemaläggare flyttar en del trådar mellan kärnorna, trots att det skulle verka vettigare att köra en tråd på varje kärna under hela programmets körtid.

Figur 1. Flertrådat program under SMP-Linux på åttakärnig CPU

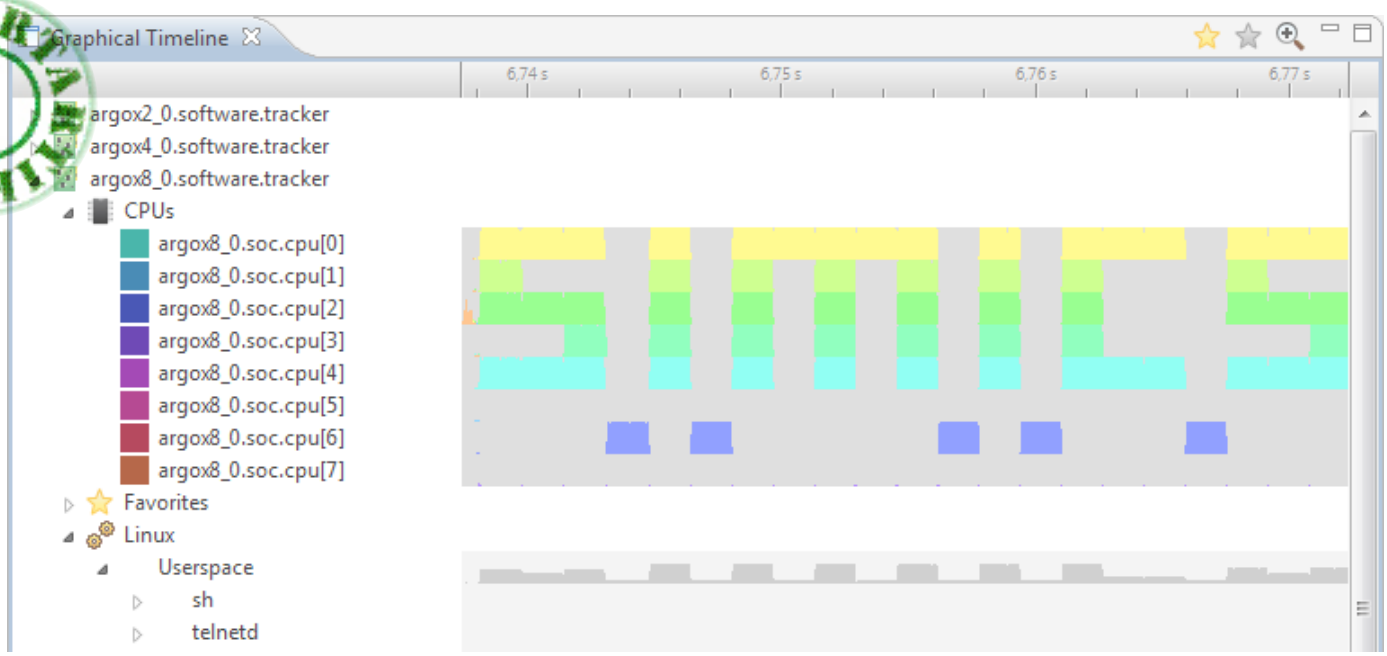


Denna typ av beteenden är väldigt svåra att upptäcka genom att bara köra programmet utan några inspektionsverktyg, eller genom att titta på processorbelastningen på systemet som helhet. Man vill se precis vad som körs var och när – det är nyckeln till att förstå parallella system.

Analyzer tillhandahåller två olika vyer: tidslinjen (**figur 1**) och en systeminformationsvy. Systeminformationen visar systemets hårdvara och mjukvara i ett visst ögonblick och kompletterar den historiska tidslinjen. Den ger även en inblick i målsystemets hårdvaru- och mjukvarukonfiguration.

Hur fungerar det?

