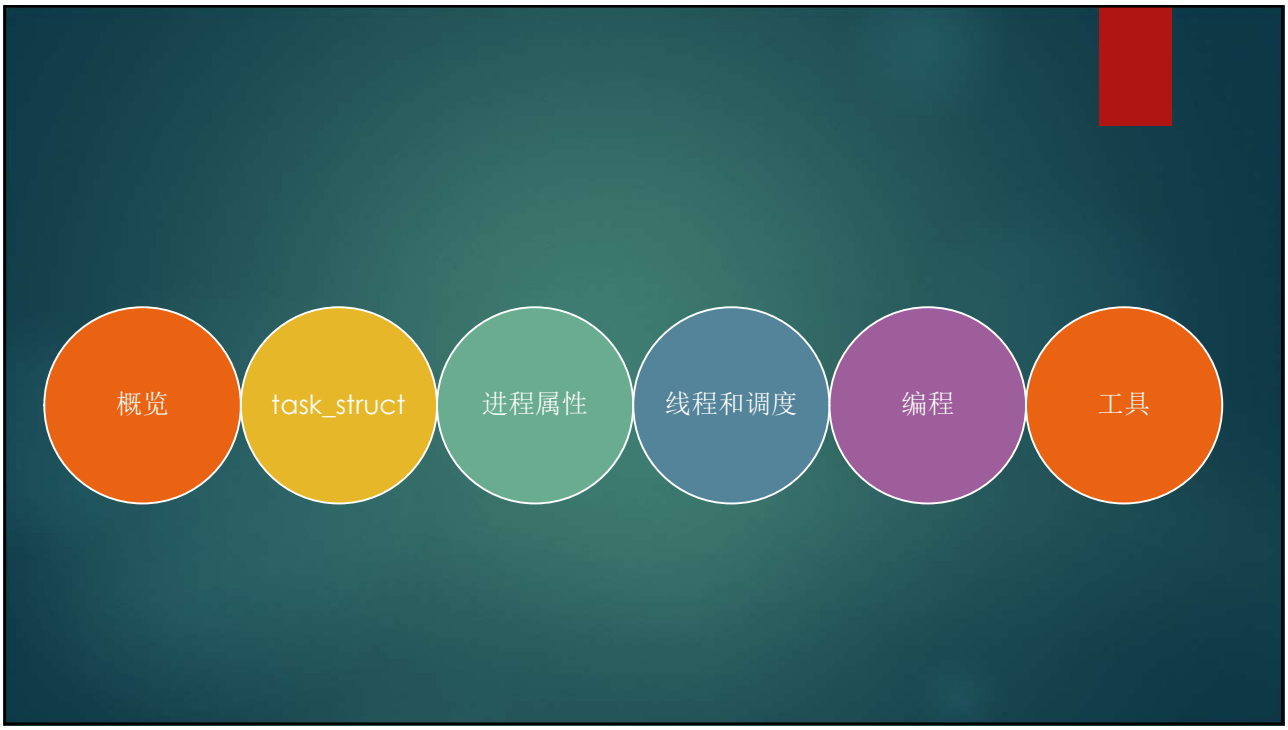
LINUX内核开发与调试 -- 任务管理

张银奎和程煜明

2016/11/6（2018/4更新）

1



2

任务

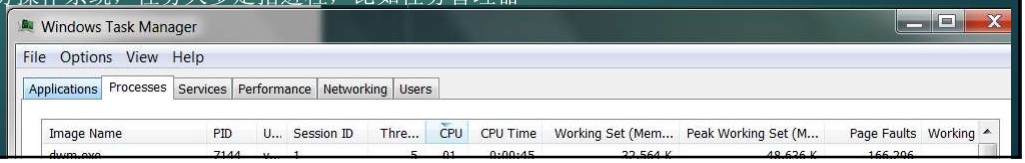
- 相对的概念
- 与主体密切相关
  - 党的任务…
  - 今天的任务…
- CPU
  - 多任务支持，任务的概念大体相当于线程
  - “A task is a unit of work that a processor can dispatch, execute, and suspend.”
- 操作系统
  - 多任务操作系统，任务大多是指进程，比如任务管理器

CHAPTER 7
TASK MANAGEMENT

This chapter describes the IA-32 architecture's task management facilities. These facilities are only available when the processor is running in protected mode.

This chapter focuses on 32-bit tasks and the 32-bit TSS structure. For information on 16-bit tasks and the 16-bit TSS structure, see Section 7.6, "16-Bit Task-State Segment (TSS)." For information specific to task management in 64-bit mode, see Section 7.7, "Task Management in 64-bit Mode."

7.1 TASK MANAGEMENT OVERVIEW



3

进程与线程

进程

- 空间
- 线程的住所
- OS组织和管理系统资源的重要单位

线程

- 时间
- 活动的生命
- CPU的调度单位
- 任务状态段(Task State Segment)

4

Windows下的数据结构

进程

- KPROCESS
- EPROCESS
- PEB

线程

- KTHREAD
- ETHREAD
- TEB

5

Linux下的数据结构

- **task_struct结构体一体二用**
  - 把线程看作是共享一个地址空间的'进程'
  - 创建新线程时，创建一个新的task_struct，大多数字段与其它线程相同，主要由do_fork完成
  - 因此LINUX线程又被称为轻进程（LWP），传统UNIX进程称为重进程
  - 历史原因
  - 对于单线程的进程来说经济一些

6

格物

```
gedu      2806  2552  2806 57    7 20:47 pts/4    00:10:07 ./gemalloc
gedu      2806  2552  2811  0    7 20:48 pts/4    00:00:00 ./gemalloc
gedu      2806  2552  2812  0    7 20:48 pts/4    00:00:00 ./gemalloc
gedu      2806  2552  2813  0    7 20:48 pts/4    00:00:00 ./gemalloc
gedu      2806  2552  2822  4    7 20:49 pts/4    00:00:45 ./gemalloc
gedu      2806  2552  2823  4    7 20:49 pts/4    00:00:44 ./gemalloc
gedu      2806  2552  2824  4    7 20:49 pts/4    00:00:43 ./gemalloc
root      2825     2  2825  0    1 21:02 ?        00:00:00 [kworker/u2:1]
gedu      2843  2541  2843  0    1 21:03 pts/7    00:00:00 bash
gedu      2876  2843  2876  0    1 21:05 pts/7    00:00:00 ps -eLf
```

```
(gdb) monitor ps
80 sleeping system daemon (state M) processes suppressed,
use 'ps A' to see all.
Task Addr               Pid   Parent [*] cpu State Thread             Command
0xffff88003588e580     2806     2552  1    0   R  0xffff88003588efc0 *gemalloc

0xffff88003588e580     2806     2552  1    0   R  0xffff88003588efc0 *gemalloc
0xffff88003a4e4880     2811     2552  0    0   S  0xffff88003a4e52c0  gmain
0xffff88003770ba00     2812     2552  0    0   S  0xffff88003770c440  gdbus
0xffff8800393c8e80     2813     2552  0    0   S  0xffff8800393c98c0  dconf worker
0xffff88003a4a0000     2822     2552  0    0   R  0xffff88003a4a0a40  gemalloc
0xffff88003a4a2b80     2823     2552  0    0   R  0xffff88003a4a35c0  gemalloc
0xffff88003a240e80     2824     2552  0    0   R  0xffff88003a2418c0  gemalloc
```

