Mjukvarutrace synliggör buggar i Linuxsystem

Verktyget ger dig drygt 20 grafiska perspektiv på vad som pågår i ditt multi-trådade program

Av Johan Kraft, Percepio AB

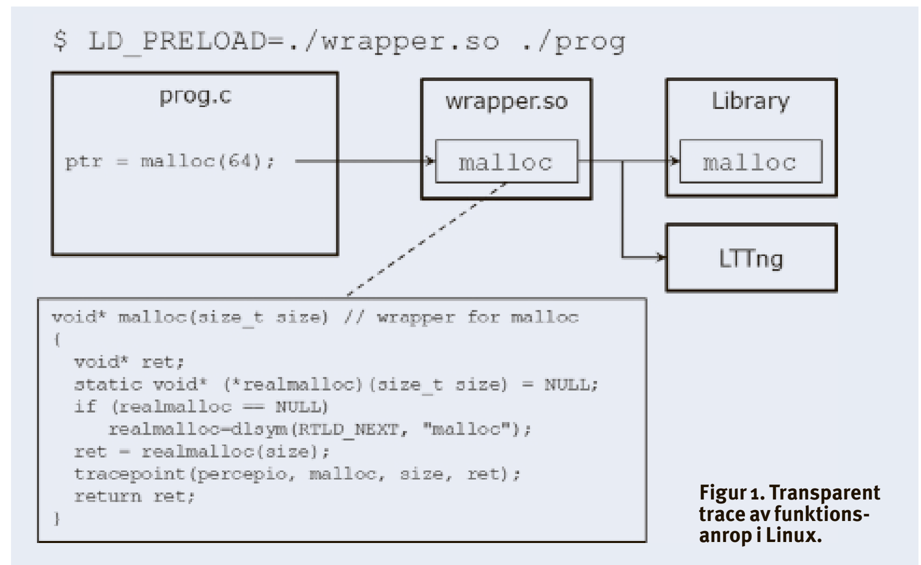
Doktor **Johan Kraft** har disputerat i Datavetenskap och utvecklade den första prototypen av traceverktyget Tracealyzer för ganska exakt 10 år sedan under sitt doktorandprojekt, i samarbete med ABB Robotics. Han grundade Percepio år 2009 och är idag vd för bolaget, som nyligen tog in expansionskapital från bland andra ALMI Invest, Stockholms Affärsänglar och Tomas Wolf, som tidigare var VD för IAR Systems.

Felsökning av programvara för inbyggda system är ofta svårt, tidskrävande och en avsevärd riskfaktor i utvecklingsprojekt, då svåranalyserade fel kan visa sig i sena skeden och orsaka förseningar. När ett fel visar sig är det första steget typiskt att studera symptomen i en debugger. Nästa fråga är ofta ”Hur kunde mjukvaran hamna i detta tillstånd?” Det vill säga, vilken input och timing orsakade felet, och varför?

Svaret kan ofta fås genom inspelning av mjukvarans beteende under drift, på engelska kallat *tracing*.

Traceinspelning är typiskt en aktivitet för utvecklingslabbet men kan också ske under skarp drift för att analysera fel som är svåra att återskapa i en debugger, till exempel sällsynta fel som visar sig under vissa omständigheter, till synes slumpmässigt.

Traceinspelningen kan ske i hårdvaran (i processorn) eller i mjukvaran. Hårdvarubaserad inspelning har fördelen att ge en detaljerad bild av programvarans exekvering utan att påverka systemet som analyseras. Nackdelar är att det kräver speciell traceutrustning och att processor och kort måste stödja hårdvarutrace, vilket snabbare processorer ofta inte gör. Dessutom kan man oftast inte spela in data, till exempel argument i funktionsanrop.



Figur 1. Transparent trace av funktionsanrop i Linux.