En virtuell plattform som Simics kan i princip iaktta allt som målsystemet gör (med obegränsad bandbredd att plocka ut information med och utan probeffekter). De rådata som erbjuds är dock ganska primitiva, eftersom allt Simics ser är maskininstruktioner, minnesaccesser, aktiviteter i periferienheter, avbrott, ändringar i processortillstånd och andra händelser på hårdvarunivå. Det är givetvis mycket svårt att få ut något vettigt av sådan data i en modern maskin med många processorkärnor som går i många gigahertz. För att få fram användbar information från rådata måste vi återskapa de programvaruabstraktioner som används i systemet, typiskt operativsystemets kärna, dess användarprocesser och deras trådar. Detta brukar kallas **OS-medvetenhet**, eller operativsys-

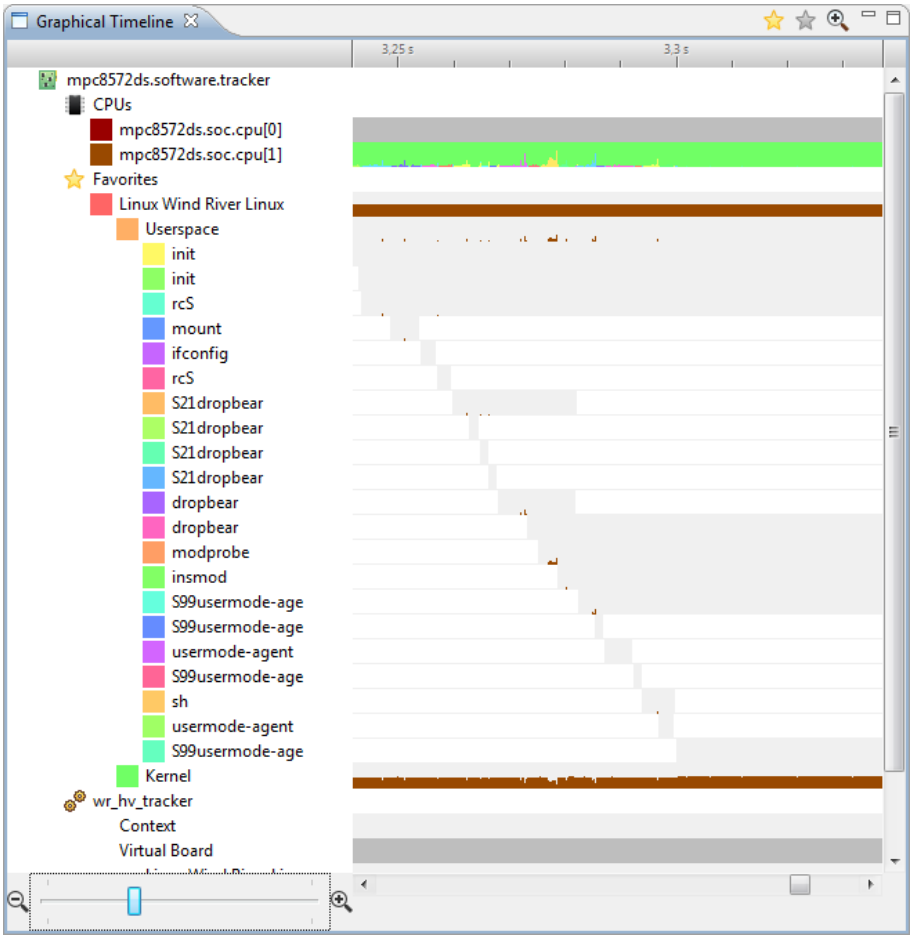


Här har Jakob Engblom roat sig och schemalagt ett program så att lasten på olika CPUer bildar namnet på verktyget.

temsmedvetenhet på svenska. Simics 4.4 innehåller ett nytt och kraftigt förbättrat system för OS-medvetenhet.

Simics kan nu hålla reda på flera operativsystem samtidigt på samma målmaskin, programvarustrukturer i flera lager inklusive hypervisor, och skapande och avslutande av både processer och trådar. Simics upptäcker när trådar schemaläggs och när operativsystemet tar kontroll efter ett avbrott eller systemanrop. OS-medvetenhet-informationen är tillgänglig för samtliga moduler som kör i en Simicssimulering, inklusive skript och program skrivna av användare och specialfunktioner som tidslinjevyn. I Simics-skript kan man använda OS-medvetenhet för att göra olika saker när trådar och processer startar och stoppar, byter kärna eller anropar operativsystemet.