- 在gemalloc进程中创建10个子线程

续

```
0xffff88003588e580     2806     2552  0    0   R  0xffff88003588efc0  gemalloc
0xffff88003a4e4880     2811     2552  0    0   S  0xffff88003a4e52c0  gmain
0xffff88003770ba00     2812     2552  0    0   S  0xffff88003770c440  gdbus
0xffff8800393c8e80     2813     2552  0    0   S  0xffff8800393c98c0  dconf worker
0xffff88003a4a0000     2822     2552  0    0   R  0xffff88003a4a0a40  gemalloc
0xffff88003a4a2b80     2823     2552  0    0   R  0xffff88003a4a35c0  gemalloc
0xffff88003a240e80     2824     2552  1    0   R  0xffff88003a2418c0 *gemalloc
0xffff880037fc4880     2843     2541  0    0   S  0xffff880037fc52c0  bash
(gdb) p ((struct task_struct *)0xffff88003a240e80)->mm
$12 = (struct mm_struct *) 0xffff880037f18400
(gdb) p ((struct task_struct *)0xffff88003a4a0000)->mm
$13 = (struct mm_struct *) 0xffff880037f18400
(gdb) p ((struct task_struct *)0xffff88003a4a0000)->pid
$14 = 2822
(gdb) p ((struct task_struct *)0xffff88003a240e80)->pid
$15 = 2824
(gdb) p ((struct task_struct *)0xffff88003a4e4880)->mm
$16 = (struct mm_struct *) 0xffff880037f18400
(gdb) p ((struct task_struct *)0xffff88003770ba00)->mm
$17 = (struct mm_struct *) 0xffff880037f18400
```

相同的mm意味着相同地址空间，它们是属于同一个进程

重要头文件

- 用于任务管理的重要头文件:
  - **include/linux/sched.h – declarations for most task data structures**
  - **include/linux/threads.h – some configuration constants (unrelated to threads)**
  - **include/linux/times.h – time structures**
  - **include/linux/time.h – time declarations**
  - **include/linux/timex.h – wall clock time declarations**

9

sched.h

- struct signal_struct
- struct sched_info
- struct task_struct

Lister - [D:\bench\linux-4.4.14\include\linux\sched.h]

File Edit Options Help

```
};

struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        unsigned int wakee_flips;
        unsigned long wakee_flip_decay_ts;
        struct task_struct *last_wakee;

        int wake_cpu;
#endif
        int on_rq;

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
#ifdef CONFIG_CGROUP_SCHED
        struct task_group *sched_task_group;
#endif
        struct sched_dl_entity dl;

#ifdef CONFIG_PREEMPT_NOTIFIERS
        /* list of struct preempt_notifier: */
        struct hlist_head preempt_notifiers;
```

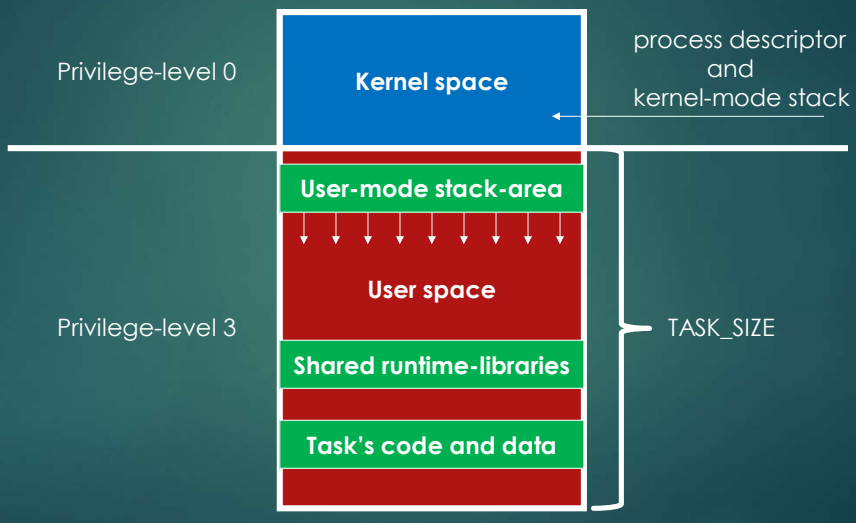
10

源代码

- The source code for process and thread management is in the kernel directory:
  - sched.c – task scheduling routines （3.11之前，后来拆分到多个文件）
  - signal.c – signal handling routines
  - fork.c – process/thread creation routines
  - exit.c – process exit routines
  - time.c – time management routines
  - timer.c – timer management routines
- The source code for the program initiation routines is in fs/exec.c.

11

进程空间布局



12

TASK_SIZE

```
#ifndef TASK_SIZE_OF
#define TASK_SIZE_OF(tsk)      TASK_SIZE
#endif
```

- 硬件架构相关
- X86 32位时 TASK_SIZE (same as PAGE_OFFSET, usually 0xc0000000)

```
#ifdef CONFIG_X86_32
/*
 * User space process size: 3GB (default).
 */
#define TASK_SIZE                PAGE_OFFSET
#define TASK_SIZE_MAX            TASK_SIZE
#define STACK_TOP                TASK_SIZE
#define STACK_TOP_MAX            STACK_TOP
```


PAGE_OFFSET(32位)

#define PAGE_OFFSET ((unsigned long)__PAGE_OFFSET)

Linux/arch/x86/include/asm/page_32_types.h

```
 1 #ifndef _ASM_X86_PAGE_32_DEFS_H
 2 #define _ASM_X86_PAGE_32_DEFS_H
 3
 4 #include <linux/const.h>
 5
 6 /*
 7  * This handles the memory map.
 8  *
 9  * A __PAGE_OFFSET of 0xC0000000 means that the kernel has
10  * a virtual address space of one gigabyte, which limits the
11  * amount of physical memory you can use to about 950MB.
12  *
13  * If you want more physical memory than this then see the CONFIG_HIGHMEM4G
14  * and CONFIG_HIGHMEM64G options in the kernel configuration.
15  */
16 #define __PAGE_OFFSET           _AC(CONFIG_PAGE_OFFSET, UL)
17
18 #define __START_KERNEL_map      __PAGE_OFFSET
```

X86-64

```
/*
 * User space process size. 47bits minus one guard page.  The guard
 * page is necessary on Intel CPUs: if a SYSCALL instruction is at
 * the highest possible canonical userspace address, then that
 * syscall will enter the kernel with a non-canonical return
 * address, and SYSRET will explode dangerously.  We avoid this
 * particular problem by preventing anything from being mapped
 * at the maximum canonical address.
 */
#define TASK_SIZE_MAX    ((1UL << 47) - PAGE_SIZE)

/* This decides where the kernel will search for a free chunk of vm
 * space during mmap's.
 */
#define IA32_PAGE_OFFSET        ((current->personality & ADDR_LIMIT_3GB) ? \
                                        0xc0000000 : 0xFFFFe000)

#define TASK_SIZE               (test_thread_flag(TIF_ADDR32) ? \
                                        IA32_PAGE_OFFSET : TASK_SIZE_MAX)
#define TASK_SIZE_OF(child)     ((test_tsk_thread_flag(child, TIF_ADDR32)) ? \
                                        IA32_PAGE_OFFSET : TASK_SIZE_MAX)
```

15

概览 task_struct 进程属性 线程和调度 编程 工具

16

进程描述符

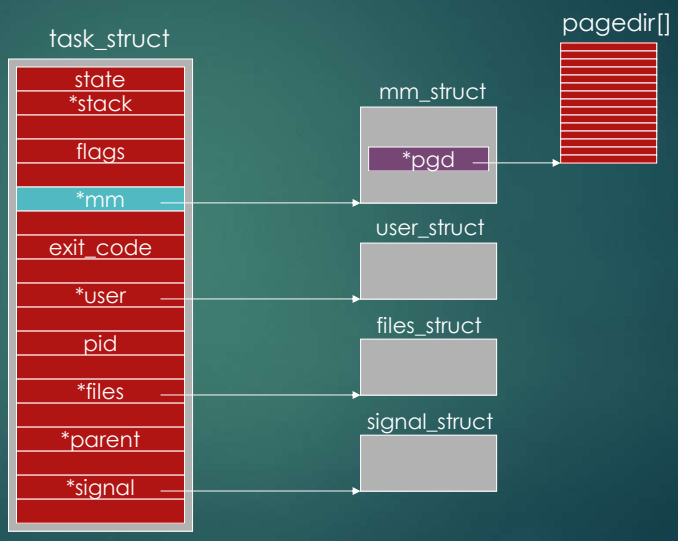
- Process – dynamic, program in motion
  - Kernel data structures to maintain "state"
  - Descriptor, PCB (control block), task_struct
  - Larger than you think! (about 1K)
  - Complex struct with pointers to others
- Type of info in task_struct
  - state, id, priorities, locks, files, signals, memory maps, locks, queues, list pointers, …
- Some details
  - Address of first few fields hardcoded in asm
  - Careful attention to cache line layout

