Linux系统编程与调试调优

宋宝华 <baohua@kernel.org>

2016年

课程安排

主要内容

- Linux进程管理与调试
- Linux内存管理与调试
- Linux内核调试
- Linux系统性能profiling
- Linux多进程与多线程

虚拟机下载：

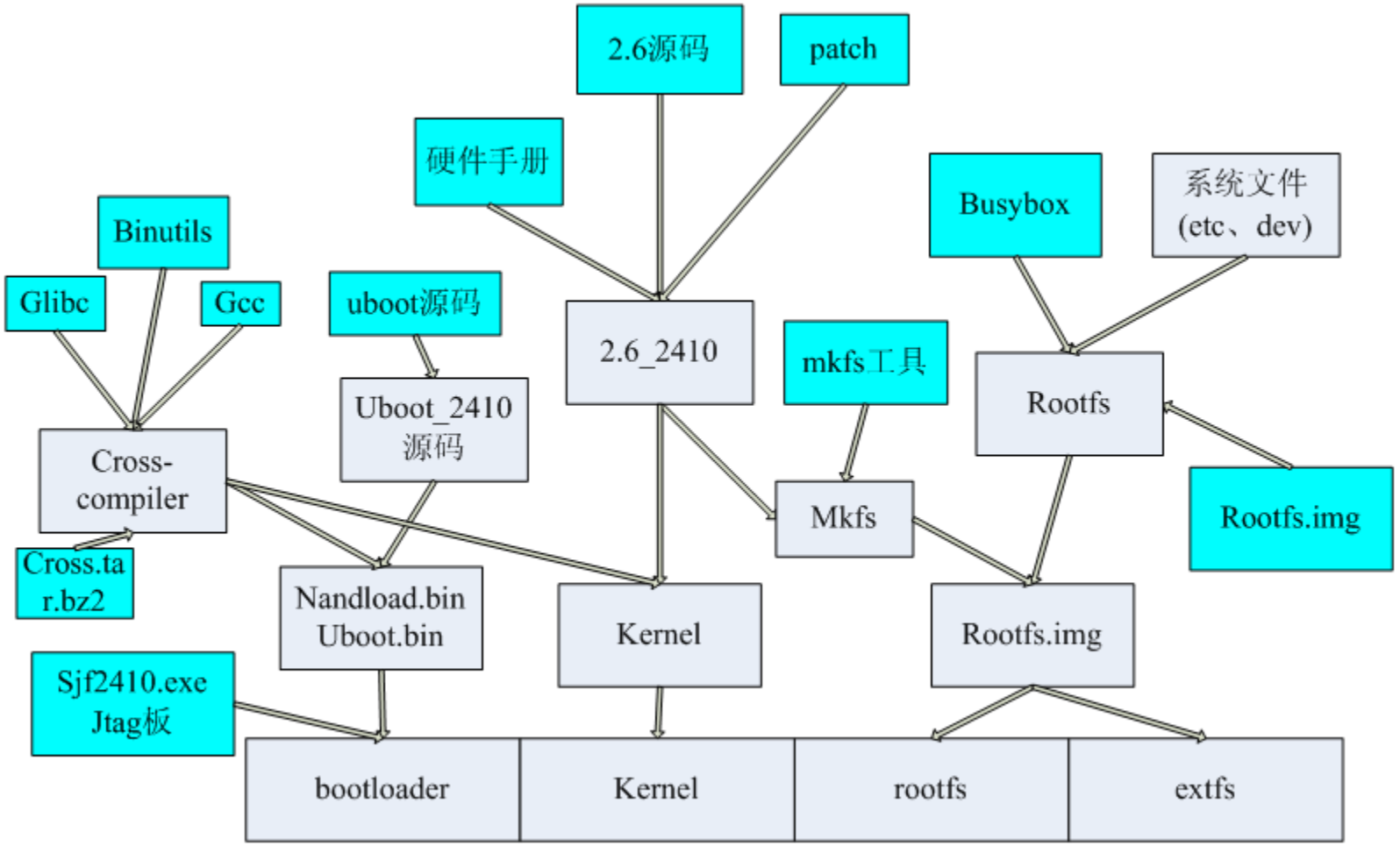
链接: http://pan.baidu.com/s/1gejDU1t 密码: vqd6

案例下载（虚拟机内已经包含案例）：

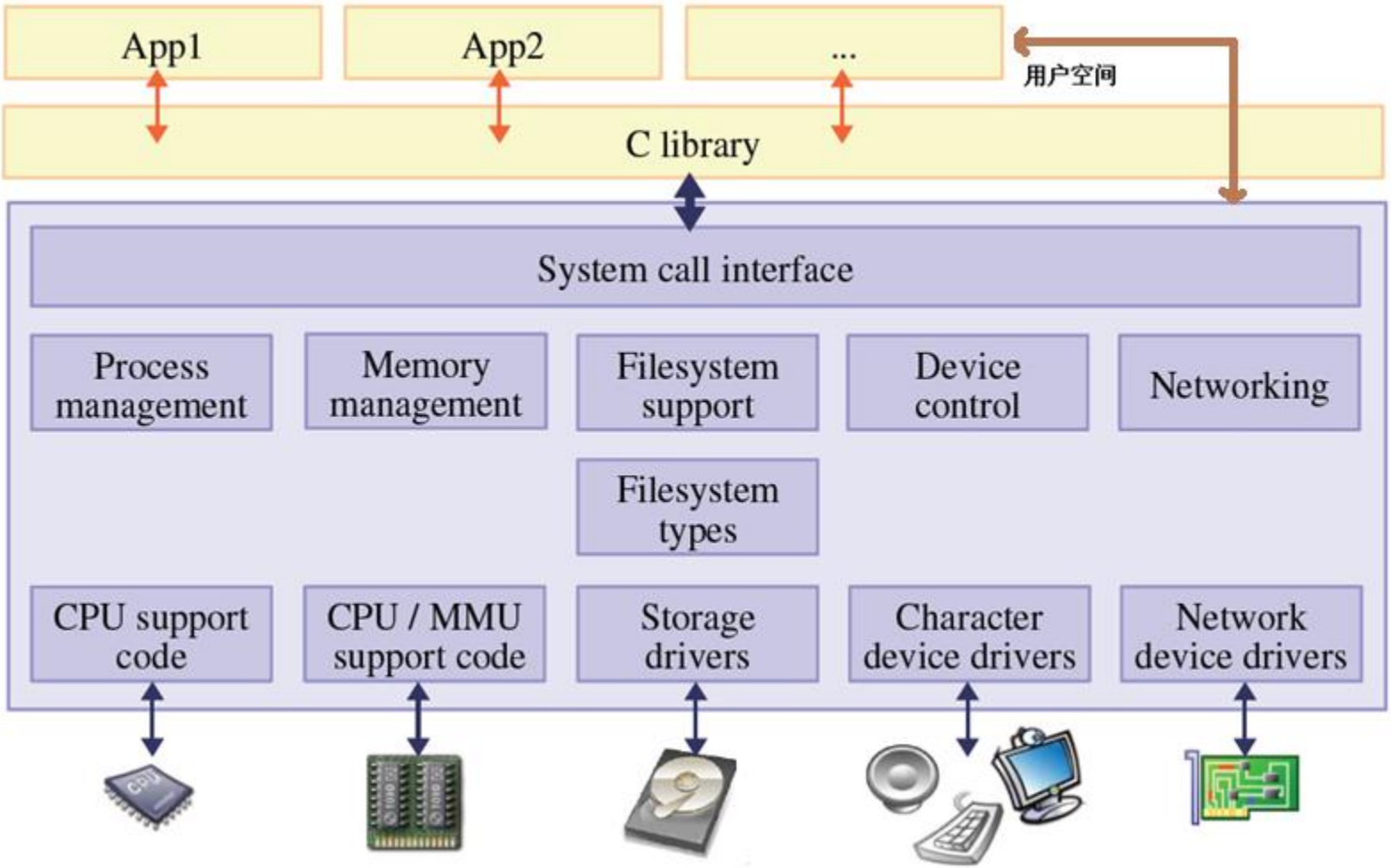
git clone https://github.com/21cnbao/training

Linux 系统开发与工具

Linux系统的组成


2.6源码
patch
硬件手册
Busybox
系统文件
(etc、dev)
Binutils
Glibc
Gcc
uboot源码
2.6_2410
mkfs工具
Rootfs
Uboot_2410
源码
Cross-
compiler
Mkfs
Rootfs.img
Cross.ta
r.bz2
Nandload.bin
Uboot.bin
Kernel
Rootfs.img
Sjf2410.exe
Jtag板
bootloader
Kernel
rootfs
extfs

Linux层次结构


App1
App2
...
用户空间
C library
System call interface
Process management
Memory management
Filesystem support
Device control
Networking
Filesystem types
CPU support code
CPU / MMU support code
Storage drivers
Character device drivers
Network device drivers

strace和ltrace

- strace:跟踪系统调用
- ltrace:跟踪库函数调用

鸟哥的Linux私房菜

strace实例

```
4  main()
5  {
6    int fd, num, pos;
7    char wr_ch[200] = "This is a test of globalmem";
8    char rd_ch[200];
9   //打开/dev/globalmem
10   fd = open("/dev/globalmem", O_RDWR, S_IRUSR | S_IWUSR);
11   if (fd != -1 )
12   {
13    //清除globalmem
14     if(ioctl(fd, MEM_CLEAR, 0) < 0)
15     {
16                        printf("ioctl command failed\n");
17    }
18     //读globalmem
19     num = read(fd, rd_ch, 200);
20     printf("%d bytes read from globalmem\n",num);
21
22   //写globalmem
23     num = write(fd, wr_ch, strlen(wr_ch));
24     printf("%d bytes written into globalmem\n",num);
25     ...
26     close(fd);
27   }
```

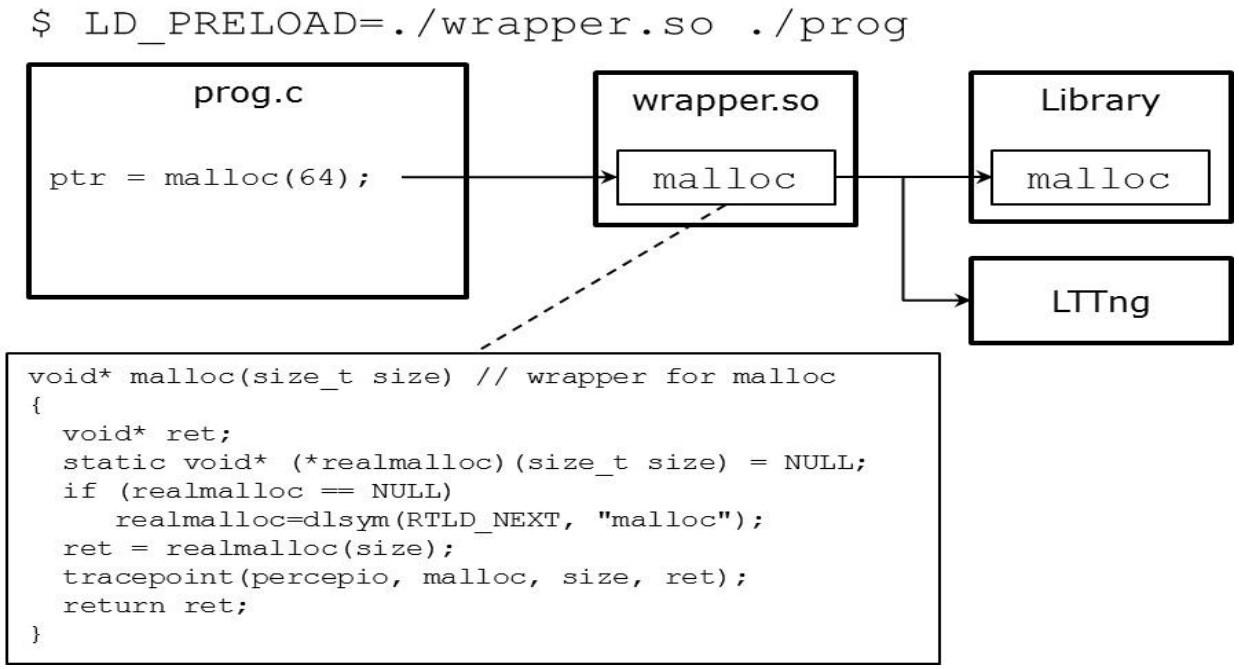
```
execve("./globalmem_test", ["./globalmem_test"], [/* 24 vars */]) = 0
...
open("/dev/globalmem", O_RDWR)          = 3          ---------打开的/dev/globalmem的fd是3
ioctl(3, FIBMAP, 0)                     = 0
read(3, 0xbff17920, 200)                = 200        ---------读取到200个字节
...
write(1, "200 bytes read from globalmem\n", 30200 bytes read from globalmem
) = 30                                  -----向标准输出设备(fd为1)写入printf中的字符串
write(3, "This is a test of globalmem", 27) = 27
write(1, "27 bytes written into globalmem\n", 3227 bytes written into globalmem
) = 32
...
```

ltrace- library call tracer

```
__libc_start_main(0x400844, 1, 0x7fff01389398, 0x400880, 0x400870
      <unfinished ...>
      _ZNSt8ios_base4InitC1Ev(0x600e2c, 65535, 0x7fff013893a8, 3,
      0x31e1752350)                    = 2
      __cxa_atexit(0x40082c, 0, 0x400960, 3,
      0x31e1752350)                                  = 0
      _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc(0x600
      d10, 0x400968, 0x7fff013893a8, 4, 0x31e1752370) = 0x600d10
      _ZNSolsEPFRSoS_E(0x600d10, 0x4006e0, 0, 0xfbad2a84, 0xffffffff
      <unfinished ...>
      _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(0x
      600d10, 0x4006e0, 0, 0xfbad2a84, 0xffffffffHello World !
      ) = 0x600d10
      _ZNSt8ios_base4InitD1Ev(0x600e2c, 0, 0x31e1752370, -1,
      0x2b2e79280530)                  = 3
```

LD_PRELOAD

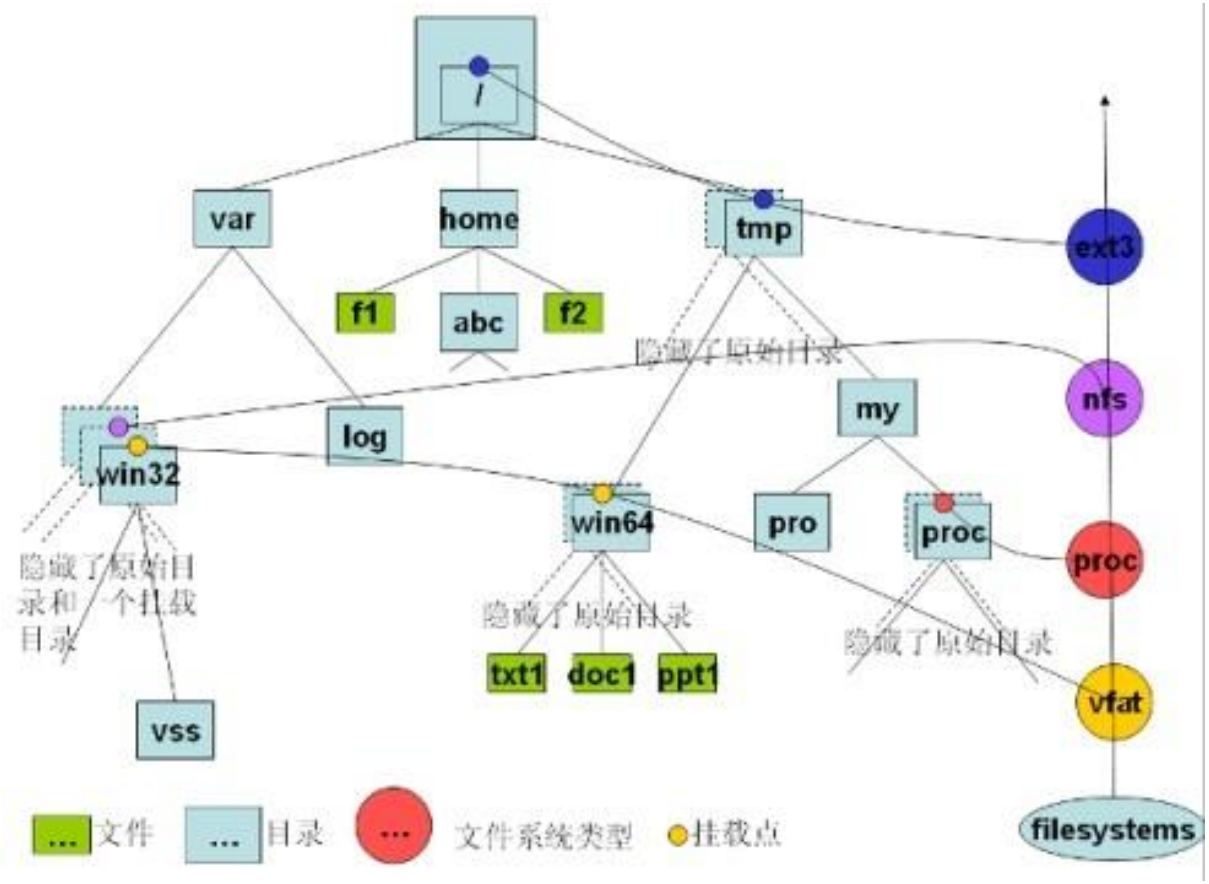
- 动态链接的时候，会将有相同符号名的符号覆盖成LD_PRELOAD指定的so文件中的符号。可以用我们自己的so库中的函数替换原来库里有的函数，从而达到hook的目的。



例如：UMEM_DEBUG=default UMEM_LOGGING=transaction LD_PRELOAD=libumem.so.1 ./Test &
export LD_PRELOAD=/usr/local/lib/liblttng-ust-fork.so

mount

- Linux使用mount机制扩展文件系统，使不同类型的文件系统可以挂载在系统的文件树的任何位置



文件系统

- /proc:基于内存的文件系统，内核的状态、计算机的属性、正在运行的进程的状态等信息。
- /sys: 基于内存的文件系统，查看和设定内核参数功能，Linux 统一设备模型管理。
- /dev: 设备文件结点
- …

init

- Busybox init
- System V init
- Upstart
- Systemd
- Android init

RTOS与Linux区别

	RTOS such as VxWorks	Linux
内核结构	微内核，内核只提供了基本的服务，如：任务管理，内存管理，中断处理等	宏内核，除了基本的服务，内核还包括文件系统，网络协议等
运行模式	应用程序运行在“实模式”下，无用户模式和内核模式之分	采用“保护模式”，用户进程、线程运行在用户模式下，内核线程运行于内核模式
内存访问和内存保护	内核采用实存储管理方式，所有任务运行于同一物理地址空间，用户程序直接操作物理地址，不能直接地提供内存保护，不能防止错误蔓延	内核采用虚拟存储管理方式，用户具有独立的地址空间，用户进程只能访问本进程的虚拟空间，提供了内存保护，可以防止错误蔓延
执行单元	任务	进程、线程
请求内核服务方式	函数调用，更快	系统调用，更安全
实时性	硬实时	软实时
发行版	windriver vxworks	Motivista Linux、Lineo Embedix、Bluecat Linux

工具链

- **GNU Binutils**
  - GNU Binutils 的主要工具有两个，一个是连接程序ld，另外一个是汇编程序as。其主要目的是为GNU 系统，提供汇编和连接工具。
- **GNU GCC**
  - GNU GCC 就是上面提到的GCC，GCC 主要是为GNU 系统提供C 编译器。现在支持多种语言，这其中包括C/C++、Fortran、Java、Objective-C、甚至还有Ada 。
- **GNU GLibc**
  - 用于定义系统调用和其它一些基本的函数调用。

工具链提供者

- **CodeSourcery**
- **Linaro**

Binutils

- addr2line - Converts addresses into filenames and line numbers.
- ar - A utility for creating, modifying and extracting from archives.
- gprof - Displays profiling information.
- nm - Lists symbols from object files.
- objcopy - Copys and translates object files.
- objdump - Displays information from object files.
- ranlib - Generates an index to the contents of an archive.
- readelf - Displays information from any ELF format object file.
- size - Lists the section sizes of an object or archive file.
- strings - Lists printable strings from files.
- strip - Discards symbols.
- gcov - gcov is a tool you can use in conjunction

gcc + glibc + linux-header + make

- The "make headers_install" command exports the kernel's header files in a form suitable for use by userspace programs.
- Kernel headers are **backwards compatible**, but not forwards compatible. This means that a program built against a C library using older kernel headers should run on a newer kernel (although it may not have access to new features), but a program built against newer kernel headers may not work on an older kernel.
- Embedded GLIBC (EGLIBC) was a variant of the GNU C Library (GLIBC) that was designed to work well on embedded systems. EGLIBC strived to be source and binary compatible with GLIBC. EGLIBC's goals included reduced footprint, configurable components, better support for cross-compilation and cross-testing.

/usr/arm-linux-gnueabi/include/linux/version.h

GCC compilation process

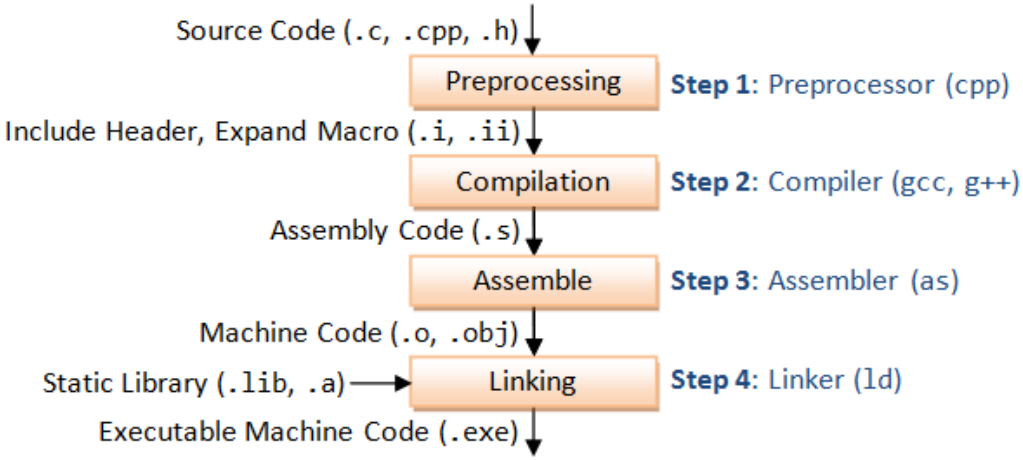
- Preprocessing
- Compilation
- Assembly
- Linking

```
$ ls
helloworld.c
$ gcc -Wall -save-temps helloworld.c -o helloworld
$ ls
helloworld  helloworld.c  helloworld.i helloworld.o helloworld.s
```

The temporary files produced by -save-temps flag in one go can be produced one by one by using the gcc flags -E, -C and -S at each of the preprocessing, compilation and assembly steps respectively.

preprocessing

- **The preprocessing step:** all the header files that you have included in your program are actually expanded and included in source code of your program. Other than this, all the macros are replaced by their respective values all over the code and all the comments are stripped off.
- **The compilation step:** the source code is actually compiled by the compiler to produce an assembly code. Assembly code consist of set of instructions that determine what you program wants to do.
- **The assembly step:** the assembler understands the assembly instructions and converts each of them in to the corresponding machine level code or a bit stream that consists of 0's and 1's. This machine level code is known as object code and this code can be executed by the processor.
- **The linking step:** Once the object file containing the machine code is produced in the step above, the linking step makes sure that all the undefined symbols in code are resolved.



Options: -O, -O1, -O2, -O3, -O0, -Os

- Various levels of optimization of the code
- -O1 to -O3 are various degrees of optimization targeted for speed
- If -O is added, then the code size is considered
- -O0 means “no optimization”
- -Os targets generated code size (forces not to use optimizations resulting in bigger code).

objdump

- objdump : extracts pieces from object file

- objdump -D ; disassemble
- objdump -x ; shows headers
- objdump -t ; symbol table
- objdump -r ; relocation information (show offsets into file)
- objdump -s ; show all segments
- …

- xxd/bvi : hex dump

DWARF

- DWARF is a debugging format used to describe programs in C and other similar programming languages. It is most widely associated with the ELF object format but it has been used with other object file formats.

readelf

gdb

dwarfdump

Dynamic Libraried

- Compiling with Position Independent Code

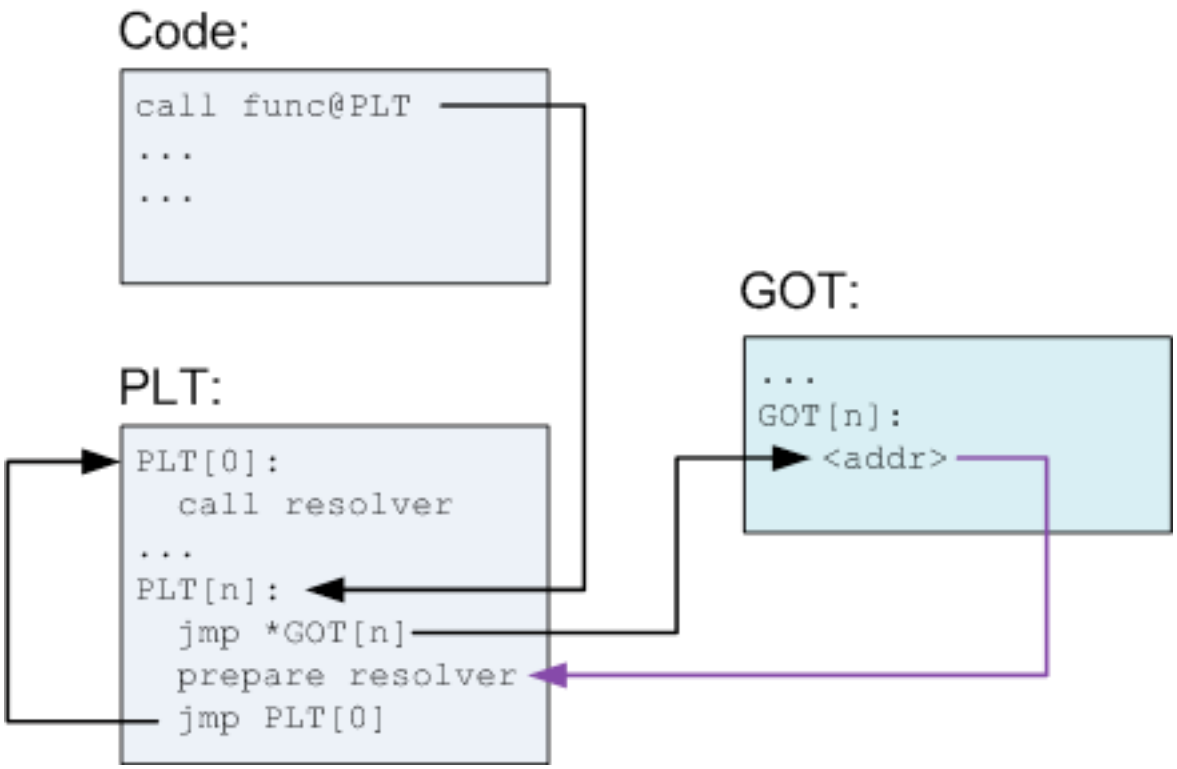
```
gcc -fpic -g -c -Wall a.c
gcc -fpic -g -c -Wall b.c
```

- Creating a shared library from an object file

```
gcc -shared -Wl,-soname,libmystuff.so.1 -o libmystuff.so.1.0.1 a.o b.o
```

PLT/GOT

Procedure Linkage Table (PLT)
Global Offset Table (GOT)



DL APIs

```
void invoke_method( char *lib, char *method, float argument )
{
  void *dl_handle;
  float (*func)(float);
  char *error;

  /* Open the shared object */
  dl_handle = dlopen( lib, RTLD_LAZY );
  if (!dl_handle) {
    printf( "!!! %s\n", dlerror() );
    return;
  }

  /* Resolve the symbol (method) from the object */
  func = dlsym( dl_handle, method );
  error = dlerror();
  if (error != NULL) {
    printf( "!!! %s\n", error );
    return;
  }

  /* Call the resolved method and print the result */
  printf("  %f\n", (*func)(argument) );

  /* Close the object */
  dlclose( dl_handle );

  return;
}
```

LD_DEBUG

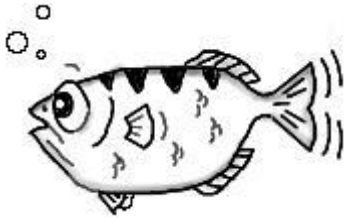
```
baohua@baohua-VirtualBox:~$ LD_DEBUG=libs ls
      3630:     find library=libselinux.so.1 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libselinux.so.1
      3630:
      3630:     find library=libacl.so.1 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libacl.so.1
      3630:
      3630:     find library=libc.so.6 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libc.so.6
      3630:
      3630:     find library=libpcre.so.3 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libpcre.so.3
      3630:
      3630:     find library=libdl.so.2 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libdl.so.2
      3630:
      3630:     find library=libattr.so.1 [0]; searching
      3630:      search cache=/etc/ld.so.cache
      3630:       trying file=/lib/i386-linux-gnu/libattr.so.1
      3630:
      3630:
      3630:     calling init: /lib/i386-linux-gnu/libc.so.6
      3630:
      3630:
      3630:     calling init: /lib/i386-linux-gnu/libattr.so.1
      3630:
```

Barriers and compilers

- The compiler can:
  - Reorder code as long as it correctly maintains data flow dependencies within a function and with called functions
  - Reorder the *execution* of code to optimize performance
- The processor can:
  - Reorder instruction execution as long as it correctly maintains register flow dependencies
  - Reorder memory modification as long as it correctly maintains data flow dependencies
  - Reorder the execution of instructions (for performance optimization)

GDB

- GDB is a debugger for free.
- A debugger is a program that runs other programs allowing the user to exercise control over these programs, and to examine variables when problems arise.
- The program being debugged can be written in C, C++, Pascal, Objective-C. Those programs might be executing on the same machine as GDB (native) or on another machine (remote). GDB can run on most popular UNIX
Microsoft Windows variants.
- The latest version of GDB, version



GDB cheatsheet - page 1

Running

`# gdb <program> [core dump]`
: Start GDB (with optional core dump).

`# gdb --args <program> <args…>`
: Start GDB and pass arguments

`# gdb --pid <pid>`
: Start GDB and attach to process.

`set args <args...>`
: Set arguments to pass to program to be debugged.

`run`
: Run the program to be debugged.

`kill`
: Kill the running program.

Breakpoints

`break <where>`
: Set a new breakpoint.

`delete <breakpoint#>`
: Remove a breakpoint.

`clear`
: Delete all breakpoints.

`enable <breakpoint#>`
: Enable a disabled breakpoint.

`disable <breakpoint#>`
: Disable a breakpoint.

Watchpoints

`watch <where>`
: Set a new watchpoint.

`delete/enable/disable <watchpoint#>`
: Like breakpoints.

<where>

`function_name`
: Break/watch the named function.

`line_number`
: Break/watch the line number in the current source file.

`file:line_number`
: Break/watch the line number in the named source file.

Conditions

`break/watch <where> if <condition>`
: Break/watch at the given location if the condition is met.
Conditions may be almost any C expression that evaluate to true or false.

`condition <breakpoint#> <condition>`
: Set/change the condition of an existing break- or watchpoint.

Examining the stack

`backtrace`
`where`
: Show call stack.

`backtrace full`
`where full`
: Show call stack, also print the local variables in each frame.

`frame <frame#>`
: Select the stack frame to operate on.

Stepping

`step`
: Go to next instruction (source line), diving into function.

`next`
: Go to next instruction (source line) but don't dive into functions.

`finish`
: Continue until the current function returns.

`continue`
: Continue normal execution.

Variables and memory

`print/format <what>`
: Print content of variable/memory location/register.

`display/format <what>`
: Like „print“, but print the information after each stepping instruction.

`undisplay <display#>`
: Remove the „display“ with the given number.

`enable display <display#>`
`disable display <display#>`
: En- or disable the „display“ with the given number.

`x/nfu <address>`
: Print memory.
n: How many units to print (default 1).
f: Format character (like „print“).
u: Unit.

Unit is one of:

b: Byte,
h: Half-word (two bytes)
w: Word (four bytes)
g: Giant word (eight bytes)).

GDB cheatsheet - page 2

Format

`a`	Pointer.
`c`	Read as integer, print as character.
`d`	Integer, signed decimal.
`f`	Floating point number.
`o`	Integer, print as octal.
`s`	Try to treat as C string.
`t`	Integer, print as binary (*t* = „two“).
`u`	Integer, unsigned decimal.
`x`	Integer, print as hexadecimal.

<what>

`expression`
: Almost any C expression, including function calls (must be prefixed with a cast to tell GDB the return value type).

`file_name::variable_name`
: Content of the variable defined in the named file (static variables).

`function::variable_name`
: Content of the variable defined in the named function (if on the stack).

`{type}address`
: Content at *address*, interpreted as being of the C type *type*.

`$register`
: Content of named register. Interesting registers are $esp (stack pointer), $ebp (frame pointer) and $eip (instruction pointer).

Threads

`thread <thread#>`
: Chose thread to operate on.

Manipulating the program

`set var <variable_name>=<value>`
: Change the content of a variable to the given value.

`return <expression>`
: Force the current function to return immediately, passing the given value.

Sources

`directory <directory>`
: Add *directory* to the list of directories that is searched for sources.

`list`
`list <filename>:<function>`
`list <filename>:<line_number>`
`list <first>,<last>`
: Shows the current or given source context. The *filename* may be omitted. If *last* is omitted the context starting at *start* is printed instead of centered around it.

`set listsize <count>`
: Set how many lines to show in „list“.

Signals

`handle <signal> <options>`
: Set how to handle signles. Options are:

  (no)print: (Don't) print a message when signals occurs.

  (no)stop: (Don't) stop the program when signals occurs.

  (no)pass: (Don't) pass the signal to the program.

Informations

`disassemble`
`disassemble <where>`
: Disassemble the current function or given location.

`info args`
: Print the arguments to the function of the current stack frame.

`info breakpoints`
: Print informations about the break- and watchpoints.

`info display`
: Print informations about the „displays“.

`info locals`
: Print the local variables in the currently selected stack frame.

`info sharedlibrary`
: List loaded shared libraries.

`info signals`
: List all signals and how they are currently handled.

`info threads`
: List all threads.

`show directories`
: Print all directories in which GDB searches for source files.

`show listsize`
: Print how many are shown in the „list“ command.

`whatis variable_name`
: Print type of named variable.

GDB – multi-threads

- Switch threads

(gdb) info threads

…

8 Thread 709 __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15

7 Thread 627 __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15

6 Thread 626 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

* 5 Thread 623 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

4 Thread 622 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

…

(gdb) thread 8

[Switching to thread 8 (Thread 709)]#0 __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15

15 ldmfd sp!, {r4, r7}

(gdb) info threads

…

9 Thread 710 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

* 8 Thread 709 __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15

7 Thread 627 __ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:15

6 Thread 626 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

5 Thread 623 __futex_syscall3 () at bionic/libc/arch-arm/bionic/atomics_arm.S:200

…

- Global mode

(gdb) set scheduler-locking off

- Single thread mode

(gdb) set scheduler-locking on

GDB – attach one process

- gdb attach <进程ID>
- detach: 当你调试结束之后,可以使用该命令断开进程与gdb的连接(结束gdb对进程的控制),在这个命令执行之后,你所调试的那个进程将继续运行;

- gdbserver :1234 --attach 2758

Core Dump

- # ulimit -c unlimited
- # echo "core_%e.%p" > /proc/sys/kernel/core_pattern
- # /data/i2c-util /dev/i2c-0 0x10 1 0

Segmentation fault (core dumped)

```
sh-4.2# gdb -c core /data/i2c-util
GNU gdb (Linaro GDB) 7.3-2011.10
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-none-linux-gnueabi".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /data/i2c-util...done.

warning: exec file is newer than core file.
[New LWP 1212]
BFD: /system/bin/linker: warning: sh_link not set for section `.ARM.exidx'
Core was generated by `/data/i2c-util /dev/i2c-0 0x10 1 0'.
Program terminated with signal 11, Segmentation fault.
#0  main (argc=5, argv=0xbee58a54) at external/i2c-util/i2c-util.c:131
131     external/i2c-util/i2c-util.c: No such file or directory.
        in external/i2c-util/i2c-util.c
(gdb) bt
#0  main (argc=5, argv=0xbee58a54) at external/i2c-util/i2c-util.c:131
```

GDB – signals

- ‘i handle’ or ‘i signals’
  - Print a table of all the signals and how gdb has been told to handle each one.