Figur 2. Linux bootad under en hypervisor på en dubbelkärna



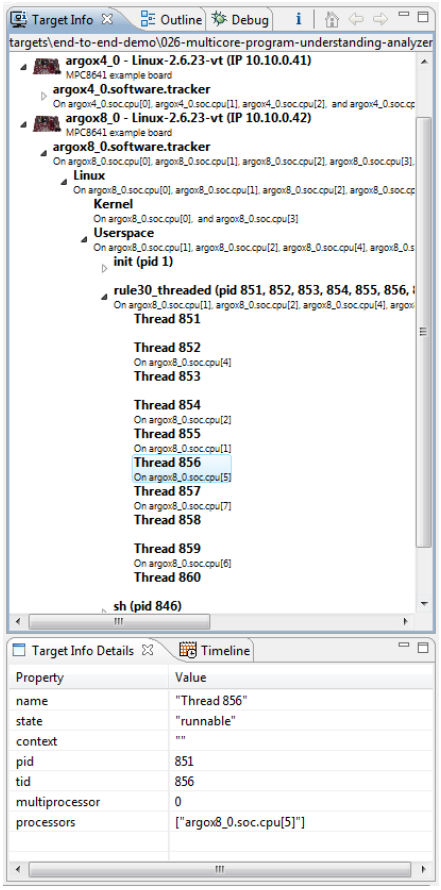
OS-medvetenhet är mycket praktiskt vid debuggning, eftersom man då vet vilken tråd som körs när ett fel inträffar. Man kan också be simulatorn köra tills en viss tråd blir aktiv.

I kombination med en källkodsdebugger får man en lösning som kan knyta händelser på hårdvarunivå (en viss minnesskrivning, till exempel) till en viss kodrad i ett visst program som körs i en viss tråd på en specifik kärna.

Att implementera OS-medvetenhet är inte helt enkelt. Det man måste göra är att läsa innehållet i processorregister och minnet, och hitta startpunkten (eller punkterna i de flesta flerkärniga system) för operativsystemets lista av processer. Sedan kan man följa en eller flera länkade listor runt i minnet för att hitta alla aktiva processer och trådar. Dessutom reagerar OS-medvetenhet på övergångar mellan användarnivå och operativsystemsnivå, så att man kan upptäcka när operativsystemet tar över.

Detta är egentligen inte unikt för Simics — verktyg som Wind Rivers hårdvaruprober erbjuder också OS-medvetenhet implementerat på liknande sätt och med samma resultat. Det är värt att påpeka att OS-medvetenheten i Simics inte stör målsystemet och att alla minnesläsningar går via en ”bakdörr” i simulatorn och inte syns eller tar tid från den simulerade systemet.

OS-medvetenhet måste känna till layouten för olika datastrukturer i operativsystemskärnan, och detta kräver ofta tillgång till debuginformation eller symboler för operativsystemkärnan. Specifikt för Linux finns även en heuristisk modul som oftast kan lista ut parameterarna, vilket möjliggör användning av OS-medvetenhet på en kärna som man bara har i binär form. När parame-



Figur 3. Aktiva trådar ett visst ögonblick

trarna för ett OS en gång har bestämts kan de sparas på en fil. Detta innebär att grupper som tar fram OS-plattformar kan tillhandahålla OS-medvetenhet till sina användare tillsammans med binära mjukvarumoduler, utan att behöva skicka med debuginformation eller symboltabeller för OS-kärnan.