17

进程描述符

- 庞大的数据结构
- 数百个字段
- 很多指针，指向其它内核数据结构，特别是线程共享的数据结构

(gdb) p sizeof(*task)
$18 = 3316



18

重要字段

- The *task_struct* is used to represent a task.
- The *task_struct* has several sub-structures that it references:
  - *tty_struct* – TTY associated with the process
  - *fs_struct* – current and root directories associated with the process
  - *files_struct* – file descriptors for the process
  - *mm_struct* – memory areas for the process
  - *signal_struct* – signal structures associated with the process
  - *user_struct* – per-user information (for example, number of current processes)

19

Task cat

$22 = {**state = 1**, **stack = 0xeeef0000**, usage = {counter = 3},
flags = 1077960704, ptrace = 0, wake_entry = {next = 0x0}, on_cpu = 0,
last_wakee = 0xeeca8000, wakee_flips = 304,
wakee_flip_decay_ts = 4294957249, on_rq = 0, prio = 120, static_prio = 120,
normal_prio = 120, rt_priority = 0, sched_class = 0xc1691820, se = {load = {
weight = 1024, inv_weight = 4194304}, run_node = {
__rb_parent_color = 3815106633, rb_right = 0xe365db48, rb_left = 0x0},
group_node = {next = 0xeec5db54, prev = 0xeec5db54}, on_rq = 0,
exec_start = 8770887600695, sum_exec_runtime = 1097174,
vruntime = 550813275, prev_sum_exec_runtime = 1090443, nr_migrations = 0,
statistics = {wait_start = 0, wait_max = 14982522, wait_count = 5,
wait_sum = 15759625, iowait_count = 0, iowait_sum = 0,
sleep_start = 8770887600695, sleep_max = 0, sum_sleep_runtime = 0,
block_start = 0, block_max = 0, exec_max = 405184, slice_max = 0,
nr_migrations_cold = 0, nr_failed_migrations_affine = 0,
nr_failed_migrations_running = 0, nr_failed_migrations_hot = 0,
nr_forced_migrations = 0, nr_wakeups = 0, nr_wakeups_sync = 0,
nr_wakeups_migrate = 0, nr_wakeups_local = 0, nr_wakeups_remote = 0,
nr_wakeups_affine = 0, nr_wakeups_affine_attempts = 0,
nr_wakeups_passive = 0, nr_wakeups_idle = 0}, parent = 0xeee74200,
cfs_rq = 0xe6755800, my_q = 0x0, avg = {runnable_avg_sum = 14447,
runnable_avg_period = 14447, last_runnable_update = 8770887600695,
decay_count = 8364571, load_avg_contrib = 1023}}, rt = {run_list = {

20

Pg. 2

next = 0xeec5dc88, prev = 0xeec5dc88}, timeout = 0, watchdog_stamp = 0,
time_slice = 25, back = 0x0, parent = 0x0, rt_rq = 0xef9e35d8,
my_q = 0x0}, sched_task_group = 0xeee74600, preempt_notifiers = {
first = 0x0}, fpu_counter = 0 '\000', btrace_seq = 0, policy = 0,
nr_cpus_allowed = 1, cpus_allowed = {bits = {1}}, sched_info = {pcount = 5,
run_delay = 15759625, last_arrival = 8770887593964, last_queued = 0},
tasks = {next = 0xee9869e4, prev = 0xdda29be4}, pushable_tasks = {
prio = 140, prio_list = {next = 0xeec5dcf0, prev = 0xeec5dcf0},
node_list = {next = 0xeec5dcf8, prev = 0xeec5dcf8}}, **mm = 0xd17db700**,
active_mm = 0xd17db700, rss_stat = {events = 57, count = {54, 3, 0}},
exit_state = 0, exit_code = 0, exit_signal = 17, pdeath_signal = 0,
jobctl = 0, personality = 0, did_exec = 1, in_execve = 0, in_iowait = 0,
no_new_privs = 0, sched_reset_on_fork = 0, sched_contributes_to_load = 1,
pid = 2208, tgid = 2208, stack_canary = 2101085047,
real_parent = 0xeeca8000, parent = 0xeeca8000, children = {
next = 0xeec5dd48, prev = 0xeec5dd48}, sibling = {next = 0xeeca8248,
prev = 0xeeca8248}, group_leader = 0xeec5db00, ptraced = {
next = 0xeec5dd5c, prev = 0xeec5dd5c}, ptrace_entry = {next = 0xeec5dd64,
prev = 0xeec5dd64}, pids = {{node = {next = 0x0, pprev = 0xddbcc788},
pid = 0xddbcc780}, {node = {next = 0xdda29c78, pprev = 0xee986a78},
pid = 0xe37b3600}, {node = {next = 0xdda29c84, pprev = 0xee986a84},
pid = 0xe37b3600}}, thread_group = {next = 0xeec5dd90,
prev = 0xeec5dd90}, vfork_done = 0x0, set_child_tid = 0xb599e7e8,

21

Pg. 3

clear_child_tid = 0x0, utime = 0, stime = 0, utimescaled = 0,
stimescaled = 0, gtime = 0, prev_cputime = {utime = 0, stime = 0},
nvcsw = 1, nivcsw = 4, start_time = {tv_sec = 260, tv_nsec = 192919173},
real_start_time = {tv_sec = 260, tv_nsec = 192919173}, min_flt = 232,
maj_flt = 0, cputime_expires = {utime = 0, stime = 0, sum_exec_runtime = 0},
cpu_timers = {{next = 0xeec5ddf0, prev = 0xeec5ddf0}, {next = 0xeec5ddf8,
prev = 0xeec5ddf8}, {next = 0xeec5de00, prev = 0xeec5de00}},
real_cred = 0xd17d9900, cred = 0xd17d9900, comm = "**cat**\000geist-fts\000n",
link_count = 0, total_link_count = 0, sysvsem = {undo_list = 0x0},
last_switch_count = 0, thread = {tls_array = {{{{a = 1493237759,
b = 3084907104}, {limit0 = 65535, base0 = 22784, base1 = 96,
type = 2, s = 1, dpl = 3, p = 1, limit = 15, avl = 1, l = 0,
d = 1, g = 1, base2 = 183}}}, {{{a = 0, b = 0}, {limit0 = 0,
base0 = 0, base1 = 0, type = 0, s = 0, dpl = 0, p = 0, limit = 0,
avl = 0, l = 0, d = 0, g = 0, base2 = 0}}}, {{{a = 0, b = 0}, {
limit0 = 0, base0 = 0, base1 = 0, type = 0, s = 0, dpl = 0, p = 0,
limit = 0, avl = 0, l = 0, d = 0, g = 0, base2 = 0}}}},
sp0 = 4008648696, sp = 4008647956, sysenter_cs = 96, ip = 3244790983,
gs = 0, ptrace_bps = {0x0, 0x0, 0x0, 0x0}, debugreg6 = 0, ptrace_dr7 = 0,
cr2 = 0, trap_nr = 0, error_code = 0, fpu = {last_cpu = 4294967295,
has_fpu = 0, state = 0xd7f9ad80}, vm86_info = 0x0, screen_bitmap = 0,
v86flags = 0, v86mask = 0, saved_sp0 = 0, saved_fs = 0, saved_gs = 0,
io_bitmap_ptr = 0x0, iopl = 0, io_bitmap_max = 0}, fs = 0xddbccdc0,

