Emulacija specijalnih operacija nad deskriptorima datoteka u korisničkom režimu QEMU-a

Aleksandar Marković, Lena Đokić, Goran Ferenc, Petar Jovanović, *Istraživačko-razvojni Institut RT-RK, Novi Sad, Srbija*

***Apstrakt*—Korisnički režim emulatora QEMU se bavi emuliranjem funkcionalnosti jezgra Linuksa jednog procesora koristeći jezgro Linuksa drugog procesora, omogućavajući time izvršavanje jednostavnih Linuks aplikacija pisanih za jedan procesor na sistemu sa drugim procesorom. Iako je taj režim QEMU-a razvijan više od decenije od strane zajednice otvorenog koda, njegova emulacija nije sasvim verna, najviše zbog nepostojanja podrške za neke složenije funkcionalnosti jezgra Linuksa. U ovom radu je izloženo rešenje koje upotpunjuje funkcionalnost korisničkog režima QEMU-a u vidu dodavanja podrške za specijalne operacije nad deskriptorima datoteka. U sadašnjoj fazi tog rešenja, implementirana je podrška za manipulaciju deskriptorima datoteka koji su u vezi sa podsistemom jezgra Linuksa za notifikaciju fanotify. Međutim, rešenje je od početka zamišljeno da može da bude lako prošireno i na preostale slučajeve specijalnih operacija nad deskriptorima datoteka.**

***Ključne reči*—QEMU, Linuks, jezgro Linuksa, emulacija**

I. Uvod

RAZNOLIKOST elektronskih uređaja kontrolisanih mikroprocesorom sa ne smanjuje poslednjih godina. To predstavlja značajan problem prilikom razvoja i testiranja softverske podrške za te uređaje. Jedan od načina prevazilaženja ovog problema jeste korišćenje portabilnih operativnih sistema kao što je Linuks. Međutim, čak i postojanje podrške za Linuks za ciljni računarski sistem ne rešava problem ograničene raspoloživosti - u mnogim situacijama potrebno je testirati aplikaciju koja treba da radi na sistemu koji fizički nije raspoloživ u samo vreme razvoja. Imajući takve i slične situacije u vidu, razvijen je emulator QEMU, softverski alat pomoću kojeg je moguće izvršiti aplikaciju pisanu za jedan procesor jedne arhitekture na sistemu koji ima procesor druge arhitekture. Konkretnije, tzv. korisnički režim emulatora QEMU emulira funkcionalnost jezgra Linuksa jednog procesora koristeći funkcionalnosti jezgra Linuksa drugog procesora.

Takva emulacija pomoću QEMU-a, međutim, nije kompletna, jer neke složenije (a po pravilu ređe korišćene) funkcionalnosti sprege aplikacije i jezgra Linuksa nisu podržane. Ovaj rad se bavi jednom od takvih nedostajućih funkcionalnosti korisničkog režima QEMU-a – problemom podrške za specijalne operacije nad deskriptorima datoteka.

Ovakvim upotpunjavanjem funkcionalnosti korisničkog režima QEMU-a proširuje se spektar aplikacija koje se mogu izvršavati u tom režimu, i time omogućuje efikasniji rad inženjera koji se bave razvojem softvera za sisteme sa raznolikim procesorima. To je bila glavna motivacija za kreiranje rešenja opisanog u ovom radu.

Organizacija rada je sledeća: u sekcijama II-IV se definišu i opisuju pojmovi i sistemi relevantni za problem koji se rešava; sekcije V i VI predstavljaju srž opisa rešenja, dok se u sekcijama VII-IX diskutuju njegovi razni drugi aspekti.

U zajednici otvorenog koda za razvoj QEMU-a ne postoji tradicija pisanja i objavljivanja radova o rešenjima koja se predlažu ili usvajaju. Ipak, neki radovi srodnih tematika sa predmetom ovog rada zaslužuju da budu spomenuti. Članci [1]-[4] predstavljaju dobar izvor informacija o arhitekturi, organizaciji i osnovnim načinima korišćenja QEMU-a. Neki od mnogih naprednih koncepata u vezi sa QEMU-om pokriveni su u radovima [5] i [6]. Radovi [7] i [8] bave se problematikom sistemskih poziva kod Linuksa u emuliranom ili virtueliziranom okruženju. Neki koncepti slični osnovnoj ideji rešenja koje će biti izloženo u ovom radu predstavljeni su u radovima [9], [10], [11], [12], [13] i [14].

