International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 10 (2015) pp. 26073-26083

http://www.ripublication.com

Enhancing Run-Time Configurable Features Using Kernel Information

Rajamohan.L[1*], Dr.Ravi.S[2] and P.Revanth Mahesh Kumar Sai,

[1]Research Scholar, Department of ECE, St.Peters' University, Chennai,*
Email: rajamohan151@gmail.com
[2]Professor and Head, Department of ECE, Dr.M.G.R. Educational and Research Institute University, Madhuravoyal,Chennai,
Email: ravi_mls@yahoo.com
Student, ECE department,
Dr.M.G.R. Educational and Research Institute,
Chennai - 95.

Abstract

An interface to internal data structures in the kernel is the proc file system. The metadata information about the system, process id and changes occurring in certain kernel parameters at runtime is obtained using this. The directory /proc maintain one subdirectory for individual process running on the system, which lists the process ID and the kernel data files give information about the running kernel. The files used to obtain this information are contained in /proc. Not all of these needs to be present in system and depending on the form-factor the modules loaded in the kernel configuration are kept dynamic. This is achieved using bmenuconfig and kmenuconfig commands. However, once configured, the functionality gets customized and scalability is not possible (without reloading). This paper includes how to use the System/Kernel Information from /Proc file system and build suitable macros that can be enabled or disabled to regulate the loading of kernel modules and also keeping the design form-factor optimal.

Keywords: *Kernel customization, Macros, form-factor, user interface*

Introduction

proc file system is an interface provided to the user, to interact with the kernel and get the required information about processes running on the system. This allows one to change some parameters (by reloading a different image or environment setup into the hardware) rather than on the fly (on current running system with immediate effect.).

This has the advantage of just in time compilation or ahead of time compilation. However, the root file system, environment details, device drivers activated etc. becomes customized but not instantly reconfigurable. To overcome this limitation, this paper proposes macro enable and disable approach. Also, to know the process behavior at realtime level (i.e. cellular level), virtual process timers are implemented.

Related Works

Prasad and Akhilesh Upadhyay (2012) implemented hybrid kernel in which the input given by the Application Program interface will be submitted to the kernel. Microkernel layer takes control, which are special for Interrupt Handler mechanisms and Specific schedulers. Micro Kernel deals with real-time tasks and gives them main priority. Monolithic Kernel is dealing with non-real-time applications and tasks. However the intermediate layer of Micro kernel deals with the applications, but the non-real applications will be scheduled by the monolithic kernel. Thus the advantages of both the kernels will be achieved and make the system General purpose System.

Prasanna and VenkateswaraRao (2012) described an embedded monitoring system based on µC/OS II RTOS operating system using ARM7. It dealt with the porting of Micro C/OS-II kernel in ARM powered microcontroller for the implementation of multitasking and time scheduling. Here a real-time kernel is the software that manages the time of a micro controller to ensure that all time critical events are processed as efficiently as possible. Different interface modules of ARM7 microcontroller like UART, ADC, and LCD are used and data acquired from these interfaces is tested using µC/OS-II based real-time operating system. This paper acts as a gateway to implement RTOS for high end applications.

Jae Hwan Koh and byoung wook choi (2013) aimed to analyze the response characteristics of real-time mechanisms in kernel and user space for real-time embedded Linux: RTAI and Xenomai. The performance evaluations of real-time mechanisms depending on the changes of task periods and load are also conducted in kernel and user space. Real-time systems are generally implemented using multiple tasks. Communication, synchronization, and resource management between tasks are performed through real-time mechanisms. Therefore, the performance of real-time systems can be determined by the time responses of real-time mechanisms. Thus, benchmarking the time characteristics of real-time mechanisms is really important to estimating the deterministic real-time performance. Test metrics are jitter of periodic tasks and response time of real-time mechanisms including semaphore, real-time FIFO, Mailbox and Message queue. The results are promising to estimate deterministic real-time task execution in implementing real-time systems using RTAI or Xenomai.

Implementation

A user-space interface is implemented to the /Proc file system. This operates in three modes.

1. Standard Mode

2. Short Mode
3. Long mode

A. Standard Mode

It displays the CPU information (cpu and model), kernel version, and Uptime by extracting from the */proc/cpuinfo, /proc/version, /proc/uptime* respective files in /Proc file system. The file format is listed in table 1.