22

Pg. 4

files = 0xd7fa6900, nsproxy = 0xc196a924, **signal = 0xeee8e780**,
sighand = 0xeef5e4c0, blocked = {sig = {0, 0}}, real_blocked = {sig = {0,
0}}, saved_sigmask = {sig = {0, 0}}, pending = {list = {
next = 0xeec5dee0, prev = 0xeec5dee0}, signal = {sig = {0, 0}}},
sas_ss_sp = 0, sas_ss_size = 0, notifier = 0, notifier_data = 0x0,
notifier_mask = 0x0, task_works = 0x0, audit_context = 0x0,
loginuid = 4294967295, sessionid = 4294967295, seccomp = {mode = 0,
filter = 0x0}, parent_exec_id = 11, self_exec_id = 12, alloc_lock = {{
rlock = {raw_lock = {{head_tail = 25186, tickets = {head = 98 'b',
tail = 98 'b'}}}}}, pi_lock = {raw_lock = {{head_tail = 1542,
tickets = {head = 6 '\006', tail = 6 '\006'}}}}, pi_waiters = {
node_list = {next = 0xeec5df28, prev = 0xeec5df28}}, pi_blocked_on = 0x0,
journal_info = 0x0, bio_list = 0x0, plug = 0x0, reclaim_state = 0x0,
backing_dev_info = 0x0, io_context = 0x0, ptrace_message = 0,
last_siginfo = 0x0, ioac = {rchar = 1299, wchar = 1, syscr = 6, syscw = 1,
read_bytes = 0, write_bytes = 0, cancelled_write_bytes = 0},
acct_rss_mem1 = 0, acct_vm_mem1 = 0, acct_timexpd = 0, mems_allowed = {
bits = {1}}, mems_allowed_seq = {sequence = 0},
cpuset_mem_spread_rotor = -1, cpuset_slab_spread_rotor = -1,
cgroups = 0xc1b11c60, cg_list = {next = 0xeec5dfb4, prev = 0xeec5dfb4},
robust_list = 0x0, pi_state_list = {next = 0xeec5dfc0, prev = 0xeec5dfc0},
pi_state_cache = 0x0, perf_event_ctxp = {0x0, 0x0}, perf_event_mutex = {
count = {counter = 1}, wait_lock = {{rlock = {raw_lock = {{head_tail = 0,

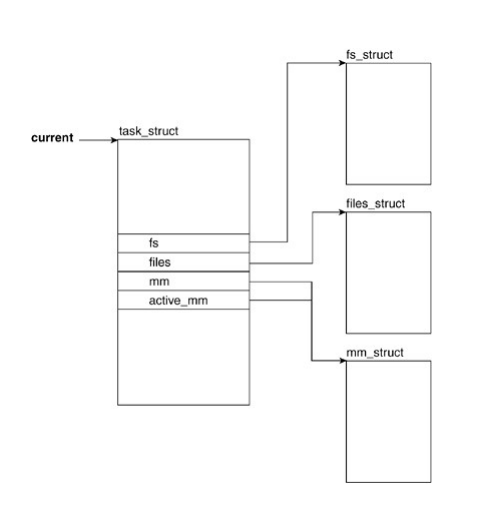
23

Pg. 5

tickets = {head = 0 '\000', tail = 0 '\000'}}}}}, wait_list = {
next = 0xeec5dfdc, prev = 0xeec5dfdc}, owner = 0x0, spin_mlock = 0x0},
perf_event_list = {next = 0xeec5dfec, prev = 0xeec5dfec}, rcu = {next = 0x0,
func = 0}, splice_pipe = 0x0, task_frag = {page = 0x0, offset = 0,
size = 0}, delays = 0xeec06780, nr_dirtied = 0, nr_dirtied_pause = 32,
dirty_paused_when = 0, latency_record_count = 0, latency_record = {{
backtrace = {0 <repeats 12 times>}, count = 0, time = 0,
max = 0} <repeats 32 times>}, timer_slack_ns = 50000,
default_timer_slack_ns = 50000, curr_ret_stack = -1, ret_stack = 0x0,
ftrace_timestamp = 0, trace_overrun = {counter = 0}, tracing_graph_pause = {
counter = 0}, trace = 0, trace_recursion = 0, memcg_batch = {do_batch = 0,
memcg = 0x0, nr_pages = 0, memsw_nr_pages = 0},
memcg_kmem_skip_account = 0, memcg_oom = {memcg = 0x0, gfp_mask = 0,
order = 0, may_oom = 0}, utask = 0x0, sequential_io = 0,
sequential_io_avg = 0}

24

文件系统和地址空间




```
(gdb) p task->mm
$17 = (struct mm_struct *) 0xd17db700
(gdb) p task->active_mm
$18 = (struct mm_struct *) 0xd17db700
```

25

pid

- In Linux, each process has a unique process identifier (pid). This pid is stored in the task_struct as a type pid_t. Although this type can be traced back to an integer type, the default maximum value of a pid is 32,768 (the value pertaining to a short int).

```
(gdb) p task->pid
$19 = 2208
(gdb) p task->tgid
$20 = 2208
```

26

comm

- A process is often created by means of a command-line call to an executable. The comm field holds the name of the executable as it is called on the command line.

27

ptrace

- ptrace is set when the ptrace() system call is called on the process for performance measurements. Possible ptrace() flags are defined in include/ linux/ptrace.h.

28

uid

- The uid field holds the user ID number of the user who created the process.
- This field is used for protection and security purposes. Likewise, the gid field holds the group ID of the group who owns the process.
- A uid or gid of 0 corresponds to the root user and group.

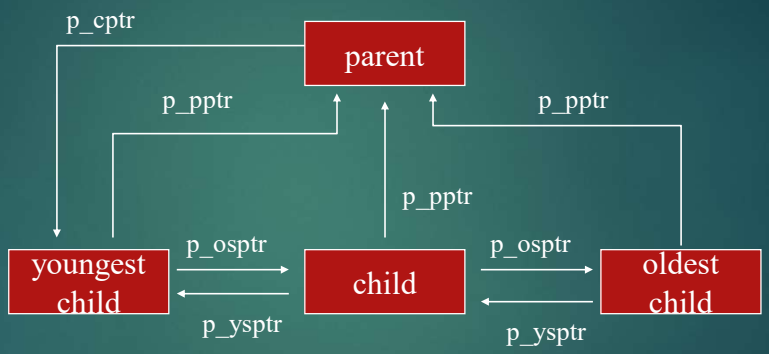
29

进程关系

- Processes are related
  - Parent/child (fork()), siblings
  - Possible to "re-parent"
    - Parent vs. original parent
  - Parent can "wait" for child to terminate
- Process groups
  - Possible to send signals to all members
- Sessions
  - Processes related to login

30

进程关系图



pid_t getpid(void);
pid_t getppid(void);

结构体指引

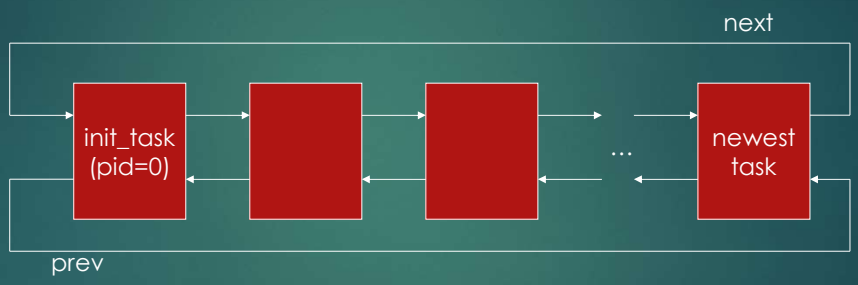
- Several pointers exist between *task_struct*s:
  - *parent* – pointer to parent process
  - *children* – pointer to linked list of child processes
  - *sibling* – pointer to task of "next younger sibling" of current process
- *children* and *sibling* point to the *task_struct* for the first thread created in a process.
- The *task_struct* for every thread in a process has the same pointer values.