- handle signal [keywords...]
  - keywords: nostop|stop, print|noprint and pass|nopass

Ex:

handle SIG35 nostop print pass

handle SIG36 stop (implies the ‘print’ as well)

handle SIG37 nostop print nopass

handle SIG38 nostop noprint nopass

GDB – call

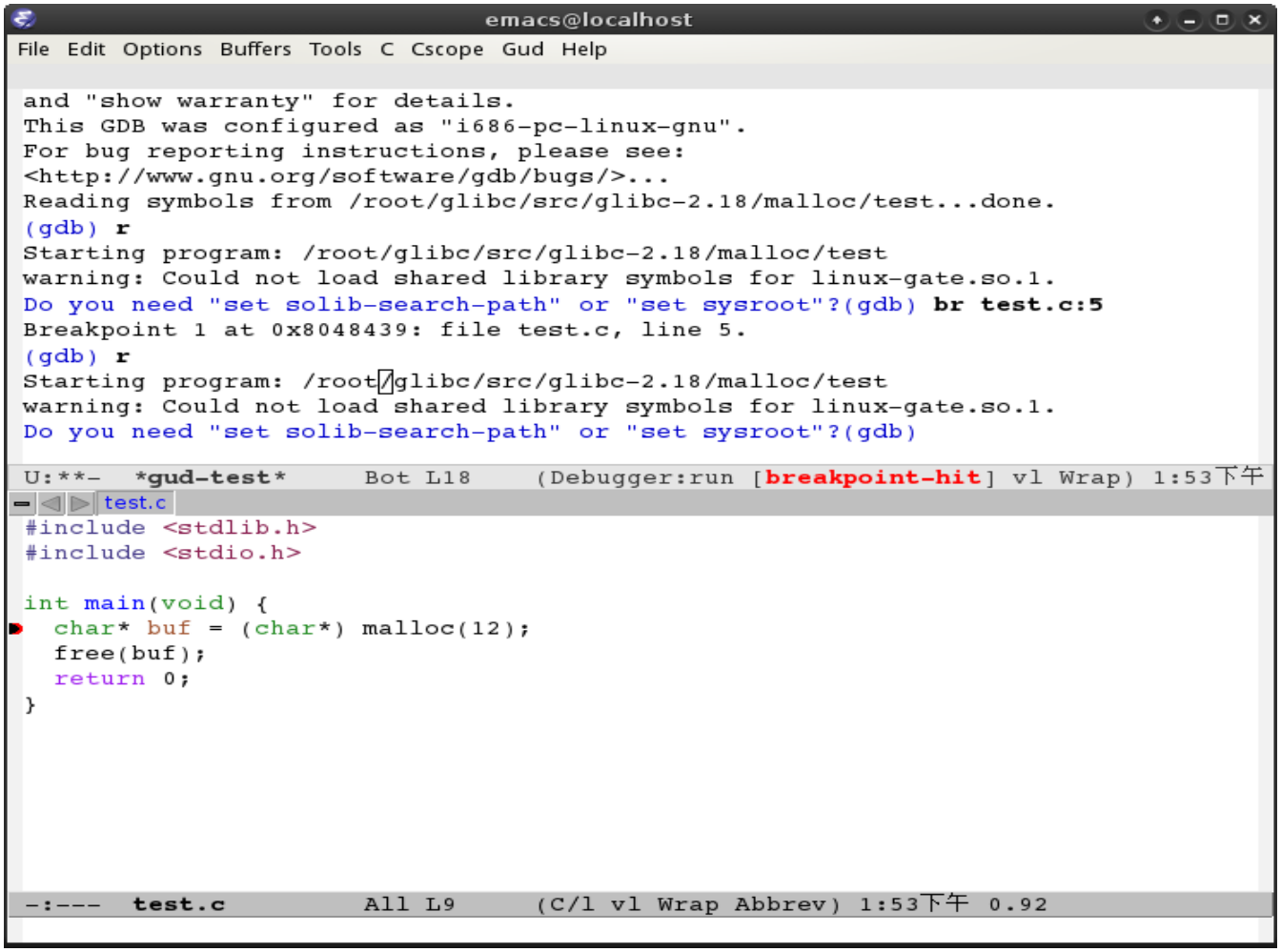
- call name 调用和执行一个函数

(gdb) call gen_and_sork(1234,1,0)

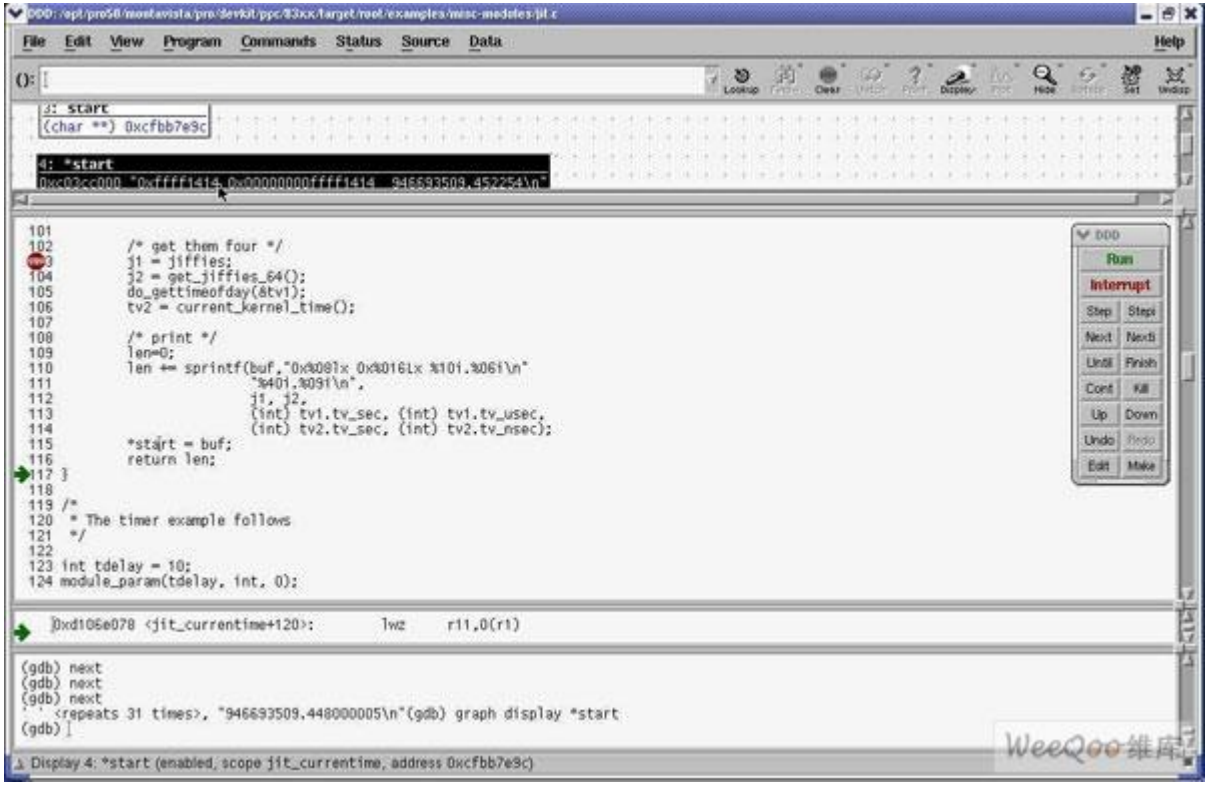
(gdb) call printf(“abcd”)

$1=4

GDB – TUI

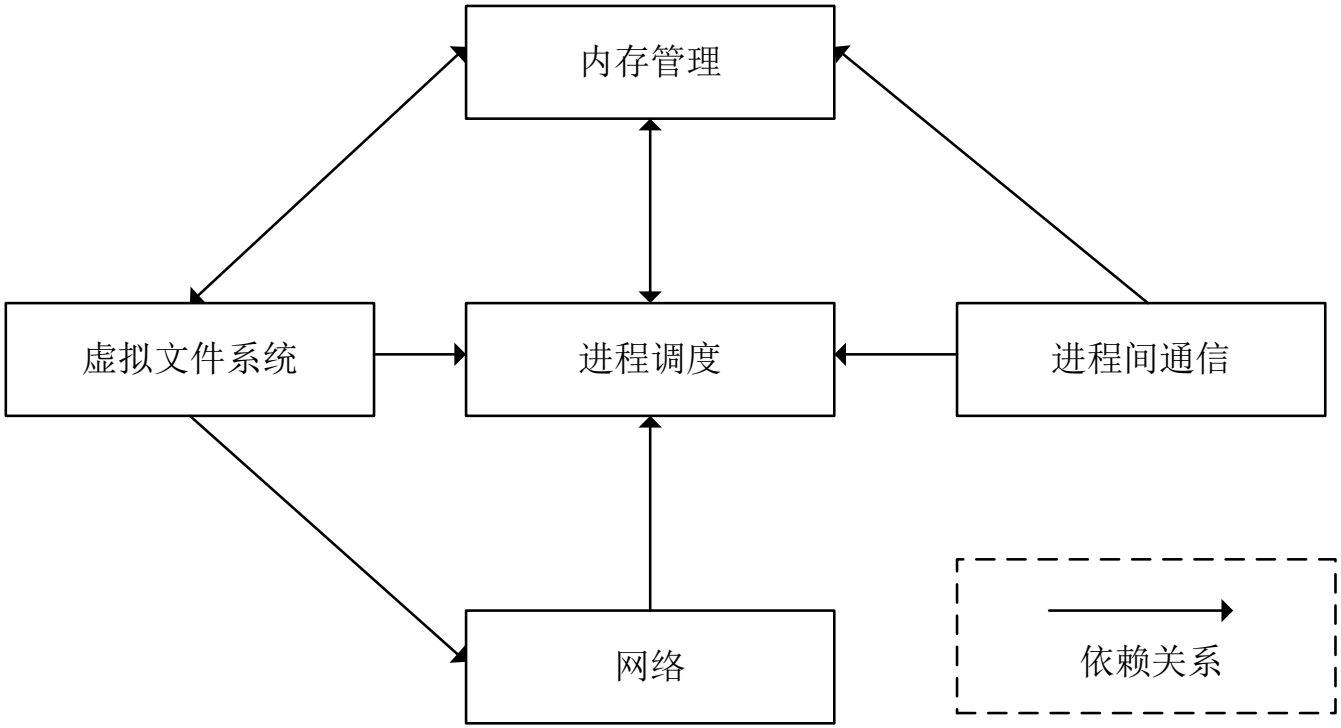

emacs@localhost
File Edit Options Buffers Tools C Cscope Gud Help
and "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/glibc/src/glibc-2.18/malloc/test...done.
(gdb) r
Starting program: /root/glibc/src/glibc-2.18/malloc/test
warning: Could not load shared library symbols for linux-gate.so.1.
Do you need "set solib-search-path" or "set sysroot"?(gdb) br test.c:5
Breakpoint 1 at 0x8048439: file test.c, line 5.
(gdb) r
Starting program: /root/glibc/src/glibc-2.18/malloc/test
warning: Could not load shared library symbols for linux-gate.so.1.
Do you need "set solib-search-path" or "set sysroot"?(gdb)
U:**- *gud-test* Bot L18 (Debugger:run [breakpoint-hit] vl Wrap) 1:53下午
test.c
#include <stdlib.h>
#include <stdio.h>
int main(void) {
char* buf = (char*) malloc(12);
free(buf);
return 0;
}
-:--- test.c All L9 (C/l vl Wrap Abbrev) 1:53下午 0.92

GDB – DDD


File Edit View Program Commands Status Source Data Help
3: start
(char **) 0xcfbb7e9c
4: *start
0xc03cc000 "0xffff1414 0x00000000ffff1414 946693509.452254\n"
/* get them four */
j1 = jiffies;
j2 = get_jiffies_64();
do_gettimeofday(&tv1);
tv2 = current_kernel_time();
/* print */
len=0;
len += sprintf(buf,"0x%08lx 0x%016Lx %10i.%06i\n"
"%40i.%09i\n",
j1, j2,
(int) tv1.tv_sec, (int) tv1.tv_usec,
(int) tv2.tv_sec, (int) tv2.tv_nsec);
*start = buf;
return len;
}
/*
* The timer example follows
*/
int tdelay = 10;
module_param(tdelay, int, 0);
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
0xd106e078 <jit_currentime+120>: lwz r11,0(r1)
(gdb) next
(gdb) next
(gdb) next
(gdb) graph display *start
Display 4: *start (enabled, scope jit_currentime, address 0xcfbb7e9c)

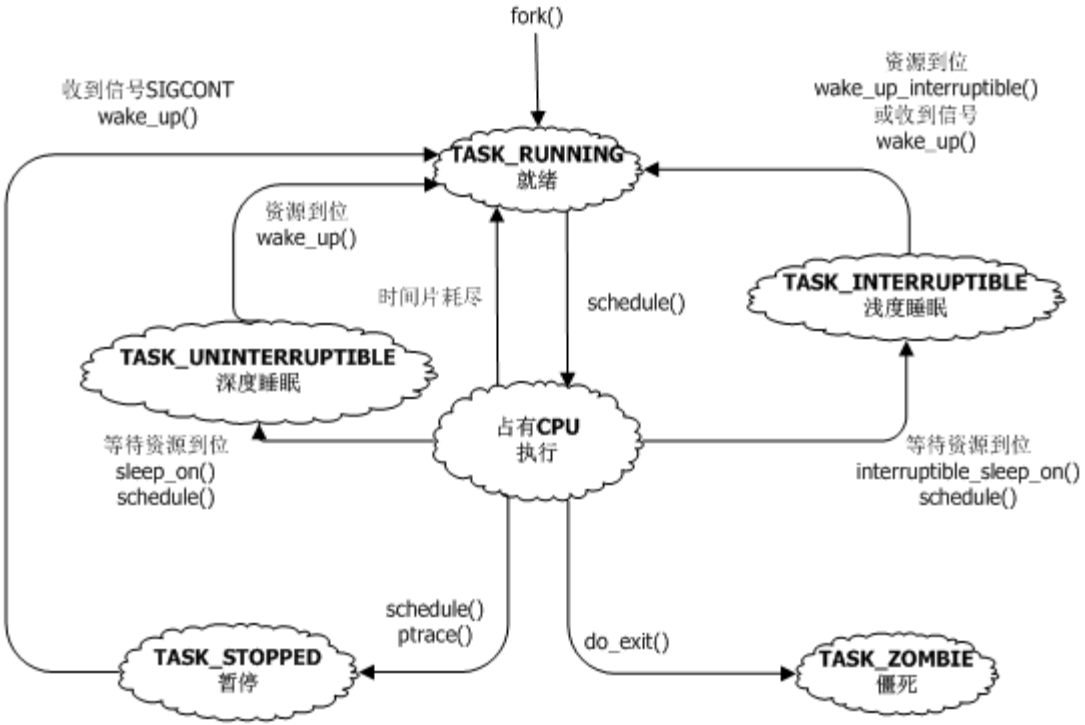
Linux内核总体

Linux内核的组成



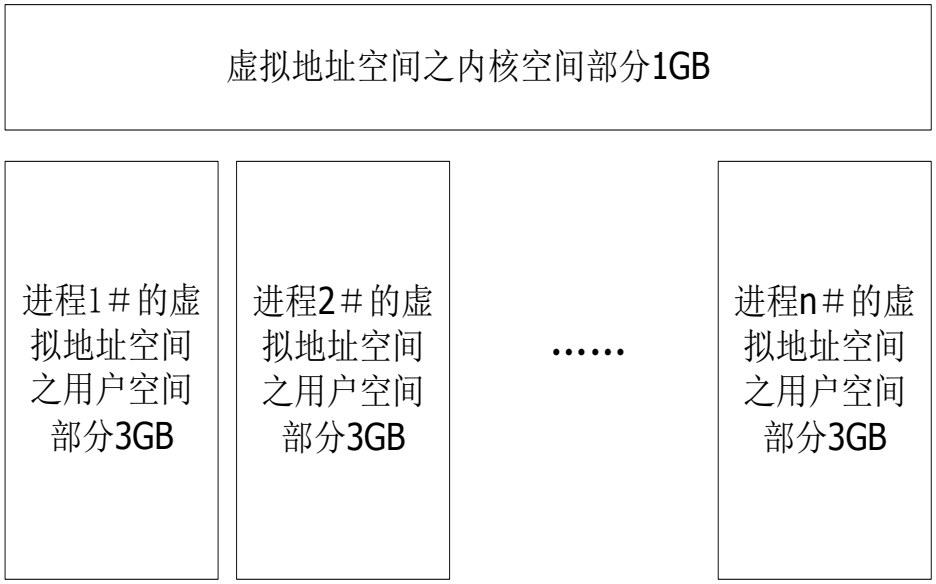
Linux内核的组成—进程调度

- 多任务，共享CPU：需要调度
- 调度的方式：通行与阻止
- 调度的结果：状态切换



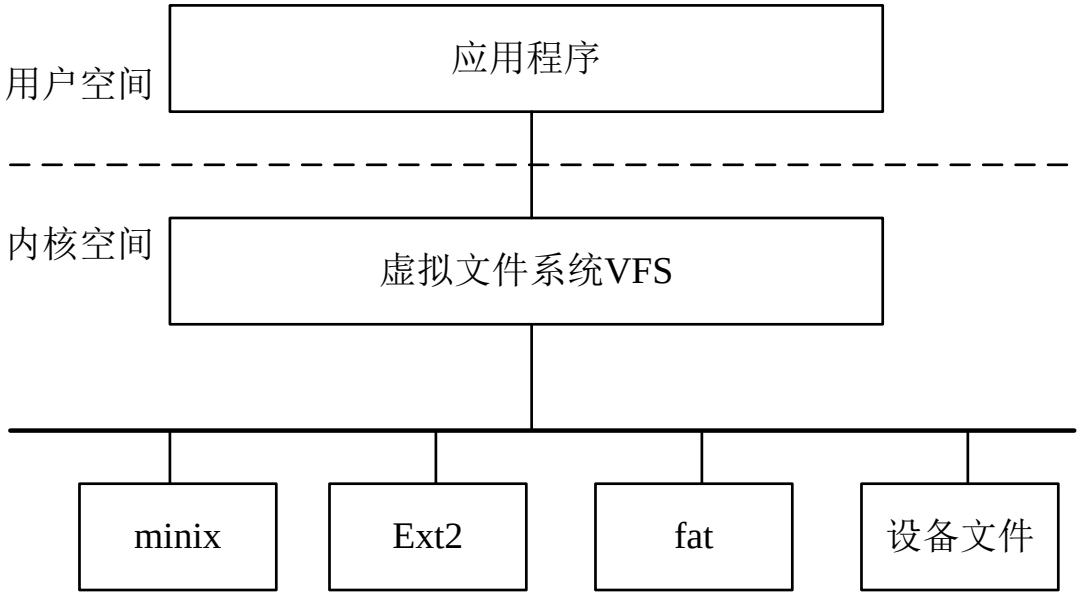
Linux内核的组成—内存管理

- 每个进程都看到4GB的内存
- 依赖于硬件MMU的支持



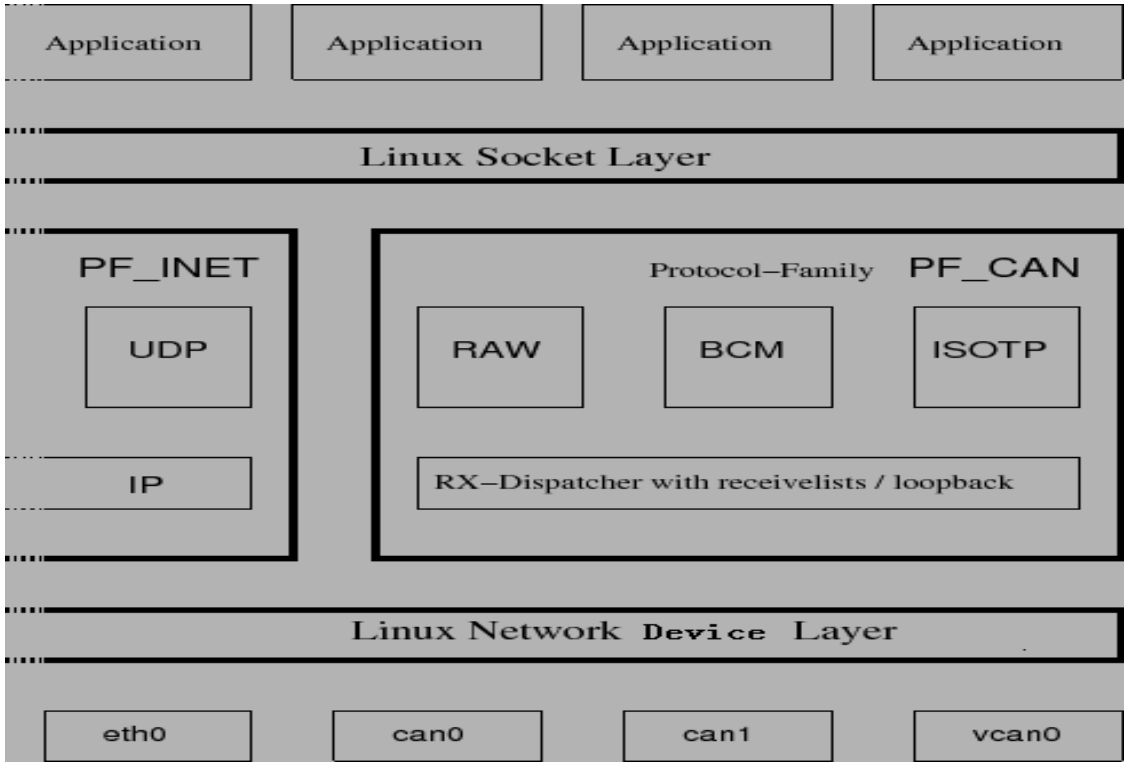
Linux内核的组成—文件系统

- 虚拟文件系统（VFS）隐藏硬件的具体细节，为所有的设备提供了统一的接口
- 磁盘/FLASH文件系统：数据在存储设备上的组织



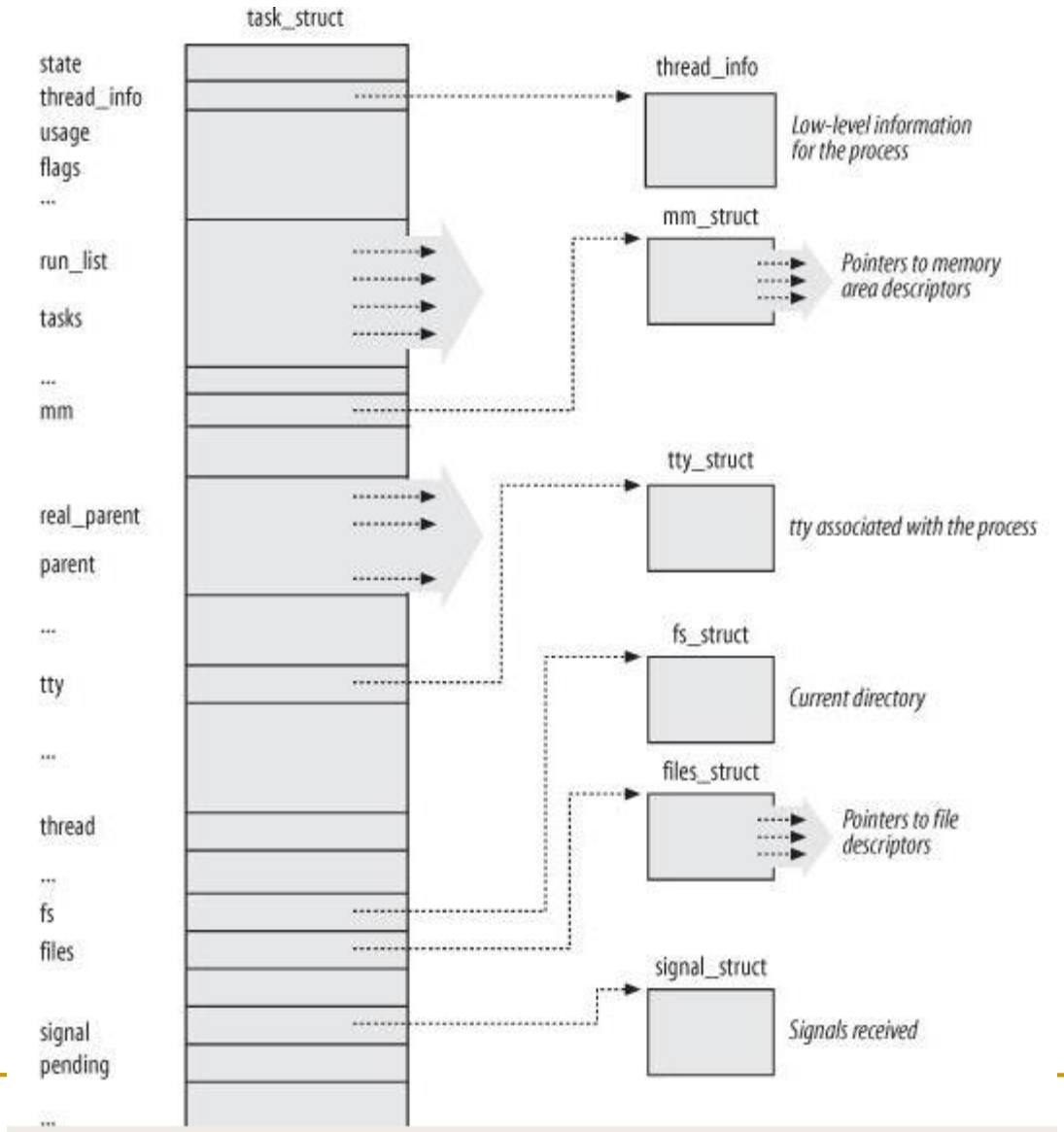
Linux内核的组成—网络

- 网络协议栈
- 网络设备驱动



Linux进程调度与调试

Linux进程描述符:task_struct



Linux进程描述符:PID

- 新进程的PID是上一个进程PID+1，最大值为32767，64位系统可达4194303

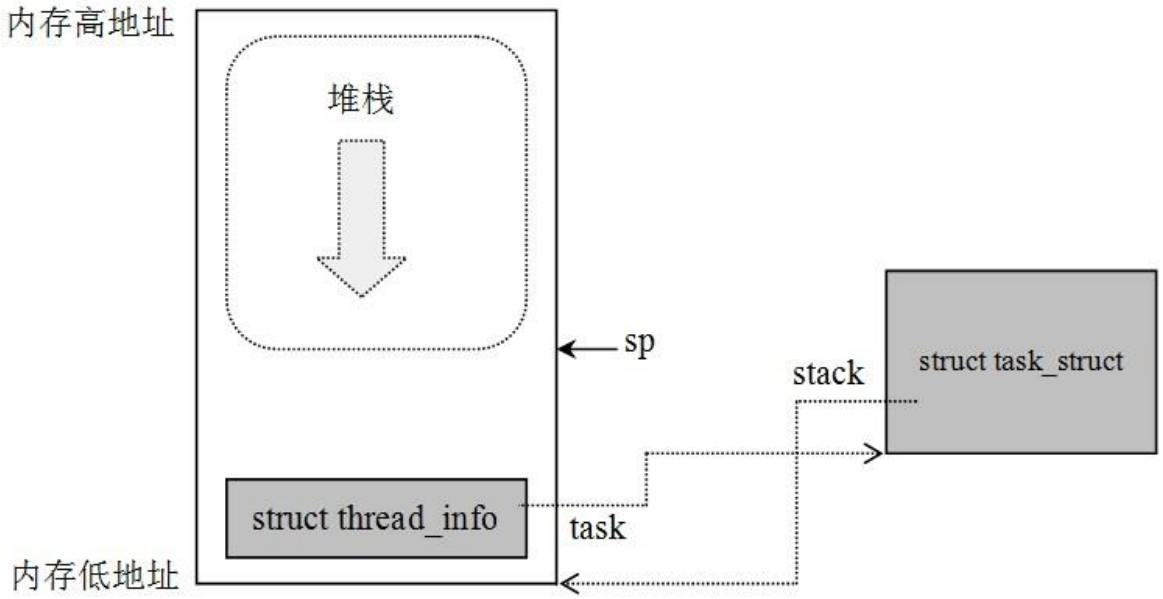
>cat /proc/sys/kernel/pid_max

32768

- PID的回收：pidmap_array每一位表示某PID是否被占用，32位系统中pidmap_array占据一页。
- Linux每个进程和轻量级进程都有一个不同的PID，而POSIX 1003.1c要求多线程应用中的每个线程有相同的PID，因此Linux使用了线程群Leader的PID作为同群中其他轻量级进程的tgid，并在getpid()系统调用返回tgid。

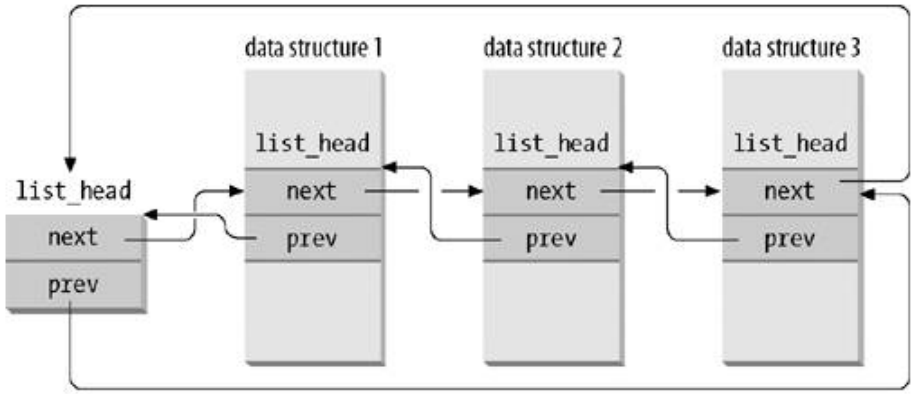
Linux进程描述符:thread_info和内核栈

- 内核栈：8KB-52byte=8140
- esp & 0xfffe0000可以得到目前的thread_info，通过current_thread_info()->task可以得到目前的task_struct（current宏）

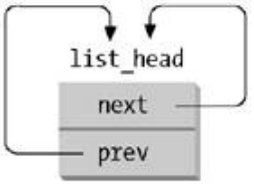


Linux进程描述符:task_struct链表

- List_head数据结构：通过嵌入宿主数据结构将其宿主链接位双向链



(a) a doubly linked listed with three elements



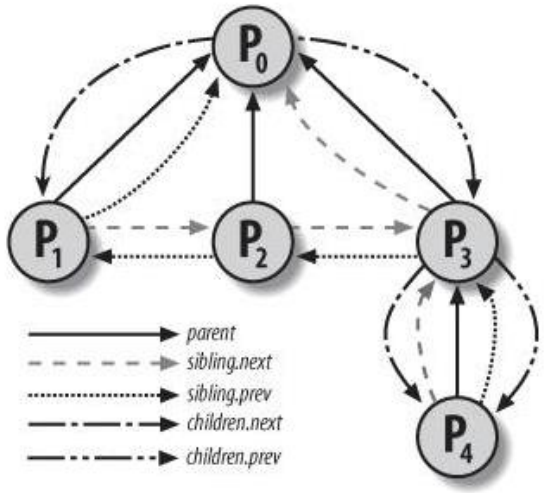
(b) an empty doubly linked list

- 前一个进程：list_entry(task->tasks.prev, struct task_struct, tasks)
- 后一个进程： list_entry(task->tasks.next, struct task_struct, tasks)
- 遍历所有进程:

```
#define for_each_process(p) \
for (p=&init_task; (p=list_entry((p)->tasks.next, \
                    struct task_struct, tasks) \
                   ) != &init_task; )
```

Linux进程描述符:进程父子关系

- parent,children,sibling



- 遍历所有子进程

```
struct task_struct *task; struct list_head *list;
list_for_each(list, &current->children) {
        task = list_entry(list, struct task_struct, sibling);
        /* task now points to one of current's children */

}
```

- PID哈希表：快速通过PID得到task_struct

```
static inline struct task_struct *find_task_by_pid(int pid)
{
        struct task_struct *p, **htable = &pidhash[pid_hashfn(pid)];
        for(p = *htable; p && p->pid != pid; p = p->pidhash_next);
        return p;
}
```

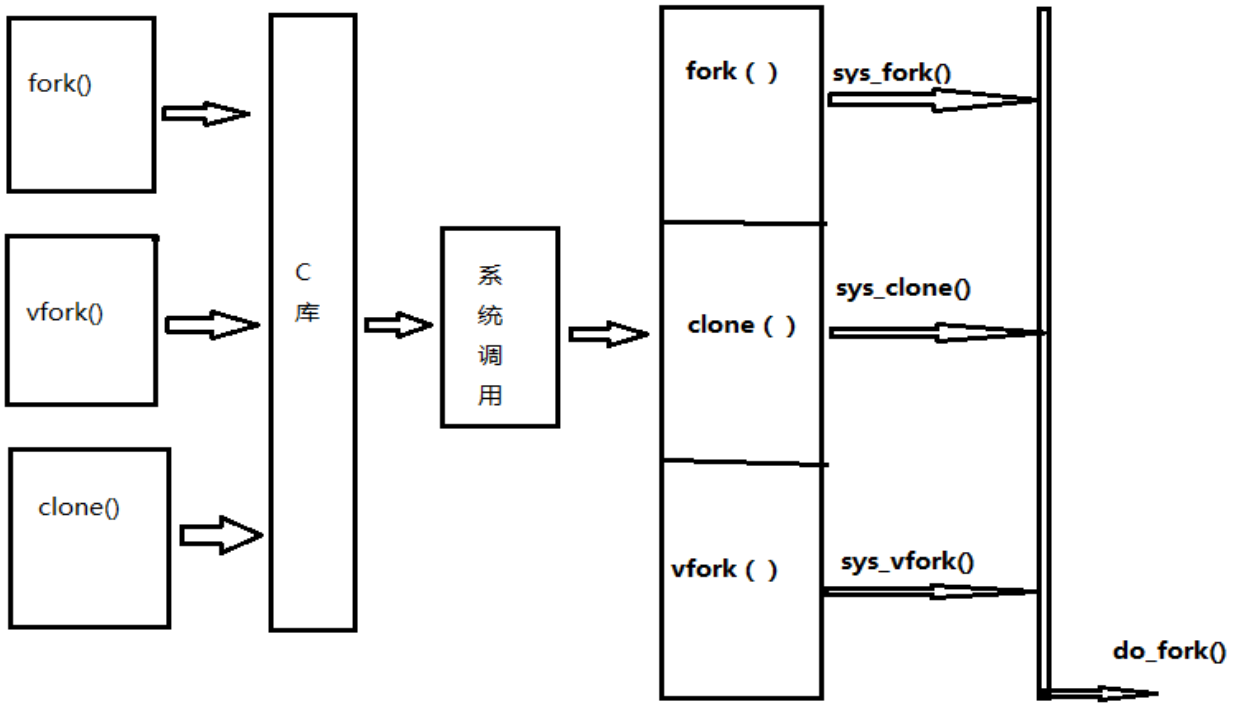
Linux进程生命周期：创建

- 写时拷贝

传统的fork()直接讲父进程资源拷贝给新进程，效率低下。

copy_on_write：父子进程共享同一拷贝，只有在写时才复制。

- clone()、fork()和vfork()



Linux进程生命周期：进程0、进程1

◆进程0，Idle进程，使用静态初始化的变量：

INIT_TASK: init_task, task_struct

INIT_THREAD_INFO: init_thread_union, thread_info和内核栈

INIT_MM

INIT_FS

INIT_FILES

INIT_SIGNALS

INIT_SIGHAND

Page Global Directory: swapper_pg_dir

执行时机：当没有其他处于TASK_RUNNING的进程时

◆进程1

(1) start_kernel( ) -> kernel_thread(init, NULL, CLONE_FS|CLONE_SIGHAND); init进程

(2) init( ) ->execve( ) 加载可执行init程序

Linux进程生命周期：内核线程创建

- 内核线程

只有内核空间，没有用户空间

*int kernel_thread(int (*fn)(void *), void * arg, unsigned long flags)*
->do_fork(flags | CLONE_VM | CLONE_UNTRACED, 0, &regs, 0, NULL, NULL);

keventd
kapmd
kswapd
pdflush
kblockd
ksoftirqd

Linux进程生命周期：终止

- *exit_group( ) 系统调用-> do_group_exit( )*

exit all threads in a process

This call is present since Linux 2.5.35.

