Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za računarsku tehniku i računarske komunikacije

Linuks u ugrađenim sistemima i razvoj rukovalaca

Razvoj rukovalaca
Treći deo

Sadržaj

- Razvoj rukovalaca
  - Debagovanje
  - Udaljeno debagovanje

Debagovanje

RAZVOJ RUKOVALACA

Debagovanje sa printk (1/2)

- Univerzalna tehnika debagovanja sa ispisima
- Ispisi se pojavljuju u konzoli ili u /var/log/messages u zavisnosti od prioriteta
  - Kontrolisano pomoću kernel parametra loglevel ili kroz /proc/sys/kernel/printk
- Dostupni prioriteti (include/linux/kernel.h)
  - #define KERN_EMERG "<0>" /* system is unusable */
    #define KERN_ALERT "<1>" /* action must be taken immediately */
    #define KERN_CRIT "<2>" /* critical conditions */
    #define KERN_ERR "<3>" /* error conditions */
    #define KERN_WARNING "<4>" /* warning conditions */
    #define KERN_NOTICE "<5>" /* normal but significant condition */
    #define KERN_INFO "<6>" /* informational */
    #define KERN_DEBUG "<7>" /* debug-level messages */

Debagovanje sa printk (2/2)

- Zastarela tehnika, više nije preporučljiva
- pr_* familija funckija: pr_emerg(), pr_alert(), pr_critic(), pr_err(), pr_warning(), pr_notice(), pr_info(), pr_cont() i specijalna pr_debug()
  - Argument je string
  - Definisane u zaglavlju include/linux/printk.h
- dev_* familija funkcija: dev_emerg(), dev_alert(), dev_crit(), dev_err(), dev_warning(), dev_notice(), dev_info() i specijalna dev_debug()
  - Prvi argument je pokazivač na struct device strukturu, a drugi je string
  - Definisane u zaglavlju include/linux/device.h
  - Koriste se u rukovaocima koji koiste Linuksov model rukavaoca

pr_debug() i dev_debug() (1/2)

- Kada se rukovalac prevede sa definisanim simbolom DEBUG, sve ove poruke se prevode i prikazuju se na debug nivou

- DEBUG simbol se može definisati
  - Iskozom #define DEBUG na pocetku rukovaoca
  - Korišćenjem ccflags-$(CONFIG_DRIVER) += -DDEBUG u Makefile-u

pr_debug() i dev_debug() (2/2)

- Kada se kernel prevede sa CONFIG_DYNAMIC_DEBUG uključenim, ove poruke se mogu dinamički uključivati na nivou datoteke, modula ili poruke
  - Više detalja u Documentation/dynamic-debug-howto.txt
- Kada DEBUG simbol nije definisan ove poruke se ne prevode

Debagovanje sa /proc i /sys (1/2)

- Umesto ispisivanja poruka u kernelski log informacije se mogu učiniti dostupnim korisničkom prostoru kroz datoteku u /proc ili /sys koja se registruje u rukovaocu
- Može da prikaže bilo kakvu informaciju o uređaju ili rukovaocu
- Može da se koristi i za slanje podataka i za kontrolu rukovaoca
- Upozorenje: Dostupno svima
  - U fazi proizvodnje treba ukloniti sprege za debagovanje
- Od pojave debugfs-a nije više preferirani način debagovanja

Debagovanje sa /proc i /sys (2/2)

- Primeri
  - cat /proc/acme/stats
    - Prikazuje statistiku o acme rukovaocu
  - cat /proc/acme/globals
    - Prikazuje vrednosti globalnih varijabli koje koristi rukovalac
  - echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
    - Podešava brzinu procesora

Debugfs

- Virtuelni sistem datoteka za prikaz debag informacija u korisničkom prostoru
- Konfiguracija kernela
  - Kernel hacking > Debug Filesystem (DEBUG_FS)
- Jednostavnije za kodovanje od sprega u /proc ili /sys
  - Debag sprega nestaje kada se isključi u konfiguraciji
- Primer mauntovanja
  - sudo mount -t debugfs none /mnt/debugfs

Debugfs API (1/2)

- Pravljenje poddirektorijuma :
  - struct dentry *debugfs_create_dir(const char *name, struct dentry *parent);
- Prikaz celobrojne promenljive kao datoteke u DebugFS-u
  - struct dentry *debugfs_create_{u,x}{8,16,32} (const char *name, mode_t mode, struct dentry *parent, u8 *value);
    - u je decimalni prikaz
    - x je heksadecimalni prikaz

Debugfs API (2/2)

- Prikaz binarnog bloba u DebugFS-u
  - struct dentry *debugfs_create_blob(const char *name, mode_t mode, struct dentry *parent, struct debugfs_blob_wrapper *blob);
- Moguće je koristiti i generičkiju funkciju debugfs_create_file() u razne svrhe