进程链表

- Kernel keeps a list of process descriptors
- A 'doubly-linked' circular list is used
- The 'init_task' serves as a fixed header
- Other tasks inserted/deleted dynamically
- Tasks have forward & backward pointers, implemented as fields in the 'tasks' field
- To go forward: task = next_task(task);
- To go backward: task = prev_task(task);

33

Doubly-linked Circular List



34

0号进程

- 也称为Idle进程，或者swapper进程
- 系统启动时，捏造而出，不像其它集成是do_fork而出
- 每个CPU一个Idle线程，没有其它任务时执行Idle线程
- 创建Init进程，是所有其它进程的祖先

```
(gdb) p current_task->comm
$1 = "swapper/0\000\000\000\000\000\000"
(gdb) p current_task->pid
$2 = 0
(gdb) p current_task->tgid
$3 = 0
(gdb) p current_task
$4 = (struct task_struct *) 0xc1954980
```

```
#define INIT_TASK_COMM "swapper“

    .mm             = NULL,                         \
    .active_mm = &init_mm,

//include/linux/init_task.h
```

35

全局变量init_task

```
(gdb) p init_task
$1 = {state = 0, stack = 0xffffffff81e00000 <init_thread_union>, usage = {
    counter = 2}, flags = 2097408, ptrace = 0, wake_entry = {
    next = 0x0 <irq_stack_union>}, on_cpu = 0, wakee_flips = 0,
  wakee_flip_decay_ts = 0, last_wakee = 0x0 <irq_stack_union>, wake_cpu = 0,
  on_rq = 1, prio = 120, static_prio = 120, normal_prio = 120,
  rt_priority = 0, sched_class = 0xffffffff81a18be0 <idle_sched_class>, se = {
    load = {weight = 1048576, inv_weight = 4194304}, run_node = {
      __rb_parent_color = 0, rb_right = 0x0 <irq_stack_union>,
      rb_left = 0x0 <irq_stack_union>}, group_node = {
```

```
(gdb) p init_task.comm
$2 = "swapper/0\000\000\000\000\000\000"
(gdb) p init_task.pid
$3 = 0
(gdb) p init_task.tgid
$4 = 0
```

36

多CPU系统

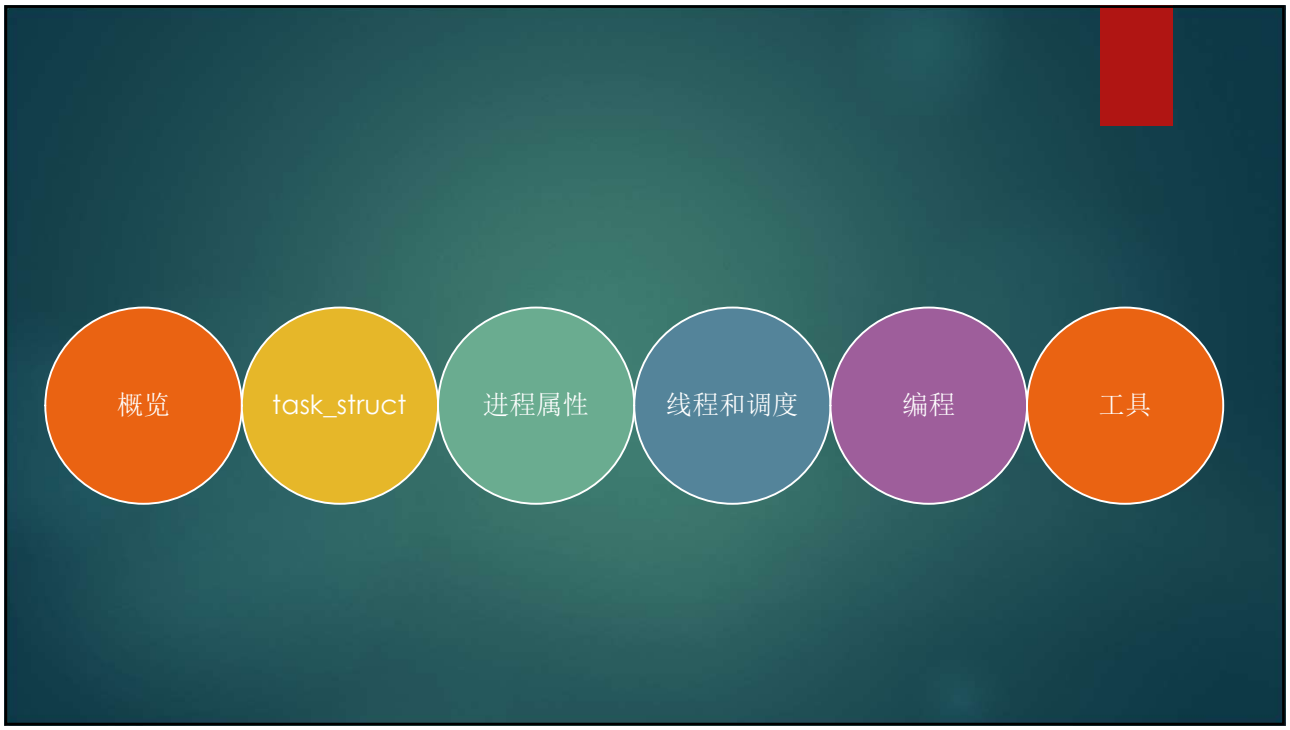
- On SMP systems, this task uses *clone* to create duplicate tasks which run as the idle task on each of the other processors.
- All of these tasks have process ID zero.
- Each of these tasks is used only by its associated processor.
- 与Windows非常类似

37

```
root@gedu-VirtualBox:/home/gedu# ps -e -o pid,ppid,comm,args --sort pid
  PID  PPID COMMAND         COMMAND
    1     0 systemd         /sbin/init splash nokaslr
    2     0 kthreadd        [kthreadd]
    3     2 ksoftirqd/0     [ksoftirqd/0]
    5     2 kworker/0:0H    [kworker/0:0H]
    7     2 rcu_sched       [rcu_sched]
    8     2 rcu_bh          [rcu_bh]
    9     2 migration/0     [migration/0]
   10     2 lru-add-drain   [lru-add-drain]
   11     2 watchdog/0      [watchdog/0]
   12     2 cpuhp/0         [cpuhp/0]
   13     2 kdevtmpfs       [kdevtmpfs]
   14     2 netns           [netns]
   15     2 khungtaskd      [khungtaskd]
   16     2 oom_reaper      [oom_reaper]
   17     2 writeback       [writeback]
   18     2 kcompactd0      [kcompactd0]
   19     2 ksmd            [ksmd]
   20     2 khugepaged      [khugepaged]
   21     2 crypto          [crypto]
   22     2 kintegrityd     [kintegrityd]
   23     2 bioset          [bioset]
   24     2 kblockd         [kblockd]
   25     2 ata_sff         [ata_sff]
   26     2 md              [md]
   27     2 devfreq_wq      [devfreq_wq]
   28     2 watchdogd       [watchdogd]
   32     2 kswapd0         [kswapd0]
   33     2 vmstat          [vmstat]
   34     2 ecryptfs-kthrea [ecryptfs-kthrea]
   73     2 kthrotld        [kthrotld]
```

ps -e -o pid,ppid,comm,args --sort pid

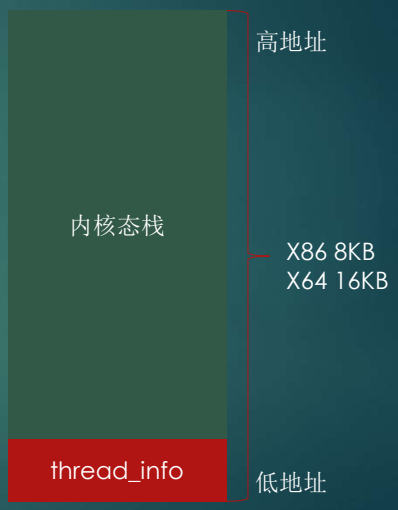
38



39

线程的内核态栈

- 创建线程时，创建内核态栈
- 32位下通常两个内存页大小，即8KB
- 最低地址处放CPU架构相关的thread_info结构



40

```
static inline struct task_struct * get_current(void)
{
        struct task_struct *current;
        __asm__("andl %%esp,%0; ":"=r" (current) : "0" (~8191UL));
        return current;
 }
```