_exit( ) 系统调用->do_exit( )

- *wait( )系统调用：*

在父进程调用wait()后，子进程的task_struct才被删除，之前处于EXIT_ZOMBIE 状态。

Linux进程状态:等待队列

- 等待队列：代表一些睡眠进程的集合

```
struct _ _wait_queue_head {        spinlock_t lock;
      struct list_head task_list;
  };
  typedef struct _ _wait_queue_head wait_queue_head_t;

struct _ _wait_queue {        unsigned int flags;
      struct task_struct * task;
      wait_queue_func_t func;
      struct list_head task_list;
  };
  typedef struct _ _wait_queue wait_queue_t;
```

DECLARE_WAIT_QUEUE_HEAD(name)

DEFINE_WAIT(wait)

*void sleep_on(wait_queue_head_t *wq)*

*long sleep_on_timeout(wait_queue_head_t *q, signed long timeout);*

*void interruptible_sleep_on(wait_queue_head_t *q);*

*long interruptible_sleep_on_timeout(wait_queue_head_t *q, signed long timeout);*

wait_event(wq,condition)

wait_event_interruptible_timeout(wq, condition, timeout)

wake_up_interruptible(x)

Linux进程状态：等待队列的用法

- 阻塞与非阻塞使用模板

```
1  static ssize_t xxx_write(struct file *file, const char *buffer, size_t count,
2    loff_t *ppos)
3  {
4    ...
5    DECLARE_WAITQUEUE(wait, current); //定义等待队列
6    add_wait_queue(&xxx_wait, &wait);  //添加等待队列
7
8    ret = count;
9    /* 等待设备缓冲区可写 */
10   do
11   {
12     avail = device_writable(...);
13     if (avail < 0)
14       __set_current_state(TASK_INTERRUPTIBLE);//改变进程状态
15
16     if (avail < 0)  {
18     if (file->f_flags &O_NONBLOCK)   {
21         ret =  - EAGAIN;
22       goto out;
23     }
24     schedule();   //调度其他进程执行
25     if (signal_pending(current)) {
26           ret =  - ERESTARTSYS;
29         goto out;
30     }
31   }
32  }while (avail < 0);
33
34  /* 写设备缓冲区 */
35  device_write(...)
36  out:
37  remove_wait_queue(&xxx_wait, &wait);//将等待队列移出等待队列头
38  set_current_state(TASK_RUNNING);//设置进程状态为TASK_RUNNING
39  return ret;
40 }
```

Linux进程调度

- 进程的类型：

a.I/O消耗型

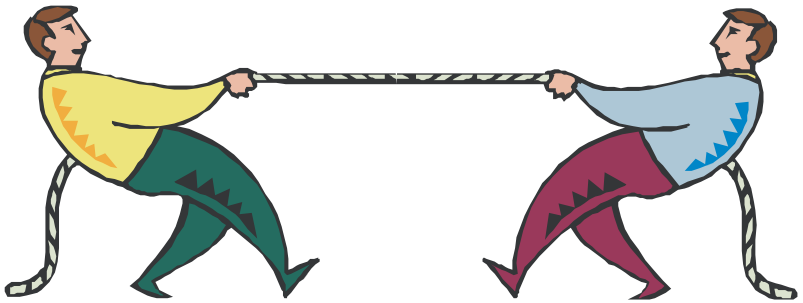
b.CPU消耗型

- 调度的目标

a.快速响应

b.高吞吐率

Throughput



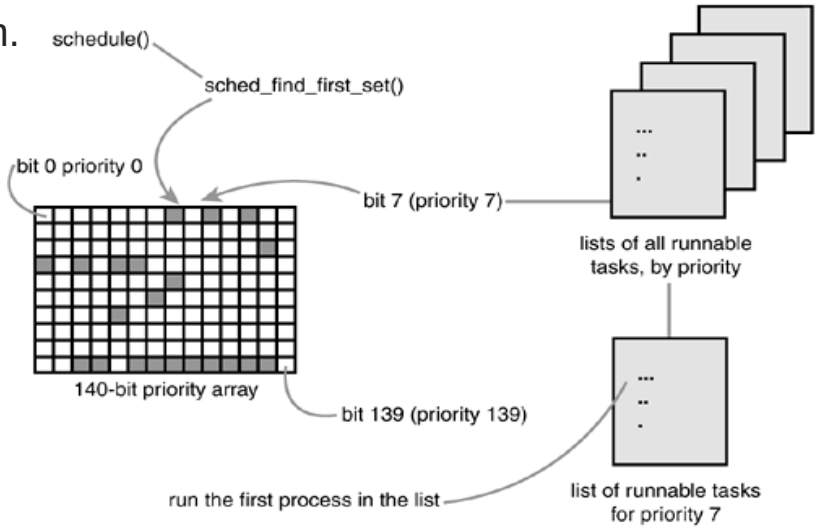
High responsiveness

Linux进程调度:schedule()

- SCHED_FIFO and SCHED_RR are so called "real-time" policies. They implement the fixed-priority real-time scheduling specified by the POSIX standard. Tasks with these policies preempt every other task.

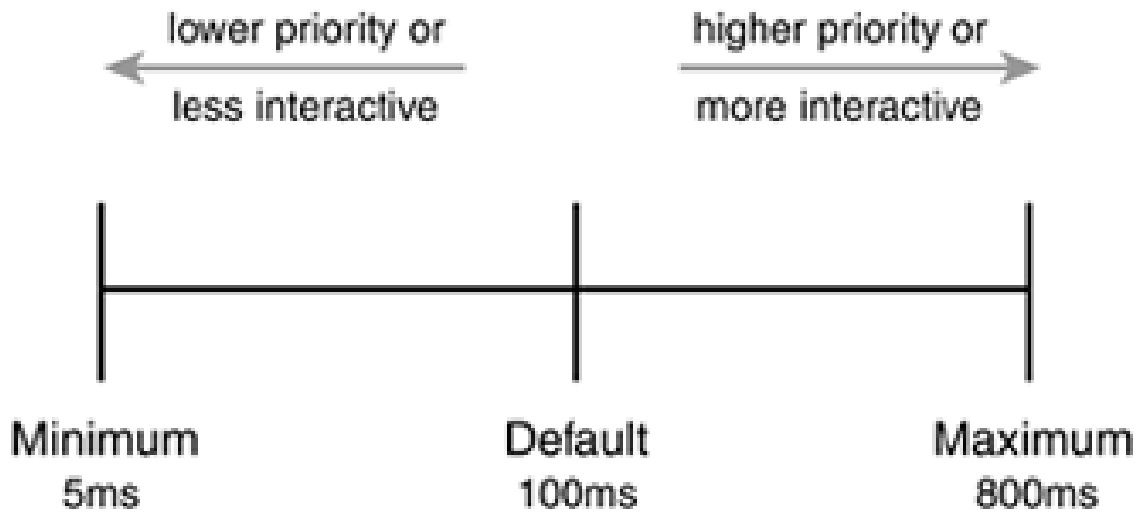
- The difference between SCHED_FIFO and SCHED_RR is that among tasks with the same priority, SCHED_RR performs a round-robin with a certain timeslice; SCHED_FIFO, instead, needs the task to explicitly yield the processor.

- SCHED_OTHER is the common round-robin time-sharing scheduling policy that schedules a task for a certain timeslice depending on the other tasks running in the system.



Linux进程调度:动态优先级和时间片

- task_struct. static_prio反映进程创建时的优先级,nice值；
- effective_prio() 返回一个进程的动态优先级，根据task_struct的sleep_avg对nice进行+-5的调整；



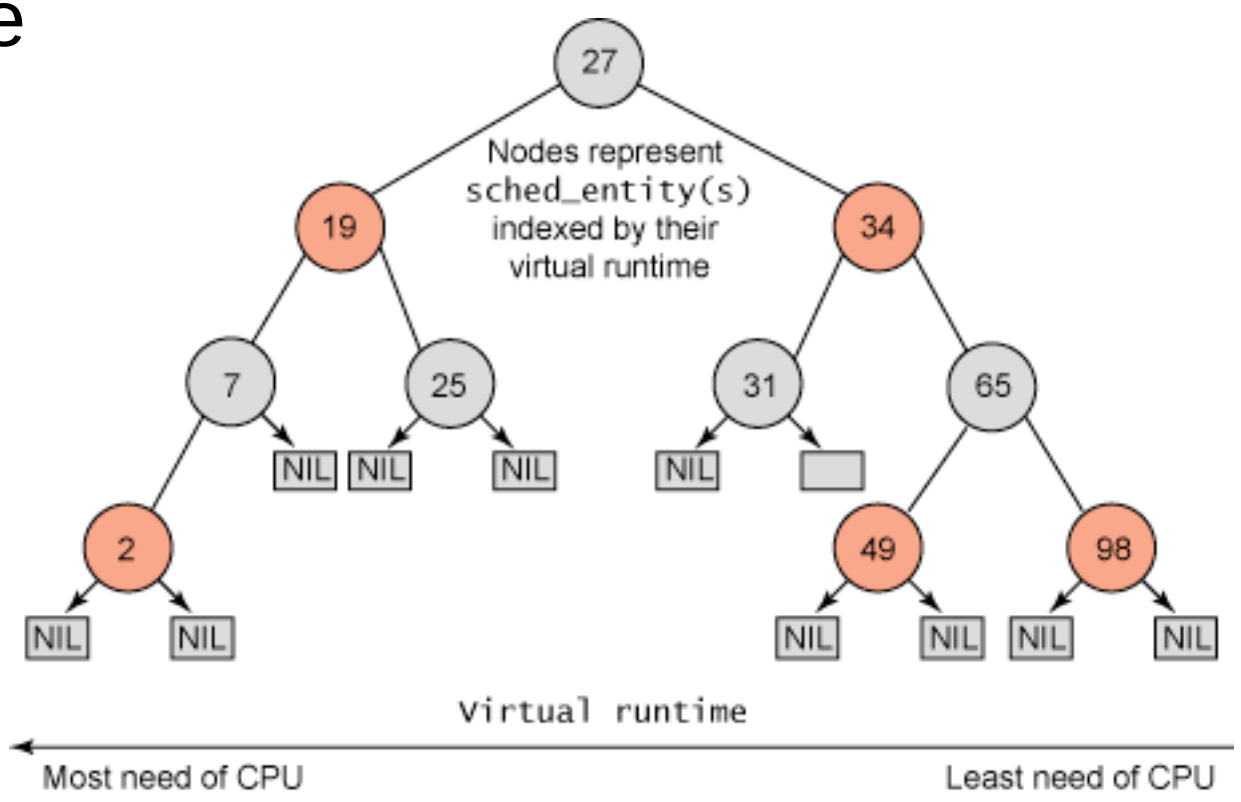
sched_rt_period_us和sched_rt_runtime_us

/proc/sys/kernel/sched_rt_period_us:
  The scheduling period that is equivalent to 100% CPU bandwidth

/proc/sys/kernel/sched_rt_runtime_us:
  A global limit on how much time realtime scheduling may use.

CFS:红黑树

- rather than maintain the tasks in a run queue, as has been done in prior Linux schedulers, the CFS maintains a time-ordered red-black tree



CFS 权重表

- When tick happens the task's CPU usage time is added to the p->se.vruntime

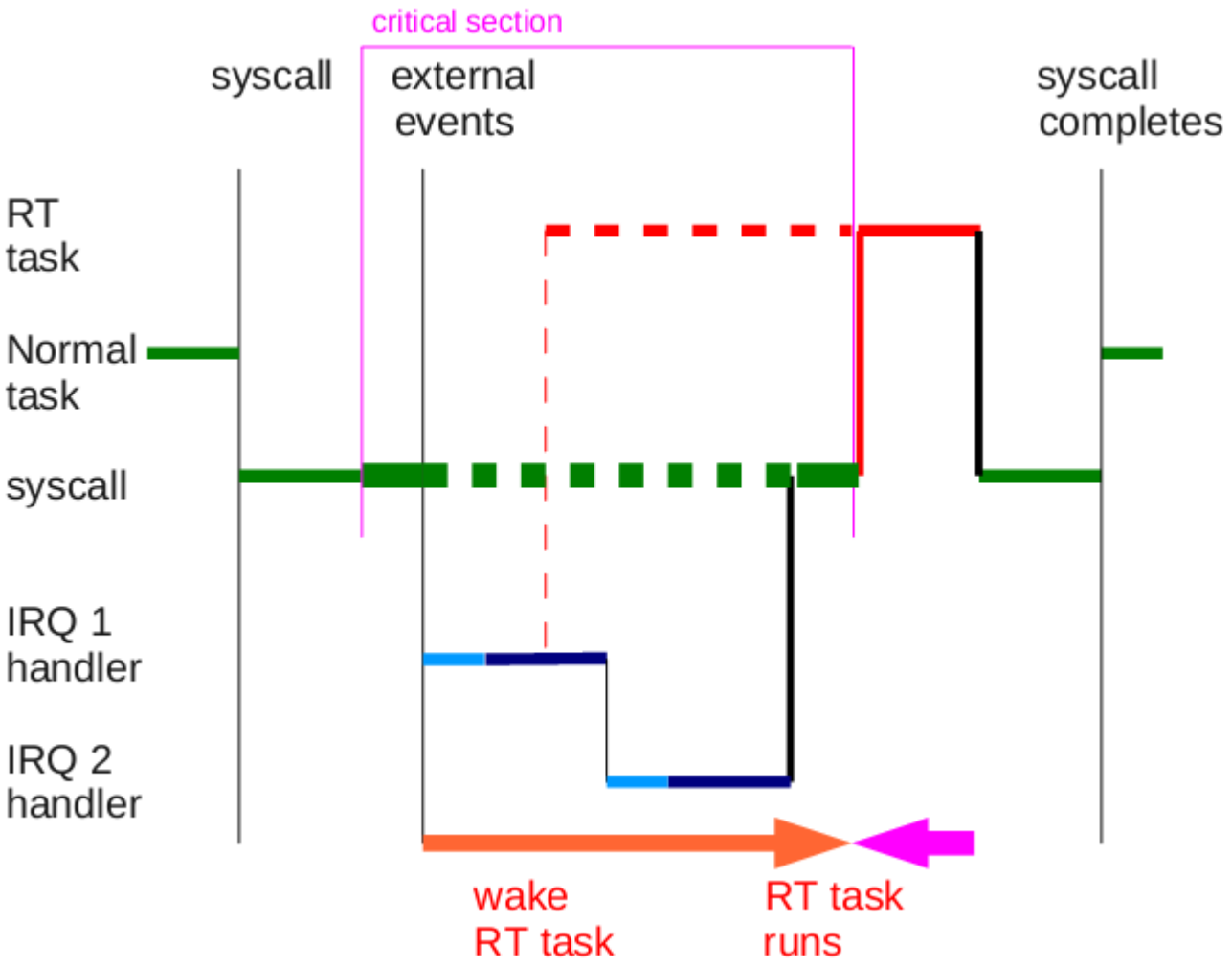
vruntime += delta NICE_0_LOAD/ se.weight;*

```
static const int prio_to_weight[40] = {
 /* -20 */    88761,    71755,    56483,    46273,    36291,
 /* -15 */    29154,    23254,    18705,    14949,    11916,
 /* -10 */     9548,     7620,     6100,     4904,     3906,
 /*  -5 */     3121,     2501,     1991,     1586,     1277,
 /*   0 */     1024,      820,      655,      526,      423,
 /*   5 */      335,      272,      215,      172,      137,
 /*  10 */      110,       87,       70,       56,       45,
 /*  15 */       36,       29,       23,       18,       15,
};
```

调度相关系统调用

System Call	Description
`nice()`	Sets a process's nice value
`sched_setscheduler()`	Sets a process's scheduling policy
`sched_getscheduler()`	Gets a process's scheduling policy
`sched_setparam()`	Sets a process's real-time priority
`sched_getparam()`	Gets a process's real-time priority
`sched_get_priority_max()`	Gets the maximum real-time priority
`sched_get_priority_min()`	Gets the minimum real-time priority
`sched_rr_get_interval()`	Gets a process's timeslice value
`sched_setaffinity()`	Sets a process's processor affinity
`sched_getaffinity()`	Gets a process's processor affinity
`sched_yield()`	Temporarily yields the processor

内核不可抢占区间


critical section
syscall
external events
syscall completes
RT task
Normal task
syscall
IRQ 1 handler
IRQ 2 handler
wake RT task
RT task runs

硬实时Linux:rt-patch和Xenomai

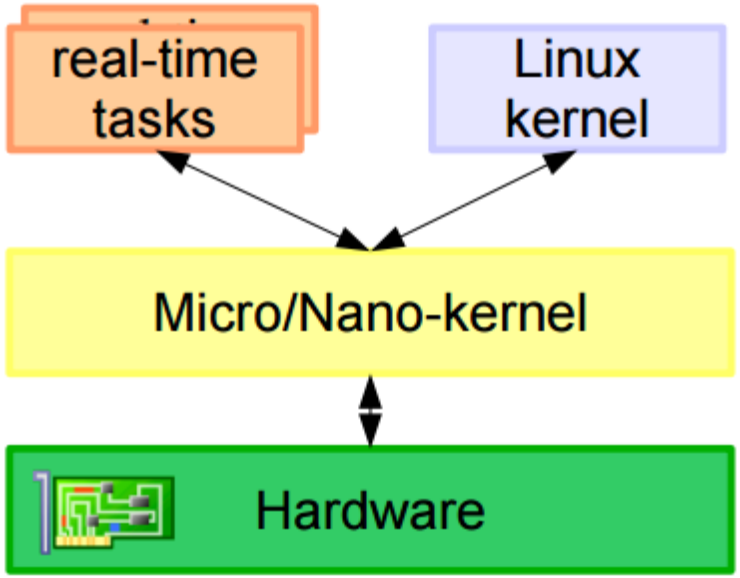
- Linux RT preempt patch: Ingo Molnar

下载地址：http://www.kernel.org/pub/linux/kernel/projects/rt/

设计思想：将IRQ和softIRQ线程化，实时任务可以有比中断线程更高的优先级；使用mutex替代spinlock来使得自旋锁可抢占。

```
# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 softirq-high/0
    ...
   18 ?        00:00:00 IRQ 12
   19 ?        00:00:00 IRQ 14
....
```

- Xenomai



SMP

Load Balance

- RTC process:
  - pull_rt_task(): Pull one top-n + 1 RT process when one of top-n yield
  - push_rt_task(0: Push the 2nd RT to Another CPU which has no RT process or has the lowest pri RT process

- Normal process:
  - Periodic

Tick -> run_rebalance_domains()-> rebalance_domains()->

  - CPU Idle

idle_balance(): ->

for_each_domain(this_cpu, sd) load_balance()…

  - fork(), exec()

sched_exec - execve() is a valuable balancing opportunity

CPU task affinity

- System call

#include <sched.h>

int sched_setaffinity(pid_t pid, unsigned int cpusetsize, cpu_set_t *mask);

  - Whenever a process has to be moved from a runqueue rq1 to another runqueue rq2, the system call awakes the migration thread of rq1 (rq1->migration_thread), which in turn removes the process from rq1 and inserts it into rq2.

int sched_getaffinity(pid_t pid, unsigned int cpusetsize, cpu_set_t *mask);

- POSIX extensions

#include <pthread.h>

int pthread_attr_setaffinity_np(pthread_attr_t *, size_t, const cpu_set_t *);

int pthread_attr_getaffinity_np(pthread_attr_t *, size_t, cpu_set_t *);

IRQ affinity

- assign certain IRQs to specific processors or groups of processors

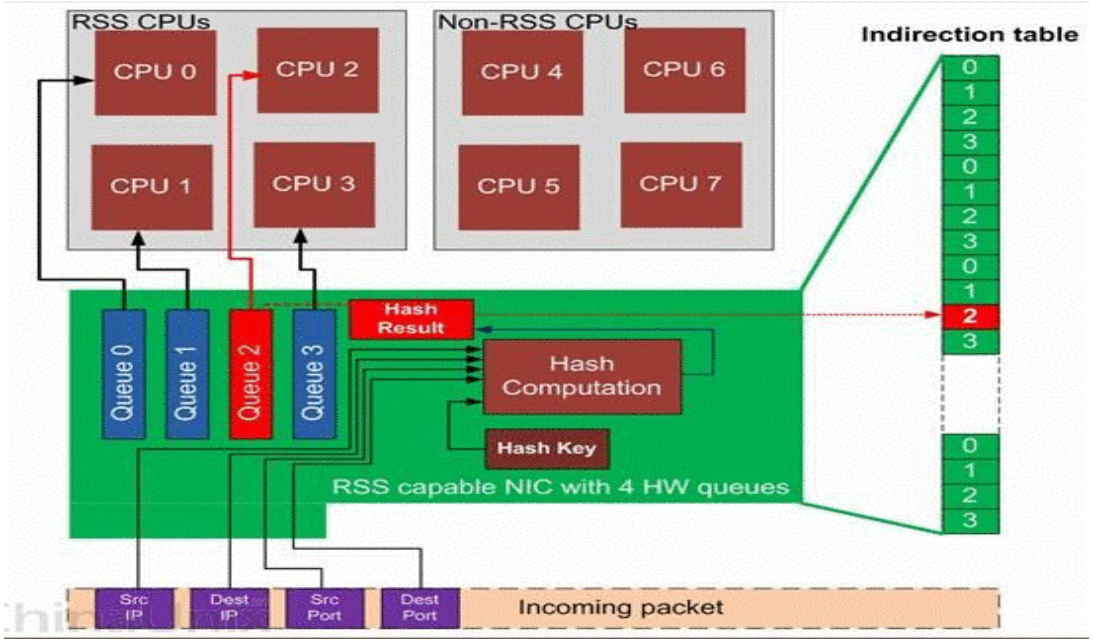
[root@boss ~]# echo 01 > /proc/irq/145/smp_affinity

[root@boss ~]# cat /proc/irq/145/smp_affinity

00000001

- 多队列网卡：把中断绑定到不同core: /proc/irq/*/smp_affinity

/proc/irq/74/smp_affinity 000001
/proc/irq/75/smp_affinity 000002
/proc/irq/76/smp_affinity 000004
/proc/irq/77/smp_affinity 000008
…



Brain Fuck Scheduler

BFS is the Brain Fuck Scheduler. It was designed to be forward looking only, make the most of lower spec machines, and not scale to massive hardware. ie it is a desktop orientated scheduler, with extremely low latencies for excellent interactivity by design rather than "calculated", with rigid fairness, nice priority distribution and extreme scalability within normal load levels.
—Con Kolivas

Brain Fuck Scheduler(BFS)

- BFS is a process scheduler designed for the Linux kernel in August 2009 as an alternative to the Completely Fair Scheduler and the O(1) scheduler. BFS was created by veteran kernel programmer Con Kolivas.
- Con Kolivas does not intend for BFS to be integrated into the mainline kernel.
- BFS has been added to an experimental branch of Google's Android development repository.It was not included in the Froyo release after blind testing did not show an improved user experience.
- BFS is the default scheduler for Zenwalk 6.4,[10] PCLinuxOS 2010,[11] NimbleX and Sabayon Linux 7.

schedutils

- chrt
- taskset
- nice/renice

cyclictest

- *This tool acquires timer jitter by measuring accuracy of sleep and wake operations of highly prioritized realtime threads.*

```
# ./cyclictest -p 80 -t5 -n
1.58 1.61 1.62 3/68 4079
T: 0 ( 3131) P:80 I:    1000 C:16469865 Min:       8 Act:      13 Max: 62
T: 1 ( 3132) P:79 I:    1500 C: 9979903 Min:       8 Act:      18 Max: 75
T: 2 ( 3133) P:78 I:    2000 C: 7934887 Min:       9 Act:      22 Max: 83
T: 3 ( 3134) P:77 I:    2500 C: 6587910 Min:       9 Act:      25 Max: 81
T: 4 ( 3135) P:76 I:    3000 C: 5489925 Min:       9 Act:      27 Max: 86
```

time – measures the runtime of a process program

- – Three figures provided

– real

– user

– sys

- \time

\time ./a.out

main pid:5500, tid:3076155136

^CCommand terminated by signal 2

4.11user 0.00system 0:02.23elapsed 183%CPU (0avgtext+0avgdata 1308maxresident)k

0inputs+0outputs (0major+66minor)pagefaults 0swaps

Kernel debug

Kernel – printk

- /proc/sys/kernel/printk
- dev_xxx
- pr_xxx
- #define pr_fmt(fmt) "hw-breakpoint: " fmt

```
#define pr_fmt(fmt) "cn_test: " fmt

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/skbuff.h>
#include <linux/slab.h>
#include <linux/timer.h>

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

Kernel – WARN_ON

- print_modules();
- dump_stack();
- print_oops_end_marker();

```
[    1.276928] ------------[ cut here ]------------
[    1.279558] WARNING: at kernel/drivers/gpio/gpiolib.c:99 gpio_ensure_requested+0x50/0x1a4()
[    1.292649] autorequest [...] [...]
[    1.295774] Modules linked in:
[    1.298812] Backtrace:
[    1.301261] [<c02df4b8>] (dump_backtrace+0x0/0x110) from [<c0652628>] (dump_stack+0x18/0x1c)
[    1.309663]  r6:c0725458 r5:00000063 r4:d8c1bde8 r3:00000000
[    1.315313] [<c0652610>] (dump_stack+0x0/0x1c) from [<c02f8a3c>] (warn_slowpath_common+0x54/0x6c)
[    1.324166] [<c02f89e8>] (warn_slowpath_common+0x0/0x6c) from [<c02f8af8>] (warn_slowpath_fmt+0x38/0x40
[    1.333618]  r8:0000003c r7:0000001c r6:00000000 r5:c07989f4 r4:c0806410
[    1.340127] r3:00000009
[    1.342741] [<c02f8ac0>] (warn_slowpath_fmt+0x0/0x40) from [<c04a4494>] (gpio_ensure_requested+0x50/0x1
[    1.352367]  r3:0000003c r2:c07254ad
[    1.355935] [<c04a4444>] (gpio_ensure_requested+0x0/0x1a4) from [<c04a478c>] (gpio_direction_output+0x8
[    1.366009] [<c04a4700>] (gpio_direction_output+0x0/0x160) from [<c05a21f4>] (wireless_init_audio_probe+0
```

Kernel – BUG_ON/BUG

- printk("BUG: failure at %s:%d/%s()!\n", __FILE__, __LINE__, __func__); \
- panic("BUG!"); \

Kernel – oops - bug

```
/kernel$ git diff
diff --git a/tools/cma/cma_test.c b/tools/cma/cma_test.c
index 7eb96db..ba02d35 100644
--- a/tools/cma/cma_test.c
+++ b/tools/cma/cma_test.c
@@ -44,6 +44,9 @@ cma_test_read(struct file *file, char __user *buf,
size_t count, loff_t *ppos)

        if (!alloc)
                return -EIDRM;
+
+       volatile int *p=0;
+       *p = 0;

        dma_free_coherent(cma_dev, alloc->size, alloc->virt,
                alloc->dma);
```

Kernel – oops - PC

```
sh-4.2# cat /dev/cma_test
[   33.491284] Unable to handle kernel NULL pointer dereference at virtual address
00000000
[   33.507357] pgd = cc914000
[   33.507379] [00000000] *pgd=0c902831, *pte=00000000, *ppte=00000000
[   33.513769] Internal error: Oops: 817 [#1] PREEMPT
[   33.518538] last sysfs file:
/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
[   33.526350] Modules linked in: vxdkernel pvrsrvkm sirfsoc_gps sirfsocfb cma_test
[   33.533729] CPU: 0    Not tainted  (2.6.38.8-sirf #28)
[   33.538857] PC is at cma_test_read+0x8c/0xec [cma_test]
[   33.544067] LR is at vfs_read+0xb8/0x144
[   33.547966] pc : [<bf06f1cc>]    lr : [<c039d64c>]    psr: a0000013
[   33.547972] sp : cc90ff08  ip : cc90ff30  fp : cc90ff2c
[   33.559421] r10: 00000000  r9 : 00000003  r8 : beafd890
[   33.564631] r7 : cc90ff68  r6 : bf06f4ec  r5 : 00000000  r4 : d3e16460
[   33.571139] r3 : 00000000  r2 : 00000000  r1 : beafd890  r0 : cc8e6e20
[   33.577650] Flags: NzCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment
user
[   33.584768] Control: 10c53c7d  Table: 0c914059  DAC: 00000015
[   33.590495]
[   33.590498] LR: 0xc039d5cc:
```

Kernel – oops -backtrace

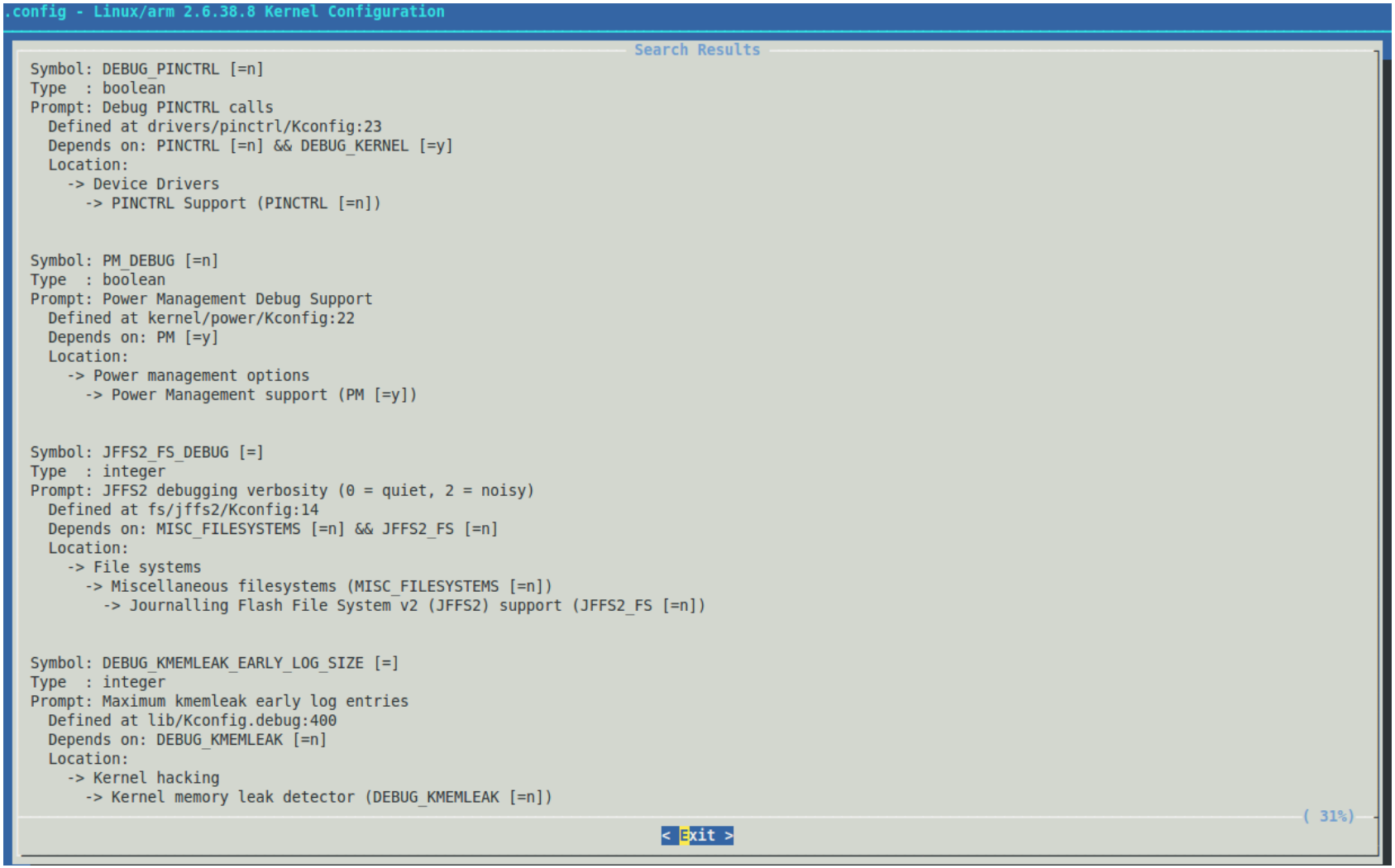
[34.152437] Backtrace:
[34.154882] [<bf06f140>] (cma_test_read+0x0/0xec [cma_test]) from [<c039d64c>] (vfs_read+0xb8/0x144)
[34.163982] r6:beafd890 r5:cc8e6e20 r4:00001000
[34.168592] [<c039d594>] (vfs_read+0x0/0x144) from [<c039d724>] (sys_read+0x4c/0x108)
[34.176394] r8:beafd890 r7:00001000 r6:beafd890 r5:cc8e6e20 r4:000a8c34
[34.183093] [<c039d6d8>] (sys_read+0x0/0x108) from [<c02dc720>] (ret_fast_syscall+0x0/0x30)
[34.191416] Code: 03e0002a 0a000012 e59f6058 e3a05000 (e5855000)
[34.294365] ---[end trace f0d7620b9f61d90d]---

Kernel – oops - objdump

[33.538857] PC is at cma_test_read+0x8c/0xec [cma_test]

```
86 00000140 <cma_test_read>:
87  140:   e1a0c00d        mov     ip, sp
88  144:   e92dd870        push    {r4, r5, r6, fp, ip, lr, pc}
89  148:   e24cb004        sub     fp, ip, #4
90  14c:   e24dd00c        sub     sp, sp, #12
91  150:   e1a0200d        mov     r2, sp
92  154:   e3c23d7f        bic     r3, r2, #8128   ; 0x1fc0
93  158:   e3c3303f        bic     r3, r3, #63     ; 0x3f
94  15c:   e5932004        ldr     r2, [r3, #4]
95  160:   e2822001        add     r2, r2, #1
96  164:   e5832004        str     r2, [r3, #4]
97  168:   e59f30a8        ldr     r3, [pc, #168]  ; 218 <cma_test_read+0xd8>

114 1ac:   e3130002        tst     r3, #2
115 1b0:   0a000000        beq     1b8 <cma_test_read+0x78>
116 1b4:   ebfffffe        bl      0 <preempt_schedule>
117 1b8:   e3540000        cmp     r4, #0
118 1bc:   03e0002a        mvneq   r0, #42 ; 0x2a
119 1c0:   0a000012        beq     210 <cma_test_read+0xd0>
120 1c4:   e59f6058        ldr     r6, [pc, #88]   ; 224 <cma_test_read+0xe4>
121 1c8:   e3a05000        mov     r5, #0
122 1cc:   e5855000        str     r5, [r5]
123 1d0:   e5960000        ldr     r0, [r6]
```

Kernel – DEBUG options



Kernel – /dev/ttyprintk

- ttyprintk is a pseudo TTY driver, which allows users to make printk messages

```
sh-4.2# echo "hello world" > /dev/ttyprintk
[12005.502521] [U] hello world
```

Kernel – DEBUG_LL/EARLY_PRINTK

Enable EARLY_PRINTK, DEBUG_LL to support early print and include definitions of printascii, printch, printhex in the kernel. This is helpful if you are debugging code that executes before the console is initialized.

Kernel – PM_DEBUG and no_console_suspend

- Enable PM_DEBUG in Kconfig

```
[*] Power Management support                                    | |
  | |                 [*]   Power Management Debug Support                                  | |
  | |                 [*]     Extra PM attributes in sysfs for low-level debugging/testing    | |
  | |                 [*]     Verbose Power Management debugging
```

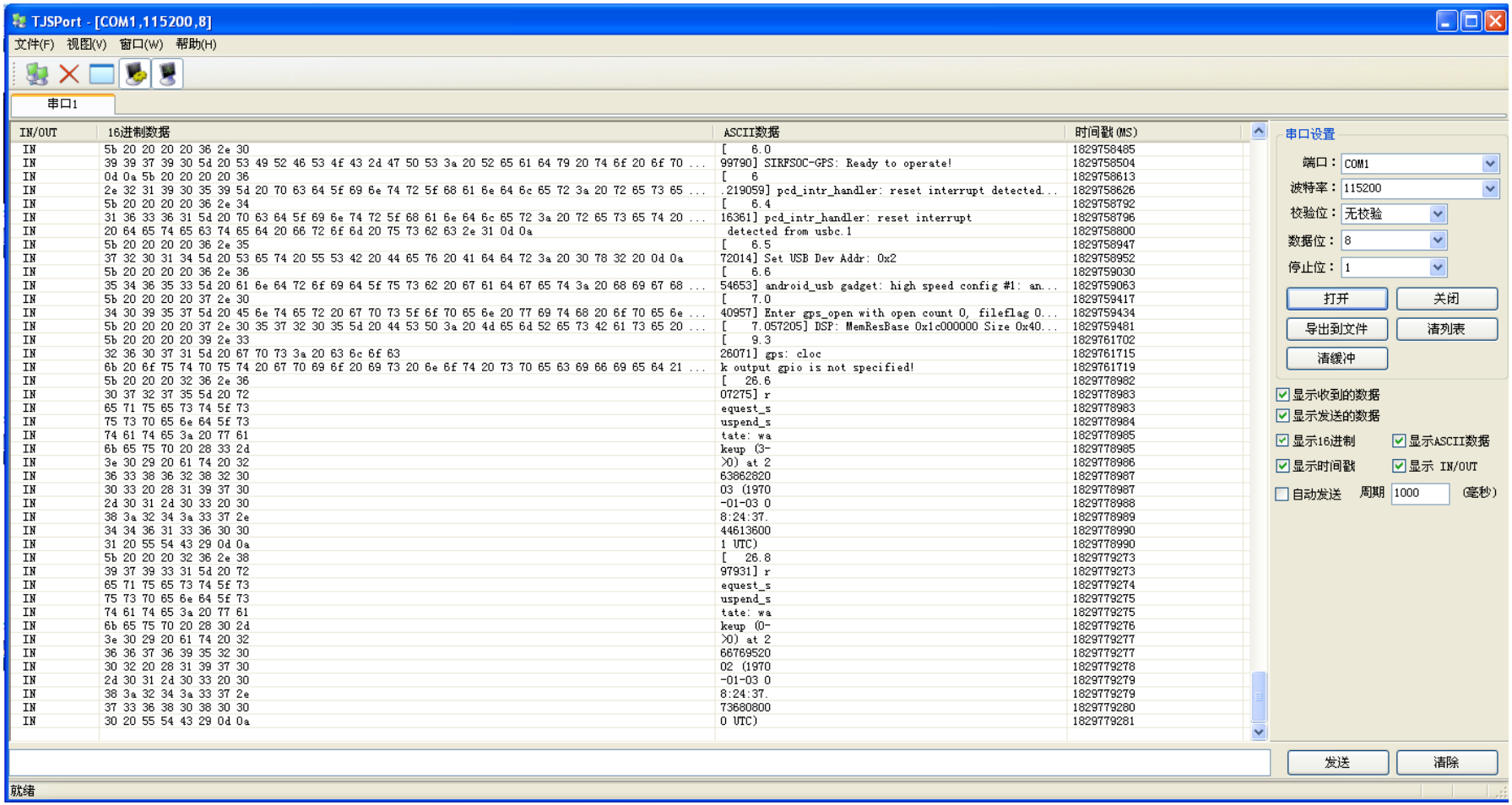
- Set no_console_suspend in bootargs

Kernel – Android init “untracked PID exited”