MJUKVARUBASERAD INSPELNING fokuserar på utvalda viktiga händelser som till exempel operativsystemsanrop, avbrottsrutiner och uppdateringar av viktiga variabler. Detta har fördelen att inte kräva någon speciell hårdvara och kan därför även inkluderas i produktionskoden, likt en mjukvaruversion av de ”svarta lådor” som används i flygplan. Till skillnad från hårdvarubaserad inspelning så kan man enkelt inkludera relevanta data, till exempel argument i funktionsanrop. Nackdelen med mjukvarubaserad inspelning är att målsystemets processor och arbetsminne används för inspelningen. På 32-bitarsprocessorer tar detta dock bara någon mikrosekund per händelse, vilket typiskt innebär att systemet exekverar med cirka 99 procent av normal hastighet. När mjukvarubaserad inspelning aktiveras uppstår en ”probe-effekt” genom att timing påverkas något av inspelningen. Det är oftast inget problem, men kan elimineras genom att man låter inspelningen vara aktiv som standard, det vill säga även i produktionskoden. Då har man dessutom alltid inspelningar tillgängliga om något problem skulle visa sig. Den processortid som åtgår till inspelningen kan tjänas in på att möjligheterna att optimera programvaran blir bättre.

Trace är extra relevant när operativsystemet används för multitrådning, som innebär att flera program (trådar) körs på samma processorkärna genom att processorn snabbt skiftar mellan dem. Multitrådning

är mycket praktiskt för inbyggda system där man har flera periodiska aktiviteter som behöver köra med olika frekvens, till exempel i reglersystem, eller när tidskritiska funktioner behöver köras vid vissa händelser och då behöver avbryta mindre tidskritiska funktioner. En multitrådad applikation innebär dock högre komplexitet, och exekveringsordningen blir mindre tydlig då den inte framgår direkt av koden utan beror av operativsystemets dynamiska schemaläggning av trådarna.

LTTNG* ÄR DEN LEDANDE lösningen för mjukvarubaserad traceinspelning i Linux. LTTng har öppen källkod och stöds av de flesta Linuxdistributioner samt av Yocto. Det är mycket beprövat och effektivt. LTTng erbjuder en kernelinspelare samt en applikationsinspelare (LTTng-UST, User-Space Tracer). Kernelinspelaren spelar in exekveringen av trådar och avbrottsrutiner, systemanrop, minneshantering och många andra typer av kernelhändelser. Med applikationsinspelaren kan man spela in egna händelser genom att lägga in så kallade tracepoints i koden, det vill säga explicita anrop till LTTng-UST från applikationen, till exempel allmänna felsökningsmeddelanden eller specifika funktionsanrop. Om LTTng inte är aktiverat så är prestandaeffekten av tracepoints minimal; endast någon enstaka klockcykel.

Traceinspelningen sparas initialt i en buffert i arbetsminnet, men LTTng kan instrueras att regelbundet tömma bufferten till ett filsystem eller nätverksanslutning. Tömningen sker i en user-space-tråd, som enkelt kan studeras med Percepios mjukvarubaserade traceverktyg Tracealyzer. Det är också möjligt att enbart använda arbetsminne i form av en ringbuffert där äldre data skrivs över vartefter ny data tillkommer. Vid behov sparas då ett ”snapshot” som innehåller den senaste historiken.

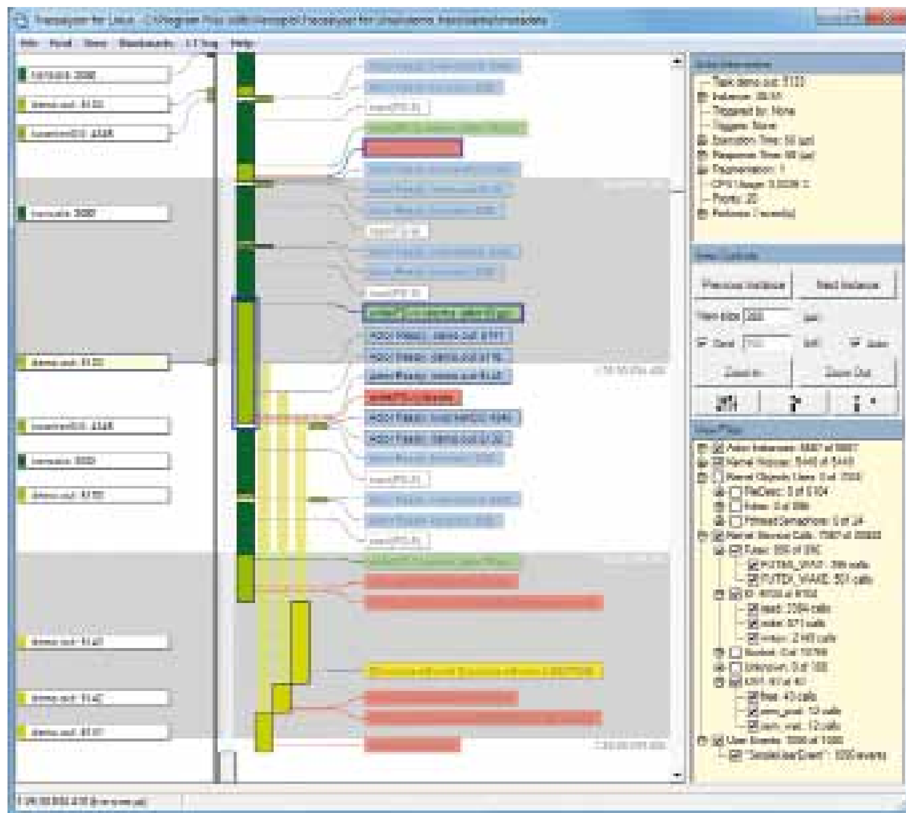
Trots att LTTng bygger på mjukvarubaserad inspelning så krävs ingen modifiering av befintlig källkod, i alla fall inte om man använder en Linuxkärna av version 2.6.38 eller senare. Man behöver inte ens kompilera om kärnan, då den redan innehåller tracepoints som kernelinspelaren utnyttjar. LTTng v2.x fungerar även på äldre kärnor tillbaka till v2.6.32, men fram till v2.6.37 behövs några patchar av kärnan.