Table 1: File format details in standard mode

File format	Details	Inference
/Proc/cpuinfo	processor: 0 vendor_id:GenuineIntel cpu family: 6 model:28 model name: Intel(R) Atom(TM) CPU D410 @ 1.66GHz stepping:10	The speed of processor 0 in group 0 is being limited by system firmware. The processor has been in this reduced performance state for 71 seconds since the last report.
/Proc/version	Linux version 2.6.32-38-generic (buildd@zirconium) (gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)) #83-Ubuntu SMP Wed Jan 4 11:13:04 UTC 2012	This file specifies the version of the Linux kernel and **gcc** in use, as well as the version of Red Hat Enterprise Linux installed on the system.
/Proc/uptime	8701.53 16009.36	Specifies how long the system has been running.

cpuinfo*:* processor (the value of which is zero for single-processor systems), vendor_id (the value of which is Genuine Intel in the case of an Intel processor), cpu family, model_name, cpu MHz (processor speed in millions of cycles per second), cache size (the amount of high speed cache memory built into the processor)

Version: This string identifies the kernel version that is currently running. It includes the contents of /proc/sys/ectype.

uptime: The current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5 and 15 minutes.

B. Short Mode

It displays kernel statistics and memory information along with *Standard* mode information by extracting from the */proc/stat, /proc/memento* respective files in /Proc file system. The file format is listed in table 2.

Table 2: File format details in short mode

File format	Details	Inference
/Proc/stat	text 19611962 time 1423453071 processes 2193 procs_running 4 procs_blocked 0 softirq 9625163 0 7983432 1 26229 37805 0 6883 459811 11191109883 Switches.	Context Switch is the process of storing and restoring the state of process orthread so that execution can be resumed from the same point at a later time. This enables multiple processes to share a single CPU and is an essential feature of a multitasking operating system.
/Proc/ meminfo	MemTotal: 2044764 kB MemFree: 537368 kB Buffers: 249588 kB Cached: 638084 kB SwapCached: 0 kB Active: 916128 kB Inactive: 455424 kBs Active(anon):492444 kB Inactive(anon):57076 kB	This is used to report the amount of free and used memory (both physical and swap) on the system as well as the shared memory and buffers used by the kernel.

C. Long Mode

It displays mounted devices and average load of CPU along with Short mode information by extracting from the */proc/mounts*, */proc/loadavg* respective files in /Proc file system.

C.1./**Proc/mounts** file format

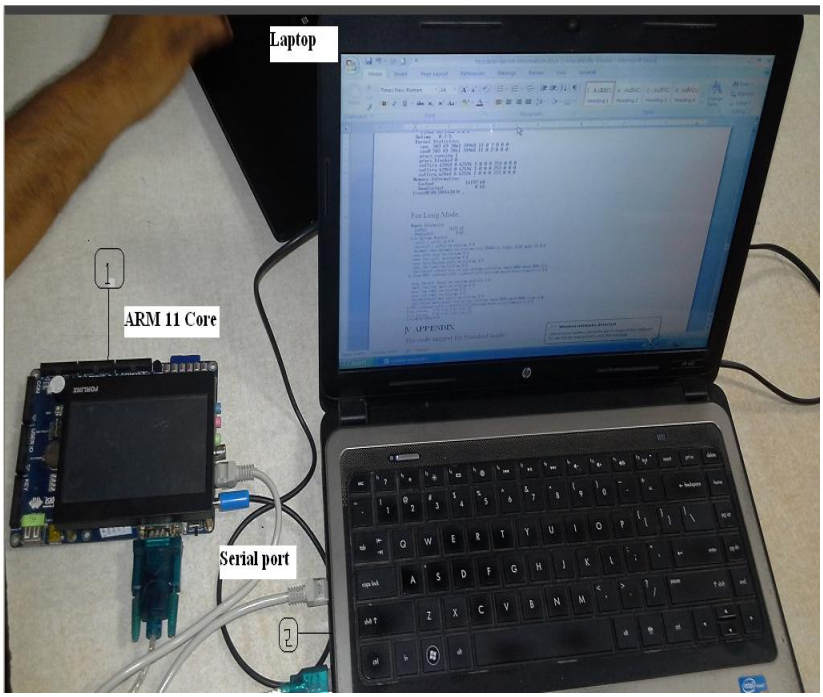
```
rootfs / rootfs rw 0 0
none /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
none /proc proc rw,nosuid,nodev,noexec,relatime 0 0
none /dev devtmpfs rw,relatime,size=1018144k,nr_inodes=216368,mode=755 0 0
none /dev/pts devpts rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000 0 0
/dev/disk/by-uuid/c0358d6e-3074-40e5-a1a1-ca713774a753 /
ext4rw,relatime,errors=remount-ro,barrier=1,data=ordered 0 0
none /sys/fs/fuse/connections fusectl rw,relatime 0 0
none /sys/kernel/debug debugfs rw, relatime 0 0
none /sys/kernel/security securityfs rw, relatime 0 0
none /dev/shm tmpfs rw,nosuid,nodev,relatime 0 0
none /var/run tmpfs rw,nosuid,relatime,mode=755 0 0
none /var/lock tmpfs rw,nosuid,nodev,noexec,relatime 0 0
none /lib/init/rw tmpfs rw,nosuid,relatime,mode=755 0 0
binfmt_misc /proc/sys/fs/binfmt_misc binfmt_misc rw,nosuid,nodev,noexec,relatime
0 0
gvfs-fuse-daemon /home/rytelyne/.gvfs fuse.gvfs-fuse-daemon
rw,nosuid,nodev,relatime,user_id=1000,group_id=1000 0 0
```