```
diff --git a/kernel/exit.c b/kernel/exit.c
index 7cdb3a6..36ead86 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -909,6 +909,16 @@ NORET_TYPE void do_exit(long code)
        struct task_struct *tsk = current;
        int group_dead;

+       /*
+        * we want to track the exit of "untracked PID exited" events in Android, eg,
+        * in the process of suspend/resume:
+        * [  222.660982] task exit: tgid(PID):1293 pid:1302 name:synergy_service
+        * [  222.731344] init: untracked pid 1293 exited
+        */
+       if (tsk->ppid == 1)
+               printk(KERN_DEBUG "task exit: tgid(PID):%d pid:%d name:%s\n",
+                       tsk->tgid, tsk->pid, tsk->comm);
+
        profile_task_exit(tsk);

        WARN_ON(atomic_read(&tsk->fs_excl));
```

Print with timestamp

- Tjsport and similar tools

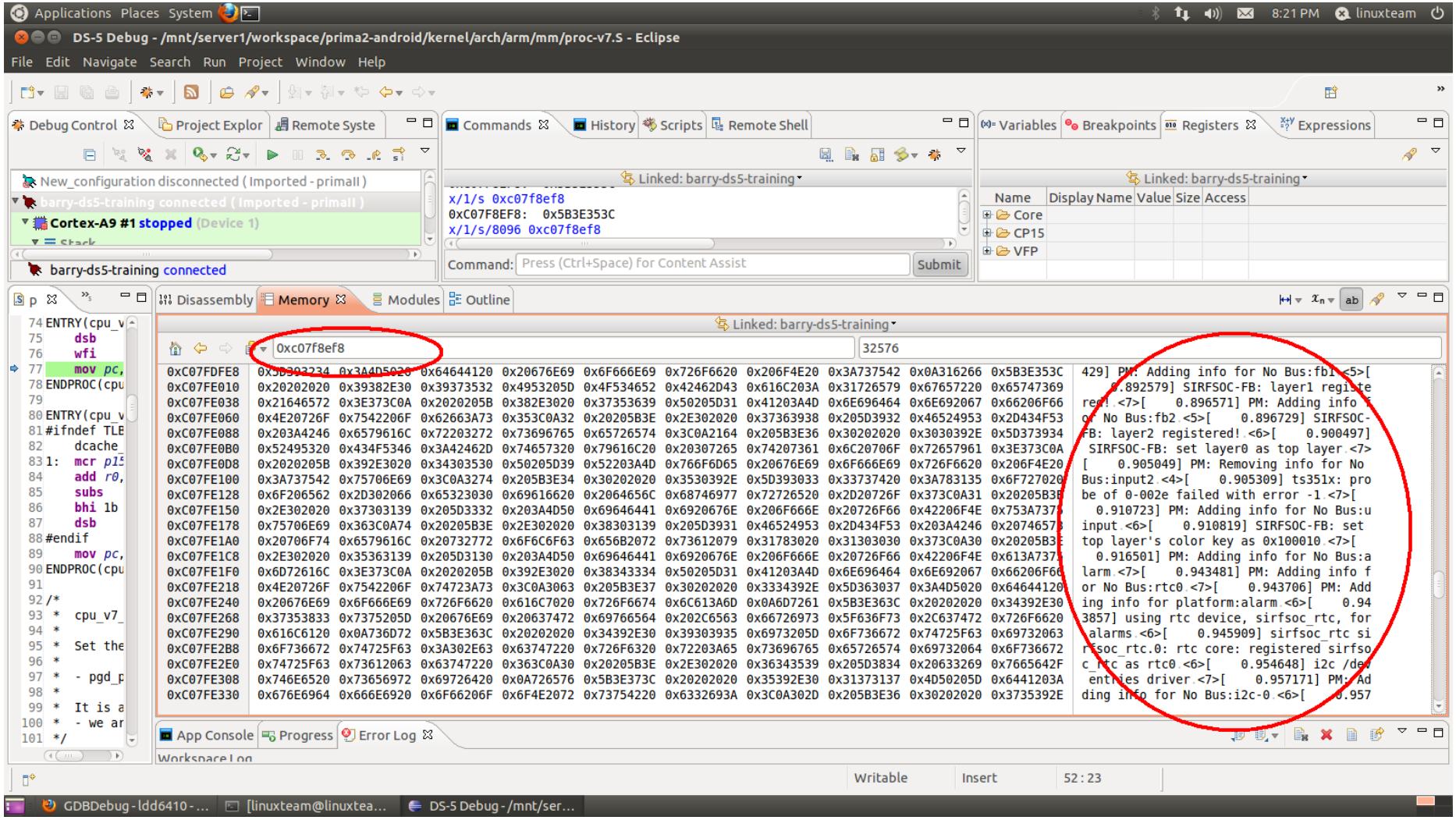


Print with timestamp

- grabserial: http://elinux.org/Grabserial

```
[tbird@timdesk data]$ ../grabserial -v -d /dev/ttyUSB1 -e 30 -t -m "Starting kernel.*"
Opening serial port /dev/ttyUSB1
115200:8N1:xonxoff=0:rtcdtc=0
Program will end in 30 seconds
Printing timing information for each line
Matching pattern 'Starting kernel.*' to set base time
Use Control-C to stop...
[0.000001 0.000001]
[0.000433 0.000432]
[0.001908 0.001475] Texas Instruments X-Loader 1.41 (Sep  1 2010 - 13:43:00)
[0.295940 0.294032] mmc read: Invalid size
[0.299905 0.003965] Starting OS Bootloader from MMC/SD1 ...
[0.311140 0.011235]
[0.313113 0.001973]
[0.313168 0.000055] U-Boot 1.1.4-gcebe815a-dirty (Aug 16 2010 - 10:34:46)
[0.317314 0.004146]
[0.317353 0.000039] Load address: 0x80e80000
[0.319459 0.002106] DRAM:  512 MB
[0.321147 0.001688] Flash:  0 kB
[0.360937 0.039790] *** Warning - bad CRC, using default environment
[0.366966 0.006029]
[0.387197 0.020231] In:    serial
```

sh-4.2# grep __log_buf /proc/kallsyms
c07f8ef8 b __log_buf



Kernel – initcall_debug

Passing the option "initcall_debug" on the kernel command line will cause timing information to be printed to the console for each initcall. You will need to enable CONFIG_PRINTK_TIME and CONFIG_KALLSYMS in your kernel configuration for this to work correctly.

calling ipc_init+0x0/0x28 @ 1
msgmni has been set to 42
initcall ipc_init+0x0/0x28 returned 0 after 1872 usecs

dmesg -s 128000 | grep "initcall" | sed "s/\(.*\)after\(.*\)/\2 \1/g" | sort -n

Kernel源代码级调试

```
baohua@baohua-VirtualBox:~/develop/linux$ arm-linux-gnueabihf-gdb vmlinux
GNU gdb (crosstool-NG linaro-1.13.1-4.8-2013.05 - Linaro GCC 2013.05) 7.6-2013.05
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-build_pc-linux-gnu --target=arm-linux-gnueabihf"
For bug reporting instructions, please see:
<https://bugs.launchpad.net/gcc-linaro>...
Reading symbols from /home/baohua/develop/linux/vmlinux...done.
(gdb) target remote :1234
Remote debugging using :1234
0x60000000 in ?? ()
(gdb) b start_kernel
Breakpoint 1 at 0x805fd8ac: file init/main.c, line 490.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:490
490     {
(gdb)
```

debug module

- Target

```
sh-4.2# pwd
/sys/module/cma_test/sections

sh-4.2# ls -a
.                       .note.gnu.build-id
..                      .rodata
.bss                      .rodata.str1.1
.data                     .strtab
.exit.text                .symtab
.gnu.linkonce.this_module  .text
.init.text

sh-4.2# cat .text
0xbf06f000
sh-4.2# cat .data
0xbf06f36c
```

- Host

```
add-symbol-file kernel/tools/cma/cma_test.ko -s .text 0xbf06f000 -s .data 0xbf06f36c

b cma_test_read
```

proc & sys file system

- Proc

```
struct proc_dir_entry* entry;
entry->read_proc = read_proc_foo;
entry->write_proc = write_proc_foo;

static int proc_read_foobar(char *page, char **start,
off_t off, int count,
int *eof, void *data)
{
        int len;
        struct fb_data_t *fb_data = (struct fb_data_t *)data;

        len = sprintf(page, "%s = '%s'\n",
        fb_data->name, fb_data->value);

        return len;
}
```

- sysfs

Soft & hard lockup

- A periodic hrtimer runs to generate interrupts and kick the watchdog task.
- The watchdog task is a high priority kernel thread that updates a timestamp every time it is scheduled
- A 'softlockup' is defined as a bug that causes the kernel to loop in kernel mode for more than 20 seconds (see "Implementation" below for details), without giving other tasks a chance to run.
- A 'hardlockup' is defined as a bug that causes the CPU to loop in kernel mode for more than 10 seconds (see "Implementation" below for details), without letting other interrupts have a chance to run

Linux内存管理与调试

Linux 物理内存分页

- 系统中的每个物理页对应一个page结构体

```
struct page {
        page_flags_t        flags;//是否dirty，是否锁定...
        atomic_t            _count;//引用计数，为0时在新的分配
中可以使用
        atomic_t            _mapcount;
        unsigned long       private;
        struct address_space  *mapping;
        pgoff_t             index;
        struct list_head    lru;
        …
};
```

Linux 内存ZONE

➢Zone

●ZONE_DMA

X86上，ISA只能在16MB以下执行DMA，因此此区为0~16MB

●ZONE_NORMAL

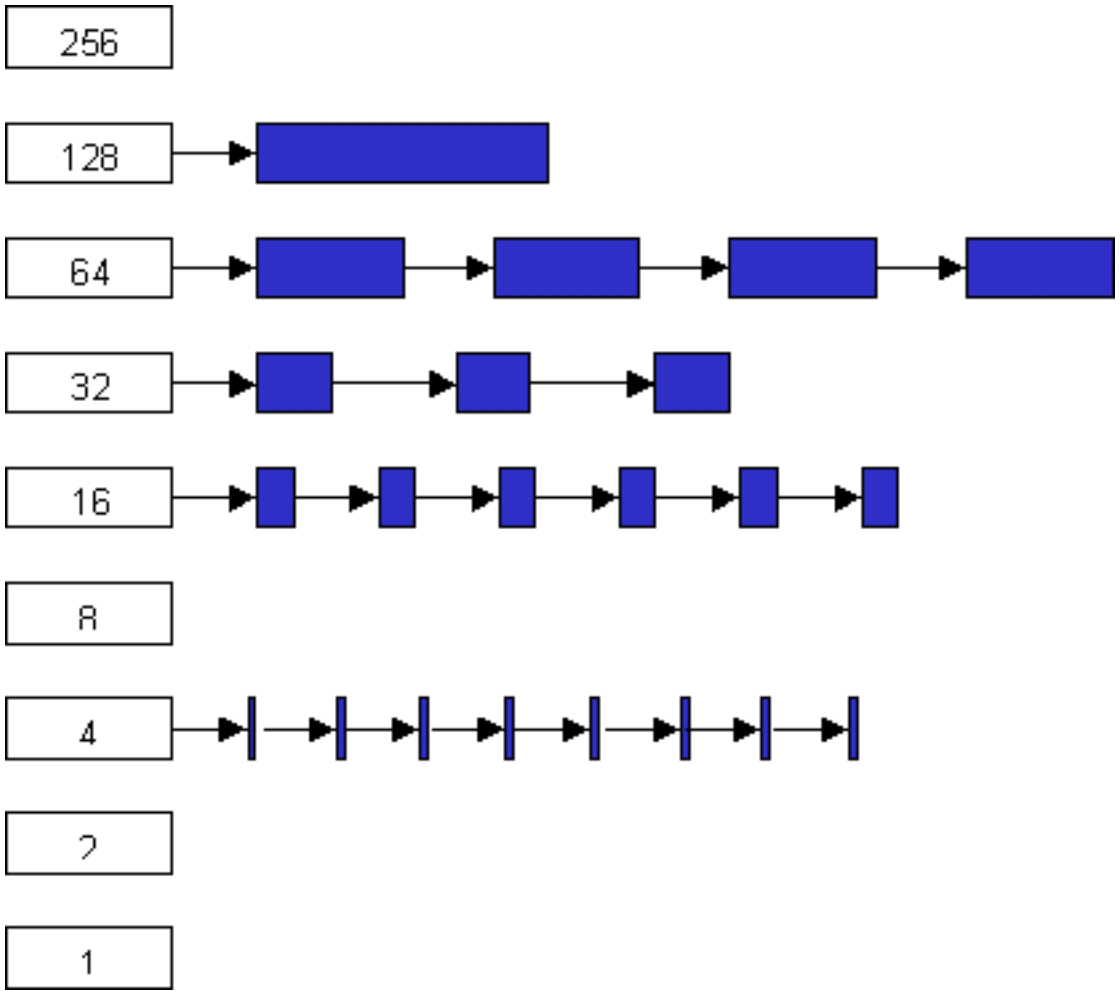
●ZONE_HIGHMEM

在x86上，为高于896MB的物理内存

➢数据结构

```
struct zone {
        spinlock_t              lock;
        unsigned long           free_pages;//该区中空闲页的数目
        unsigned long           pages_min;
        unsigned long           pages_low;
        unsigned long           pages_high;
        unsigned long           protection[MAX_NR_ZONES];
        spinlock_t              lru_lock;
        struct list_head        active_list;
        struct list_head        inactive_list;
        unsigned long           nr_scan_active;
        unsigned long           nr_scan_inactive;
        unsigned long           nr_active;
        unsigned long           nr_inactive;
        int                     all_unreclaimable;
        unsigned long           pages_scanned;
        struct free_area        free_area[MAX_ORDER];//2^order连续空闲页
        wait_queue_head_t       *wait_table;
        …
};
```

Linux Buddy分配算法



Linux 内核页操作

- 获得页

struct page * alloc_pages(unsigned int gfp_mask, unsigned int order) ;
*void * page_address(struct page *page) ;*
unsigned long __get_free_pages(unsigned int gfp_mask, unsigned int order) ;
仅需要一页:
*struct page * alloc_page(unsigned int gfp_mask) ;*
unsigned long __get_free_page(unsigned int gfp_mask) ;
获得填充为0的页：
unsigned long get_zeroed_page(unsigned int gfp_mask);

- 释放页

*void __free_pages(struct page *page, unsigned int order) ;*
void free_pages(unsigned long addr, unsigned int order) ;
void free_page(unsigned long addr) ;

- gfp_mask

Linux 内核slab层结构

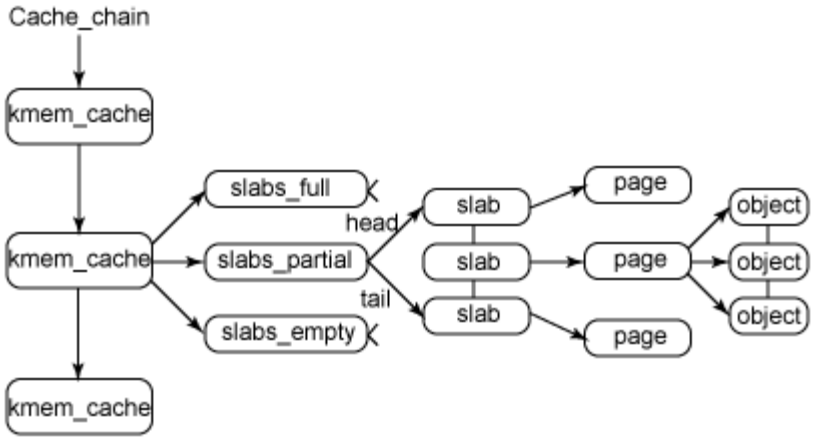
➢设计思想

●内核通常依赖于对小对象的分配，它们会在系统生命周期内进行无数次分配。slab 缓存分配器通过对类似大小的对象进行缓存而提供这种功能，从而避免了常见的碎片问题；

●slab 分配器支持通用对象的初始化，从而避免了为同一目而对一个对象重复进行初始化；

●slab 分配器支持硬件缓存对齐和着色，这允许不同缓存中的对象占用相同的缓存行，从而提高缓存的利用率并获得更好的性能。

➢结构



cache_chain： slab 缓存的链接列表；

kmem_cache ： cache，它定义了一个要管理的给定大小的对象池；

slabs 列表：一段连续的内存块。

/proc/slabinfo

```
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount
ctive_slabs> <num_slabs> <sharedavail>
isofs_inode_cache     44     44    360   22    2 : tunables    0    0    0 : slabdata      2      2      0
ext4_groupinfo_4k    156    156    104   39    1 : tunables    0    0    0 : slabdata      4      4      0
UDPLITEv6              0      0    768   21    4 : tunables    0    0    0 : slabdata      0      0      0
UDPv6                 84     84    768   21    4 : tunables    0    0    0 : slabdata      4      4      0
tw_sock_TCPv6          0      0    192   21    1 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                 44     44   1472   22    8 : tunables    0    0    0 : slabdata      2      2      0
zcache_objnode         0      0    272   30    2 : tunables    0    0    0 : slabdata      0      0      0
kcopyd_job             0      0   2344   13    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent              0      0   2464   13    8 : tunables    0    0    0 : slabdata      0      0      0
dm_rq_clone_bio_info      0      0     88   46    1 : tunables    0    0    0 : slabdata      0      0      0
dm_rq_target_io        0      0    264   31    2 : tunables    0    0    0 : slabdata      0      0      0
bsg_cmd                0      0    288   28    2 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache    28     28    576   28    4 : tunables    0    0    0 : slabdata      1      1      0
fuse_request          42     42    376   21    2 : tunables    0    0    0 : slabdata      2      2      0
fuse_inode            36     36    448   18    2 : tunables    0    0    0 : slabdata      2      2      0
ecryptfs_inode_cache      0      0    640   25    4 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache        0      0    416   19    2 : tunables    0    0    0 : slabdata      0      0      0
```

slabtop

```
Active / Total Objects (% used)    : 227125 / 230281 (98.6%)
Active / Total Slabs (% used)      : 5040 / 5040 (100.0%)
Active / Total Caches (% used)     : 69 / 100 (69.0%)
Active / Total Size (% used)       : 37463.56K / 38318.42K (97.8%)
Minimum / Average / Maximum Object : 0.01K / 0.17K / 8.00K
```

OBJS	ACTIVE	USE	OBJ SIZE	SLABS	OBJ/SLAB	CACHE SIZE	NAME
37728	37443	99%	0.12K	1179	32	4716K	dentry
29638	29638	100%	0.05K	406	73	1624K	buffer_head
24732	24732	100%	0.59K	916	27	14656K	ext4_inode_cache
22592	22592	100%	0.06K	353	64	1412K	kmalloc-64
17408	17408	100%	0.01K	34	512	136K	ext4_io_page
16128	15529	96%	0.03K	126	128	504K	kmalloc-32
15204	14742	96%	0.09K	362	42	1448K	kmalloc-96
12800	12800	100%	0.01K	25	512	100K	kmalloc-8
8576	8576	100%	0.03K	67	128	268K	anon_vma
8040	8040	100%	0.33K	335	24	2680K	inode_cache
6846	6208	90%	0.19K	326	21	1304K	kmalloc-192
5120	5120	100%	0.02K	20	256	80K	kmalloc-16
3536	2739	77%	0.30K	136	26	1088K	radix_tree_node
1870	1870	100%	0.05K	22	85	88K	Acpi-State
1856	1856	100%	0.06K	29	64	116K	journal_head
1584	1584	100%	0.36K	72	22	576K	proc_inode_cache
1600	1462	91%	0.12K	50	32	200K	kmalloc-128
1328	1245	93%	0.50K	83	16	664K	kmalloc-512
1020	1020	100%	0.02K	6	170	24K	nsproxy
1012	1012	100%	0.36K	46	22	368K	shmem_inode_cache

Out Of Memory

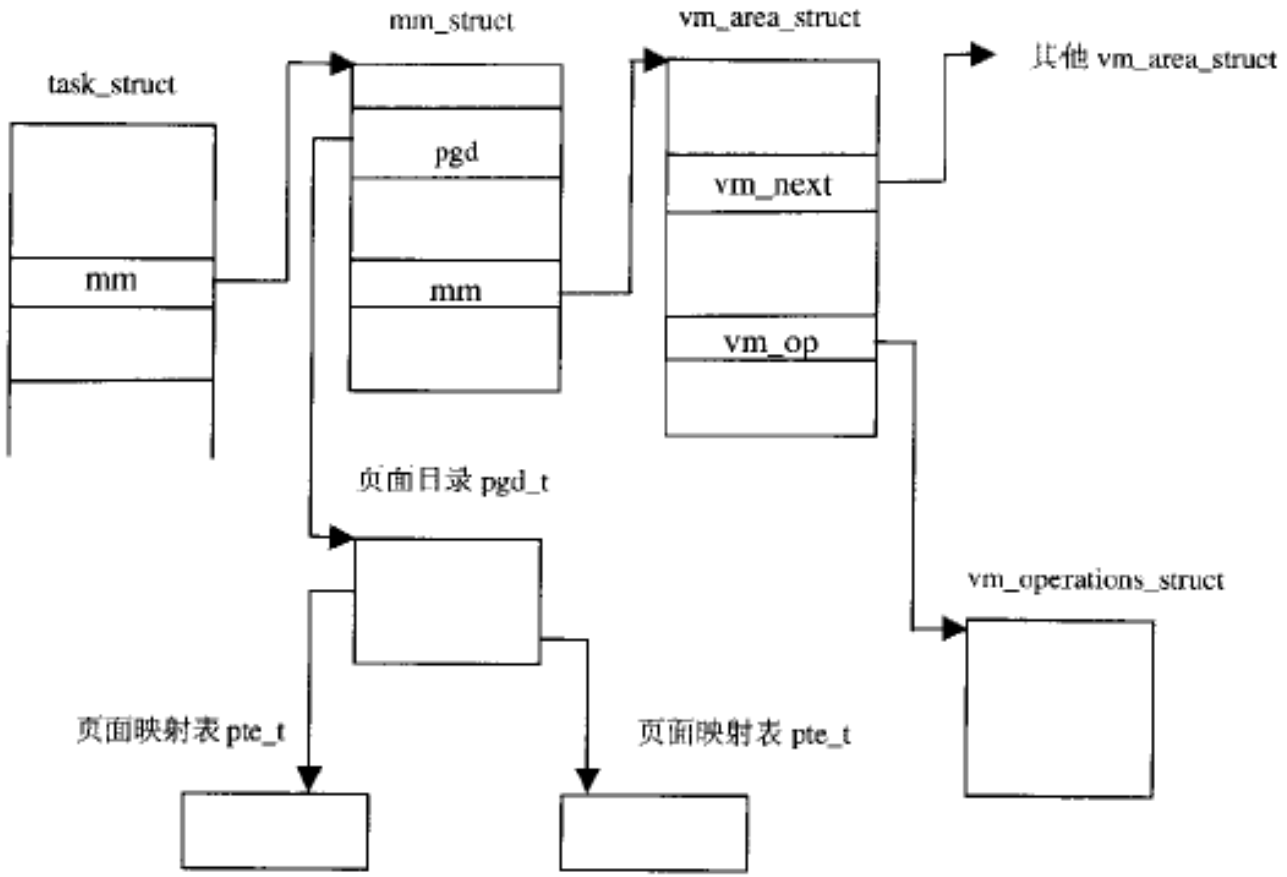
- An OOM (Out Of Memory) error is what happens when the kernel runs out of memory in its own internal pools and is unable to reclaim memory from any other sources. It basically starts killing random processes, and spits a lot of logging into dmesg.
- very few OOM events are genuine kernel bugs. Virtually all of them are user applications which are behaving badly.

```
Sep 19 00:33:10 mette kernel: Out of Memory: Killed process 8631 (xterm).
Sep 19 00:33:34 mette kernel: Out of Memory: Killed process 9154 (xterm).
Sep 19 00:34:05 mette kernel: Out of Memory: Killed process 6840 (xterm).
Sep 19 00:34:42 mette kernel: Out of Memory: Killed process 9066 (xterm).
Sep 19 00:35:15 mette kernel: Out of Memory: Killed process 9269 (xterm).
Sep 19 00:35:43 mette kernel: Out of Memory: Killed process 9351 (xterm).
Sep 19 00:36:05 mette kernel: Out of Memory: Killed process 6752 (xterm).
```

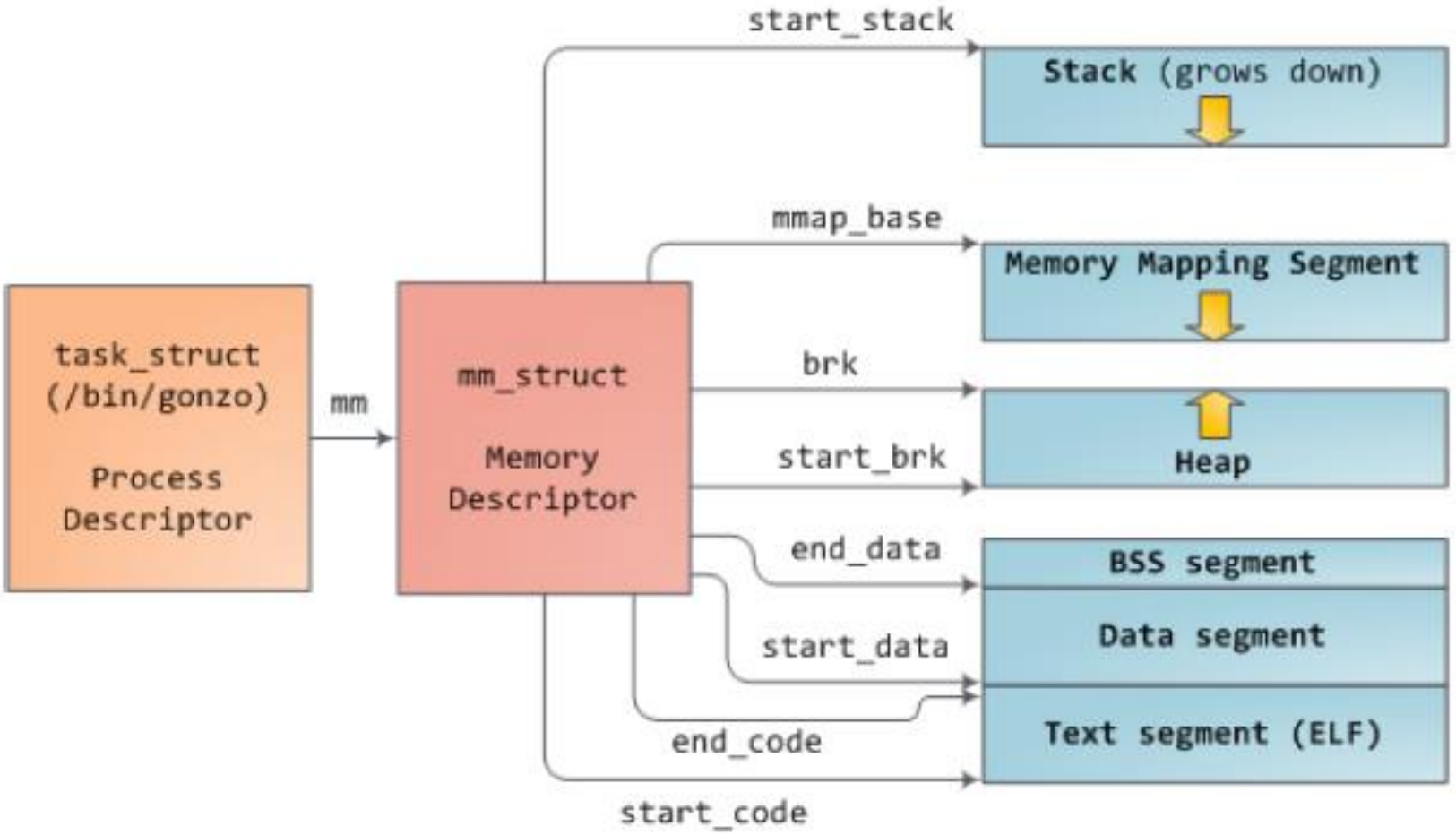
OOM killer criteria

- The function badness() in *mm/oom_kill.c* give a score to each existing processes. The one with highest score will be the victim:
  - VM size. the sum of the size of all VMAs owned by the process. The bigger the VM size, the higher the score.
  - Related to #1, the VM size of the process's children are important too. The VM size is cumulative if a process has one or more children.
  - Processes with task priorities smaller than zero (niced processes) get more points.
  - Superuser processes are important, by assumption; thus they have their scores reduced.
  - Process runtime. The longer it runs, the lower the score.
  - …

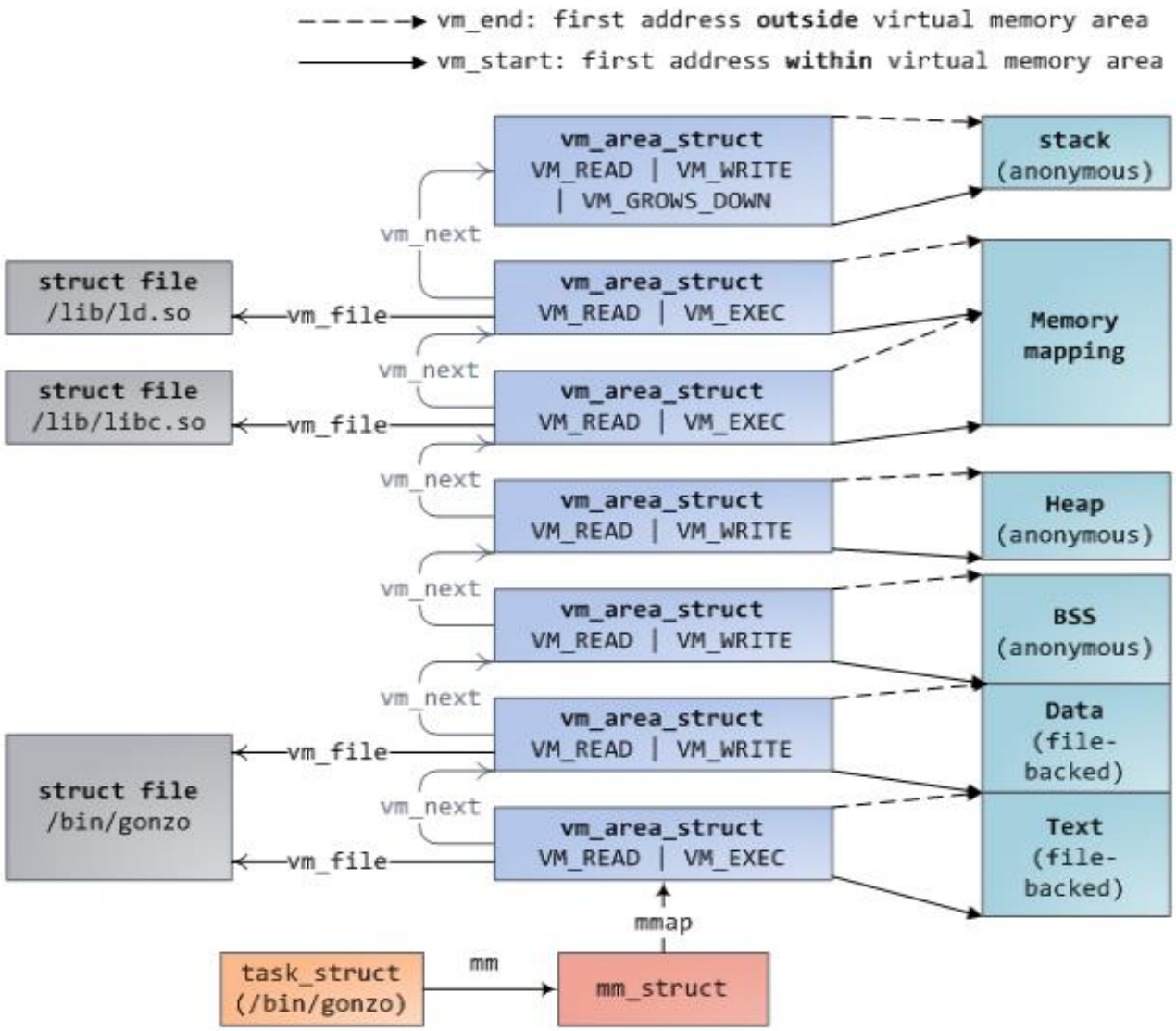
Linux虚存管理数据结构联系



Linux进程内存


start_stack
Stack (grows down)
mmap_base
Memory Mapping Segment
task_struct
(/bin/gonzo)
Process
Descriptor
mm
mm_struct
Memory
Descriptor
brk
start_brk
Heap
end_data
BSS segment
start_data
Data segment
Text segment (ELF)
end_code
start_code

mm_struct与VMA

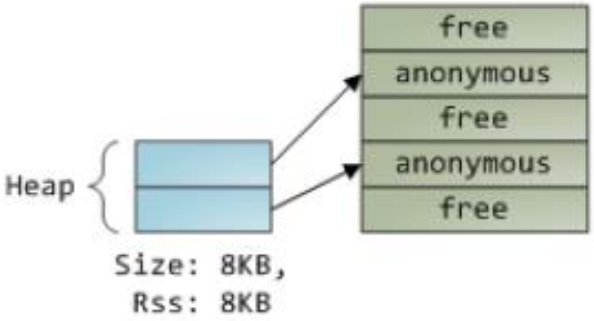


/proc/pid/maps

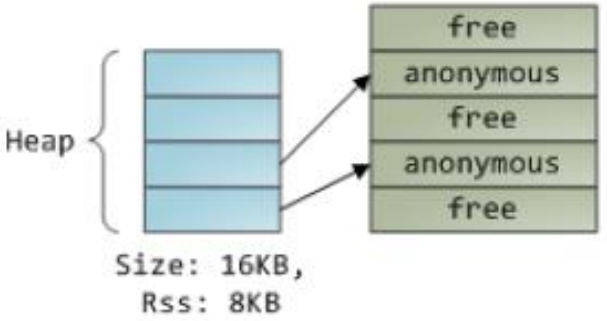
```
barry@barry-VirtualBox:~$ cat /proc/3623/maps
08048000-08049000 r-xp 00000000 08:01 781882     /home/barry/training/thread/a.out
08049000-0804a000 r--p 00000000 08:01 781882     /home/barry/training/thread/a.out
0804a000-0804b000 rw-p 00001000 08:01 781882     /home/barry/training/thread/a.out
b759c000-b759d000 rw-p 00000000 00:00 0
b759d000-b7741000 r-xp 00000000 08:01 671749     /lib/i386-linux-gnu/libc-2.15.so
b7741000-b7743000 r--p 001a4000 08:01 671749     /lib/i386-linux-gnu/libc-2.15.so
b7743000-b7744000 rw-p 001a6000 08:01 671749     /lib/i386-linux-gnu/libc-2.15.so
b7744000-b7747000 rw-p 00000000 00:00 0
b775b000-b775d000 rw-p 00000000 00:00 0
b775d000-b775e000 r-xp 00000000 00:00 0          [vdso]
b775e000-b777e000 r-xp 00000000 08:01 682278     /lib/i386-linux-gnu/ld-2.15.so
b777e000-b777f000 r--p 0001f000 08:01 682278     /lib/i386-linux-gnu/ld-2.15.so
b777f000-b7780000 rw-p 00020000 08:01 682278     /lib/i386-linux-gnu/ld-2.15.so
bfc8c000-bfcad000 rw-p 00000000 00:00 0          [stack]
```

VSS vs. RSS

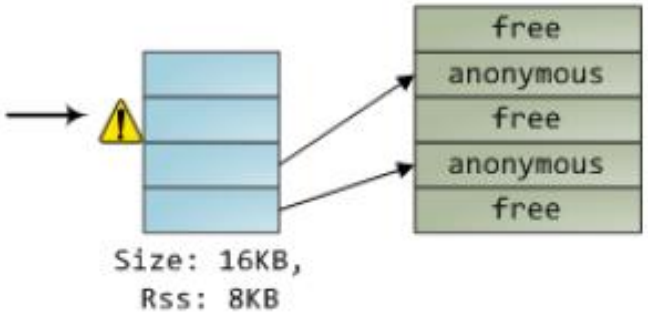
1. Program calls brk() to grow its heap



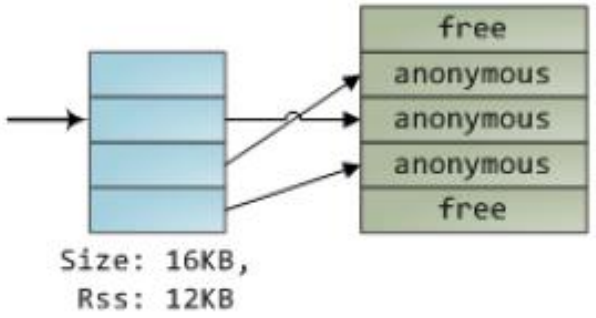
2. brk() enlarges heap VMA.
New pages are **not** mapped onto physical memory.



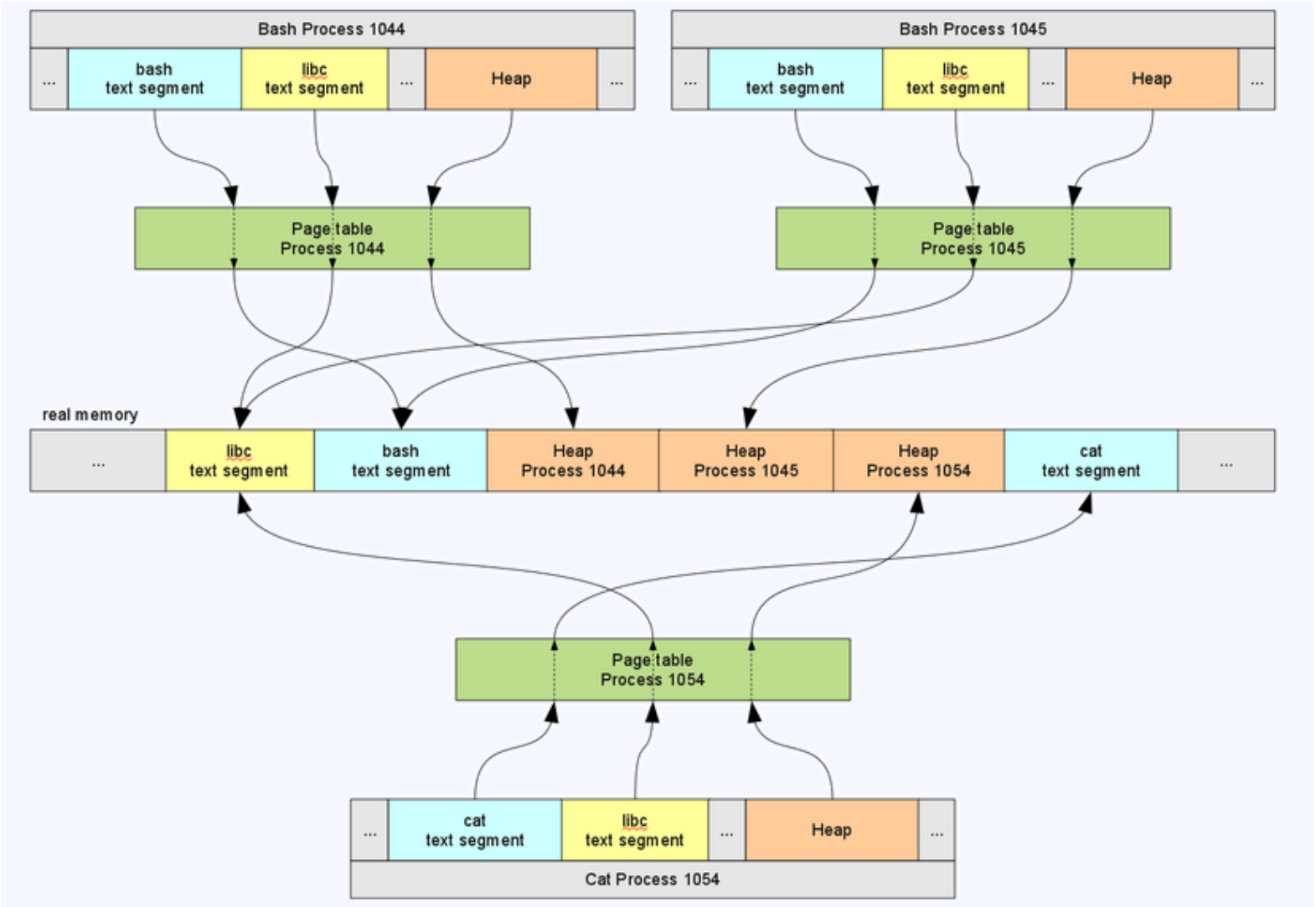
3. Program tries to access new memory.
Processor page faults.



4. Kernel assigns page frame to process, creates PTE, resumes execution. Program is unaware anything happened.

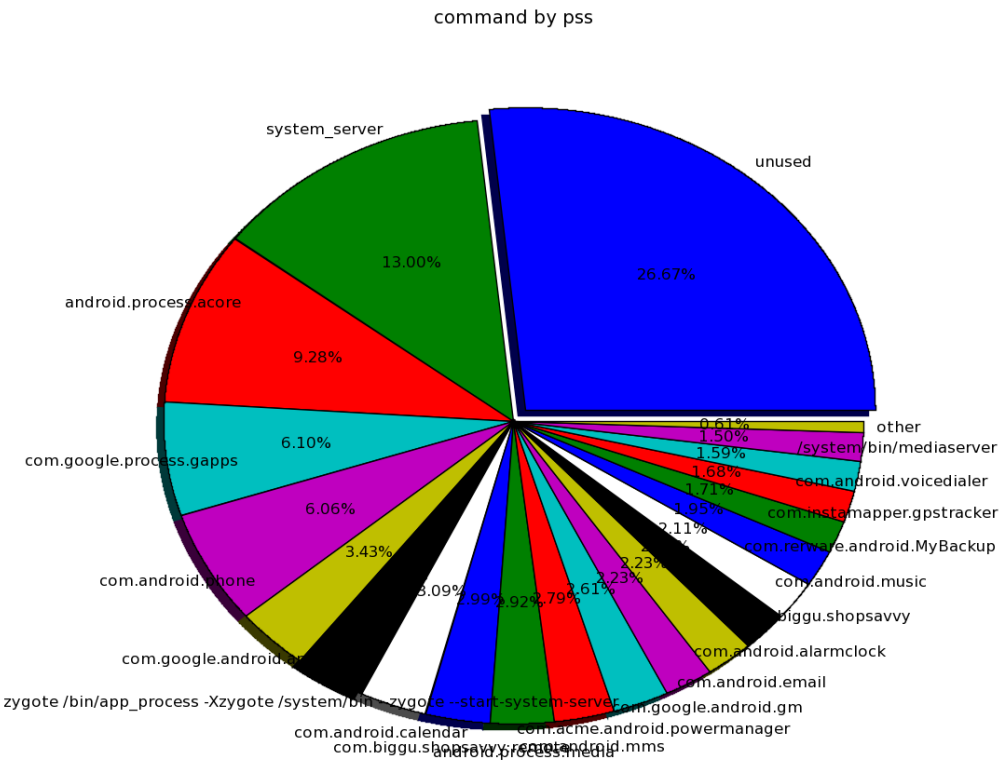


vss/rss/pss/uss



smem – process memory

$ ./smemcap >memdata.tar

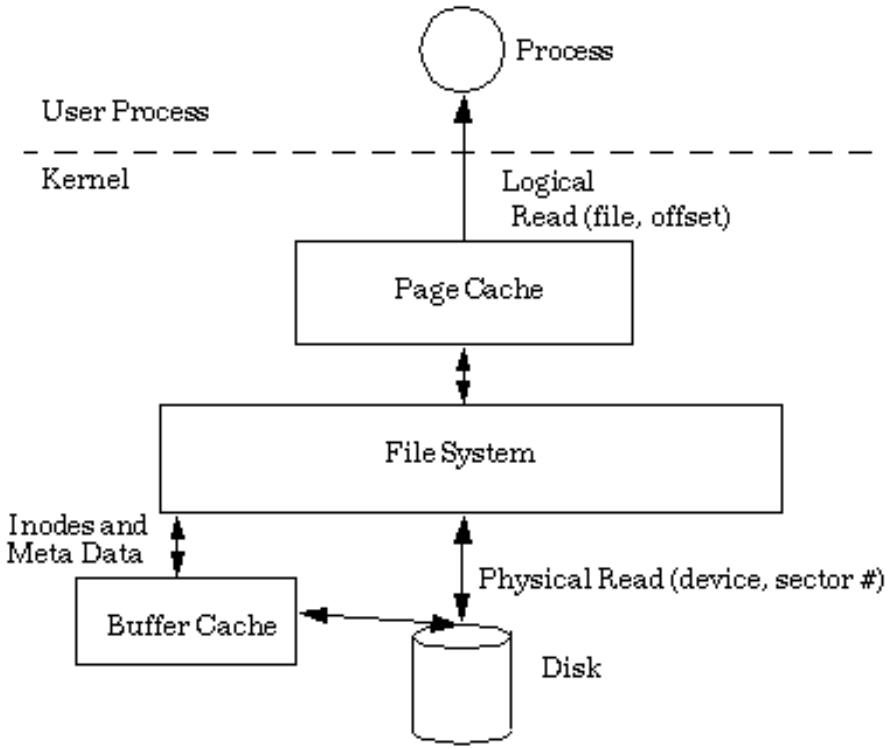


page cache的目的

- Two serious problems must be solved by the OS when it comes to files.
  - The first one is the mind-blowing slowness of hard drives, and disk seeks in particular, relative to memory.
  - The second is the need to load file contents in physical memory once and share the contents among programs.

Page cache的位置