II. Specijalne operacija nad deskriptorima datoteka kod jezgra Linuksa

Gotovo svi savremeni operativni sistemi se oslanjaju na koncept datoteke na čijem temelju grade sisteme za čuvanje, dobijanje ili prenos podataka. Kod Linuksa, svaki proces na određeni način dobija jedan ceo broj koji identifikuje datoteku koja se koristi, i taj broj se naziva deskriptor datoteke. Pomoću takvog deskriptora moguće je izvršiti određene operacije nad datotekom. U velikoj većini slučajeva, to su samo operacije čitanja i pisanja. Međutim, postoje i složenije operacije. Recimo, u Linuksu, pomoću deskriptora podataka moguće je pročitati podatke o performansama izvršenja aplikacije, ili ispitati stanje notifikacionog mehanizma operativnog sistema, ili čekati da određeni časovnik istekne. Takve operacije će u ovom radu biti nazvane specijalne operacije nad deskriptorima datoteka.

Specijalne operacije nad deskriptorima datoteka su najčešće usko vezane sa određeni podsistem operativnog sistema. U tabeli I je dat pregled nekih takvih podsistema u slučaju jezgra

Aleksandar Marković, Lena Đokić, Goran Ferenc, Petar Jovanović – Istraživačko-razvojni Institut RT-RK, Narodnog fronta 23a, Novi Sad, Srbija (e-mail adrese: aleksandar.markovic@rt-rk.com, lena.djokic@rt-rk.com, goran.ferenc@rt-rk.com, petar.jovanovic@rt-rk.com)

Linuksa, a za svaki takav podsistem opisan je ukratko način korišćenja deskriptora datoteka:

TABELA I
NEKE SPECIJALNE OPERACIJE NAD DESKRIPTORIMA DATOTEKA PODRŽANE OD STRANE JEZGRA LINUKSA

sistemski poziv koji stvara deskriptor datoteke	*opis funcionalnosti specijalnih operacija nad deskriptorima datoteka*
eventfd()	*novostvoreni deskriptor predstavlja objekat koji se koristi za generalni mehanizam čekanja i notifikacije; sistemskim pozivima read(), write(), poll(), select(), ppoll() i pselect() vrši se interakcija sa tim mehanizmom*
fanotify_init()	*deskriptor predstavlja objekat skupa događaja u vezi sa sistemom datoteka; taj objekat se može ažurirati pomoću fanotify_mark() i write(), pomoću sistemskog poziva read() moguće je čekati na unapred definisani događaj*
perf_event_open()	*pomoću sistemskih poziva read(), mmap(), ioctl() i fcntl() nad novostvorenim deskriptorom dobijaju se informacije o određenim merama performansi, a moguće je i podešavanje samog sistema za merenje performansi*
signalfd()	*ovako stvoreni deskriptor može se koristiti za prihvatanje signala; pomoću sistemskih poziva poll(), ppoll(), epoll(), select() i pselect() vrši se nadgledanje tog deskriptora*
timerfd_create()	*primenom sistemskih poziva poll(), ppoll(), epoll(), select() i pselect() nad novostvorenim deskriptorom dobija se informacija o isticanju izabranog časovnika*

III. PODSISTEM ZA NOTIFIKACIJU FANOTIFY

Jezgro Linuksa nudi API za notifikaciju u vezi sa sistemima datoteka koji se zove fanotify.

TABELA II
PODSISTEMI ZA NOTIFIKACIJU KOD RAZLIĆITIH OPERATIVNIH SISTEMA

podsistem za notifikaciju	*operativni sistem*
inotify	Linux
kqueue	BSD, OS X, iOS
ReadDirectoryChangesW	Windows
FSEvents	OS X
File Alteration Monitor (FAM)	Unix
File Event Notifications (FEN)	Solaris 11
fanotify	Linux 2.6.37+

U tabeli II su navedeni podsistemi za notifikaciju kod raznih operativnih sistema, pomoću koje može da se šire i jasnije sagleda značaj podsistema fanotify kod Linuksa.

Pomoću podsistema fanotify moguće je, recimo, dobiti notifikaciju o tome da je neka datoteka otvorena od strane bilo kog procesa u sistemu, ili da li je pristupljeno sadržaju određenog direktorijuma, ili pak da li su promenjena prava pristupa specificiranoj datoteci ili direktorijumu.