2.4内核X86实现，栈的底部直接放的task_struct

```
DECLARE_PER_CPU(struct task_struct *, current_task);

static __always_inline struct task_struct *get_current(void)
{
        return this_cpu_read_stable(current_task);
}
```

4.16内核X86实现，访问per-CPU的变量

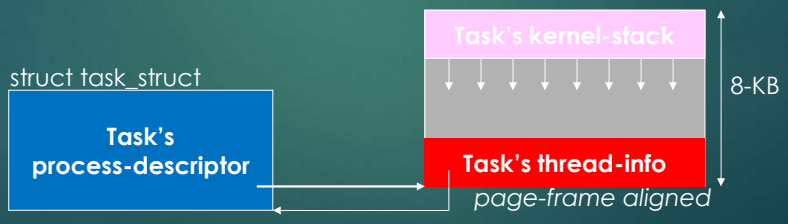
```
static __always_inline struct task_struct *get_current(void)
{
        unsigned long sp_el0;

        asm ("mrs %0, sp_el0" : "=r" (sp_el0));

        return (struct task_struct *)sp_el0;
}
```

ARM64实现，访问per-CPU的变量

41

线程上下文

- Linux uses part of a task's kernel-stack page-frame to store thread information
- The thread_info includes a pointer to the task's process-descriptor data-structure



42

CPU架构相关

- 供不同架构的CPU记录架构相关的信息
- 从执行角度看，CPU先找到这个架构，再通过这个结构访问到task_struct

thread_info

Defined as a struct type in:

- arch/microblaze/include/asm/switch_to.h, line 13
- arch/microblaze/include/asm/thread_info.h, line 66
- arch/sh/include/asm/thread_info.h, line 28
- arch/cris/include/asm/thread_info.h, line 29
- arch/powerpc/include/asm/thread_info.h, line 40
- arch/avr32/include/asm/thread_info.h, line 22
- arch/score/include/asm/thread_info.h, line 29
- arch/sparc/include/asm/trap_block.h, line 22
- arch/sparc/include/asm/thread_info_32.h, line 27
- arch/sparc/include/asm/thread_info_64.h, line 36
- arch/tile/include/asm/thread_info.h, line 27
- arch/frv/include/asm/thread_info.h, line 32
- arch/arm/include/asm/thread_info.h, line 60
- arch/xtensa/include/asm/thread_info.h, line 45
- arch/blackfin/include/asm/thread_info.h, line 38
- arch/um/include/asm/thread_info.h, line 15
- arch/m68k/include/asm/thread_info.h, line 26
- arch/arc/include/asm/thread_info.h, line 43
- arch/unicore32/include/asm/switch_to.h, line 14
- arch/unicore32/include/asm/thread_info.h, line 68
- arch/s390/include/asm/thread_info.h, line 35
- arch/mips/include/asm/thread_info.h, line 24

43

x86

```
struct thread_info {
    struct task_struct*task;        /* main task structure */
    struct exec_domain *exec_domain;  /* execution domain */
    __u32               flags;      /* low level flags */
    __u32               status;     /* thread synchronous flags */
    __u32               cpu;        /* current CPU */
    int             saved_preempt_count;
    mm_segment_t        addr_limit;
    struct restart_block    restart_block;
    void __user         *sysenter_return;
    unsigned int        sig_on_uaccess_error:1;
    unsigned int        uaccess_err:1;    /* uaccess failed */
};
```

44

访问线程的‘thread-info’

- During a task's execution in kernel-mode, it's very quick to find that task's *thread_info* object
- Just use two assembly-language instructions:

```
movl    $0xFFFFE000, %eax
andl    %esp, %eax
```

对于8KB的栈

Ok, now %eax = the thread-info's base-address

- Masking off 13 bits of the stack yields *thread_info*
- Macro *current_thread_info* implements this computation
- *thread_info* points to *task_struct*
- *current* macro yields the *task_struct*
- *current* is not a static variable, useful for SMP

```
#define get_current() (current_thread_info()->task)
#define current get_current()
```

45

current_thread_info

```
static inline struct thread_info *current_thread_info(void)
{
    register unsigned long sp asm ("sp");
    return (struct thread_info *)(sp & ~(THREAD_SIZE - 1));
}
```

46

结构体互访

- Use a macro '**task_thread_info**(task)' to get a pointer to the 'thread_info' structure:

```
struct thread_info  *info = task_thread_info( task );
```

- Then one more step gets you back to the address of the task's process-descriptor:

```
struct task_struct  *task = info->task;
```

47

task_thread_info

//sched.h

```
3066 #define task_thread_info(task)  ((struct thread_info *)(task)->stack)
3067 #define task_stack_page(task)   ((task)->stack)
```

48

thread_struct

- 用于保存CPU相关的线程状态信息（寄存器）
- Task_struct中的thread结构体指向
  - /* CPU-specific state of this task */
  - struct thread_struct thread;
- 结构体较大，如果放在thread_info中，会占用宝贵的内核态栈空间

49

x86

```
struct thread_struct {
        /* Cached TLS descriptors: */
        struct desc_struct      tls_array[GDT_ENTRY_TLS_ENTRIES];
        unsigned long           sp0;
        unsigned long           sp;
#ifdef CONFIG_X86_32
        unsigned long           sysenter_cs;
#else
        unsigned long           usersp;   /* Copy from PDA */
        unsigned short          es;
        unsigned short          ds;
        unsigned short          fsindex;
        unsigned short          gsindex;
#endif
#ifdef CONFIG_X86_32
        unsigned long           ip;
#endif
#ifdef CONFIG_X86_64
        unsigned long           fs;
#endif
        unsigned long           gs;
        /* Save middle states of ptrace breakpoints */
        struct perf_event       *ptrace_bps[HBP_NUM];
        /* Debug status used for traps, single steps, etc... */
```

50

task->thread

(gdb) p task->thread
$24 = {tls_array = {{{{a = 1493237759, b = 3084907104}, {limit0 = 65535,
base0 = 22784, base1 = 96, type = 2, s = 1, dpl = 3, p = 1,
limit = 15, avl = 1, l = 0, d = 1, g = 1, base2 = 183}}}, {{{a = 0,
b = 0}, {limit0 = 0, base0 = 0, base1 = 0, type = 0, s = 0, dpl = 0,
p = 0, limit = 0, avl = 0, l = 0, d = 0, g = 0, base2 = 0}}}, {{{
a = 0, b = 0}, {limit0 = 0, base0 = 0, base1 = 0, type = 0, s = 0,
dpl = 0, p = 0, limit = 0, avl = 0, l = 0, d = 0, g = 0,
base2 = 0}}}}, **sp0 = 4008648696, sp = 4008647956**, sysenter_cs = 96,
ip = 3244790983, gs = 0, ptrace_bps = {0x0, 0x0, 0x0, 0x0}, debugreg6 = 0,
ptrace_dr7 = 0, cr2 = 0, trap_nr = 0, error_code = 0, fpu = {
last_cpu = 4294967295, has_fpu = 0, state = 0xd7f9ad80}, vm86_info = 0x0,
screen_bitmap = 0, v86flags = 0, v86mask = 0, saved_sp0 = 0, saved_fs = 0,
saved_gs = 0, io_bitmap_ptr = 0x0, iopl = 0, io_bitmap_max = 0}