```
# free -m
             total       used       free     shared    buffers     cached
Mem:         15976      15195        781          0        167       9153
-/+ buffers/cache:       5874      10102
Swap:         2000          0       1999
```

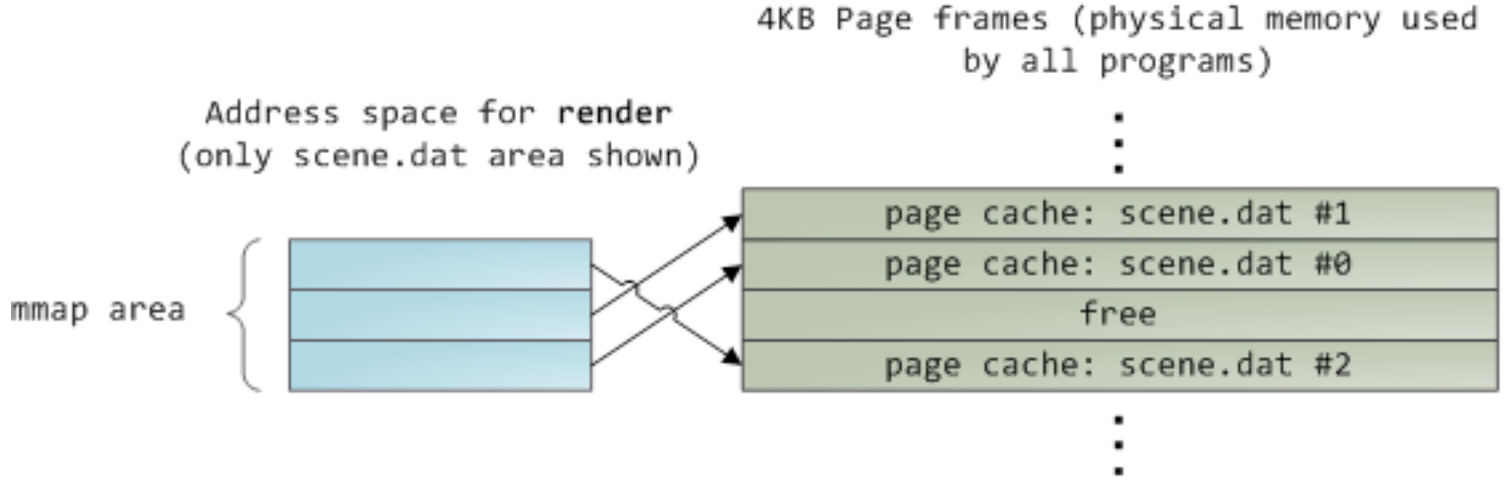


mmap

- **mmap** is a POSIX-compliant Unix system call that maps files or devices into memory. It is a method of memory-mapped file I/O. It naturally implements demand paging, because initially file contents are not entirely read from disk and do not use physical RAM at all. The actual reads from disk are performed in a "lazy" manner, after a specific location is accessed.
  - *caddr_t mmap(caddr_t addr, size_t len, int prot, int flags, int fd, off_t off);*
  - *int munmap(caddr_t addr, size_t len);*

mmap & page cache

- When you use file mapping, the kernel maps your program's virtual pages directly onto the page cache.
- When you map a file its contents are not brought into memory all at once, but rather on demand via page faults. The fault handler maps your virtual pages onto the page cache after obtaining a page frame with the needed file contents.

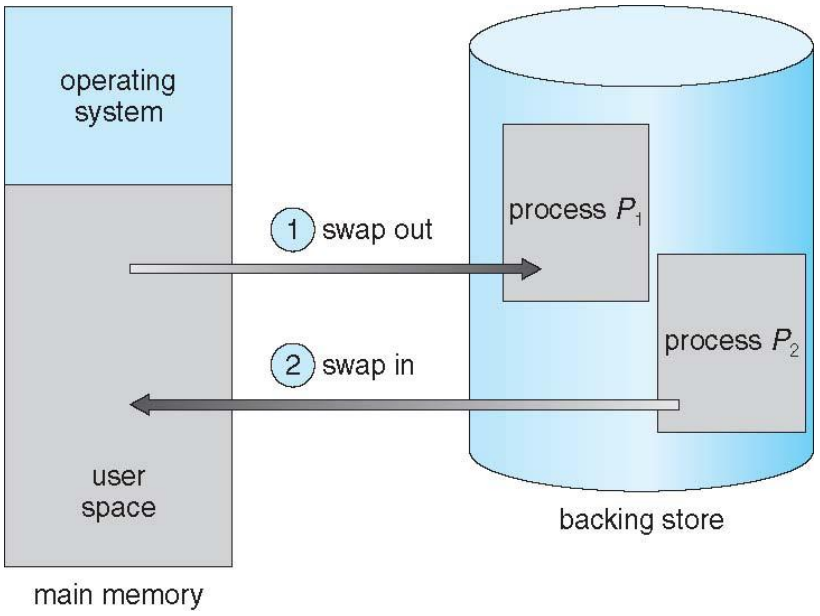


File-backed vs. Anonymous

- *File-backed mapping* maps an area of the process's virtual memory to files
- *Anonymous mapping* maps an area of the process's virtual memory not backed by any file

Swapping

- anonymous pages (those without a disk file to serve as backing storage - pages of malloc()'d memory, for example) are assigned an entry in the system swapfile, and those pages are maintained in the swap cache.



Linux Page Replacement

- Page replacement strategy applies to all page cache and anonymous pages
- Linux uses a variation of the clock algorithm to approximate an LRU page-replacement strategy

valgrind - memory check

- Illegal read / Illegal write errors
- Use of uninitialised values
- Use of uninitialised or unaddressable values in system calls
- Illegal frees
- When a heap block is freed with an inappropriate deallocation function
- Overlapping source and destination blocks
- Memory leak detection

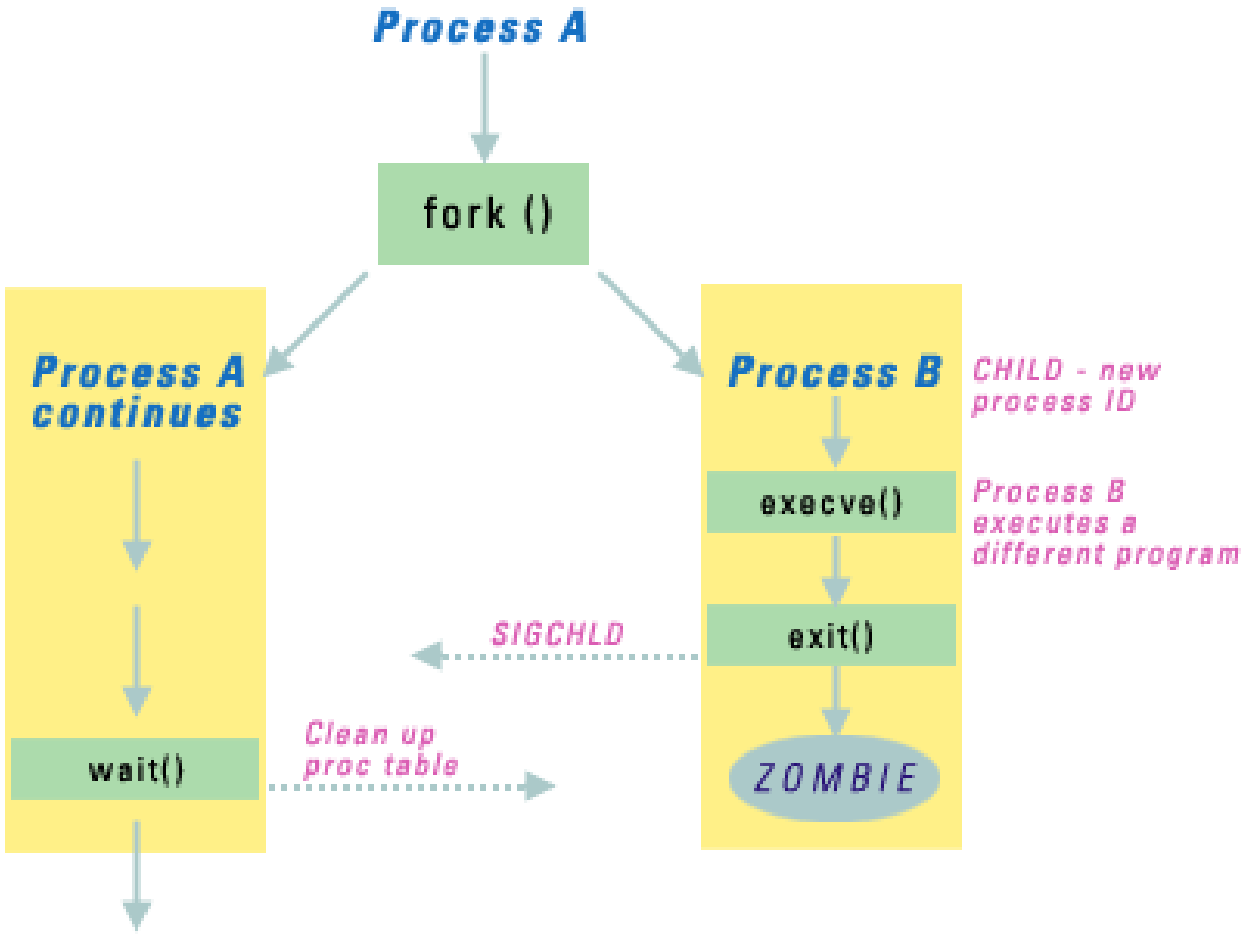
valgrind - reports

valgrind --leak-check=full --track-origins=yes /data/check

```
cdlyangj:/home/workspace/intro # valgrind ./sample
==32372== Memcheck, a memory error detector.
==32372== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
==32372== Using LibVEX rev 1854, a library for dynamic binary translation.
==32372== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP.
==32372== Using valgrind-3.3.1, a dynamic binary instrumentation framework.
==32372== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al.
==32372== For more details, rerun with: -v
==32372==
==32372== Invalid write of size 4
==32372==    at 0x80483CF: fun (sample.c:6)
==32372==    by 0x80483EC: main (sample.c:11)
==32372==  Address 0x416f050 is 0 bytes after a block of size 40 alloc'd
==32372==    at 0x4023888: malloc (vg_replace_malloc.c:207)
==32372==    by 0x80483C5: fun (sample.c:5)
==32372==    by 0x80483EC: main (sample.c:11)
==32372==
==32372== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 3 from 1)
==32372== malloc/free: in use at exit: 40 bytes in 1 blocks.
==32372== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==32372== For counts of detected errors, rerun with: -v
==32372== searching for pointers to 1 not-freed blocks.
==32372== checked 56,780 bytes.
==32372==
==32372== LEAK SUMMARY:
==32372==    definitely lost: 40 bytes in 1 blocks.
==32372==      possibly lost: 0 bytes in 0 blocks.
==32372==    still reachable: 0 bytes in 0 blocks.
==32372==         suppressed: 0 bytes in 0 blocks.
==32372== Rerun with --leak-check=full to see details of leaked memory.
```

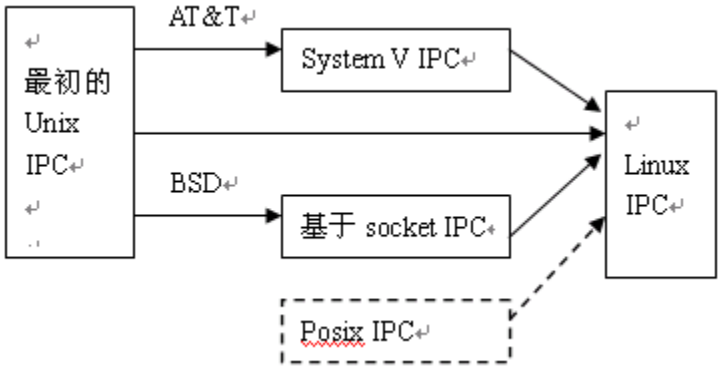
多进程开发

fork特点


Process A
fork ()
Process A continues
wait()
Process B
CHILD - new process ID
execve()
Process B executes a different program
exit()
SIGCHLD
Clean up proc table
ZOMBIE

Linux下进程间通信概述

- **管道(Pipe)及有名管道(named pipe)：**管道可用于具有亲缘关系进程间的通信，有名管道，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信
- **信号(Signal)：**信号是在软件层次上对中断机制的一种模拟，它是比较复杂的通信方式，用于通知接受进程有某事件发生，一个进程收到一个信号与处理器收到一个中断请求在效果上可以说是一样的
- **消息队列：**消息队列是消息的链接表

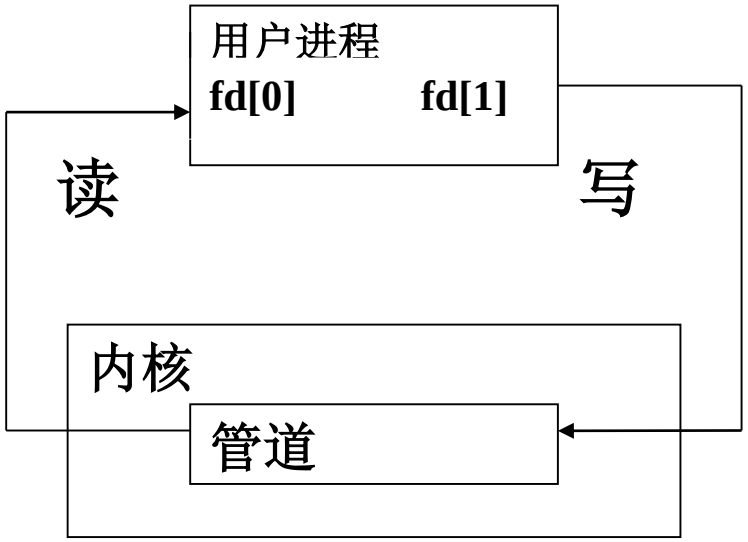


Linux下进程间通信概述

- **共享内存：**可以说这是最有用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以即时看到对方进程中对共享内存中数据的更新。这种通信方式需要依靠某种同步机制，如互斥锁和信号量等
- **信号量：**主要作为进程间以及同一进程不同线程之间的同步手段
- **套接字(Socket)：**这是一种更为一般的进程间通信机制，它可用于不同机器之间的进程间通信，应用非常广泛

管道

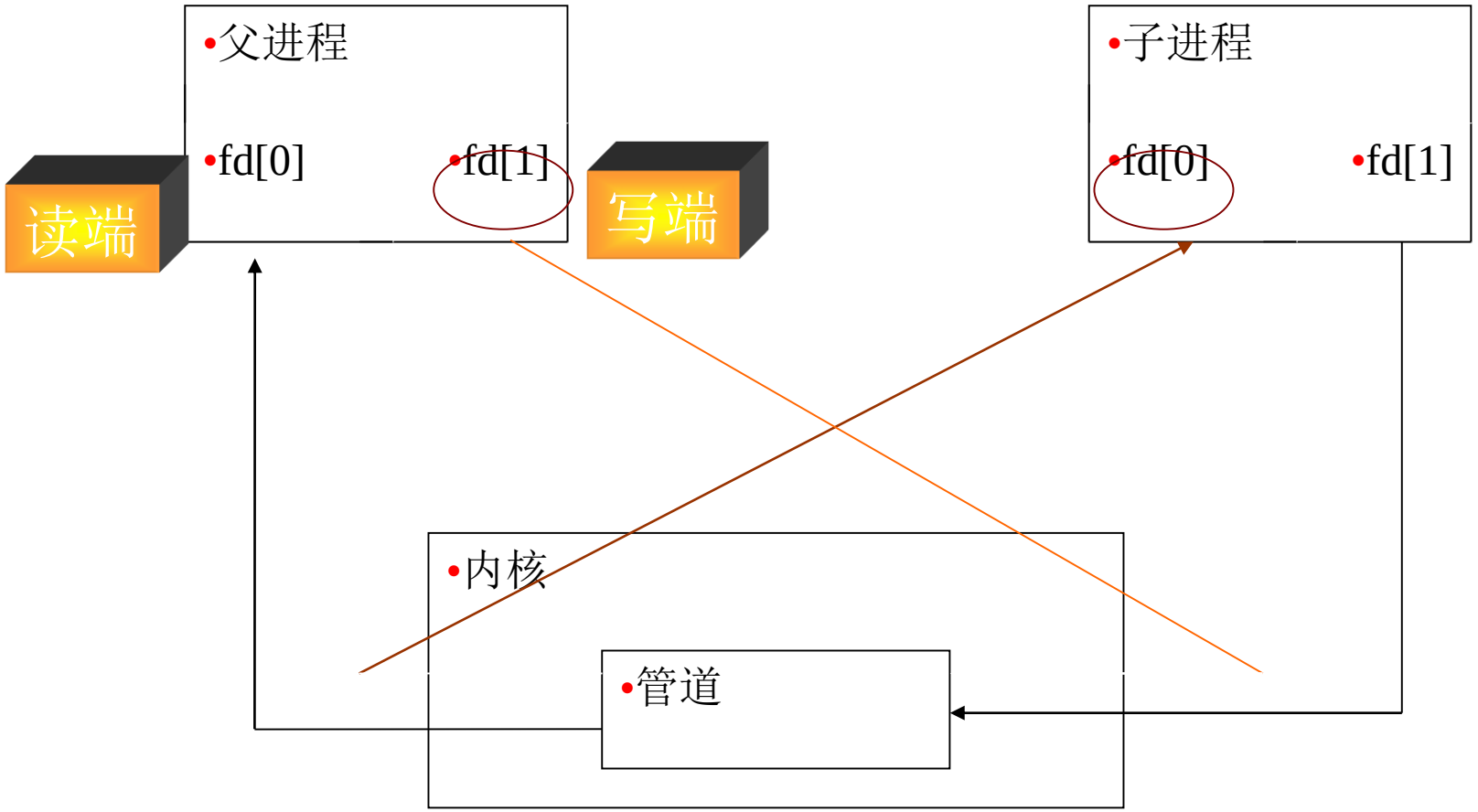
- 管道是基于文件描述符的通信方式，当一个管道建立时，它会创建两个文件描述符fds[0]和fds[1]，其中fds[0]固定用于读管道，而fd[1]固定用于写管道
- 构成了一个半双工的通道。



管道创建

所需头文件	**#include <unistd.h>**
函数原型	**int pipe(int fd[2])**
函数传入值	**fd[2]：管道的两个文件描述符，之后就可以直接操作这两个文件描述符**
函数返回值	**成功：0**
	出错：-1

管道创建与关闭



FIFO

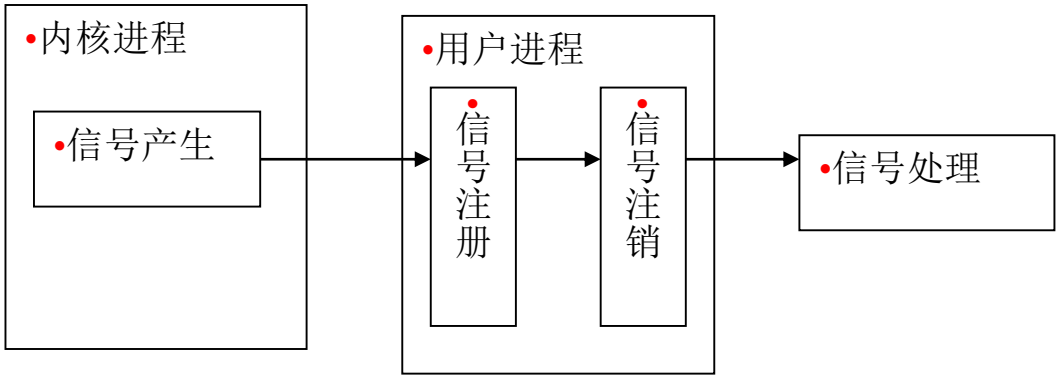
- 无名管道只能用于具有亲缘关系的进程之间，这就大大限制了管道的使用
- 有名管道可以使互不相关的两个进程实现彼此通信。该管道可以通过路径名来指出，并且在文件系统中是可见的
- 两个进程就可以把FIFO当作普通文件一样进行读写操作，使用非常方便
- FIFO是严格遵循先进先出规则的，对管道及FIFO的读总是从开始处返回数据，对它们的写则把数据添加到末尾
- 不支持如lseek()等文件定位操作

信号通信

- 信号是在软件层次上对中断机制的一种模拟，是一种异步通信方式
- 信号可以直接进行用户空间进程和内核进程之间的交互，内核进程也可以利用它来通知用户空间进程发生了哪些系统事件。它可以在任何时候发给某一进程，而无需知道该进程的状态。如果该进程当前并未处于执行态，则该信号就由内核保存起来，直到该进程恢复执行再传递给它；如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消时才被传递给进程

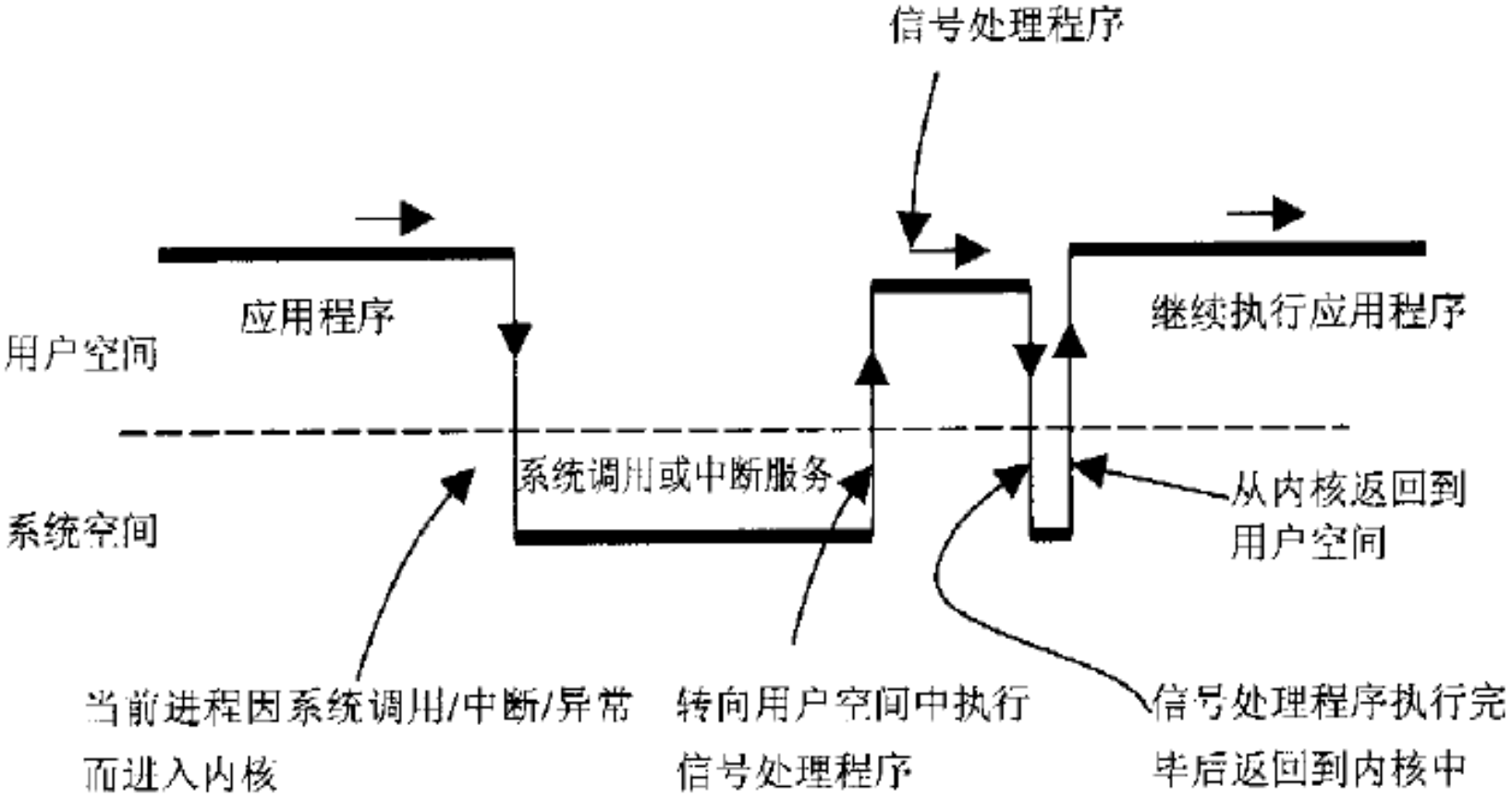
信号通信

- 信号的生存周期



- 用户进程对信号的响应方式：
  - 忽略信号，即对信号不做任何处理，但是有两个信号不能忽略：即SIGKILL及SIGSTOP。
  - 捕捉信号。定义信号处理函数，当信号发生时，执行相应的处理函数。
  - 执行缺省操作，Linux对每种信号都规定了默认操作

信号检测与处理流程



信号的检测与处理流程图

信号发送与捕捉

- kill()和raise()
  - kill函数同读者熟知的kill系统命令一样，可以发送信号给进程或进程组(实际上，kill系统命令只是kill函数的一个用户接口)。这里要注意的是，它不仅可以中止进程(实际上发出SIGKILL信号)，也可以向进程发送其他信号
  - kill －l查看系统支持的信号列表
  - raise函数允许进程向自身发送信号

信号发送与捕捉

- alarm()和pause()
- alarm也称为闹钟函数，它可以在进程中设置一个定时器，当定时器指定的时间到时，它就向进程发送SIGALARM信号。要注意的是，一个进程只能有一个闹钟时间，如果在调用alarm之前已设置过闹钟时间，则任何以前的闹钟时间都被新值所代替
- pause函数是用于将调用进程挂起直到捕捉到信号为止。这个函数非常常用，通常可以用于判断信号是否已到

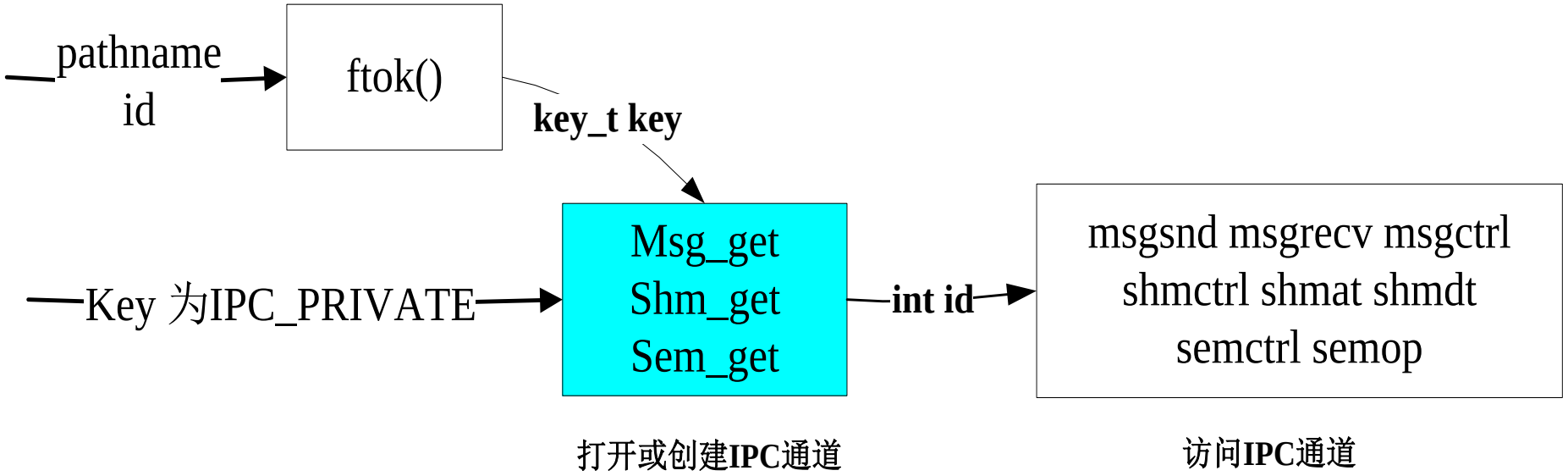
信号的处理

- 特定的信号是一定的进程相联系的
- 一个进程可以决定在该进程中需要对哪些信号进程什么样的处理
- 信号处理的主要方法有两种
  - 使用简单的signal函数
  - 使用信号集函数组
- signal
- sigaction

ps s查看信号状态