Korišćenje podsistema fanotify organizovano je na sledeći način: prvo se dobije deskriptor datoteke pomoću jedne od funkcija podsistema, a zatim se pomoću specijalnih operacija nad tim deskriptorom dobija željena funkcionalnost notifikacije. Konkretnije, sledeći sistemski pozivi se koriste u vezi sa deskriptorima datoteka podsistema fanotify fanotify_init(), fanotify_mark(), read(), write() i close(2). Stvaranje takvih deskriptora datoteka se vrši korišćenjem sistemskog poziva fanotify_init().

Ukoliko se na tako stvoreni deskriptor primeni sistemski poziv read(), on će blokirati sve dok se ne dogodi neki od događaja u vezi sa datotekama (specificiranim u toku kreiranja i manipulacije deskriptora datoteka), ili dok poziv ne bude prekinuta signalom. U slučaju uspešnog izvršavanja, prihvatna memorija sistemskog poziva read() će sadržati jednu ili više struktura sledeće organizacije

```
struct fanotify_event_metadata {
    __u32 event_len;
    __u8 vers;
    __u8 reserved;
    __u16 metadata_len;
    __aligned_u64 mask;
    __s32 fd;
    __s32 pid;
};
```

Ukoliko se, pak, koristi sistemski poziv write() nad deskriptorom datoteka stvorenim pomoću fanotify_init(), prihvatna memorija treba da ima sledeću organizaciju:

```
struct fanotify_response {
    __s32 fd;
    __u32 response;
};
```

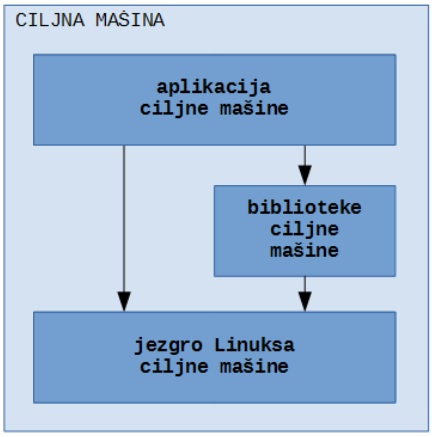
Kao što se vidi iz gornjih segmenata koda, elementi struktura koje se koriste prilikom sistemskih poziva read() i write() uglavnom su celobrojnog tipa dužine 16, 32 ili 64. Značaj te činjenice biće detaljnije obrazložen i razjašnjen u sledećim sekcijama.

IV. PRINCIPI EMULACIJE U KORISNIČKOM REŽIMU QEMU-A

Korisnički režima emulatora QEMU stremi emuliranju funkcionalnosti jezgra Linuksa jednog računarskog sistema pomoću korišćenja funkcionalnosti jezgra Linuksa drugog računarskog sistema. Ova dva sistema se, u žargonu QEMU-a, nazivaju ciljna mašina i mašina domaćin. Svaka komunikacija aplikacije ciljne mašine sa jezgrom Linuksa se presreće i emulira najčešće oslanjanjem na analognu funkcionalnost jezgra Linuksa mašine domaćina. U svom sadašnjem stanju, takva emulacija funkcionalnosti jezgra nije stopostotna, ali se nedostajući fragmenti neprestano dodaju.

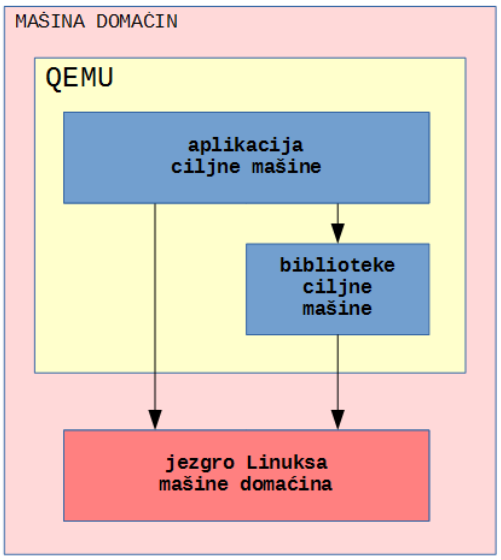
Na slici 1 je prikazana klasična organizacija interakcije

aplikacije, biblioteka i jezgra Linuksa.