Med applikationsinspelaren (LTTng-UST) kan användaren lägga in egna händelser (tracepoints) i applikationkod eller bibliotek. Detta kan göras utan att man behöver modifiera den befintliga koden, med hjälp av miljövariabeln LD_PRELOAD och ”wrapper”-funktioner (se figur 1). Wrapperfunktionerna kompileras som ett delat bibliotek (wrapper.so) och när detta anges i LD_PRELOAD så används wrapperfunktionerna istället för de ordinarie funktionerna med samma namn. Wrapperfunktionerna spelar in anropet med hjälp av en LTTng-UST-tracepoint och anropar sedan den ordinarie funktionen. Notera att detta kräver en dynamisk uppslagning av adressen för den ordinarie funktionen vid första anropet.

Genom kombinationen av LTTng-UST, wrapperfunktioner och LD_PRELOAD så blir inspelningen helt transparent och kräver inte ens att den befintliga koden kompileras om.

LTTNG SPARAR INSPELNINGARNA i ett öppet binärt format (CTF, Common Trace Format). LTTng-verktyget Babeltrace kan sedan översätta inspelningarna till textformat. *Men inspelningsdata är ofta mycket omfattande och svåra att överblicka i textformat.* Ett visuellt analysverktyg är till stor hjälp eftersom den mänskliga hjärnan är mycket bättre på att hitta mönster i visuella bilder än i text. Nyttan av trace är ju inte själva ▶

* LTTng-projektet: http://lttng.org/



Figur 2. Huvudfönstret i Tracealyzer – trådar och olika händelser.

inspelningen av data. Nyttan uppstår först när utvecklaren får en bättre förståelse av sitt problem och därigenom kan hitta en lämplig lösning.

Percepio AB utvecklar *Tracealyzer*, ett tracevisualiseringsverktyg som erbjuder ett stort antal grafiska vyer med olika perspektiv som underlättar analys. Tracealyzer finns för flera operativsystem, inklusive Linux, VxWorks, FreeRTOS, SafeRTOS, Micrium µC/OS och RTXC Quadros. *Tracealyzer for Linux* är avsett för LTTng-inspelningar och stödjer både den nya generationen LTTng (v2.x) och den äldre generationen LTTng (v0.x) som används i till exempel Wind River Linux 5.

HUVUDFÖNSTRET I TRACEALYZER (figur 2) visar exekveringen av trådar längs en vertikal tidslinje, och andra händelser (till exempel operativsystemsanrop) med hjälp av horisontella etiketter. Etiketternas bakgrundsfärg visar händelsens typ. Till exempel visar röda etiketter blockerade systemanrop och gröna etiketter var ett blockerande systemanrop slutar och tråden därmed fortsätter exekvera. Händelser från applikationen (via LTTng-UST) kan visas på två sätt, antingen som systemanrop (till exempel malloc) eller som ”user events”, det vill säga allmänna meddelanden (gula etiketter).