51

Finding task-related kernel-data

- Use a macro '**task_thread_info**(task)' to get a pointer to the 'thread_info' structure:

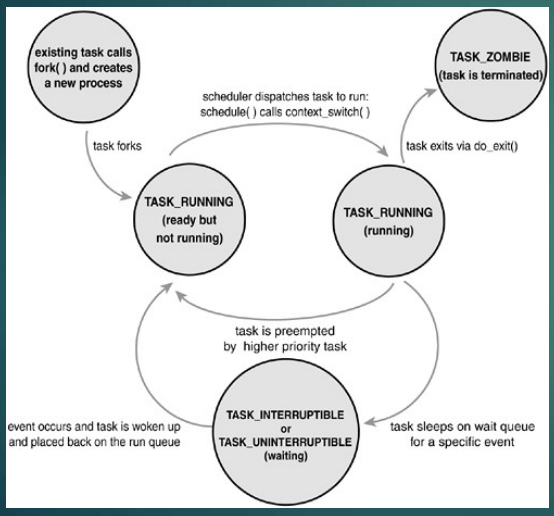
  struct thread_info *info = task_thread_info(task);

- Then one more step gets you back to the address of the task's process-descriptor:

  struct task_struct *task = info->task;

52

运行状态



53

Task States

From kernel-header: <linux/sched.h>

- #define TASK_RUNNING 0
- #define TASK_INTERRUPTIBLE 1
- #define TASK_UNINTERRUPTIBLE 2
- #define TASK_STOPPED 4
- #define TASK_TRACED 8
- #define EXIT_ZOMBIE 16
- #define EXIT_DEAD 32
- #define TASK_NONINTERACTIVE 64
- #define TASK_DEAD 128

54

Task States

- TASK_RUNNING – the thread is running on the CPU or is waiting to run
- TASK_INTERRUPTIBLE – the thread is sleeping and can be awoken by a signal (EINTR)
- TASK_UNINTERRUPTIBLE – the thread is sleeping and cannot be awakened by a signal
- TASK_STOPPED – the process has been stopped by a signal or by a debugger
- TASK_TRACED – the process is being traced via the ptrace system call
- TASK_NONINTERACTIVE – the process has exited
- TASK_DEAD – the process is being cleaned up and the task is being deleted

55

Exit States

```
135 * We have two separate sets of flags: task->state
136 * is about runnability, while task->exit_state are
137 * about the task exiting. Confusing, but this way
138 * modifying one set can't modify the other one by
139 * mistake.
```

- EXIT_ZOMBIE – the process is exiting but has not yet been waited for by its parent
- EXIT_DEAD – the process has exited and has been waited for