C.2**. /proc/loadavg** file format
0.38 0.55 0.48 3/289 2292

Table 3: Features extracted with reported works compared with present approach

Feature	**Reported works**	**Present Approach**
Release Date	1999–present	2009–present
Preemptive Multitasking	Yes	Yes
Maximum number of tasks	255	Dynamic
Number of tasks at each priority level	1	Dynamic
Round robin scheduling	No	Yes
Semaphores	Yes	Yes
Mutual exclusion semaphores	Yes	Yes(Nestable)
Event flags	Yes	Yes
Message mailboxes	Yes	No (not needed)
Message queues	Yes	Yes
Fixed sized memory management	Yes	Yes
Signal a task without requiring a semaphore	No	Yes
Option to post without scheduling	No	Yes
Send messages to a task without requiring a message queue	No	Yes
Software timers	Yes	Yes
Task suspend/resume	Yes	Yes(Nestable)
Deadlock prevention	Yes	Yes
Scalable	Yes	Yes
Code footprint	6K to 26K	6K to 24K
Data footprint	1K+	1K+
ROMable	Yes	Yes
Run-time configurable	No	Yes
Compile-time configurable	Yes	Yes
Pend on multiple objects	Yes	Yes
Task registers	Yes	Yes
Built-in performance measurements	Limited	Extensive
User definable hook functions	Yes	Yes
Time stamps on posts	No	Yes
Built-in kernel awareness support	Yes	Yes
Optimizable scheduler in assembly language	No	Yes
Catch a task that returns	No	Yes
Tick handling at task level	No	Yes
Source code available	Yes	Yes
Deadlock prevention	Yes	Yes
Number of services	~90	~90

Hardware Implementation



1-----(a) ARM core with root file system mounted in Sd-card
(b) Linux Kernel mounted as image file
2-----Hyperterminal to view the results.

Results and Discussion

1. Run as follows for *standard* mode,

./output

By default the program will run in standard mode and displays the hostname by reading from the */proc/sys/kernel/hostname* file and displays CPU info, kernel version and model and program exists.

2. Run as follows for *short* mode,

./output –s

It displays kernel statistics and memory information along with *Standard* mode information.

3. Run as follows for *Long* mode,

./output –l 2 4

It displays mounted devices and average load of CPU along with Short mode information. Here 2 specify the interval and 4 specifies the duration for which the average load is calculated.

For Standard Mode

```
  Linux version 3.0.1
Uptime:  0:7:5
Kernel Statistics
  cpu  503 69 2041 39968 11 0 2 0 0 0
  cpu0 503 69 2041 39968 11 0 2 0 0 0
  procs_running 1
  procs_blocked 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
Memory Information:
  Cached:           14192 kB
  SwapCached:           0 kB
[root@FORLINX6410]# ./output
Status report type Standard at  Sat Jan  1 00:42:21 2000

Machine hostname: FORLINX6410
CPU Information
  BogoMIPS      : 528.79
  Features      : swp half thumb fastmult edsp java
Kernel version:
  Linux version 3.0.1
Uptime:  0:9:24
[root@FORLINX6410]# _
```

For Short Mode

```
etc         mnt         output      sdcard      usr
lib         ms          proc        sys         var
[root@FORLINX6410]# ./output -s
Status report type Short at  Sat Jan  1 00:40:02 2000

Machine hostname: FORLINX6410
CPU Information
  BogoMIPS      : 528.79
  Features      : swp half thumb fastmult edsp java
Kernel version:
  Linux version 3.0.1
Uptime:  0:7:5
Kernel Statistics
  cpu  503 69 2041 39968 11 0 2 0 0 0
  cpu0 503 69 2041 39968 11 0 2 0 0 0
  procs_running 1
  procs_blocked 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
  softirq 42848 0 42594 1 0 0 0 253 0 0 0
Memory Information:
  Cached:           14192 kB
  SwapCached:           0 kB
[root@FORLINX6410]# _
```