Sl. 1. Interakcije aplikacije, biblioteka i jezgra Linuksa.

Kao pandan prethodnoj organizaciji, organizacija odnosa aplikacije i biblioteka ciljne mašine i jezgra mašine domaćina kod korisničkog režima QEMU-a prikazana je na slici 2:



Sl. 2. Arhitektura korisničkog režima QEMU-a.

V. Rešenje emulacije glavne funkcionalnosti podsistema fanotify

Ova sekcija se odnosi na implementaciju emulacije sistemskih poziva fanotify_init() i fanotify_mark(). Oba sistemska poziva ciljne mašine biće translirana u odgovarajuce sistemske pozive mašine domaćina. Međutim, ta translacija ne može da bude direktna, iz razloga opisanih u sledećim paragrafima.

Jedan problem su različite dužine osnovne jedinice adresiranja ciljne mašine i mašine domaćina, kao i različite dužine nekih tipova podataka kao direktna posledica toga. Određene konverzije moraju biti izvršene da bi emulacija radila u svim podržanim slučajevima i dostigla precizno zadovoljavanje specifikacija sistemskih poziva ciljne mašine. Podržane kombinacije u tom pogledu su prikazane u tabeli III:

TABELA III
PODRŽANE KOMBINACIJE CILJNE MAŠINE I MAŠINE DOMAĆINA (U SMISLU DUŽINE OSNOVNE JEDINICE ADRESIRANJA) KOD KORISNIČKOG REŽIMA QEMU-A

		ciljna mašina	
		32-bitna	64-bitna
mašina domaćin	32-bitna	podržano	nepodržano
	64-bitna	podržano	podržano

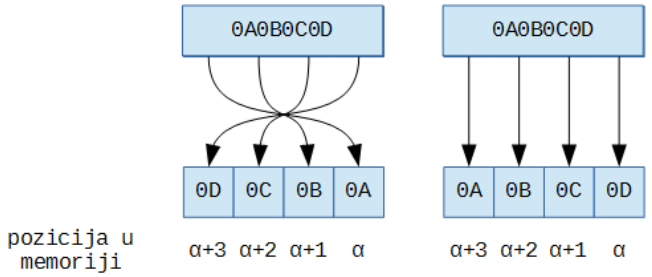
Još jedan značajan problem jeste raznolikost određenih konstanti u jezgru Linuksa u vezi sa različitim procesorima. Potrebno je izvršiti konverziju takvih konstanti između ciljne mašine i mašine domaćina da bi se očuvala semantika parametara konkretnog sistemskog poziva. U tabeli IV je naveden primer konstanti koje određuju poziciju zastavica unutar jednog argumenta sistemskog poziva fcntl(), kao i vrednosti tih pozicija koje te konstante specificiraju za razlicite procesore:

TABELA IV
RAZNOLIKOST POZICIJA ZASTAVICA JEDNOG ARGUMENTA SISTEMSKOG POZIVA FCNTL() U POGLEDU PROCESORA

konstanta	***procesor***				
	alpha	***arm***	***intel***	***mips***	***sparc***
O_RDONLY	0	0	0	0	0
O_WRONLY	1	1	1	1	1
O_RDWR	2	2	2	2	2
O_CLOEXEC	22	19	19	19	22
O_NONBLOCK	4	11	11	7	14

VI. Rešenje emulacije operacija nad deskriptorima datoteka kod podsistema fanotify

Deskriptor datoteka stvoren pomoću fanotify_init() mora da podržava operacije read() i write() po specifikacijama opisanim u sekciji III. Ukoliko bi se koristila direktna translacija sistemskih poziva read() i write() ciljne mašine na sistemske poziva read() i write() mašine domaćina, podaci u strukturama koje učestvuju u tom procresu bi bili netačni ukoliko se redosled zapisa brojeva u memoriji (eng. endianness) cilje mašine i mašine domačina razlikuju. To je ilustorvano na slici 3:

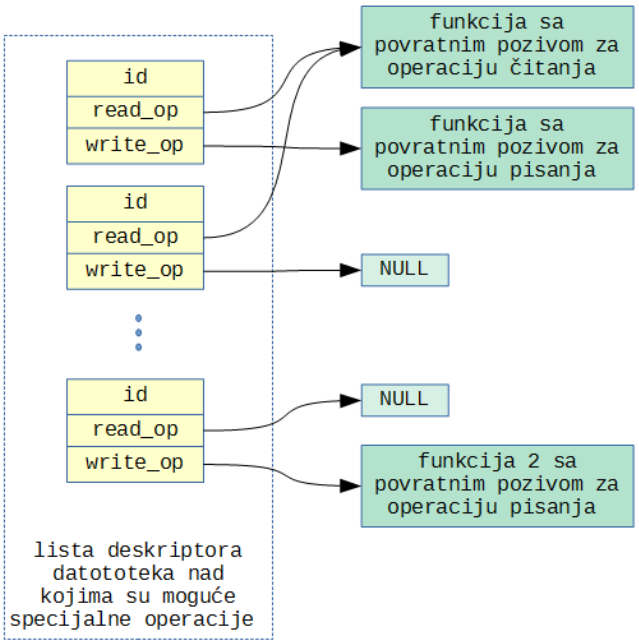


Sl. 3. Reprezentacija 32-bitnog celog broja kod dva sistema sa različitim redosledom zapisa brojeva u memoriji.

Stoga je bilo potrebno identifikovati takve slučajeve u vreme izvršavanja, i na osnovu toga, ukoliko je potrebno, primeniti rutine za konverziju binarnih reprezentacija.

Sledeći značajan izazov je bio problem distinkcije deskriptora koji su stvoreni pomoću fanotify_init() i ostalih deskriptora – QEMU mora da poseduje informaciju kada je u toku specijalna operacija nad deskriptorom, a kada ne.

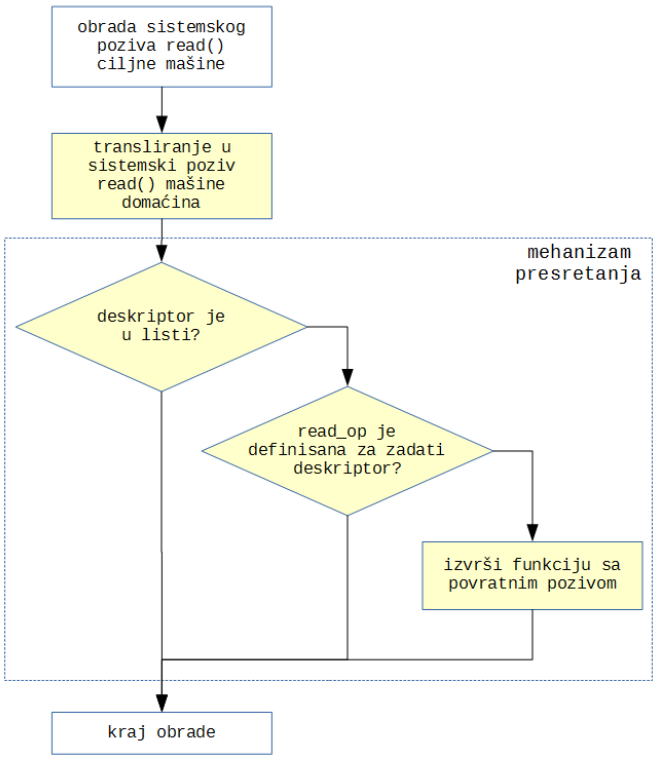
Da bi se rešio ovaj problem, projektovan je, unutar korisničkog režima QEMU-a, sistem za vođenje evidencija o deskriptorima koji zahtevaju specijalne operacije. Nešto detaljnije, svakom deskriptoru stvorenim putem sistemskog poziva fanotify_init() dodeljuje se jedan element u listi fanotify deskriptora (ta lista se održava unutar QEMU-a i nezavisna je od jezgra Linuksa). Svaki takav element sadrži identifikator deskritora i pokazivače na određeni broj funkcija koje treba da budu pozvane u određenim situacijama, u skladu sa specifikacijom podsistema fanotify (takav aranžman funkcija i pokazivača se često zove mehanizam funkcija sa povratnim pozivom). U našem slučaju, funkcionalnost od interesa se odnosi na korišćenje deskriptora putem sistemskih poziva read() i write(), pa postoje dva pokazivača koji su prikladno nazvani read_op i write_op. Celokupna organizacija liste deskriptora je prikazana na slici 4:



Sl. 4. Konceptualni primer predložene liste deskriptora.