Visualiseringen i Tracealyzer är intelligent på flera sätt. Till exempel markeras relaterade händelser. Detta gör det enklare att förstå operativsystemets beteende, till exempel varför trådar aktiveras eller blir blockerade. Ett exempel kan ses i figur 2, där två relaterade händelser i en blockerande write-operation visas med blå markeringar på de röda och gröna etiketterna. Detta visar att tråden ”demo.out: 5133” blockerades på grund en skrivning till Standard Output (”write(FD-1) blocks”). Tråden aktiverades nästan omedelbart igen (”Actor Ready: demo.out: 5133”) men det tog relativt lång tid innan tråden fortsatte exekvera (”write(FD-1) returns after 69 µs”).

HUVUDFÖNSTRET KOMPLETTERAS av mer än 20 andra grafiska vyer som visar CPU-användning, statistik på exekveringstider och responstider, blockering i kärnan, intensitet i schemaläggningen, kommunikation mellan olika trådar, med mera. En inspelning innehåller ofta en stor mängd data om mindre intressanta repetitiva förlopp, men de olika vyerna i Tracealyzer gör det enklare att hitta de intressanta delarna, till exempel där ett systemanrop misslyckas eller någon tråd exekverar längre än vanligt.

Applikationshändelser visas som gula etiketter i huvudfönstret men kan även visas i en separat textlogg vilket ger en bra översikt över applikationens övergripande beteende, till exempel ändringar av viktiga tillståndsvariabler. Om numerisk data inkluderas i sådan applikationsloggning, till exempel signaler i en regleralgoritm, så kan Tracealyzer visa dem i grafiska interaktiva diagram. Detta kan liknas vid en logikanalysator i mjukvara, som kan vara till stor nytta inom de flesta typer av utveckling.

De flesta vyer i Tracealyzer är ihopkopplade, vilket gör det enklare att växla mellan flera vyer när man studerar en specifik del av inspelningen. När man till exempel studerar ett diagram med applikationsdata så kan man dubbelklicka på en intressant datapunkt för att fokusera det huvudfönstret och studera vilka trådar och avbrott som är aktiva vid den tidpunkten.

I EN DEL LINUXSYSTEM används fasta prioritetsnivåer för tidskritiska trådar. Om sådana prioriteter sätts fel så kan prestanda och responsivitet bli lidande, och i värsta fall uppstår låsningar. Om en högprioriterad tråd tar för mycket CPU-tid så syns det tydligt i CPU-belastningsgrafen, som visar den exakta mängden CPU-tid varje tråd använder över tiden. Tracealyzers statistikrapport ger dessutom en bra översikt, som kan vara ett lämpligt underlag för att studera och optimera prioritetsnivåerna.

Tracealyzer har flera funktioner för att plotta tidsegenskaper hos trådar, till exempel trådarnas periodicitet. För periodiska aktiviteter syns där störningar i periodtid (det vill säga jitter) mycket tydligt och man kan snabbt analysera detta genom att dubbelklicka på datapunkten i fråga. Då visas den aktuella tidpunkten i huvudfönstret och man ser vilken tråd eller avbrottsrutin som orsakade fördröjningen. Om periodiska trådar ofta krockar (startar samtidigt) så framgår det tydligt i flera vyer, till exempel i plottningen av ”response interference”, som visar trådarnas responstider i förhållande till deras faktiska CPU-utnyttjande. Med informationen från Tracealyzer kan man hitta alternativa lösningar som ger bättre responsivitet, till exempel genom att förskjuta någon periodisk aktivitet i tiden.

Visualiseringen i Tracealyzer är utvecklad i .NET, ursprungligen för Windows. Från och med version 2.7 kan Tracealyzer köras under Linux med hjälp av Mono**, en alternativ öppen implementation av .NET som finns för flera operativsystem.

SAMMANFATTNING. Traceanalys ger bättre möjligheter att förstå, felsöka och optimera inbyggd programvara, speciellt multitrådade system. För Linux är LTTng en beprövad och effektiv lösning för traceinspelning, som kan användas utan att man modifierar befintlig kod. *Tracealyzer for Linux* visualiserar inspelningar från LTTng med hjälp av ett tjugotal smart ihopkopplade grafiska vyer. Tracealyzer gör traceinspelningar mer visuella och lättillgängliga för utvecklare, vilket ger dem bättre möjligheter att producera effektiv och robust mjukvara, i tid och inom budget. ■

** Monoprojektet: http://mono-project.org/