```
$ ps s
  UID   PID        PENDING         BLOCKED         IGNORED         CAUGHT STAT TTY
TIME COMMAND
 1000  3509 0000000000000000 0000000000010000 0000000000380004 000000004b817efb Ss
pts/2      0:00 bash
 1000  4030 0000000000000000 0000000000010000 0000000000380004 000000004b817efb Ss
pts/4      0:00 bash
 1000  4389 0000000000000000 0000000000000000 0000000000380004 000000004b817efb Ss+
pts/14     0:00 bash
 1000  4587 0000000000000000 0000000000000000 0000000000380004 000000004b817efb Ss+
pts/19     0:00 bash
 1000  6000 0000000000000000 0000000000000000 0000000000000000 0000000073d3fef9 R+
pts/2      0:00 ps s
```

IPC对象


pathname
id
ftok()
key_t key
Key 为IPC_PRIVATE
Msg_get
Shm_get
Sem_get
int id
msgsnd msgrecv msgctrl
shmctrl shmat shmdt
semctrl semop
打开或创建IPC通道
访问IPC通道

共享内存

- 共享内存是一种最为高效的进程间通信方式，进程可以直接读写内存，而不需要任何数据的拷贝
- 为了在多个进程间交换信息，内核专门留出了一块内存区，可以由需要访问的进程将其映射到自己的私有地址空间
- 进程就可以直接读写这一内存区而不需要进行数据的拷贝，从而大大提高的效率。
- 由于多个进程共享一段内存，因此也需要依靠某种同步机制，如互斥锁和信号量等

共享内存实现

- 共享内存的实现分为两个步骤，
  - 创建共享内存，这里用到的函数是shmget，也就是从内存中获得一段共享内存区域
  - 映射共享内存，也就是把这段创建的共享内存映射到具体的进程空间中去，这里使用的函数是shmat
  - 到这里，就可以使用这段共享内存了，也就是可以使用不带缓冲的I/O读写命令对其进行操作
  - 撤销映射的操作，其函数为shmdt

信号灯

- 二值信号灯：值为0或1。与互斥锁类似，资源可用为1，被锁住不可用则为0。
- 计数信号灯：值在0到n之间。用来统计资源，其值代表可用资源数

System V信号灯

- System V的信号灯是指一个或者多个信号灯的一个集合。其中的每一个都是单独的计数信号灯。
- 主要函数semget，semop，semctrl

ipcs

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0xe9b42ad8 2228225    oracle     640        88080384   13
0x8cdf18fc 2359299    oracle     640        528482304  26

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xbbe791cc 98304      oracle     640        44
0x9d085d3c 491521     oracle     640        154
```

ipcs - Semaphores

- The ncount is the number of processes blocking on the semaphore, waiting for it to increment. The zcount is the number of processes blocking on the semaphore, waiting for it to go to zero（若sem_op是0，那么调用者希望等到semval变为0）. The pid column identifies the most recent process to complete a semaphore operation; it does not identify processes waiting on the semaphore

```
baohua@baohua-VirtualBox:/lib$  ipcs -s -i 0

Semaphore Array semid=0
uid=0      gid=0      cuid=0    cgid=0
mode=0666, access_perms=0666
nsems = 2
otime = Sun May 31 22:02:34 2015
ctime = Sun May 31 22:02:34 2015
semnum     value      ncount     zcount     pid
0          1          0          0          1854
1          100        0          0          1854
```

ipcs – shared memory

The creator's PID is listed as cpid and the last PID to attach or detach is listed as lpid. You may think that as long as nattch is 2, these are the only processes.
Don't forget that there is no guarantee that the object creator is still attached (or still running).

```
$ ipcs -m -p
------ Shared Memory Creator/Last-op --------
shmid owner cpid lpid
163840 john 2790 2906
32769 john 2788 0
65538 john 2788 0
98307 john 2788 0
131076 john 2788 0
196613 john 2897 2754
```

POSIX shm

- shm_open(): open/create SHM object
- mmap(): map SHM object
- shm_unlink(): remove SHM object pathname
- fstat(): retrieve info (size, ownership, permissions)
- ftruncate(): change size
- fchown(): fchmod(): change ownership, permissions

POSIX sem

- sem_init(semp, pshared, value): initialize semaphore pointed to by semp to value
- sem_post(semp): add 1 to value
- sem_wait(semp): subtract 1 from value
- sem_destroy(semp): free semaphore, release resources back to system
- sem_open(): open/create semaphore
- sem_unlink(): remove semaphore pathname

POSIX共享内存实例

```
int main(int argc, char **argv)
{
        /* Create shared memory object and set its size */
        fd = shm_open(argv[1], O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
        if (fd == -1) {
                perror("shm open failed\n"); /* Handle error */;
                return -1;
        }

        if (ftruncate(fd, sizeof(struct region)) == -1) {
                perror("ftruncate failed\n")    /* Handle error */;
                return -1;
        }

        /* Map shared memory object */
        rptr = mmap(NULL, sizeof(struct region),
                        PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        if (rptr == MAP_FAILED) {
                perror("mmap failed\n") /* Handle error */;
                return -1;
        }

        /* Now we can refer to mapped region using fields of rptr;
           for example, rptr->len */
        memset(rptr->buf, 0, MAX_LEN);
        rptr->len=1000;
        strcpy(rptr->buf, "hello world");
}
```

netstat – socket communication status

```
barry@barry-VirtualBox:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 barry-VirtualBox.:34386 58.63.236.40:http       ESTABLISHED
tcp        0      0 barry-VirtualBox.:58192 183.136.195.42:http     ESTABLISHED
tcp        0      0 barry-VirtualBox.:58172 183.136.195.42:http     ESTABLISHED
tcp        0      0 barry-VirtualBox.:39824 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:39899 113.108.216.252:https   ESTABLISHED
tcp        0      0 barry-VirtualBox.:39939 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:39900 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:43820 183.60.187.41:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:58171 183.136.195.42:http     ESTABLISHED
tcp        0      0 barry-VirtualBox.:39473 a23-65-11-27.deplo:http FIN_WAIT2
tcp        0      0 barry-VirtualBox.:39826 218.30.66.145:http      TIME_WAIT
tcp        0      0 barry-VirtualBox.:50834 162.78.142.219.bro:http ESTABLISHED
tcp        0      0 barry-VirtualBox.:50829 162.78.142.219.bro:http ESTABLISHED
tcp        0      0 barry-VirtualBox.:39837 218.30.66.145:http      TIME_WAIT
tcp        0      0 barry-VirtualBox.:43846 183.60.187.41:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:39827 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:49905 ip223.hichina.com:http  ESTABLISHED
tcp        0      0 barry-VirtualBox.:39822 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:39811 218.30.66.145:http      ESTABLISHED
tcp        0      0 barry-VirtualBox.:41742 privet.canonical.c:http ESTABLISHED
tcp        0      0 barry-VirtualBox.:59902 243.78.142.219.bro:http ESTABLISHED
tcp        0      0 barry-VirtualBox.:39968 113.108.216.252:https   ESTABLISHED
tcp        0      0 barry-VirtualBox.:50832 162.78.142.219.bro:http ESTABLISHED
```

lsof (1)

lsof will show all open files in the system

```
$ lsof -p 16894
COMMAND   PID USER   FD   TYPE DEVICE     SIZE   NODE NAME
cat     16894 john  cwd    DIR  253,0     4096 575355 /home/john
cat     16894 john  rtd    DIR  253,0     4096      2 /
cat     16894 john  txt    REG  253,0    21104 159711 /bin/cat
cat     16894 john  mem    REG  253,0   126648 608855 /lib/ld-2.3.5.so
cat     16894 john  mem    REG  253,0  1489572 608856 /lib/libc-2.3.5.so
cat     16894 john  mem    REG    0,0           0 [heap]
cat     16894 john  mem    REG  253,0 48501472 801788 .../locale-archive
cat     16894 john    0r  FIFO    0,5          176626 pipe
cat     16894 john    1u   CHR  136,2             4 /dev/pts/2
cat     16894 john    2w   CHR    1,3          1510 /dev/null
cat     16894 john    3r   REG  253,0      167 575649 /home/john/.bashrc
```

```
barry@barry-virtual-machine:~$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 1048576    barry      600        4194304    2          dest
0x00000000 229377     barry      600        524288     2          dest
0x00000000 557060     barry      700        15600      2          dest
0x00000000 950278     barry      600        33554432   2          dest
0x00000000 917512     barry      600        2097152    2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x002fa327 0          root       666        2

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

barry@barry-virtual-machine:~$ lsof | grep 1048576
gnome-ter 2710           barry  DEL       REG        0,4          1048576 /SYSV00000000
gdbus     2710 2712      barry  DEL       REG        0,4          1048576 /SYSV00000000
dconf     2710 2714      barry  DEL       REG        0,4          1048576 /SYSV00000000
gmain     2710 2715      barry  DEL       REG        0,4          1048576 /SYSV00000000
pool      2710 3373      barry _DEL       REG        0,4          1048576 /SYSV00000000
```

lsof (2)

lsof will show all open files in the system

```
$ ls -l !$
ls -l /proc/19991/fd
total 5
lrwx------  1 john john 64 Apr 12 23:33 0 -> /dev/pts/4
lrwx------  1 john john 64 Apr 12 23:33 1 -> /dev/pts/4
lrwx------  1 john john 64 Apr 12 23:33 2 -> /dev/pts/4
lr-x------  1 john john 64 Apr 12 23:33 3 -> pipe:[318960]
l-wx------  1 john john 64 Apr 12 23:33 4 -> pipe:[318960]
```

```
$ lsof | head -1 && lsof | grep 318960
COMMAND  PID    USER   FD      TYPE     DEVICE     SIZE       NODE NAME
ppipe  19991    john    3r      FIFO        0,5                318960 pipe
ppipe  19991    john    4w      FIFO        0,5                318960 pipe
ppipe  19992    john    3r      FIFO        0,5                318960 pipe
ppipe  19992    john    4w      FIFO        0,5                318960 pipe
```

```
$ lsof -n -i tcp
COMMAND     PID    USER   FD   TYPE DEVICE SIZE NODE NAME
portmap    1853     rpc    4u  IPv4   4847       TCP *:sunrpc (LISTEN)
rpc.statd  1871 rpcuser    6u  IPv4   4881       TCP *:32769 (LISTEN)
smbd       2120    root   20u  IPv4   5410       TCP *:microsoft-ds (LISTEN)
smbd       2120    root   21u  IPv4   5411       TCP *:netbios-ssn (LISTEN)
X          2371    root    1u  IPv6   6310       TCP *:x11 (LISTEN)
X          2371    root    3u  IPv4   6311       TCP *:x11 (LISTEN)
xinetd    20338    root    5u  IPv4 341172       TCP *:telnet (LISTEN)
sshd      23444    root    3u  IPv6 487790       TCP *:ssh (LISTEN)
sshd      23555    root    3u  IPv6 502673       ...
       TCP 192.168.163.128:ssh->192.168.163.1:1344 (ESTABLISHED)
sshd      23557    john    3u  IPv6 502673       ...
       TCP 192.168.163.128:ssh->192.168.163.1:1344 (ESTABLISHED)
```

Linux多线程开发

POSIX标准的要求

- 1. 查看进程列表的时候，相关的一组task_struct应当被展现为列表中的一个节点；
- 2. 发送给这个"进程"的信号(对应kill系统调用)，将被对应的这一组task_struct所共享，并且被其中的任意一个"线程"处理；
- 3. 发送给某个"线程"的信号(对应pthread_kill)，将只被对应的一个task_struct接收，并且由它自己来处理；
- 4. 当"进程"被停止或继续时(对应SIGSTOP/SIGCONT信号)，对应的这一组task_struct状态将改变；
- 5. 当"进程"收到一个致命信号(比如由于段错误收到SIGSEGV信号)，对应的这一组task_struct将全部退出；

Linux线程模型 LinuxThreads

- Light weight process:在Linux的内核中，进程是进行调度的最小单位，并没有真正的对线程的支持，可以说线程只是用户空间的概念，在内核中只能用进程来实现线程。
- LinuxThreads:
  - 内核并没有线程组的概念，LinuxThreads模型只能在pthread库中对每个进程下的线程组都增加一个额外的线程来进行管理即管理线程(Manager Thread)。
  - 管理线程负责对管理同一进程下的其他线程的创建、退出，资源的分配和回收，及线程切换等，非常复杂，开销也很大，而且一旦管理线程被杀死，其他线程都不能正确回收。
  - 当时内核中缺乏对线程同步操作的支持，因此pthread库中的互斥量mutex等只能采用信号来实现，效率非常低。
  - LinuxThreads模型在某些方面也不符合POSIX标准，比如用它在同一进程中创建的线程，都有自己唯一的进程ID，而不是同一个ID，而且接收信号的时候也只能按线程接收。

LinuxThreads不符合POSIX

- linuxthreads除了第5点以外, 都没有实现(实际上是无能为力):
  - 1.如果运行了A程序, A程序创建了10个线程, 那么在shell下执行ps命令时将看到11个A进程, 而不是1个(注意, 也不是10个, 下面会解释);
  - 2.不管是kill还是pthread_kill, 信号只能被一个对应的线程所接收;
  - 3.SIGSTOP/SIGCONT信号只对一个线程起作用;

Linux线程模型NPTL

- NPTL:NPTL模型仍然采用clone()系统调用，在内核中还是通过进程来模拟线程，不过对LinuxThreads的一些主要缺点有很大的改进：
  - 在内核中改进了clone()系统调用并引入新的exit_group()系统调用，优化了线程创建和结束，并且使得pthread库中不再需要管理线程，减少了很多开销。
  - 在内核中引入新的线程同步原语Futex，并基于Futex来实现线程同步操作如互斥量、条件变量等，改进了原来LinuxThreads基于信号实现时的低效率。
  - 符合POSIX标准：发送信号时能发送到同一进程下的所有线程；getpid()会为同一进程下的所有线程返回相同的进程ID。

TGID

- 在linux 2.6中, 内核有了线程组的概念, task_struct结构中增加了一个tgid(thread group id)字段
  - 如果这个task是一个"主线程", 则它的tgid等于pid, 否则tgid等于进程的pid(即主线程的pid).
  - 在clone系统调用中, 传递CLONE_THREAD参数就可以把新进程的tgid设置为父进程的tgid(否则新进程的tgid会设为其自身的pid).
  - getpid(获取进程ID)系统调用返回的也是tast_struct中的tgid, 而tast_struct中的pid则由gettid系统调用来返回

signal_pending

- 为了应付"发送给进程的信号"和"发送给线程的信号", task_struct里面维护了两套signal_pending: 一套是线程组共享的, 一套是线程独有的.
  - 通过kill发送的信号被放在线程组共享的signal_pending中, 可以由任意一个线程来处理;
  - 通过pthread_kill发送的信号(pthread_kill是pthread库的接口, 对应的系统调用中tkill)被放在线程独有的signal_pending中, 只能由本线程来处理.

当线程停止/继续, 或者是收到一个致命信号时, 内核会将处理动作施加到整个线程组中.

线程创建与退出

- 线程的创建：pthread_create()
- 线程的退出：
  - 线程handler返回
  - pthread_exit()
  - pthread_cancel()
- 等待线程结束：pthread_join()
- 线程的分离：pthread_detach()

线程创建实例

```
#define THREAD_NUMBER 2
int retval_hello1= 2, retval_hello2 = 3;

void* hello1(void *arg)
{
  char *hello_str = (char *)arg;
  sleep(1);
  printf("%s\n", hello_str);
  pthread_exit(&retval_hello1);
}

void* hello2(void *arg)
{
  char *hello_str = (char *)arg;
  sleep(2);
  printf("%s\n", hello_str);
  pthread_exit(&retval_hello2);
}
```

```
int main(int argc, char *argv[])
{
  int i;   int ret_val;   int *retval_hello[2];
  pthread_t pt[THREAD_NUMBER];      const char
     *arg[THREAD_NUMBER];
  arg[0] = "hello world from thread1";
  arg[1] = "hello world from thread2";
  printf("Begin to create threads...\n");

  ret_val = pthread_create(&pt[0], NULL, hello1, (void *)arg[0]);
  if (ret_val != 0 ) { printf("pthread_create error!\n");   exit(1); }
  ret_val = pthread_create(&pt[1], NULL, hello2, (void *)arg[1]);
  if (ret_val != 0 ) { printf("pthread_create error!\n");  exit(1);   }
  printf("Now, the main thread returns.\n");
  printf("Begin to wait for threads...\n");
  for(i = 0; i < THREAD_NUMBER; i++) {
     ret_val = pthread_join(pt[i], (void **)&retval_hello[i]);
     …
      }
  return 0;
}
```

信号量

对POSIX信号量的操作函数：

- int sem_init(sem_t *sem, int pshared, unsigned int value);
- int sem_wait(sem_t * sem);
- int sem_trywait(sem_t * sem);
- int sem_post(sem_t * sem);
- int sem_getvalue(sem_t * sem, int * sval);
- int sem_destroy(sem_t * sem);

信号量实例

```
void *produce(void *arg)
{
  int i;
  for (i = 0; i < nitems; i++) {
    sem_wait(&shared.nempty);
    sem_wait(&shared.mutex);
    shared.buff[i % NBUFF] = i;
    cout << "Product " << shared.buff[i %
      NBUFF] << endl;
    sem_post(&shared.mutex);
    sem_post(&shared.nstored);
  }
  return(NULL);
}
```

```
void *consume(void *arg)
{
  int i;
  for (i = 0; i < nitems; i++) {
    sem_wait(&shared.nstored);
    sem_wait(&shared.mutex);
    if (shared.buff[i % NBUFF] != i)
      cout << "buff[" << i  <<"] = " <<  shared.buff[i %
      NBUFF] << endl;
    cout << "Consumer:" << shared.buff[i % NBUFF] <<
      endl;
    sem_post(&shared.mutex);
    sem_post(&shared.nempty);
  }
  return(NULL);
}
```

信号量实例(续)

```
int main(int argc, char **argv)
{
  pthread_t tid_produce, tid_consumer;
  if (argc != 2){      cout << "Usage: prodcons number" << endl; exit(0);   }
  nitems = atoi(argv[1]);
  sem_init(&shared.mutex, 0, 1);
  sem_init(&shared.nempty, 0, NBUFF);
  sem_init(&shared.nstored, 0, 0);
  pthread_create(&tid_produce, NULL, produce, NULL);
  pthread_create(&tid_consumer, NULL, consume, NULL);
  pthread_join(tid_produce, NULL);
  pthread_join(tid_consumer, NULL);
  sem_destroy(&shared.mutex);
  sem_destroy(&shared.nempty);
  sem_destroy(&shared.nstored);
  exit(0);
}
```

```
pthread_mutex_t mutex;
pthread_mutex_init (&mutex,NULL);

/* 锁定互斥锁*/
pthread_mutex_lock (&mutex);
...
/* 打开互斥锁*/
pthread_mutex_unlock(&mutex);
```

查看mutex

deadlock.c

```
34              int tid1,tid2;
35              pthread_mutex_init(&mutex_1,NULL);
36              pthread_mutex_init(&mutex_2,NULL);
37              pthread_create(&tid1,NULL,child1,NULL);
38              pthread_create(&tid2,NULL,child2,NULL);
39              do{
40                      sleep(2);
B+> 41          }while(1);
42              pthread_exit(0);
43      }
44
45
46
47
48
49
50
```

```
multi-thre Thread 0xb7df2 In: main                                  Line: 41   PC: 0x804872e
$1 = {__data = {__lock = 2, __count = 0, __owner = 4353, __kind = 0, __nusers = 1, {d = {__espins = 0,
        __elision = 0}, __list = {__next = 0x0}}},
  __size = "\002\000\000\000\000\000\000\001\021\000\000\000\000\000\000\001\000\000\000\000\000\000",
  __align = 2}
(gdb) p mutex_2
$2 = {__data = {__lock = 1, __count = 0, __owner = 4354, __kind = 0, __nusers = 1, {d = {__espins = 0,
        __elision = 0}, __list = {__next = 0x0}}},
  __size = "\001\000\000\000\000\000\000\002\021\000\000\000\000\000\000\001\000\000\000\000\000\000",
  __align = 1}
(gdb)
```

Therac-25 Accidents

Therac-25事件是在软件工程界被大量引用的案例。Therac-25是Atomic Energy of Canada Limited所生产的一种辐射治疗的机器。由于其软件设计时的瑕疵，致命地超过剂量设定导致在1985年6月到1987年1月之间，六件已知的医疗事故中，患者死亡或严重辐射灼伤。

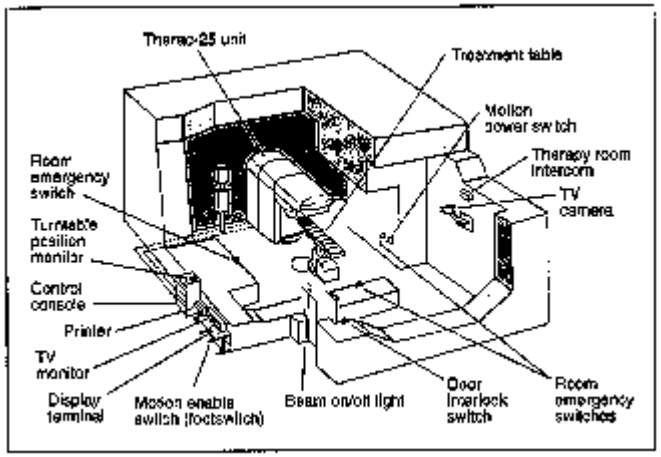
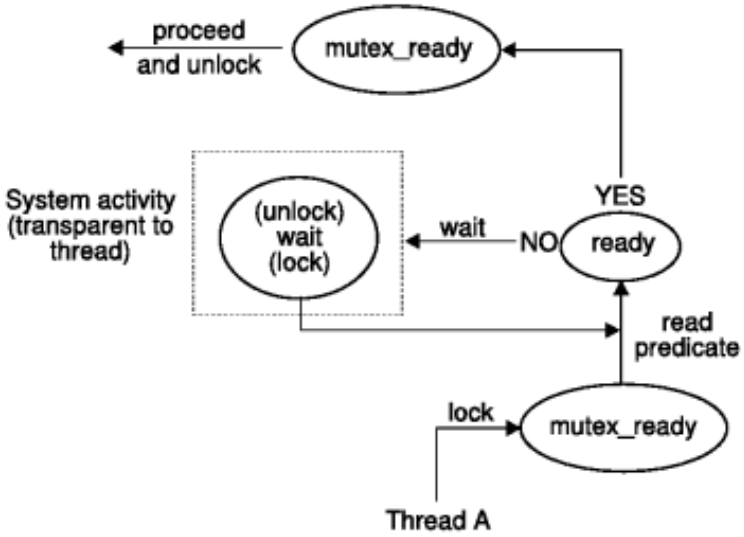


Figure 1. Typical Therac-25 facility.

在机器的编辑阶段，数据的输入速度是该错误出现的关键因素，也就是说，对于一个熟练的操作人员，在重复同样的操作千百次之后，编辑速度越来越快，最终将使‘Malfunction 54 ’信息出现，即超剂量辐射事故发生。参加实验的医生是在经过了相当长的实践后达到了临界的编辑速度。

条件变量

```
pthread_mutex_t count_lock;
pthread_cond_t count_nonzero;
unsigned count;
decrement_count()
{
        pthread_mutex_lock (&count_lock);
        while(count==0)
        pthread_cond_wait( &count_nonzero, &count_lock);
        count=count -1;
        pthread_mutex_unlock (&count_lock);
}

increment_count()
{
        pthread_mutex_lock(&count_lock);
        if(count==0)
                pthread_cond_signal(&count_nonzero);
        count=count+1;
        pthread_mutex_unlock(&count_lock);
}
```



线程属性

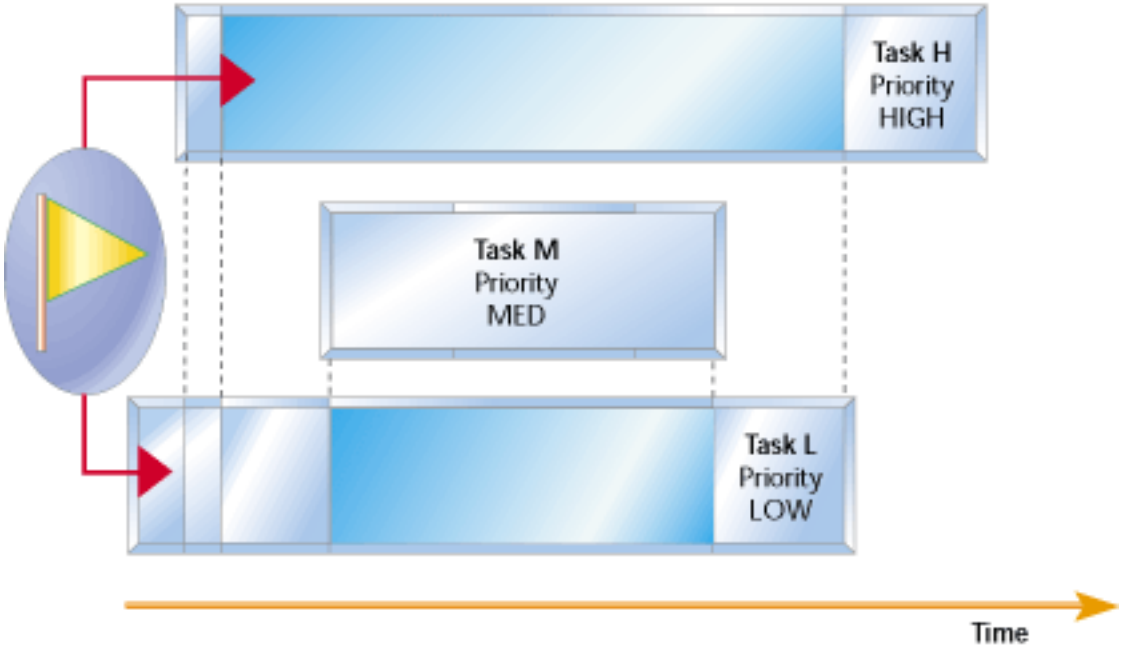
```
int tid = (int)pthread_self();
struct sched_param param;
int orig_policy, orig_prio;

rc = pthread_getschedparam(tid, &orig_policy, &param);
param.sched_priority = 99;
rc = pthread_setschedparam(tid, SCHED_FIFO, &param);
```

线程级全局变量

- 在单线程的程序里，有两种基本的数据：全局变量和局部变量。但在多线程程序里，还有第三种数据类型：线程数据（TSD: Thread-Specific Data）。
- 它和全局变量很象，在线程内部，各个函数可以象使用全局变量一样调用它，但它对线程外部的其它线程是不可见的。这种数据的必要性是显而易见的。例如常见的变量errno，它返回标准的出错信息。

Priority inversions

- when a medium-priority task preempts a lower-priority task using a shared resource on which the higher-priority task is pending. If the higher-priority task is otherwise ready to run, but a medium-priority task is currently running instead, a priority inversion is said to occur.

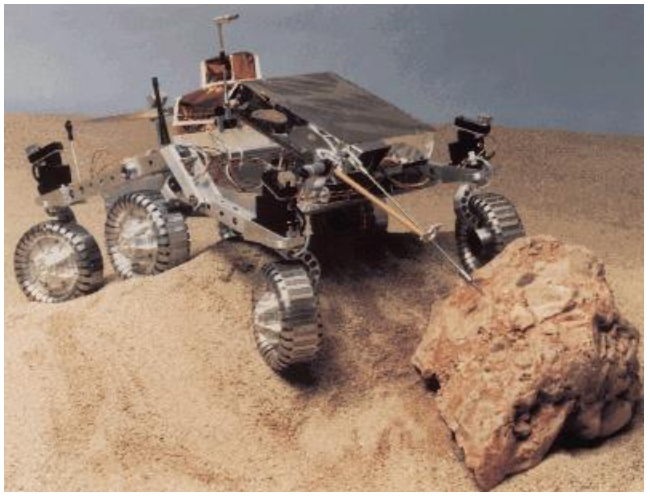


PTHREAD_PRIO_INHERIT

- *A real-time system cannot be real-time if there is no solution for priority inversion, this will cause undesired latencies and even deadlocks. On Linux there is a solution for it in user-land since kernel version 2.6.18 together with Glibc 2.5 (PTHREAD_PRIO_INHERIT).*
- *int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);*
  - ✓ *PTHREAD_PRIO_NONE: A thread's priority and scheduling are not affected by the mutex ownership.*
  - ✓ *PTHREAD_PRIO_INHERIT: This protocol value affects a thread's (such as thrd1) priority and scheduling when higher-priority threads block on one or more mutexes owned by thrd1 where those mutexes are initialized with PTHREAD_PRIO_INHERIT. thrd1 runs with the higher of its priority or the highest priority of any thread waiting on any of the mutexes owned by thrd1.*

what really happened on mars

- 探路者有一个“information bus”，总线管理任务以高优先级运行，负责在总线上放入或者取出各种数据。它被设计为最重要的任务，并且要保证能够每隔一定的时间就可以操作总线。对总线的异步访问是通过互斥锁(mutexes)来保证的。
- 另有一个气象数据搜集任务，它的运行频度不高，也只有低优先级，它只向总线写数据。写的过程是，申请/获得总线互斥量，进行写操作，完成后释放互斥量。
- 探路者上还有一个以中等优先级运行的通信任务，通信任务和总线是没有什么瓜葛的。
- 气象任务（低优先级）获得互斥量并写总线的时候，一个中断的发生导致了通信任务（中优先级）被调度并就绪，调度的时机正好是总线管理任务（高优先级）等待在总线访问互斥量上的时候。



使用GDB调试线程

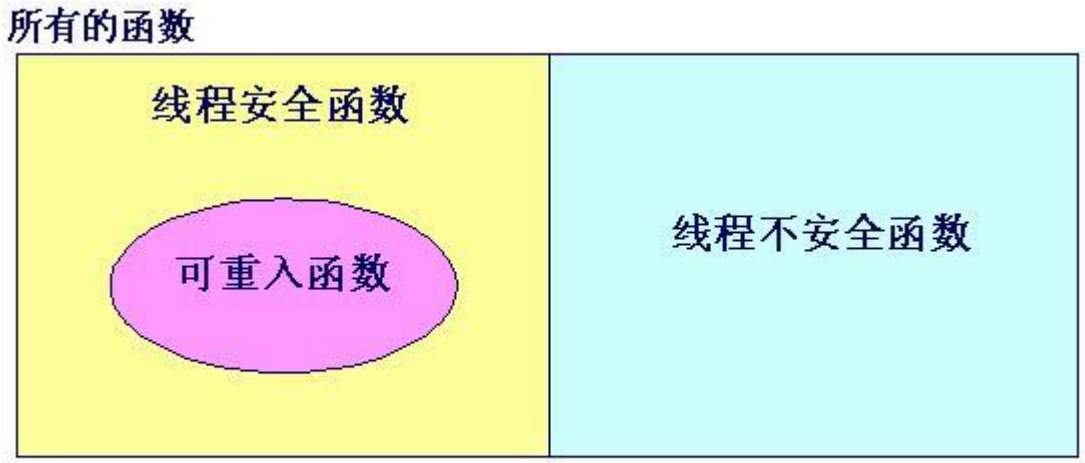
- 使用info thread来查看当前系统中的线程信息

```
(gdb) info thread
  2 Thread -1209439312 (LWP 15333)  0xb7f624d8 in clone () from
    /lib/tls/i686/cmov/libc.so.6
* 1 Thread -1209436480 (LWP 15330)  main (argc=1,
    argv=0xbf9fc2d4) at threadcon.c:62
(gdb)
```

可重入与线程安全

可重入与线程安全(1)

- 可重入函数满足两条件：一，函数是线程安全的；二，函数是可软中断的，执行了软中断处理例程后，再回过头来继续执行函数，结果仍然正确



可重入与线程安全(2)

- 如果一个函数中用到了全局或静态变量，那么它不是线程安全的，也不是可重入的；
- 如果我们对它加以改进，在访问全局或静态变量时使用互斥量或信号量等方式加锁，则可以使它变成线程安全的，但此时它仍然是不可重入的，因为通常加锁方式是针对不同线程的访问，而对同一线程可能出现问题；
- 如果将函数中的全局或静态变量去掉，改成函数参数等其他形式，则有可能使函数变成既线程安全，又可重入。

安全的信号处理函数

- 信号处理函数尽量只执行简单的操作，譬如只是设置一个外部变量，其它复杂的操作留在信号处理函数之外执行；
- errno 是线程安全，即每个线程有自己的 errno，但不是异步信号安全。如果信号处理函数比较复杂，且调用了可能会改变 errno 值的库函数，必须考虑在信号处理函数开始时保存、结束的时候恢复被中断线程的 errno 值；
- 信号处理函数只能调用可以重入的 C 库函数；譬如不能调用 malloc()，free()以及标准 I/O 库函数等；

线程安全函数

- 线程安全的概念比较直观。一般说来，一个函数被称为线程安全的，当且仅当被多个并发线程反复调用时，它会一直产生正确的结果；
- 要确保函数线程安全，主要需要考虑的是线程之间的共享变量。属于同一进程的不同线程会共享进程内存空间中的全局区和堆，而私有的线程空间则主要包括栈和寄存器。因此，对于同一进程的不同线程来说，每个线程的局部变量都是私有的，而全局变量、局部静态变量、分配于堆的变量都是共享的。在对这些共享变量进行访问时，如果要保证线程安全，则必须通过加锁的方式；
- 线程不安全可能导致的后果是显而易见的——共享变量的值由于不同线程的访问，可能发生不可预料的变化，进而导致程序的错误，甚至崩溃。

线程不安全的库函数

asctime()	ecvt()	gethostent()	getutxline()	putc_unlocked()
basename()	encrypt()	getlogin()	gmtime()	putchar_unlocked()
catgets()	endgrent()	getnetbyaddr()	hcreate()	putenv()
crypt()	endpwent()	getnetbyname()	hdestroy()	pututxline()
ctime()	endutxent()	getnetent()	hsearch()	rand()
dbm_clearerr()	fcvt()	getopt()	inet_ntoa()	readdir()
dbm_close()	ftw()	getprotobyname()	l64a()	setenv()
dbm_delete()	gcvt()	getprotobynumber()	lgamma()	setgrent()
dbm_error()	getc_unlocked()	getprotoent()	lgammaf()	setkey()
dbm_fetch()	getchar_unlocked()	getpwent()	lgammal()	setpwent()
dbm_firstkey()	getdate()	getpwnam()	localeconv()	setutxent()
dbm_nextkey()	getenv()	getpwuid()	localtime()	strerror()
dbm_open()	getgrent()	getservbyname()	lrand48()	strtok()
dbm_store()	getgrgid()	getservbyport()	mrand48()	ttyname()
dirname()	getgrnam()	getservent()	nftw()	unsetenv()
dlerror()	gethostbyaddr()	getutxent()	nl_langinfo()	wcstombs()
drand48()	gethostbyname()	getutxid()	ptsname()	wctomb()