Ovde treba napomenuti da se u vezi sa podsistemom fanotify javlja potreba za samo dva pokazivača na funkcije sa povratnim pozivom - to su oni koji su označeni sa read_op i write_op na slici 4. U slučaju nekih drugih podsistema, postojaće potreba za definisanjem dodatnih pokazivača, recimo select_op. Takvo proširenje u principu se svodi na dodavanje elementa u jednu strukturu i obradu operacija koje taj element treba da podrži. Takođe, neki drugi podsistemi možda uopšte neće koristiti pokazivače read_op i write_op i u tom slucaju treba da ih postave na NULL, kao što je i ilustrovano na slici 4.

Na slici 5 je prikazano korišćenje prethodno opisane liste deskriptora u slučaju sistemskog poziva read():



Sl. 5. Predloženi mehanizam presretanja kod primene sistemskog poziva read() na deskriptor datoteka fanotify u korisničkom režimu QEMU-a.

Analogni mehanizam presretanja projektovan je i za sistemski poziv write(). Jedna značajna razlika u odnosu na slučaj sistemskog poziva read() jeste što se mehanizam presretanja umeće pre (a ne posle) transliranja u sistemski poziv mašine domaćina, jer u tom trenutku mašina domaćin mora da poseduje pravilne podatke.

VII. Testiranje rešenja

Testiranje rešenja je bilo dvojako:

A. Testiranje jedinica

Napisan je određeni broj test programa za testiranje jedinica. Tom prilikom je vođeno računa da je pokrivenost novog koda 100%.

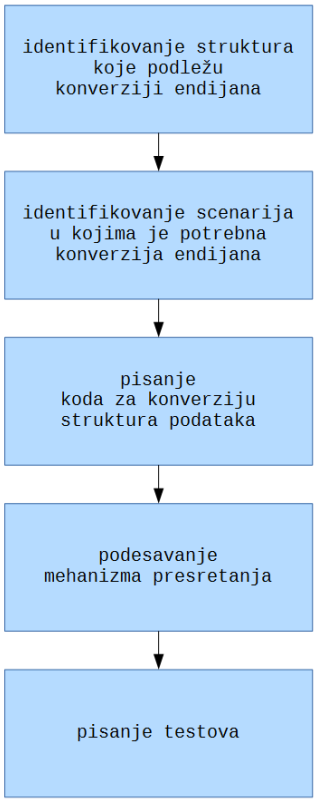
B. Integracioni testovi

Korišćen je podskup testova u skupu testova za testiranje sistemskih poziva LTP. LTP (skraćenica od Linux Test Project) je skup od više od 2000 test programa koji sadrže preko 10000 test slučajeva za testiranje funkcionalnosti jezgra Linuksa. Jedan broj tih testova koristi podsistem jezgra fanotify, i ti testovi su uzeti kao integracioni testovi predloženog rešenja. Takvi testovi su organizovani u 7 grupa sa ukupno 100 test slučajeva.

VIII. Pregled potencijalnog rešenja u slučaju emulacije drugih sistemskih poziva koji uključuju specijalne operacije nad deskriptorima datoteka

Rešenje opisano u ovom radu je fokusirano na podsistem jezgra za notifikaciju fanotify. Međutim, ono je od početka

bilo projektovano tako da se može primeniti na druge situacije koje zahtevaju specijalne operacije nad deskriptorima datoteka. Uopšteni postupak implementacije u tim slučajevima se svodi na korake prikazane na slici 6:



Sl. 6. Uopšteni postupak implementacije podrške za operacije nad deskriptorima datoteka u korisničkom režimu QEMU-a.

Planira se da se opisani pristup primeni na sve preostale slučajeve specijalnih operacija nad deskriptorima datoteka nabrojanih u tabeli I.

IX. Diskusija i evaluacija rešenja i zaključak

Glavna prednost opisanog rešenja jeste što je projektovano tako da ima što opštiji karakter. To je demonstrirano u sekciji VIII, gde se vidi da postoji standardan šablon implementacije u sličnim situacijama.

Mana rešenja su povećani zahtevi u vezi memorije u vreme izvršavanja QEMU-a. Za svaki stvoreni deskriptor datoteka potrebno je alocirati strukturu sa identifikatorom i pokazivačima na funkcije sa povratnim pozivom. Sa druge strane, ta struktura je male veličine, pa stoga ova mana može da dođe do izražaja jedino ako aplikacija otvara neobično veliki broj deskriptora datoteka. Još jedna mana rešenja jeste da unosi određeni negativni efekat na performanse operacija nad deskriptorima datoteka, ali on nije značajan. Obe opisane mane su u ovom trenutku shvaćene kao minorne, i ne planira se dodatni razvoj koji bi ih otklonio.