Jednostavan debugfs primer

```
#include <linux/debugfs.h>

static char *acme_buf;                                  // module buffer
static unsigned long acme_bufsize;
static struct debugfs_blob_wrapper acme_blob;
static struct dentry *acme_buf_dentry;

static u32 acme_state;                                  // module variable
static struct dentry *acme_state_dentry;

/* Module init */
acme_blob.data = acme_buf;
acme_blob.size = acme_bufsize;
acme_buf_dentry = debugfs_create_blob("acme_buf", S_IRUGO,          // Create
                                      NULL, &acme_blob);            // new files
acme_state_dentry = debugfs_create_bool("acme_state", S_IRUGO,      // in debugfs
                                        NULL, &acme_state);

/* Module exit */
debugfs_remove (acme_buf_dentry);                       // removing the files from debugfs
debugfs_remove (acme_state_dentry);
```

Debagovanje sa ioctl

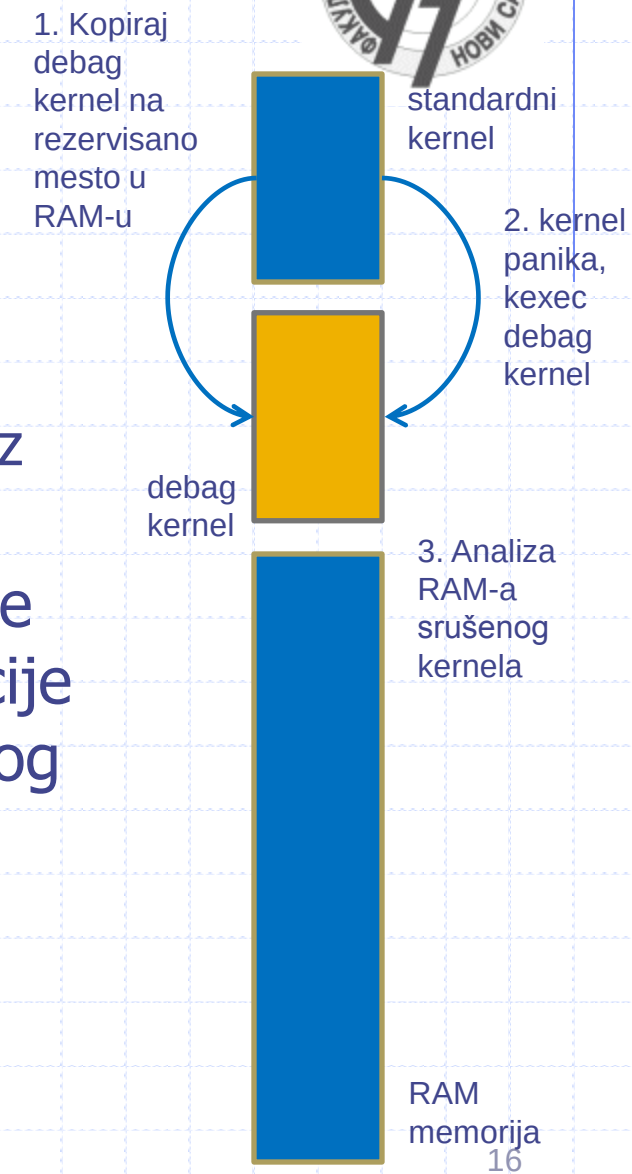
- ioctl() sistemski poziv se može koristiti za upit o informacijama ili za slanje komandi rukovaocu
- Poziva ioctl operaciju nad datotekom koja se registruje u rukovaocu
- Prednost - debag sprega nije javna (može se čak ostaviti u sistemu i kada dospe do korisnika)

Debagovanje sa GDB-om

- Ukoliko se kernel pokrene iz debagera na istom računaru uticaće na ponašanje kernela na računaru
- GDB može da pristupi trenutnom stanju kernela
  - gdb /usr/src/linux/vmlinux /proc/kcore
- Može da se pristupi kernelskim strukturama, da se prate pokazivači (samo čitanje je dozvoljeno)
- Kernel mora da bude preveden sa CONFIG_DEBUG_INFO (Kernel hacking sekcija)

Analiza kernelske greške sa kexec-om

- kexec sistemski poziv
  - omogućeva pozivanje novog kernela bez restartovanja sistema
- Ideja: posle kernel panike novi kernel se automatski učitava sa rezervisane lokacije u RAM-u i vrši analizu memorije srušenog kernela



Još saveta za debagovanje

- Omogućiti CONFIG_KALLSYMS_ALL
  - General Setup > Configure standard kernel features
  - poruke o oops-evima se dobijaju sa imenima simbola umesto sa sirovim adresama

- Ukoliko kernel ne može da se pokrene a ne ispisuje nikakve poruke može da se uključi debagovanje na niskom nivou
  - CONFIG_DEBUG_LL=y