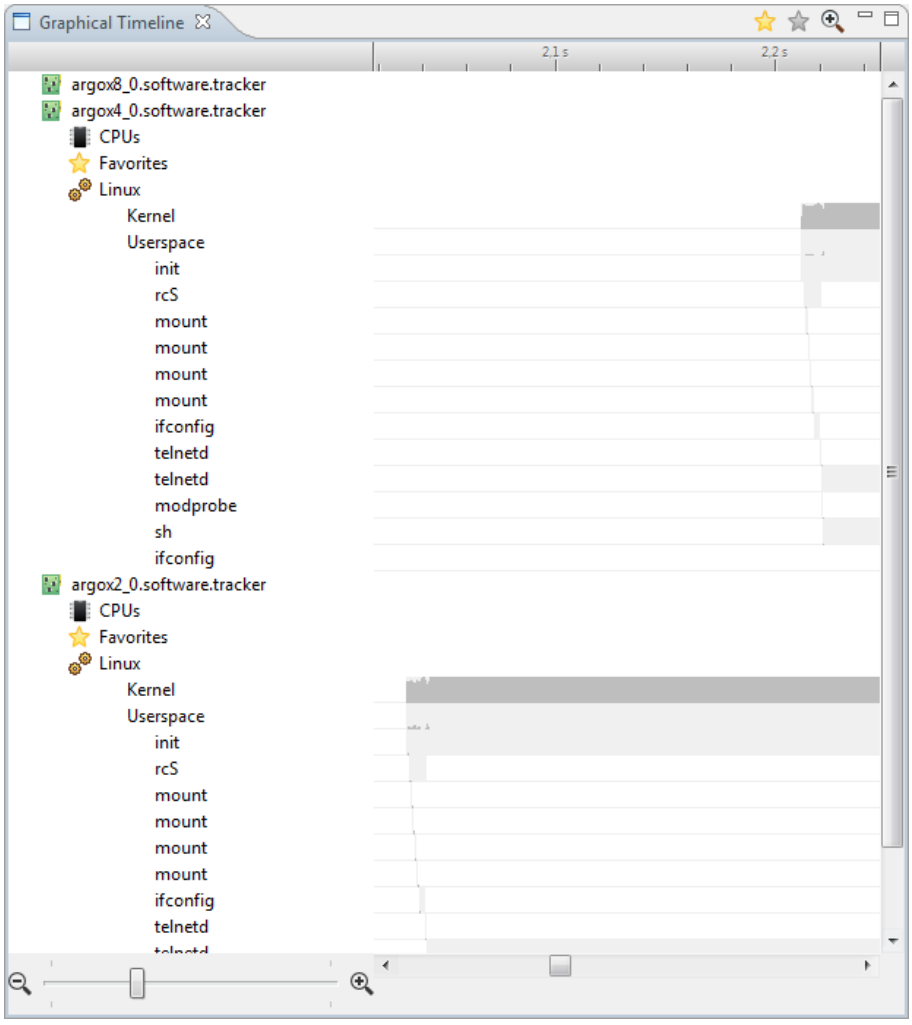
Några exempel

Figur 2 visar uppstarten hos ett Wind River Linux-system som körs under en hypervisor på ett målsystem med två processorkärnor. Vi ser de olika processer som ingår i systemets uppstart. Observera att användarprocesserna inte tar särskilt mycket processortid – för det mesta är det operativsystemkärnan som kör. Vi ser också att hypervisorn håller operativsystemkärnan på en enda processorkärna, vilket lämnar den andra processorkärnan fri att utföra annat arbete (normalt genom att köra ett annat OS som VxWorks).

Figur 3 visar systeminformationsvyn för ett specifikt ögonblick under körningen i den första tidslinjen i denna artikel. Vi ser att bara vissa trådar är aktiva just i detta ögonblick (de aktiva trådarna visar namnet på den processorkärna som de är schemalagda på). Observera att vi ser två maskiner på bilden, eftersom systemet egentligen är ett nätverk av maskiner där vi just nu bara är intresserade av den åttakärniga maskinen. Detta lyft över nivån av enskilda maskiner och enskilda operativsystem är en viktig poäng med såväl Simics och Simics Analyzer.

Det sista exemplet (**figur 4**) går tillbaka till Linuxbooten, denna gång tittar vi på olika maskiner i samma virtuella nätverk. Samtliga maskiner kör med samma operativsystemkärna och samma filsystem, men med olika antal processorkärnor. Observera att booten av en maskin med fyra processorkärnor går långsammare än för maskinen med två processorkärnor (runt 0,2 sekunder långsammare). Uppenbarligen tar Linuxkärnan längre tid att starta med fler processorkärnor i systemet. Det är också tydligt att initieringen av systemet är en sekventiell process även i en maskin med flera processorkärnor (systemet använder ett vanligt initskript utan några optimeringar).

Figur 4. En fyrkärna och en dubbelkärna bootar Linux



Ständiga förbättringar

Det vi visar här är bara en liten början av vad man kan göra med visualiseringar från de data som är tillgängliga i Simics. Tidslinjevyn som den ser ut just nu har tagits fram under de senaste kvartalen, och vi arbetar kontinuerligt med förbättringar. Detaljer i utseendet och användargränssnittet förändras fortfarande, i takt med att vi använder tidslinjen och får feedback från tidiga användare. Ett exempel på sådana ändringar är att bara trådar som ligger under ”favorites” färgas in. Tidigare, när samtliga processer och trådar i systemet färgsattes, fick vi en virtuell färgsprakande godishög som visserligen såg snygg ut men som var helt oanvändbar, eftersom färgerna återanvänds när målsystemet fick mer än några tiotal trådar. ■

Om Wind River

Wind River saluför, utvecklar och säljer tjänster kring verktyg, system och mjukvara för inbyggda system med bland annat mobila system som ett viktigt tillämpningsområde. Företaget har varit verksamt sedan 1981 och dess teknik finns idag i över 500 miljoner produkter.

Huvudkontoret ligger i Alameda i Kalifornien, USA och företaget har kontor i ytterligare 15 städer.

I februari 2010 köpte Wind River svenska Virtutech och dess systemsimulator Simics.

För mer information, besök Wind River på www.windriver.com eller www.blogs.windriver.com