Opšti zaključak napora predstavljenih u ovom radu jeste da je predloženi pristup adekvatan i da doprinosi kvalitetu QEMU-a, kao i da postoje potencijali za buduće primene i proširenja. Dalji rad na opisanoj tematici će biti dvojak: sa jedne strane će se sastojati u saradnji sa zajednicom otvorenog koda u smislu prihvatanja opisanog rešenja, a sa druge strane će uključivati napore ka upotpunjavanju i generalizovanju rešenja.

Zahvalnica

Ovaj rad je delimično finansiran od strane Ministarstva za nauku i tehnologiju Republike Srbije, na projektu tehnološkog razvoja broj III_044009_1.

Literatura

[1] F. Bellard, *QEMU, a Fast and Portable Dynamic Translator,* Proceedings of the USENIX 2005 Annual Technical Conference, pp. 41-46, 2005

[2] F. Ciacchi, *Getting started with User-Mode Linux: Linux in Linux,* Linux Magazine, Issue 54, pp. 32-36, May 2005.

[3] F. Ciacchi, *System Emulation with QEMU: Virtual Benefits,* Linux Magazine, Issue 52, pp. 46-51, March 2005.

[4] D. Bartholomew, *QEMU: A Multihost, Multitarget Emulator,* Linux Journal, Issue 145, pp. 68-71, May 2006.

[5] Y. Kinebuchi, H. Koshimae, S. Oikawa, T. Nakajima, *Dynamic Translator-based Virtualization*, Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2007), pp.182 – 195, Santorini Island, Greece, 2007.

[6] C. Guillon, *Program Instrumentation with QEMU*. Proceedings of the Conference on Design, Automation and Test in Europe (DATE '11), Grenoble, France, 2011.

[7] J. Eriksson, *Virtualization, Isolation and Emulation in a Linux Environment,* Master's Thesis in Computing Science, Umea University, Sweden, April 2009.

[8] J. Jelten, *Dynamic System Call Translation between Virtual Machines,* Bachelor's Thesis in Informatics, Technische Universitat Munchen, Germany, September 2014.

[9] X. Wang, V. Jhi, S. Zhu, P. Liu, *Detecting software theft via system call based birthmarks*,. Proceedings of the Computer Security Applications Conference 2009, p. 149–58., 2009.

[10] A. Reina, A. Fattori, L. Cavallaro, *A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors*, Proceedings of Euro Sec'13, pp. 1–6, 2013.

[11] G. Canfora, E. Medvet, F. Mercaldo, C. A. Visaggio, *Detection of malicious web pages using system calls sequences*, Proceedings of the 4th International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD 2014), pp. 226–238, 2014.

[12] D. Gao, *Gray-Box Anomaly Detection using System Call Monitoring*, Doctoral Dissertation, Carnegie Mellon University, USA, Jan 2007.

[13] W. Fadel, *Techniques for the Abstraction of System Call Traces to Facilitate the Understanding of the Behavioural Aspects of the Linux Kernel*, Masters Thesis, Concordia Universitz, Canada, Nov. 2010.

[14] C.-C. T. Bhushan, J. Nafees, A. Abdul, D. E. Porter, *A Study of Modern Linux API Usage and Compatibility: What to Support When You're Supporting*, Stony Brook University, United Kingdom, Apr. 2016.

Abstract

QEMU user mode deals with emulation of Linux kernel functionality for a certain processor using Linux kernel functionality built for another processor. It provides execution of a simple Linux applications written for one processor on a system with another processor. In spite of more than decade long development of QEMU user mode by open source community, that emulation is not entirely faithful, mostly because support for certain more complex Linux kernel functionalities is missing. A solution for amending QEMU user mode functionality in the area of file descriptor special operations support is described in this paper. In its present

state, only support for file descriptor manipulation related to Linux file system notification subsytem fanotify is implemented. However, the solution is designed from its outset to be easily extended to the remaining cases of special operations on file descriptors.

Emulation of special operations on file descriptors in user mode QEMU

Aleksandar Marković, Lena Đokić, Goran Ferenc, Petar Jovanović