可重入函数

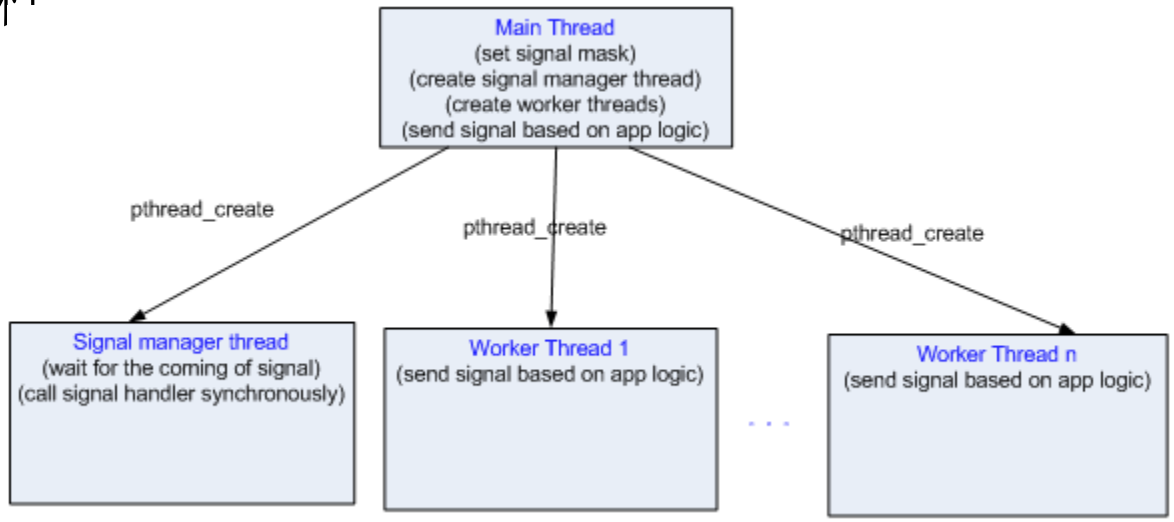
- A computer program or routine is described as reentrant if it can be safely executed concurrently; that is, the routine can be re-entered while it is already running；
- 要确保函数可重入，需满足以下几个条件：
- 1、不在函数内部使用静态或全局数据
- 2、不返回静态或全局数据，所有数据都由函数的调用者提供。
- 3、使用本地数据，或者通过制作全局数据的本地拷贝来保护全局数据。
- 4、不调用不可重入函数。
- 不可重入的后果：
- 不可重入的后果主要体现在象信号处理函数这样需要重入的情况中。如果信号处理函数中使用了不可重入的函数，则可能导致程序的错误甚至崩溃。

信号的屏蔽

```
main(void)
{
    sigset_t newmask, oldmask, pendingmask;
    //设置对信号SIGQUIT的处理函数
    if (signal(SIGQUIT, sig_quit) == SIG_ERR)
    {
        fprintf(stderr, "can't catch SIGQUIT/n");
        exit(1);
    }
    //设置一个空的信号集
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGQUIT); // 在这个信号集中增加SIGQUIT信号

    //在当前进程中增加newmask信号集作为屏蔽信号集，oldmask返回当前进程的信号集
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
    {
        fprintf(stderr, "SIG_BLOCK error/n");
        exit(1);
    }
    sleep(5);
    //返回当前进程信号集
    if (sigpending(&pendingmask) < 0)
    {
        fprintf(stderr, "sigpending error/n");
        exit(1);
    }
    //检查SIGQUIT信号是否在当前信号集中
    if (sigismember(&pendingmask, SIGQUIT))
        printf("/nSIGQUIT pending/n");
    //恢复进程的信号集
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
```

异步信号的同步化

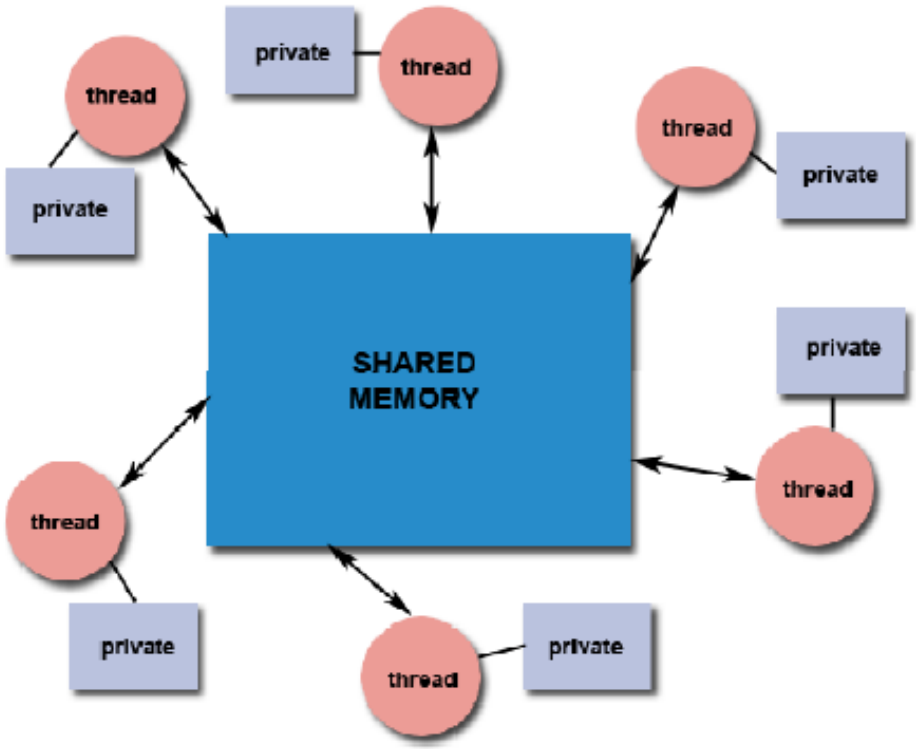
- 主线程设置信号掩码，阻碍希望同步处理的信号；主线程的信号掩码会被其创建的线程继承；
- 主线程创建信号处理线程；信号处理线程将希望同步处理的信号集设为 sigwait（）的第一个参数。
- 主线程创建工作线秤



线程栈与栈溢出

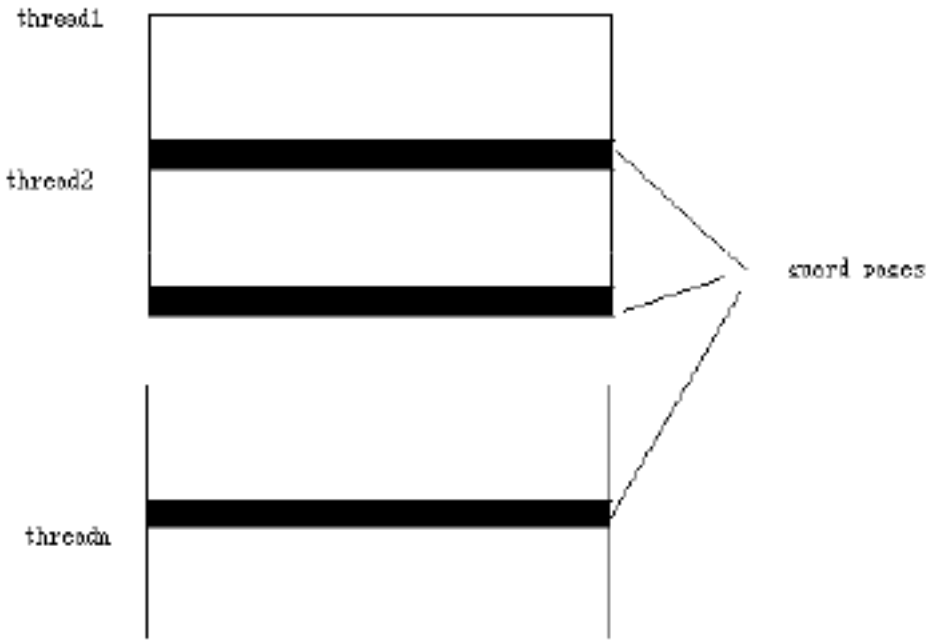
线程栈

栈：线程独有，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立（但内存属于整个进程），因此，栈是thread safe的。操作系统在切换线程的时候会自动的切换栈，栈空间不需要在高级语言里面显式的分配和释放



线程栈Guard page

```
int pthread_attr_setguardsize(
                        pthread_attr_t   *attr,
                        size_t      guardsize);
```



多线程编程模型

H.GOMMA原则

- I/O原则
- 大量运算原则
- 优先级原则
- 功能耦合原则
- 周期性原则
- 偶然耦合原则

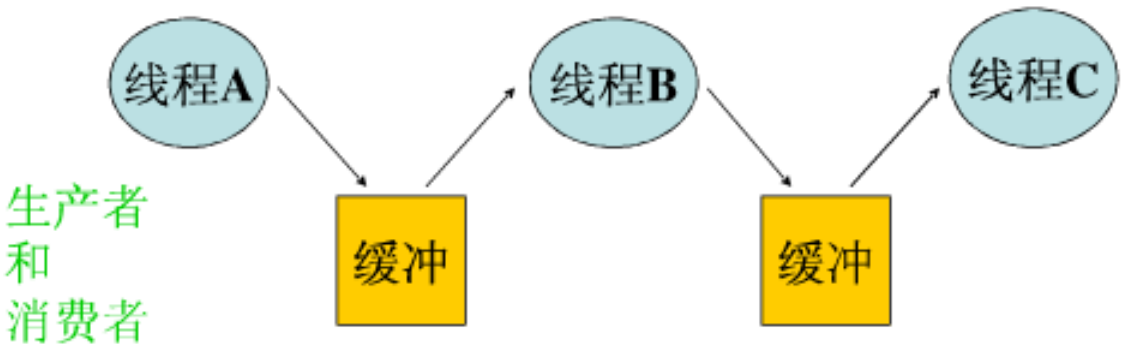
流水线模型

在流水线方式中，数据流串行地被一组线程顺序处理。

每个线程依次在该数据上执行自身特定的操作，并将执行结果传递给流水线中的下一个线程。

各线程的工作量不应差别太大！

但还是有差别！！！？



工作组模型

MIMD：

工作组中的线程，每个从共享队列中删除工作请求、然后做请求的工作。

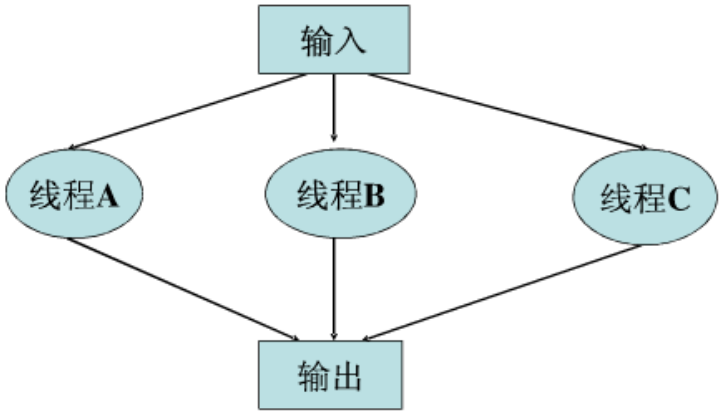
由于队列中每项要求的操作和数据不同，故该模式是类似**MIMD**（多指令多数据）。

SIMD:

工作组中的线程，每个负责处理数据的一部分（例如，某些行或列）。

由于所有的工作线程在不同的数据部分上执行相同的操作，故该模式类似**SIMD**（单令多数据流）。

在工作组模式中，数据被一组线程独立地处理。



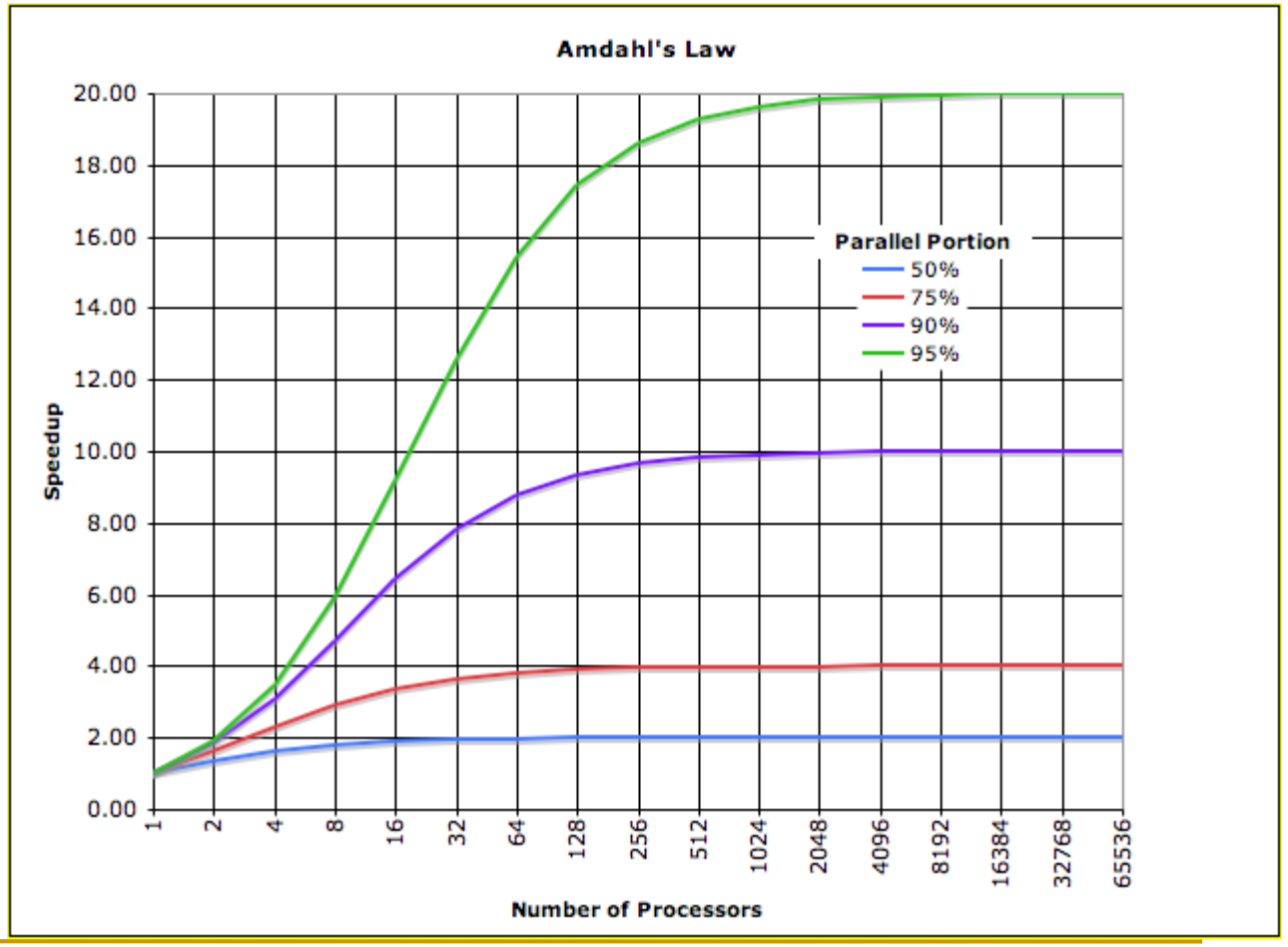
串行化方面的难题

加速系数

$S(p) = p / (1 + (p-1)*f)$

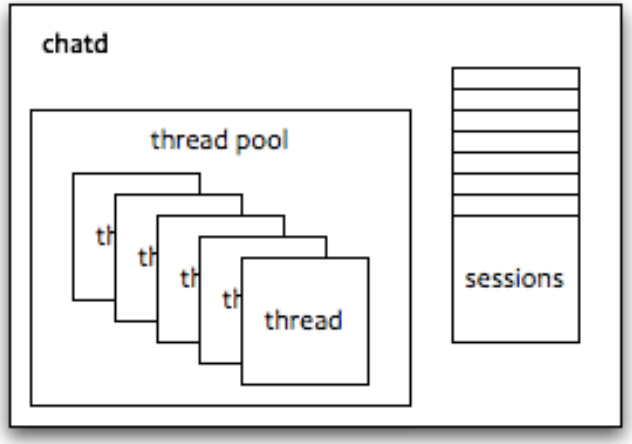
其中 S(p)表示加速系数

p 表示处理器的个数



线程池

- 线程池采用预创建技术，在应用程序启动后，立即创建一定数量的线程，并让这些线程处于阻塞状态（即空闲线程）不消耗CPU，只占用较小的内存空间。
- 当任务来临时，应用程序即唤醒一个空闲的线程，并处理此任务。
- 当此线程处理完任务后，重新回到阻塞状态，并不退出。当系统比较空闲时，大部分线程处于空闲状态，线程池会自动销毁一些线程，回收系统资源。



Linux文件与I/O编程

系统IO与服务器模型

- 在UNIX/Linux下主要的I/O 模型：
  - 阻塞I/O：
    - 最常用、最简单
  - 非阻塞I/O：
    - 可防止进程阻塞在I/O操作上
  - I/O 多路复用：
    - 允许同时对多个I/O进行控制
  - 信号驱动I/O:
    - 一种异步通信模型
  - 异步I/O:
    - 发完I/O请求后不等待其执行完

阻塞/非阻塞模式控制

- 可以通过下列操作来完成整个过程：

(1)

```
int flag
flag = fcntl(sockfd, F_GETFL, 0);
flag |=O_NONBLOCK;
fcntl(sockfd, F_SETFL, flag);
```

(2)

```
flags = 1;
return ioctl(fd, FIOBIO, &flags);
```

多路复用I/O

- 应用程序中同时处理多路输入输出流，若采用阻塞模式，将得不到预期的目的；
- 若采用非阻塞模式，对多个输入进行轮询，但又太浪费CPU时间；
- 若设置多个进程，分别处理一条数据通路，将新产生进程间的同步与通信问题，使程序变得更加复杂；
- 比较好的方法是使用I/O多路复用。其基本思想是：
  - 先构造一张有关描述符的表，然后调用一个函数，它要到这些描述符中的一个已准备好进行I/O时才返回。
  - 返回时，它告诉进程那个描述符已准备好可以进行I/O。

select()/poll()实现多路复用

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int n, fd_set *readfds, fd_set *writefds, fd_set
*exceptfds, struct timeval *timeout);

#include <sys/poll.h>
int poll(struct pollfd *ufds, unsigned int nfds, int timeout);
```

EPOLL

- API

int epoll_create(int size);

int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

EPOLL_CTL_ADD：注册新的fd到epfd中；

EPOLL_CTL_MOD：修改已经注册的fd的监听事件；

EPOLL_CTL_DEL：从epfd中删除一个fd；

int epoll_wait(int epfd, struct epoll_event * events, int maxevents, int timeout);

异步非阻塞 I/O（AIO）

- API

aio_read

aio_error

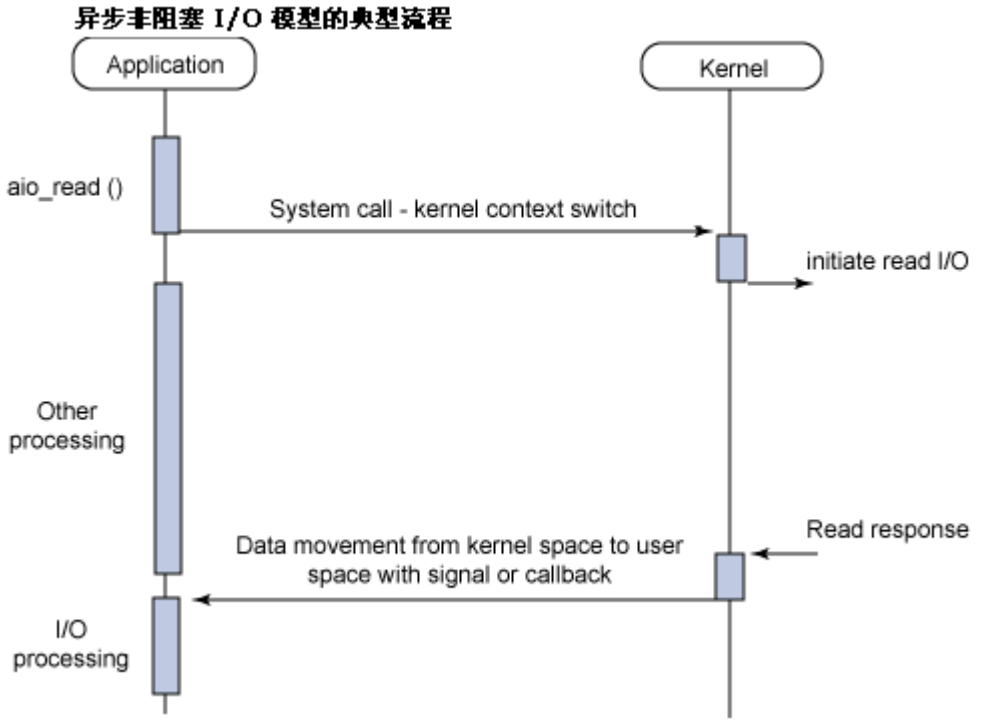
aio_return

aio_write

aio_suspend

aio_cancel

lio_listio



C10K问题

www.kegel.com/c10k.html

The C10K problem

[Help save the best Linux news source on the web -- subscribe to Linux Weekly News!]

It's time for web servers to handle ten thousand clients simultaneously, don't you think? After all, th

And computers are big, too. You can buy a 1000MHz machine with 2 gigabytes of RAM and an 1000Mbit/sec E 50Kbits/sec per client. It shouldn't take any more horsepower than that to take four kilobytes from the works out to $0.08 per client, by the way. Those $100/client licensing fees some operating systems char

In 1999 one of the busiest ftp sites, cdrom.com, actually handled 10000 clients simultaneously through who expect it to become increasingly popular with large business customers.

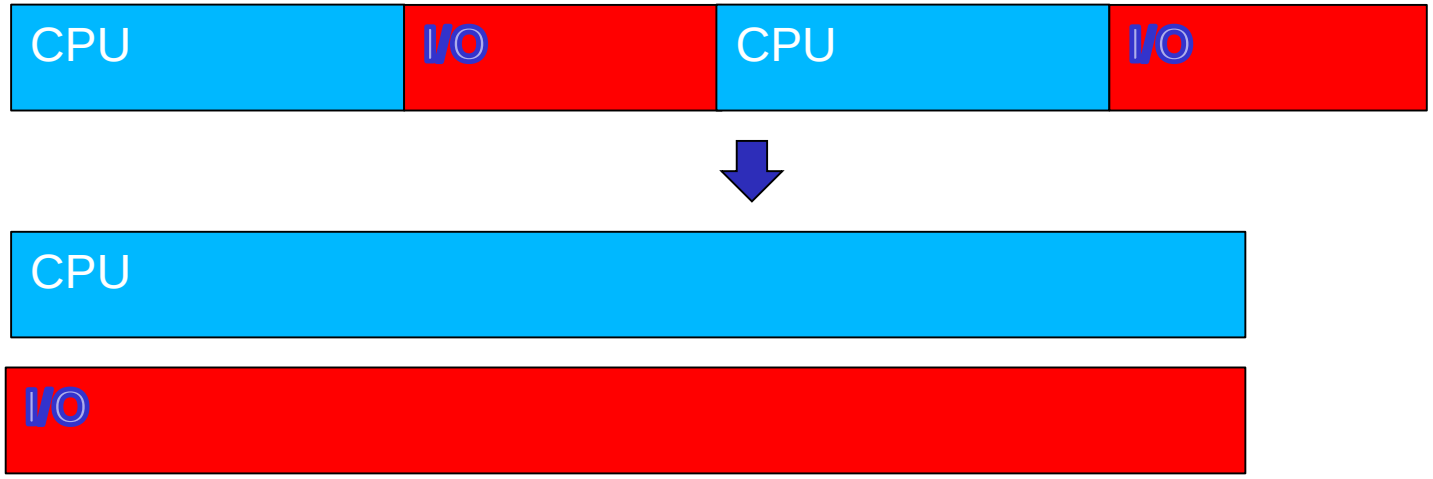
And the thin client model of computing appears to be coming back in style -- this time with the server

With that in mind, here are a few notes on how to configure operating systems and write code to support my personal area of interest, but Windows is also covered a bit.

Contents

Systemd readahead

- systemd-readahead-collect.service is a service that collects disk usage patterns at boot time. systemd-readahead-replay.service is a service that replays this access data collected at the subsequent boot.



系统调试与性能调优

top

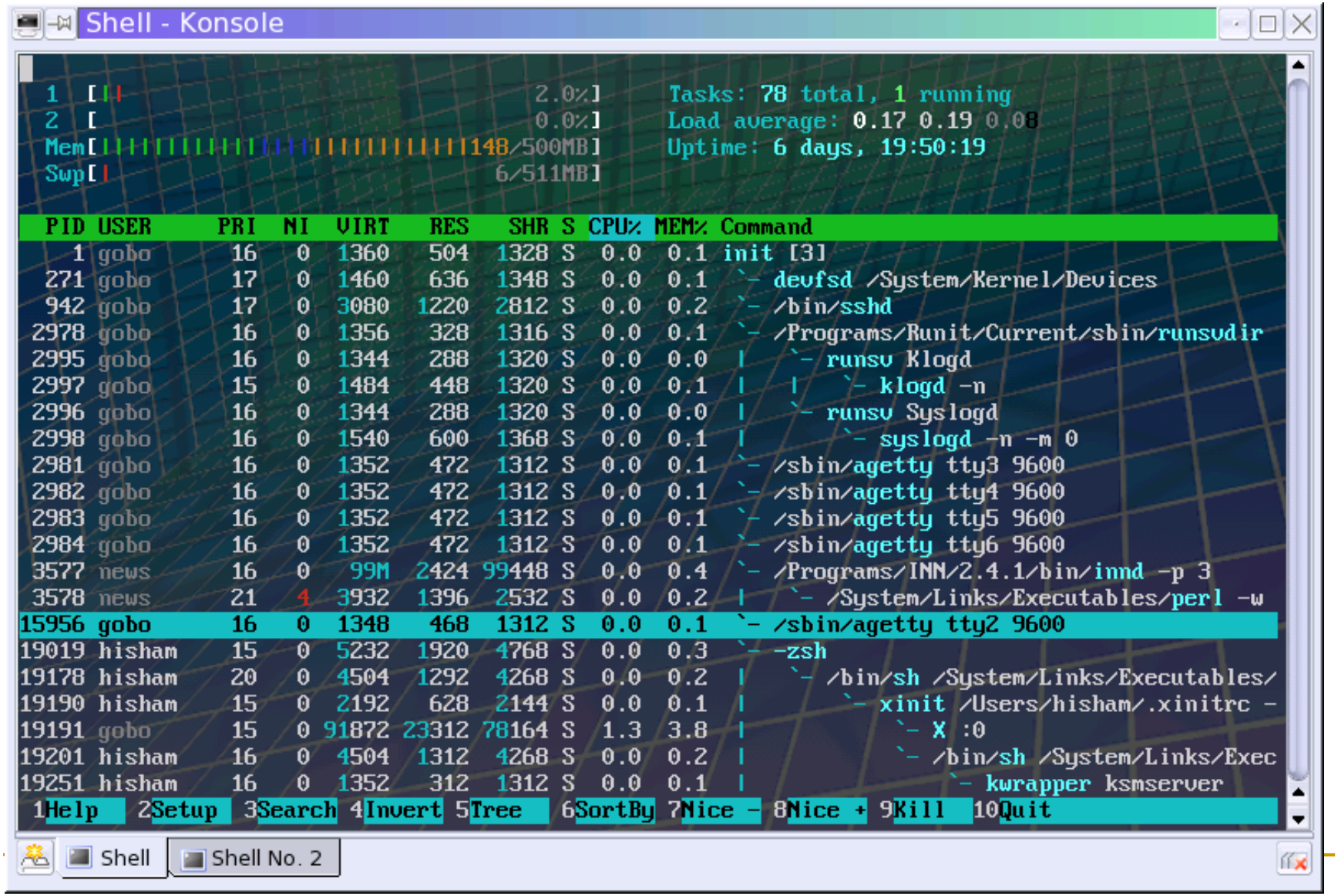
User 0%, System 0%, IOW 0%, IRQ 0%
User 2 + Nice 0 + Sys 3 + Idle 297 + IOW 1 + IRQ 0 + SIRQ 0 = 303

```
  PID CPU% S  #THR     VSS     RSS PCY UID      Name
 1017   1% R     1    964K    376K  fg root     top
  615   0% S    11  16076K   1796K  fg root     /usr/bin/gpsexe
  574   0% S    10  39756K   7576K  fg root     /system/bin/mediaserver
  555   0% S     1      0K      0K  fg root     jbd2/mmcblk0p2-
  584   0% S    10  14756K   2356K  fg root
/system/bin/synergy_service
    6   0% S     1      0K      0K  fg root     khelper
   13   0% S     1      0K      0K  fg root     suspend
   14   0% S     1      0K      0K  fg root     kworker/0:1
  198   0% S     1      0K      0K  fg root     sync_supers
  200   0% S     1      0K      0K  fg root     bdi-default
```

mpstat – show multi-core CPUs

```
barry@lianlab:~$ mpstat -P ALLLinux 2.6.32-39-server (lianlab)      04/12/2012
      _x86_64_     (16 CPU)01:54:06 PM  CPU    %usr   %nice    %sys %iowait
%irq   %soft  %steal  %guest   %idle01:54:06 PM  all    1.49    0.00    0.16
0.08    0.00    0.01    0.00    0.00   98.2701:54:06 PM    0    1.98    0.00    0.23
0.49    0.00    0.03    0.00    0.00   97.2701:54:06 PM    1    2.05    0.00    0.28
0.48    0.00    0.02    0.00    0.00   97.1701:54:06 PM    2    1.79    0.00    0.27
0.03    0.00    0.00    0.00    0.00   97.9101:54:06 PM    3    1.93    0.00    0.29
0.03    0.00    0.00    0.00    0.00   97.7401:54:06 PM    4    1.44    0.00    0.15
0.02    0.00    0.00    0.00    0.00   98.4001:54:06 PM    5    1.38    0.00    0.15
0.02    0.00    0.00    0.00    0.00   98.4501:54:06 PM    6    1.34    0.00    0.13
0.01    0.00    0.00    0.00    0.00   98.5201:54:06 PM    7    1.52    0.00    0.16
0.01    0.00    0.00    0.00    0.00   98.3001:54:06 PM    8    1.28    0.00    0.12
0.04    0.00    0.01    0.00    0.00   98.5501:54:06 PM    9    1.34    0.00    0.13
0.04    0.00    0.00    0.00    0.00   98.4901:54:06 PM   10    1.37    0.00    0.14
0.02    0.00    0.00    0.00    0.00   98.4701:54:06 PM   11    1.38    0.00    0.13
0.02    0.00    0.00    0.00    0.00   98.4601:54:06 PM   12    1.31    0.00    0.13
0.01    0.00    0.00    0.00    0.00   98.5501:54:06 PM   13    1.20    0.00    0.11
0.01    0.00    0.00    0.00    0.00   98.6701:54:06 PM   14    1.17    0.00    0.10
0.01    0.00    0.00    0.00    0.00   98.7201:54:06 PM   15    1.29    0.00    0.12
0.01    0.00    0.00    0.00    0.00   98.58
```

htop


Shell - Konsole
1 [|| 2.0%]
2 [0.0%]
Mem[||||||||||||||||||||||||148/500MB]
Swp[| 6/511MB]
Tasks: 78 total, 1 running
Load average: 0.17 0.19 0.08
Uptime: 6 days, 19:50:19
PID USER PRI NI VIRT RES SHR S CPU% MEM% Command
1 gobo 16 0 1360 504 1328 S 0.0 0.1 init [3]
271 gobo 17 0 1460 636 1348 S 0.0 0.1 `- devfsd /System/Kernel/Devices
942 gobo 17 0 3080 1220 2812 S 0.0 0.2 `- /bin/sshd
2978 gobo 16 0 1356 328 1316 S 0.0 0.1 `- /Programs/Runit/Current/sbin/runsvdir
2995 gobo 16 0 1344 288 1320 S 0.0 0.0 | `- runsv Klogd
2997 gobo 15 0 1484 448 1320 S 0.0 0.1 | | `- klogd -n
2996 gobo 16 0 1344 288 1320 S 0.0 0.0 | `- runsv Syslogd
2998 gobo 16 0 1540 600 1368 S 0.0 0.1 | `- syslogd -n -m 0
2981 gobo 16 0 1352 472 1312 S 0.0 0.1 `- /sbin/agetty tty3 9600
2982 gobo 16 0 1352 472 1312 S 0.0 0.1 `- /sbin/agetty tty4 9600
2983 gobo 16 0 1352 472 1312 S 0.0 0.1 `- /sbin/agetty tty5 9600
2984 gobo 16 0 1352 472 1312 S 0.0 0.1 `- /sbin/agetty tty6 9600
3577 news 16 0 99M 2424 99448 S 0.0 0.4 `- /Programs/INN/2.4.1/bin/innd -p 3
3578 news 21 4 3932 1396 2532 S 0.0 0.2 | `- /System/Links/Executables/perl -w
15956 gobo 16 0 1348 468 1312 S 0.0 0.1 `- /sbin/agetty tty2 9600
19019 hisham 15 0 5232 1920 4768 S 0.0 0.3 `- -zsh
19178 hisham 20 0 4504 1292 4268 S 0.0 0.2 | `- /bin/sh /System/Links/Executables/
19190 hisham 15 0 2192 628 2144 S 0.0 0.1 | `- xinit /Users/hisham/.xinitrc -
19191 gobo 15 0 91872 23312 78164 S 1.3 3.8 | `- X :0
19201 hisham 16 0 4504 1312 4268 S 0.0 0.2 | `- /bin/sh /System/Links/Exec
19251 hisham 16 0 1352 312 1312 S 0.0 0.1 | `- kwrapper ksmserver
1Help 2Setup 3Search 4Invert 5Tree 6SortBy 7Nice - 8Nice + 9Kill 10Quit
Shell
Shell No. 2

sar

- sar -u 2 5

Report CPU utilization for each 2 seconds. 5 lines are displayed.

- sar -I 14 -o int14.file 2 10

Report statistics on IRQ 14 for each 2 seconds. 10 lines are displayed. Data are stored in a file called int14.file.

- sar -B 10 3

Report paging statistics.

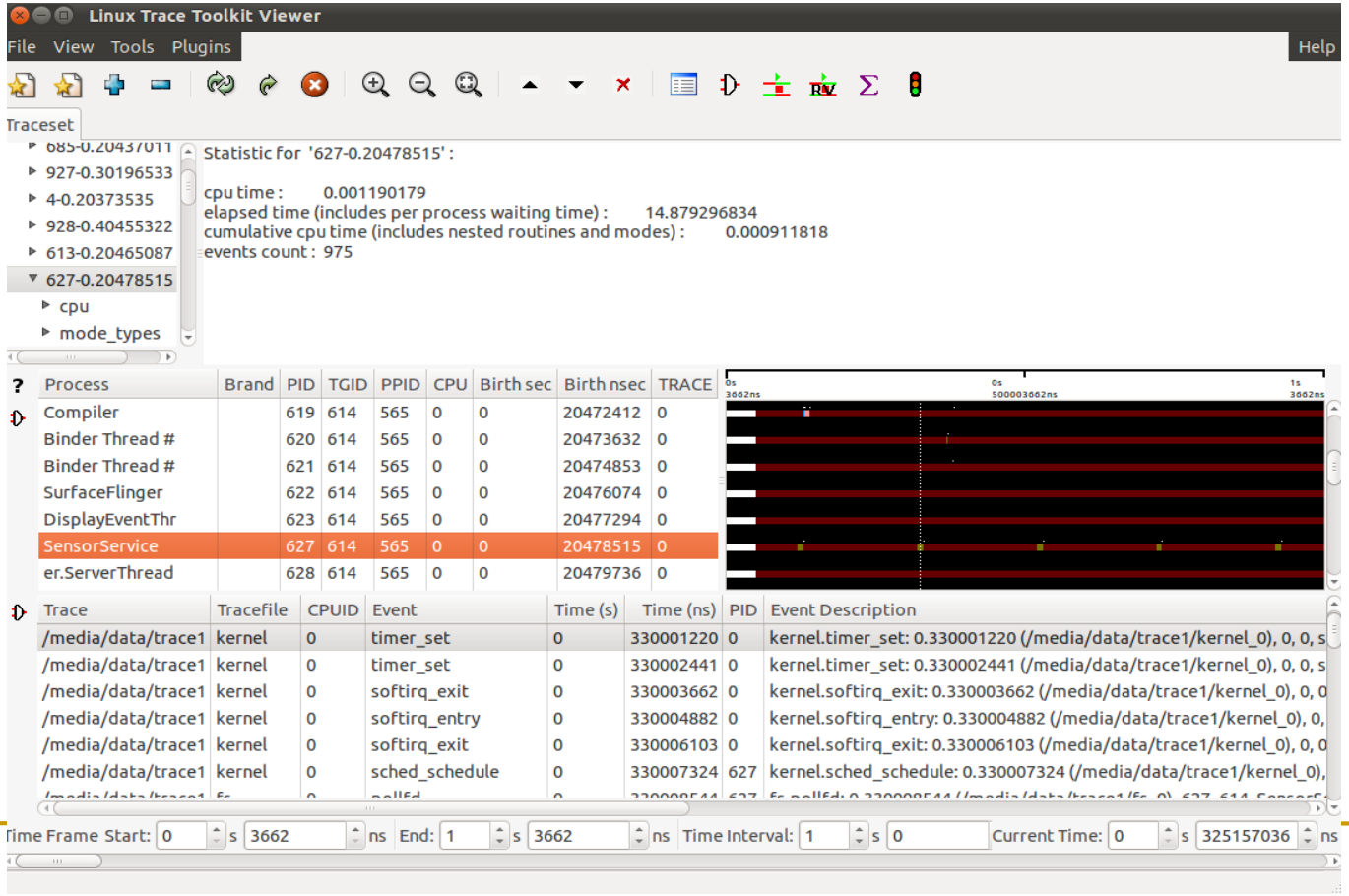
```
barry@barry-VirtualBox:~/develop/lddd3-kernel$ sar -B 1 100
Linux 3.8.0-44-generic (barry-VirtualBox)       11/30/2014      _i686_  (4 CPU)