For Long Mode

```
Memory Information:
  Cached:           14192 kB
  SwapCached:           0 kB
File Systems Mounted:
  rootfs / rootfs rw 0 0
  /dev/root / yaffs2 rw,relatime 0 0
  devtmpfs /dev devtmpfs rw,relatime,size=98804k,nr_inodes=24701,mode=755 0 0
  none /proc proc rw,relatime 0 0
  none /sys sysfs rw,relatime 0 0
  none /proc/bus/usb usbfs rw,relatime 0 0
  none /dev ramfs rw,relatime 0 0
  /dev/sdcard /sdcard vfat rw,sync,noatime,nodiratime,fmask=0000,dmask=0000,allo
w_utime=0022,codepage=cp936,iocharset=utf8,shortname=mixed,errors=remount-ro 0 0

  none /dev/pts devpts rw,relatime,mode=622 0 0
  tmpfs /dev/shm tmpfs rw,relatime 0 0
  none /tmp ramfs rw,relatime 0 0
  none /var ramfs rw,relatime 0 0
  /dev/mtdblock3 /mnt yaffs2 rw,relatime 0 0
  /dev/sdcard /mnt/sdcard vfat rw,sync,relatime,fmask=0000,dmask=0000,allow_utim
e=0022,codepage=cp936,iocharset=utf8,shortname=mixed,errors=remount-ro 0 0
Load average:  0.08 0.03 0.05 2/44 132
Load average:  0.08 0.03 0.05 1/44 132
[root@FORLINX6410]# _
```

Target Hardware Results

Case(i) Extracting Kernel Info

```
output
[root@FORLINX6410]# ./output
Status report type Standard at  Sat Jan  1 00:16:27 2000

Machine hostname: FORLINX6410
CPU Information
  BogoMIPS      : 528.79
  Features      : swp half thumb fastmult edsp java
Kernel version:
  Linux version 3.0.1
Uptime:  0:1:17
[root@FORLINX6410]#
```

Case(ii) Extracting process info at Macro level

```
output
[root@FORLINX6410]# ./output
fibarg = 30

Child 1 fib = 832040, real time =
Child 1 fib = 832040, cpu time =
Child 1 fib = 832040, user time =
Child 1 fib = 832040, kernel time

Child 2 fib = 832040, real time =
Child 2 fib = 832040, cpu time =
Child 2 fib = 832040, user time =
Child 2 fib = 832040, kernel time

Parent fib = 832040, real time =
Parent fib = 832040, cpu time = (
Parent fib = 832040, user time =
Parent fib = 832040, kernel time
[root@FORLINX6410]#
```

Conclusion

In this paper, three modes standard, short and long modes where certain parameters can be changed by using kernel info acquired using the proc file system is demonstrated. The details of individual processes running on the system(s) along with the process ID, device drivers activated are extracted dynamically. Subsequently, by using macros certain features are enabled /disabled in runtime. Also, the number of tasks in running state, priority levels, maximum number of tasks and optimal use of semaphore are made dynamic. Nested features are extended to mutual exclusion semaphore, message / Mailbox, Task suspend /resume.

References

[1] Dhruva, R, Rinku & Mohdarshad 2013, 'Design and implementation of free RTOS based online data acquisition and controlling system using cortex m3core', *International Journal of Engineering Science & Advanced Technology,* vol. 3, no.5, pp. 259-263.

[2] Jae Hwan Koh & Byoung Wook Choi 2013, 'Real-time Performance of Real-time Mechanisms for RTAI and Xenomai in Various Running Conditions', *International Journal of Control and Automation*, vol. 6, no. 1, pp. 235-245.

[3] JianFeng&Hongmei Jin 2011, **'**μC/OS-II Port for STM32F107VC Processor', *Information Engineering Letters***,** vol. 1, no. 1, pp. 1-7.

[4] Kim, W.-J., Ji, K., and Ambike, A. (2006). Real-time operating environmentfor networked control systems.Automation Science and Engineering, IEEE Transactions on[see also Robotics and Automation, IEEE Transactions on], 3(3):287–296.

[5] Koker, K. (2007).Autonomous Robots and Agents, chapter Embedded RTOS: Perfor-mance Analysis With High Precision Counters, pages 171–179. Springer Berlin / Hei-delberg.

[6] J. H. Koh and B. W. Choi, "Performance Evaluation of Real-time Mechanisms for Real-time Embedded Linux", J. of Institute of Control, Robotics and Systems (in Korean), vol. 18, no. 4, (2012), pp. 337-342.