Udaljeno debagovanje

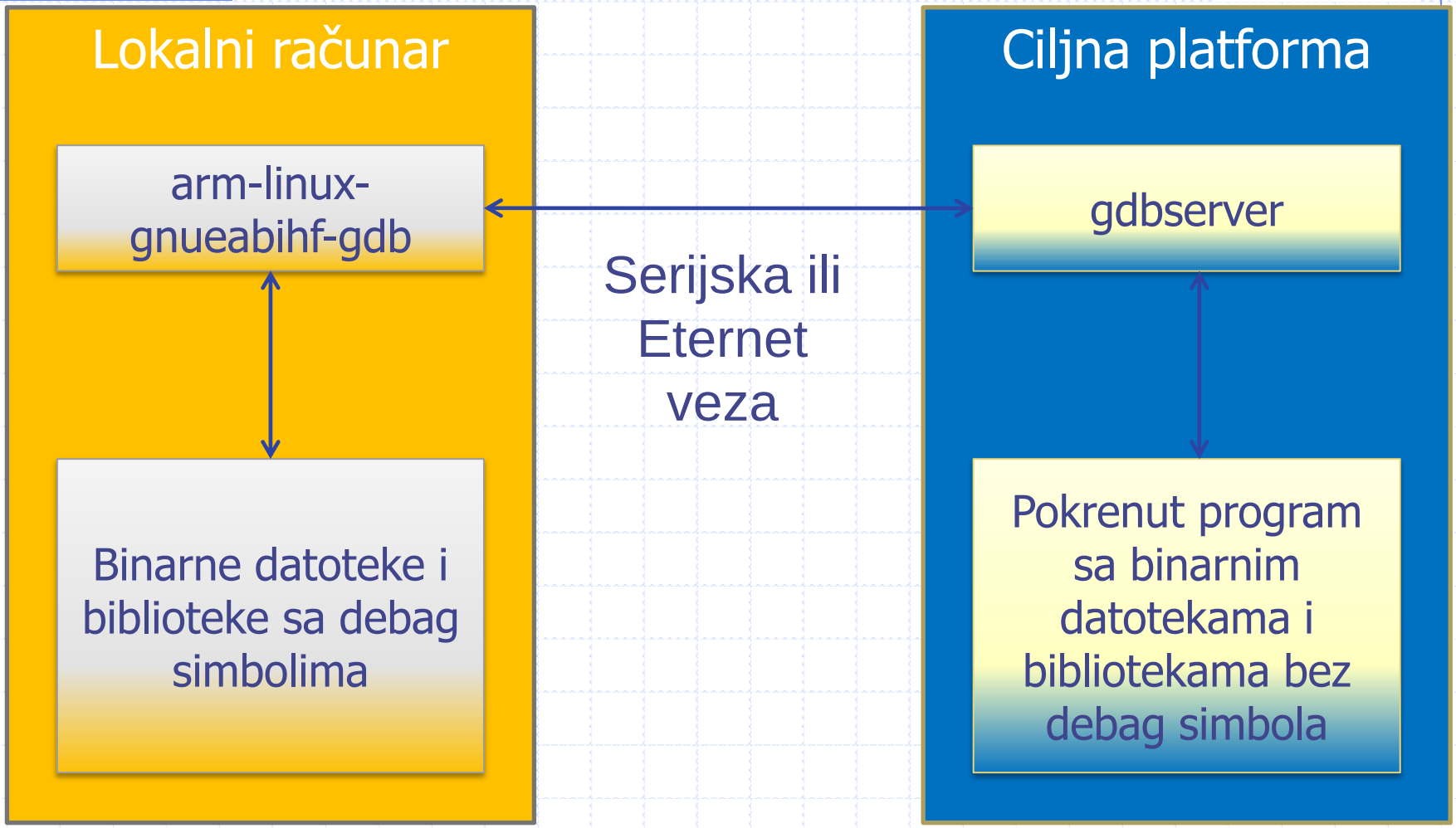
RAZVOJ UGRAĐENIH SISTEMA

Udaljeno debagovanje

- U neugrađenim sistemima debagovanje se uglavnom izvršava gdb-om
- gdb ima direktan pristup binarnim datotekama i bibliotekama prevednim sa debag simbolima
- U ugrađenim sistemima ciljna platforma je često ograničena memorijom i ne može da dozvoli direktno debagovanje gdb-om

- Bolja opcija je udaljeno debagovanje
  - gdb se koristi na lokalnom računaru
  - gdbserver se koristi na ciljnoj platformi

Udaljeno debagovanje - arhitektura



Udaljeno debagovanje - zahtevi

- Zahtevi
  - Na lokalnom računaru
    - alati za prevođenje sa podrškom za gdb (arm-linux-gnueabihf-gdb)
    - Verzija gdb-a koja se pokreće na lokalnom računaru, ali razume specifičnosti ciljne platforme
  - Na ciljnoj platformi
    - gdbserver program preveden za ciljnu arhitekturu
    - Često se nalazi kao deo alata za prevođenje
  - Serijska ili Eternet veza između lokalnog računara i ciljne platforme

Udaljeno debagovanje - korišćenje

- Povezivanje gdbservera na pokrenut program na ciljnoj platformi

```
$ gdbserver program -p <pid>
```

- Pokrenuti na lokalnom računaru

```
$ arm-linux-gnueabihf-gdb
```

- Ostvarivanje veze sa gdbserver-om

```
(gdb) target remote <host:port>
```

- Učitavanje simbola

```
(gdb) symbol-file <program>
```

- Nakon ovog koraka gdb može da se koristi kao i pri normalnom debagovanju na lokalnom računaru