10:45:54 PM  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
10:45:55 PM      0.00      0.00    700.00      0.00    740.00      0.00      0.00      0.00      0.00
10:45:56 PM      0.00      0.00    724.00      0.00    791.00      0.00      0.00      0.00      0.00
10:45:57 PM      0.00      0.00    718.00      0.00    792.00      0.00      0.00      0.00      0.00
10:45:58 PM      0.00     16.00    746.00      0.00    847.00      0.00      0.00      0.00      0.00
10:45:59 PM      0.00      0.00    725.00      0.00    790.00      0.00      0.00      0.00      0.00
10:46:00 PM      0.00      0.00    718.00      0.00    791.00      0.00      0.00      0.00      0.00
10:46:01 PM      0.00      0.00    718.00      0.00    787.00      0.00      0.00      0.00      0.00
10:46:02 PM      0.00      0.00    719.00      0.00    790.00      0.00      0.00      0.00      0.00
10:46:03 PM      0.00      4.00    719.00      0.00    840.00      0.00      0.00      0.00      0.00
10:46:04 PM      0.00     16.00    719.00      0.00    858.00      0.00      0.00      0.00      0.00
10:46:05 PM      0.00      0.00   3101.00      0.00   4230.00      0.00      0.00      0.00      0.00
```

vmstat - memory, processes, interrupts, paging and block I/O statistics

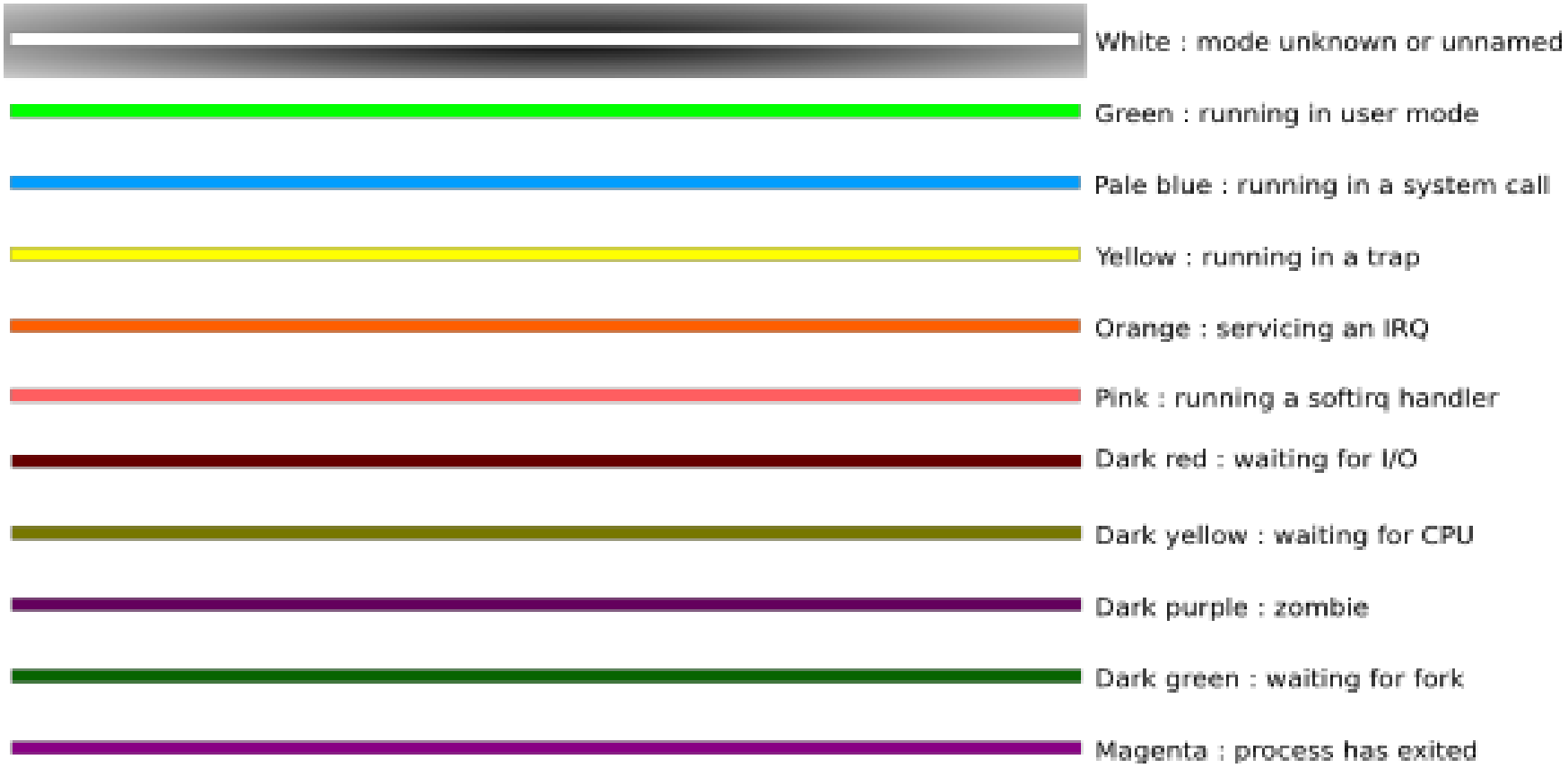
```
barry@barry-VirtualBox:~$ vmstat -n 1
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 2  0  47196 177100  51660 215592    1    3    60    22   64  107  6  1 92  1
 1  0  47196 177040  51660 215592    0    0     0     0  373  387 25  0 75  0
 1  0  47196 177048  51660 215592    0    0     0     0  566  283 25  1 75  0
 1  0  47196 177048  51660 215592    0    0     0     0  338  107 25  0 75  0
 1  0  47196 177048  51668 215584    0    0     0    16  581  242 25  0 75  0
 1  0  47196 177048  51668 215584    0    0     0     0  466  113 25  0 75  0
 1  0  47196 177048  51668 215592    0    0     0     0  420  125 25  0 75  0
 1  0  47196 177048  51668 215592    0    0     0     0  558  113 25  0 75  0
 1  0  47196 177048  51668 215592    0    0     0     0  513  117 25  0 75  0
 1  0  47196 177108  51668 215592    0    0     0     8  445  324 25  0 74  0
 1  0  47196 176488  51676 215584    0    0     0    16  492  202 25  0 75  0
 1  0  47196 176404  51676 215592    0    0     0     0  329  145 25  0 75  0
 1  0  47196 176404  51676 215592    0    0     0     0  452  117 25  0 75  0
 1  0  47196 176428  51676 215592    0    0     0     0  448  173 25  0 75  0
 1  0  47196 176428  51676 215592    0    0     0     0  490  215 25  0 75  0
 2  0  47196 176828  51676 215592    0    0     0     0  495  304 25  1 74  0
 1  0  47196 176552  51684 215592    0    0     0    16  632  462 26  0 74  0
 1  0  47196 176552  51684 215592    0    0     0     0  593  202 25  0 75  0
 1  0  47196 176552  51684 215592    0    0     0     0  594  353 25  1 74  0
 1  0  47196 176528  51684 215592    0    0     0    12  545  213 25  0 74  0
 1  0  47196 176552  51684 215592    0    0     0     0  448  113 25  0 75  0
```

LTTng - usage

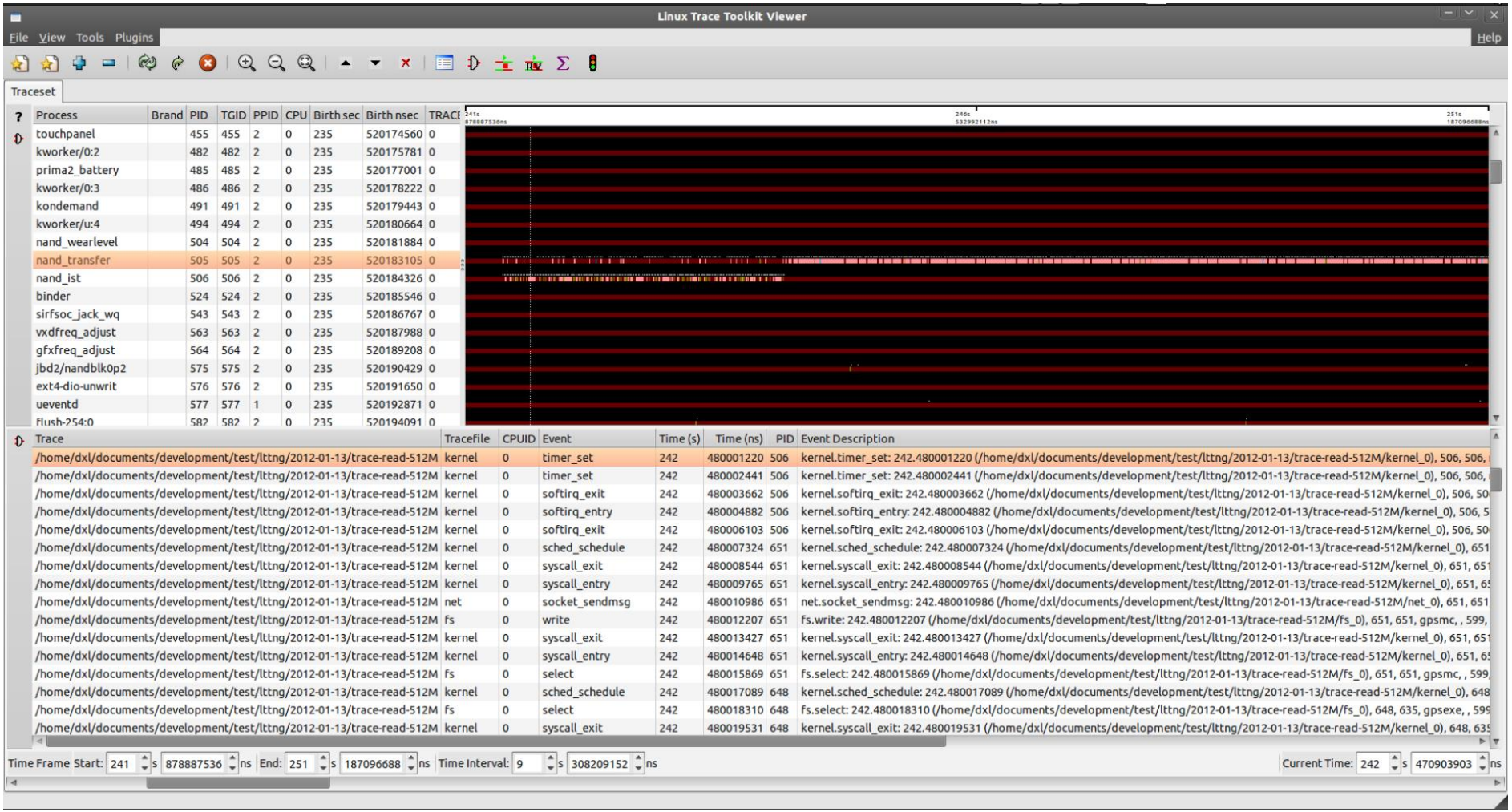
- # lttctl -C -w /mnt/obb/trace1 trace1
- # lttctl -D trace1
- lttv -m batchAnalysis -s -m textDump -s -t trace
- lttv-gui



lttv-gui - color



LTTng –NAND performance

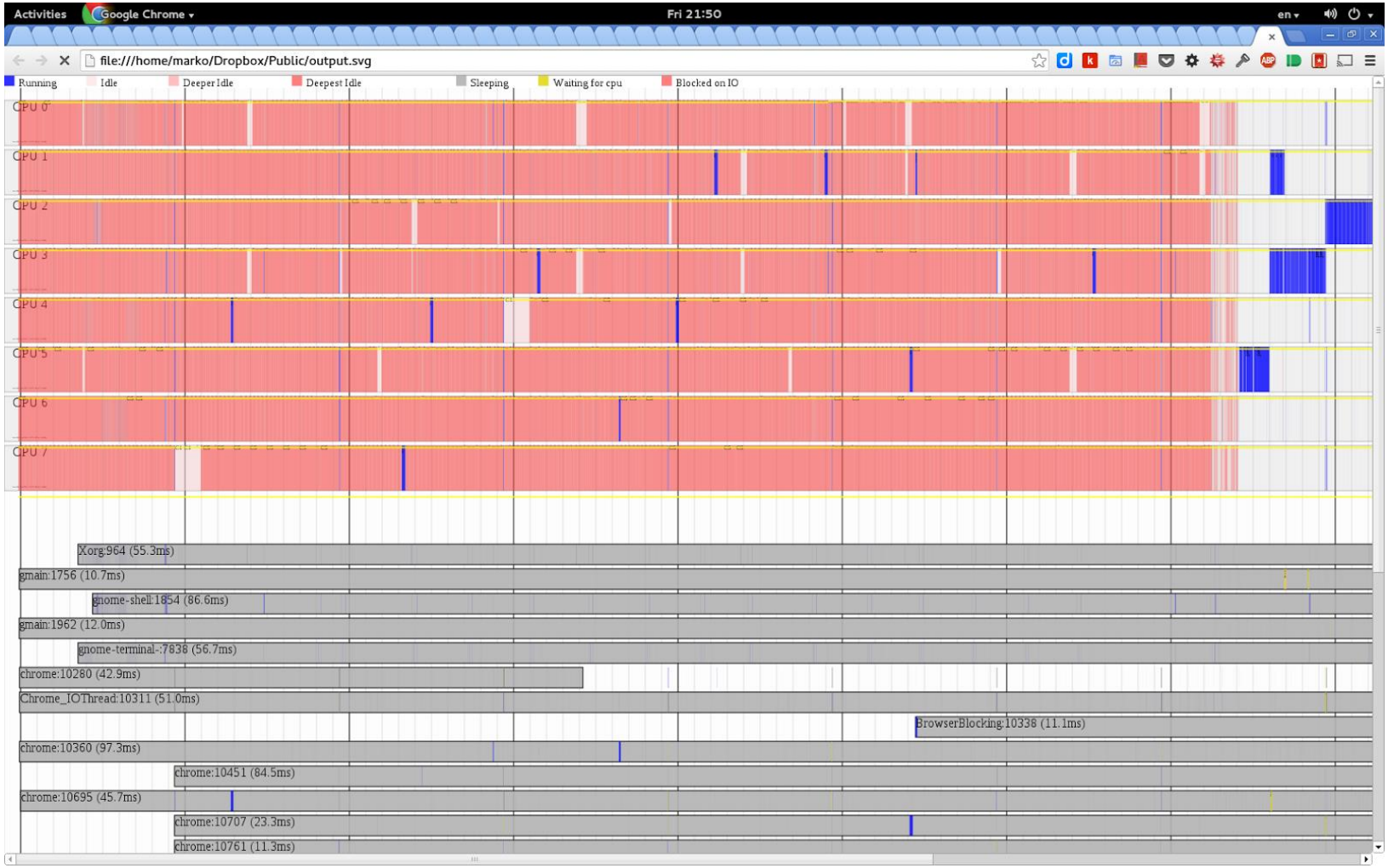


Oprofile -usage

- # opcontrol --init
- # opcontrol --start
- # opcontrol --dump
- # opreport -la

```
ounted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 150000
amples  cum. samples  %        cum. %     app name                 symbol name
934     5934          53.7257  53.7257    libskia.so               /system/lib/libskia.so
600     7534          14.4862  68.2119    libdvm.so                /system/lib/libdvm.so
070     8604           9.6876  77.8995    libc.so                  /system/lib/libc.so
17      8921           2.8701  80.7696    libcutils.so             /system/lib/libcutils.so
65      9086           1.4939  82.2635    vmlinux                  schedule
26      9212           1.1408  83.4043    vmlinux                  __memzero
02      9314           0.9235  84.3278    libutils.so              /system/lib/libutils.so
00      9414           0.9054  85.2331    vmlinux                  sirfsoc_enter_idle
5       9499           0.7696  86.0027    vmlinux                  l2x0_flush_all
9       9578           0.7153  86.7180    libGLESv1_CM_POWERVR_SGX531_110.so.1.1.16.4117 /system/lib/
6       9654           0.6881  87.4061    vmlinux                  __do_softirq
9       9723           0.6247  88.0308    libandroid_runtime.so    /system/lib/libandroid_runtime.so
4       9787           0.5794  88.6102    pvrsrvkm                 /pvrsrvkm
8       9845           0.5251  89.1354    libui.so                 /system/lib/libui.so
9       9884           0.3531  89.4885    dalvik-jit-code-cache (deleted) /dev/ashmem/dalvik-jit-code
```

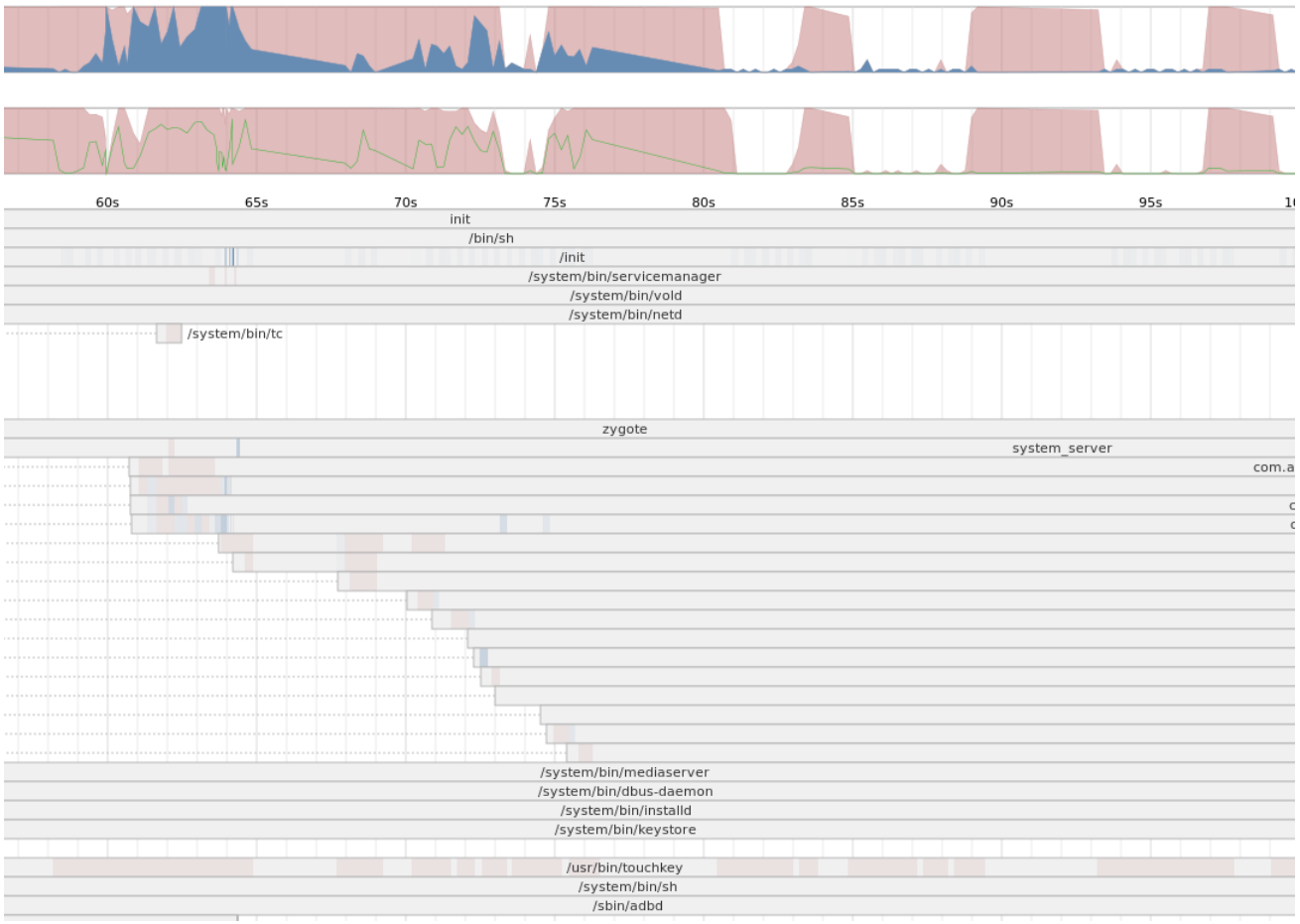
perf sched timechart



gprof

- cc -g -c myprog.c utils.c -pg
- cc -o myprog myprog.o utils.o –pg
- gprof *options* [*executable-file* [*profile-data-files*...]] [> *outfile*]

Bootchart


60s
65s
70s
75s
80s
85s
90s
95s
init
/bin/sh
/init
/system/bin/servicemanager
/system/bin/vold
/system/bin/netd
/system/bin/tc
zygote
system_server
com.a
/system/bin/mediaserver
/system/bin/dbus-daemon
/system/bin/installd
/system/bin/keystore
/usr/bin/touchkey
/system/bin/sh
/sbin/adbd

iostat - storage input and output statistics

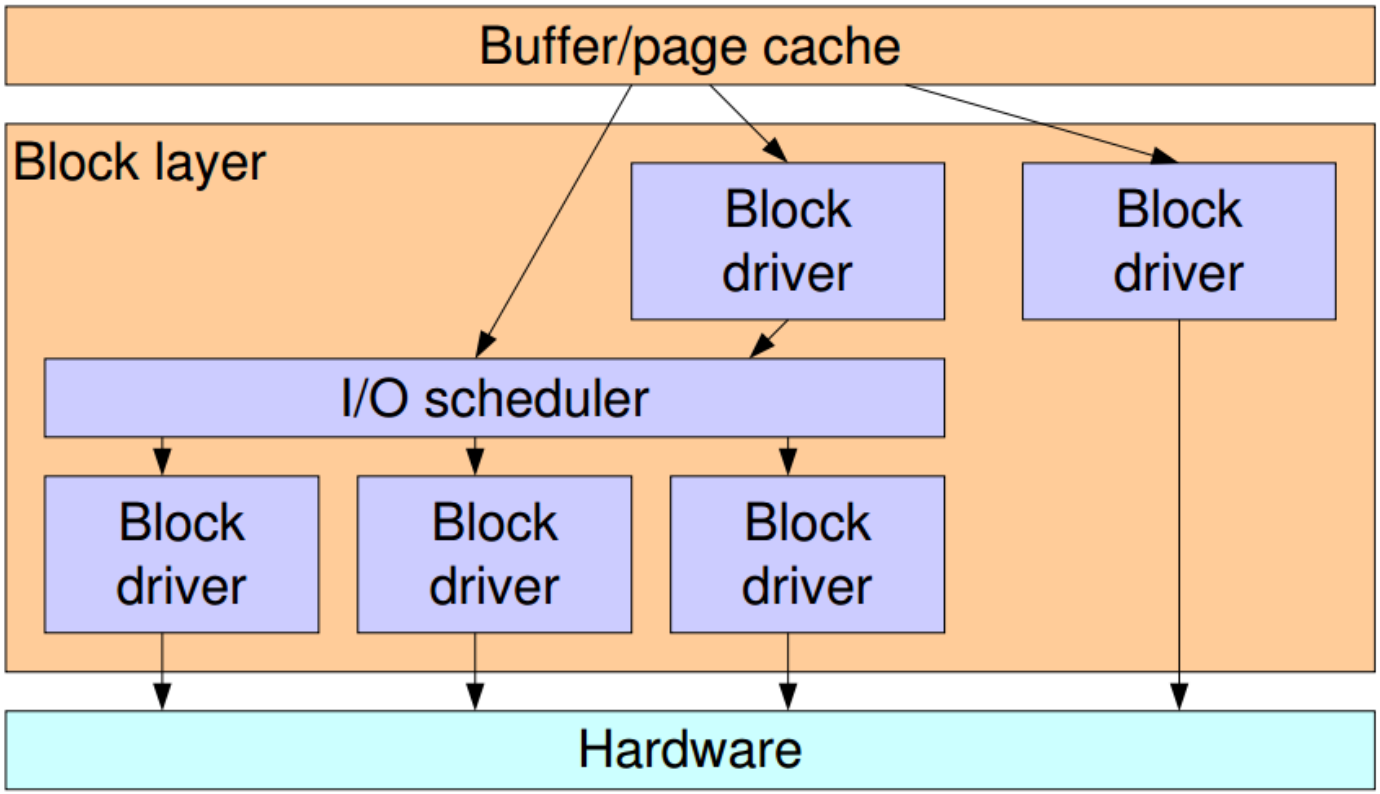
```
barry@barry-VirtualBox:~$ iostat -txz
Linux 3.8.0-44-generic (barry-VirtualBox)      11/26/2014      _i686_  (4 CPU)

11/26/2014 10:58:21 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          4.60    0.75    0.62    1.52    0.00   92.51

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
scd0              0.00     0.00    0.01    0.00     0.02     0.00     6.36     0.00   40.91   40.91    0.00  40.73   0.02
sda               0.32     2.70   18.18    1.48   234.56    75.35    31.54     0.28   14.18   12.58   33.82   3.05   6.00
sdb               0.27     0.00    0.08    0.00     0.31     0.00     7.99     0.00   27.04   27.04    0.00  27.00   0.21
```

iotop

```
Total DISK READ:       52.46 M/s | Total DISK WRITE:         0.00 B/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 5139 be/4 root       51.91 M/s    0.00 B/s  0.00 % 96.07 % cat /dev/sdb1
 4053 be/4 barry     558.89 K/s    0.00 B/s  0.00 %  7.88 % sftp-server
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
    5 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0H]
 2054 be/4 barry       0.00 B/s    0.00 B/s  0.00 %  0.00 % VBoxClien~aganddro
    7 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/u:0H]
    8 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]
    9 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_bh]
   10 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_sched]
   11 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [watchdog/0]
   12 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [watchdog/1]
   13 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/1]
   14 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/1]
 2063 be/4 barry       0.00 B/s    0.00 B/s  0.00 %  0.00 % gnome-ses~-fallbac
   16 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/1:0H]
   17 rt/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [watchdog/2]
   18 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/2]
```

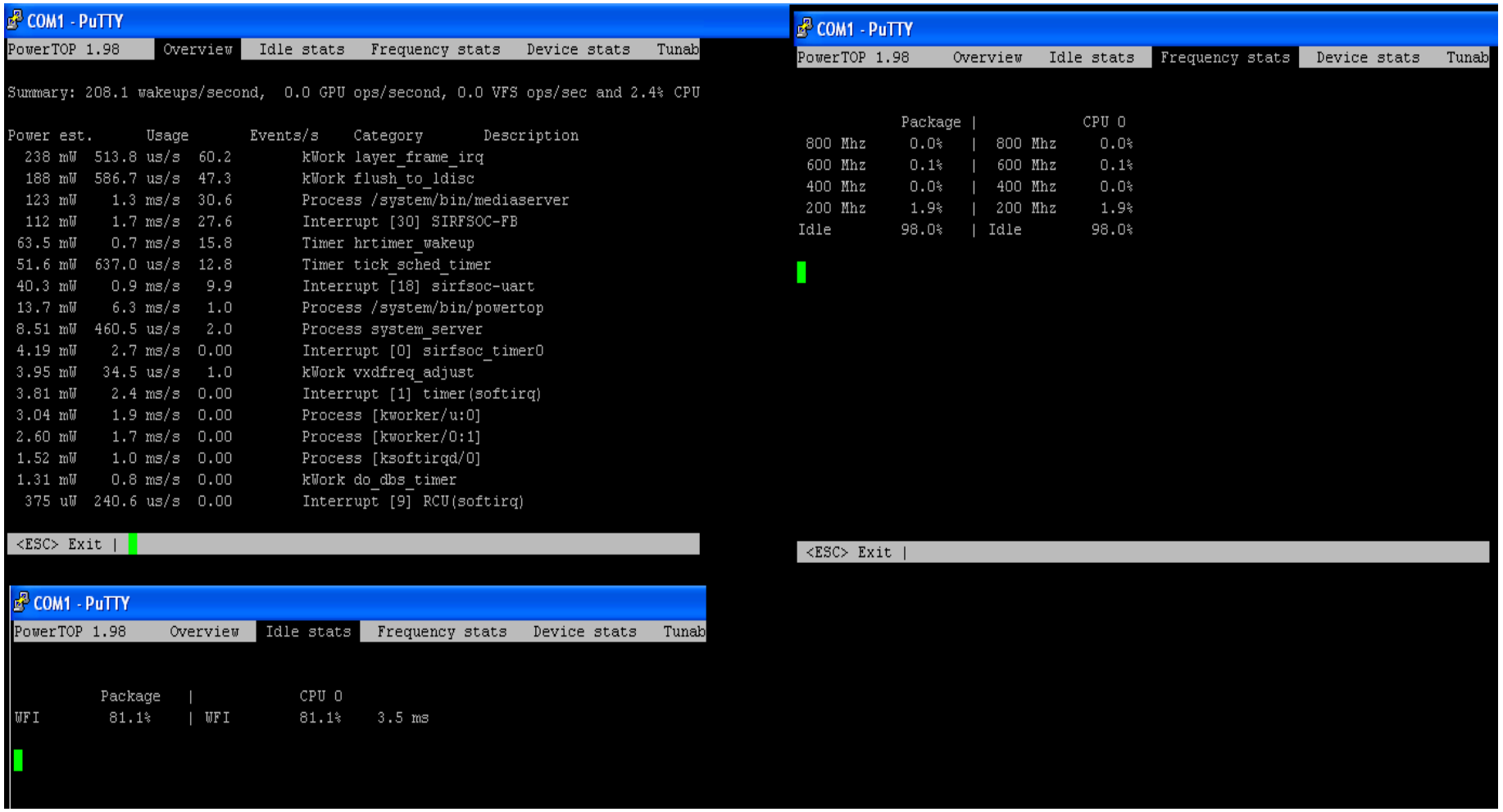
Linux块设备与I/O调度

The current scheduler for a device can be get and set in /sys/block /queue/scheduler



DVFS- powertop

- /system/bin/powertop



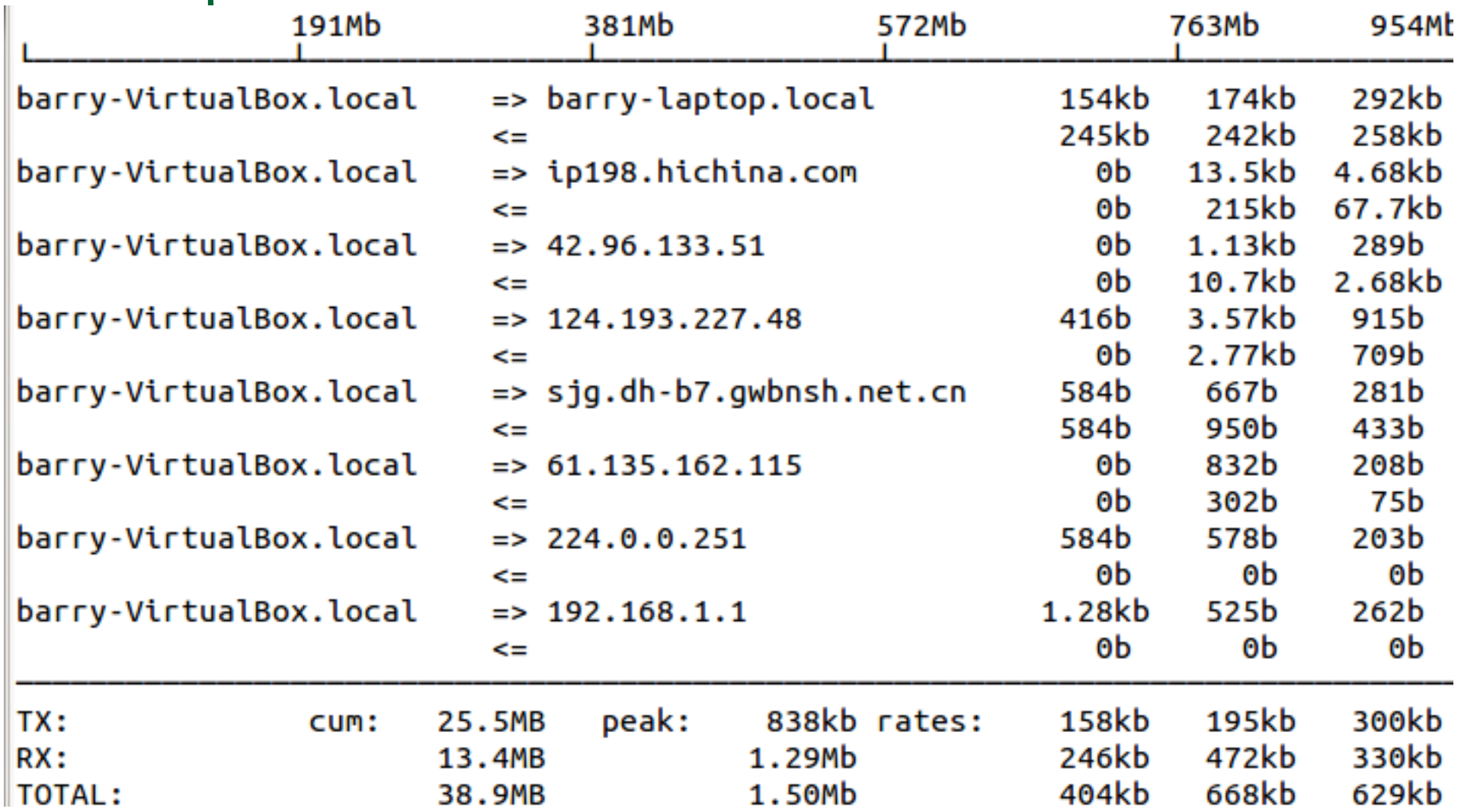
DVFS – cpufreq-bench

/usr/sbin/cpufreq-bench -l 50000 -s 100000 -x 50000 -y 100000 -g ondemand -r 5 -n 5 –v

Ondemand:
Round 1 - 39.74%
Round 2 - 36.35%
Round 3 - 47.91%
Round 4 - 54.22%
Round 5 - 58.64%

Interactive:
Round 1 - 72.95%
Round 2 - 87.20%
Round 3 - 91.21%
Round 4 - 94.10%
Round 5 - 94.93%

iftop


191Mb
381Mb
572Mb
763Mb
954Mb
barry-VirtualBox.local => barry-laptop.local 154kb 174kb 292kb
<= 245kb 242kb 258kb
barry-VirtualBox.local => ip198.hichina.com 0b 13.5kb 4.68kb
<= 0b 215kb 67.7kb
barry-VirtualBox.local => 42.96.133.51 0b 1.13kb 289b
<= 0b 10.7kb 2.68kb
barry-VirtualBox.local => 124.193.227.48 416b 3.57kb 915b
<= 0b 2.77kb 709b
barry-VirtualBox.local => sjg.dh-b7.gwbnsh.net.cn 584b 667b 281b
<= 584b 950b 433b
barry-VirtualBox.local => 61.135.162.115 0b 832b 208b
<= 0b 302b 75b
barry-VirtualBox.local => 224.0.0.251 584b 578b 203b
<= 0b 0b 0b
barry-VirtualBox.local => 192.168.1.1 1.28kb 525b 262b
<= 0b 0b 0b
TX: cum: 25.5MB peak: 838kb rates: 158kb 195kb 300kb
RX: 13.4MB 1.29Mb 246kb 472kb 330kb
TOTAL: 38.9MB 1.50Mb 404kb 668kb 629kb

nethogs

```
NetHogs version 0.8.0

  PID USER     PROGRAM                          DEV        SENT      RECEIVED
4052  barry    sshd: barry@notty                eth0      47.980     32.955 KB/sec
4501  barry    /usr/lib/firefox/firefox         eth0       0.176      0.182 KB/sec
?     root     ..6:56772-61.135.186.84:80                  0.000      0.000 KB/sec
?     root     ..:51601-61.135.185.105:80                  0.000      0.000 KB/sec
?     root     unknown TCP                                 0.000      0.000 KB/sec

  TOTAL                                                    48.156     33.138 KB/sec
```

Benchmark tools

- lmbench

Microbenchmark for operating system functions

- Bonnie++/Iozone

File system benchmark

- Netperf

Network performance benchmark

让我们一起讨论！