[7] Krodel, J. and Romanski, G. (2007). Real-time operating systems and component integration considerations in integrated modular avionics systems report. Technical report,U.S. Department of Transportation - Federal Aviation Administration.

[8] J.J.Labrosse,"uC/OS-II The Real-Time Kernel", Micrium (2009).

[9] P. A. Laplante,"Real-time System Design and Analysis", Wiley-IEEE Press,(2004).

[10] Prasad, PS & Akhilesh Upadhyay 2012, 'Design of Hybrid Kernel and the Performance Improvement of the Operating System', *International Journal of Engineering and Technology*, vol. 4, no. 2, pp. 162-165.

[11] Prasanna,SL & Venkateswara Rao, M 2012 (a), 'Design of μC/ Os II RTOS Based Scalable Cost Effective Monitoring System Using Arm Powered Microcontroller', *International Journal of Scientific and Research Publications*, vol. 2, no. 4, pp.1-4.

[12] Prasanna,SL& Venkateswara Rao, M 2012 (b), 'Implementation of a Scalable μC /OS-II Based Multitasking Monitoring System', *International Journal of Computer Science and Technology*, vol. 3, no. 2, pp. 86-89.

[13] SonaliGrover 2014, 'Real-Time Operating Systems: An Overview', *International journal of innovative research in technolog*y, vol. 1, no. 4, pp. 203-207.

[14] Yan Liping&Song Kai 2011,'Improvement and test of realtime performance of embedded Linux 2.6 kernel',*International Journal of Digital Content Technology and its Applications.*vol. 5, no. 4, pp. 247-253.

[15] Huiting Hou,Gao, Dengke Liu. 2014,"A support vector machine with maximal information coefficient weighted kernel functions for regression", *systems and informatics(ICSAI),2014 2nd international conference*, Vol.2,no.3, pp.938-942.

Appendix

The code snippet for Standard mode:

```
void sampleCpuInfo () {
char lineBuf[LB_SIZE];
FILE *thisProcFile;
thisProcFile = fopen("/proc/cpuinfo", "r");
printf("CPU Information\n");
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Skip processor
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // cpu
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // model
printf("  %s", lineBuf);
fclose(thisProcFile);    }
```

The function opens *cpuinfo*file in /proc file system and reads and displays the *cputype*and *model* from the opened file.

```
void sampleVersion() {
char lineBuf[LB_SIZE];
FILE *thisProcFile;
thisProcFile = fopen("/proc/version", "r");
fscanf(thisProcFile, "%s", lineBuf);
printf("Kernel version:\n  %s", lineBuf);
fscanf(thisProcFile, "%s", lineBuf);
printf(" %s", lineBuf);
fscanf(thisProcFile, "%s", lineBuf);
printf(" %s\n", lineBuf);
fclose(thisProcFile);
}
```

The function opens *version* file in /proc file system and reads and displays the kernel version.

```
void sampleUptime() {
int i;
char lineBuf[LB_SIZE];
FILE *thisProcFile;
thisProcFile = fopen("/proc/uptime", "r");
fscanf(thisProcFile, "%s", lineBuf);
i = atoi(lineBuf);
printf("Uptime:  %d:%d:%d\n", i/3600, i/60, i%60);
fclose(thisProcFile);
}
```

The function opens *uptime* file in /proc file system and reads and displays the time elapsed.

The code snippet for Short mode:

```
void sampleStat() {
char lineBuf[LB_SIZE];
FILE *thisProcFile;
thisProcFile = fopen("/proc/stat", "r");
printf("Kernel Statistics\n");
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Jiffies
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Disk operations
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Disk reads
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Disk writes
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Disk read sectors
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Disk written sectors
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Memory pages
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Interrupts
fgets(lineBuf, LB_SIZE+1, thisProcFile);
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);     // Boot time
printf("  %s", lineBuf);
fgets(lineBuf, LB_SIZE+1, thisProcFile);
printf("  %s", lineBuf);
fclose(thisProcFile);}
```

The function opens *stat* file in /proc file system and reads and displays Jiffies, Disk read/writes, Memory pages, Interrupts, Boot time and Process started.

The code snippet for Long Mode:

```
void sampleMounts() {char lineBuf[LB_SIZE];
FILE *thisProcFile;
thisProcFile = fopen("/proc/mounts", "r");
printf("File Systems Mounted:\n");
while(fgets(lineBuf, LB_SIZE+1, thisProcFile) != NULL)
printf("  %s", lineBuf);
fclose(thisProcFile);}
```

The function opens *mounts* file in /proc file system and reads and displays the mounted devices.