56

等待信号

```
#include <sys/types.h>
    #include <sys/wait.h>

    pid_t wait(int *status);

    pid_t waitpid(pid_t pid, int *status, int options);

    int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

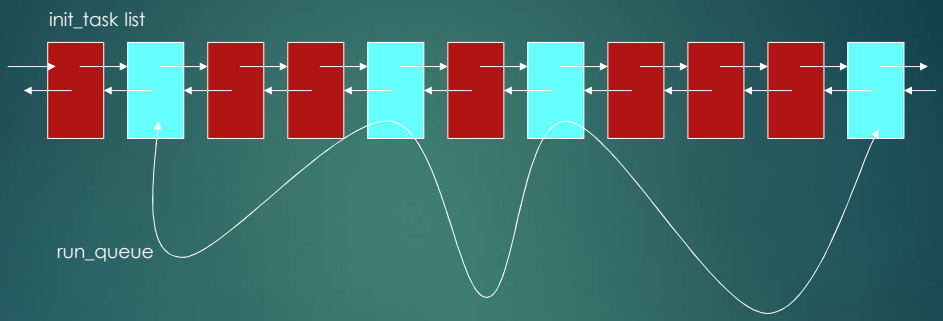
57

‘run’ queues and ‘wait’ queues

- In order for Linux to efficiently manage the scheduling of its various ‘tasks’, separate queues are maintained for ‘running’ tasks and for tasks that temporarily are ‘blocked’ while waiting for a particular event to occur (such as the arrival of new data from the keyboard, or the exhaustion of prior data sent to the printer)

58

Some tasks are 'ready-to-run'

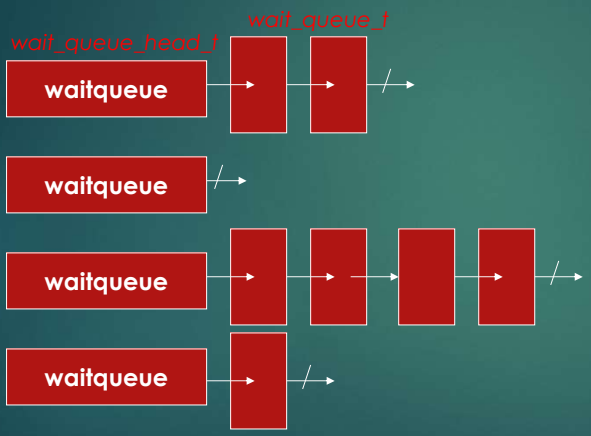


Those tasks that are ready-to-run comprise a sub-list of all the tasks, and they are arranged on a queue known as the 'run-queue'

Those tasks that are blocked while awaiting a specific event to occur are put on alternative sub-lists, called 'wait queues', associated with the particular event(s) that will allow a blocked task to be unblocked

59

Kernel Wait Queues



wait_queue_head_t can have 0 or more wait_queue_t chained onto them

However, usually just one element

Each wait_queue_t contains a list_head of tasks

All processes waiting for specific "event"

Used for timing, synch, device i/o, etc.

60

内核调度方面的牛人

people working on/interested in this stuff
Ingo Molnar, Red Hat, voluntary preemption, Ingo real-time preemption
Sven Dietrich, Monta Vista, MV real-time preemption
Daniel Walker, Monta Vista, priority inheritance??
John Cooper, Time Sys, ???
Tim Bird, Sony, port to 2.6.10-native, port to PPC
Scott Woods, Time Sys, IRQ threading??

61

Scheduler design goals

- **Judiciously apportion CPU time to all tasks in the system**
- **Ensure that the system delivers acceptable performance to each application, as long as the total workload is in the expected**
- **Application classes**
  - **Interactive - fast response time**
  - **Batch - high throughput**
  - **Real time – predictable scheduling behavior with guaranteed bounds on response times**
- **Avoidance of task starvation**

62

Linux scheduling policy

- Rules used to decide which task to run and when to switch to another
- Task priorities
  - 0 – 99 Real-time tasks (0 is the highest priority)
  - 100 – 139 time sharing tasks (139 is the lowest priority)
- Five scheduling policies
  - SCHED_NORMAL (traditionally called SCHED_OTHER), SCHED_BATCH, SCHED_IDLE, SCHED_FIFO, SCHED_RR

63

Linux scheduling policy

- SCHED_NORMAL/SCHED_BATCH/SCHED_IDLE
  - Each task is allocated a timeslice for running, switch to another task after time slice is used up
  - Always tries to run the task with the smallest virtual runtime
  - The virtual runtime of a task specifies when its next timeslice would start execution on the ideal multi-tasking CPU.
  - the virtual runtime of a task is its actual runtime normalized to the total number of running tasks
  - Tasks with SCHED_IDLE policy are by definition preempted by non-SCHED_IDLE tasks
  - SCHED_BATCH/SCHED_IDLE tasks do not preempt SCHED_NORMAL tasks (their preemption is driven by the tick)
  - Timeslice is based on task priority

64

Linux scheduling policy

- **SCHED_NORMAL/SCHED_BATCH/SCHED_IDLE**
- Timeslice is based on task priority - period * task_weight / cpu_load
  8 prio 100 tasks, 8 prio 19 tasks in one CPU
  weight [Prio100]= 88761, weight [Prio139]=15
  Timeslice100 = period*88761/((88761 + 15)*8)
  Timeslice139 = period*15/((88761+15)*8)
- **SCHED_FIFO**
- **SCHED_RR (Round-robin)**
- Time slice is 100ms

65

Linux scheduling class

- **Completely Fair Scheduler (Ingo Molnár , RedHat)**
- **Real-time scheduler**
- **Idle scheduler – only used in Kernel**
- **Stop scheduler – only used in Kernel**

66

A Simple O(1) Realtime scheduling Algorithm

- Highest priority is 255 (lsb: bit0)
  - Hash Table: $tbl[i] = \lfloor \log_2 i \rfloor$, i = 1..255
  - Uint8_t prio_bit76
  - Uint8_t prio_bit53[4]
  - Uint8_t prio_bit20[32]

67

Linux Realtime scheduling

- Use a bitmap to check highest priority RT task
- Example O(1) algorithm in RTOS, Highest priority is 255 (lsb: bit0)

Hash Table: $tbl[i] = \lfloor \log_2 i \rfloor$, i = 1..255

Uint8_t prio_bit76

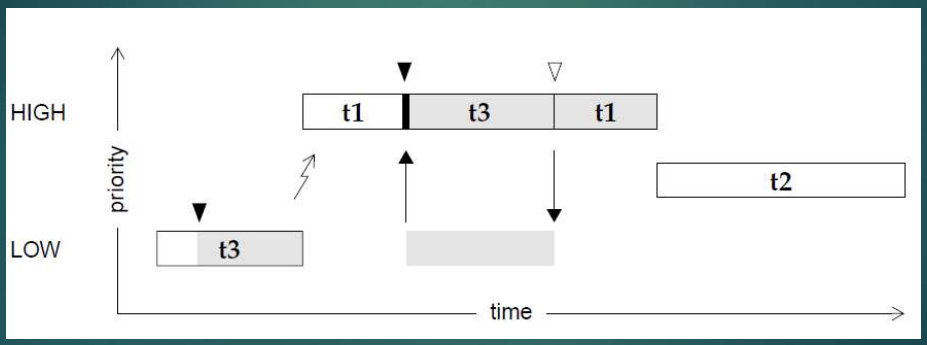
Uint8_t prio_bit53[4]

Uint8_t prio_bit20[32]

68

Linux Realtime scheduling

- Rtmutex - support priority inheritance

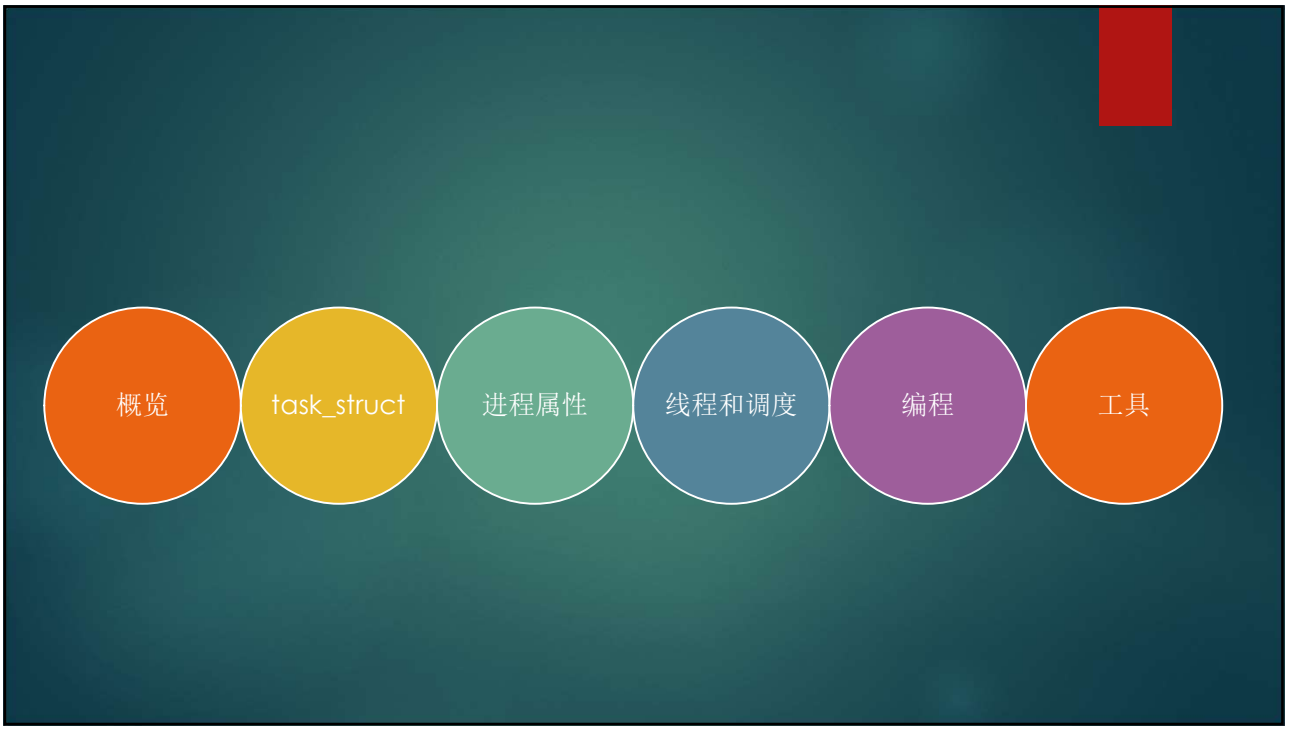


69

Linux CFS scheduling

- CFS basically models an "ideal, precise multi-tasking CPU" on real hardware.
  - "Ideal multi-tasking CPU" is a (non-existent :-)) CPU that has 100% physical power and which can run each task at precise equal speed, in parallel, each at 1/nr_running speed. For example: if there are 2 tasks running, then it runs each at 50% physical power --- i.e., actually in parallel.
- Red-Black tree – all runnable tasks are sorted by vruntime

70



71

Process and thread creation

72

cs431-cotter

- Process Creation
  - fork ( )
  - execve( )
  - clone( )
- **Thread Creation**
  - clone( )
  - pthread_create( )

72

Creating Processes - fork( )

73

- **pid_t fork(void);**
- Creates a child process that differs from the process only in its PID and PPID, and in the fact that resource utilizations are set to 0. Linux uses a copy-on-write procedure.
- **Child process returns PID = 0;**
- **Parent process returns child PID**

代码模式

74

```
pid = fork( );                  /* if the fork succeeds, pid > 0 in the parent */
if (pid < 0) {
        handle_error( );        /* fork failed (e.g., memory or some table is full) */
} else if (pid > 0) {
                                /* parent code goes here. /*/
} else {
                                /* child code goes here. /*/
}
```

Show me the code

75

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <sys/types.h>
#define LB_SIZE 128
int main ()
{
   time_t now;
   char lineBuf[LB_SIZE];
   pid_t  pid;
   void work (char *name);
   time (&now);
   printf ("Fork test at %s\n", ctime(&now));
    if ((pid = fork() ) < 0)  {
    printf("Couldn't fork process!\n");
    exit(1);
   }
   else if (pid == 0) {   // child process
    work ("child");
    return 0;
   }
   work("main");
   printf ("Closing main program...\n");
   return 0;
}
void work(char * name)  {
   printf("This is the %s work cycle\n",name);
   sleep (1);
}
```


编译和执行

76

```
ge@gewubox:~/work/silk$ gcc -o fork fork.c
ge@gewubox:~/work/silk$ ./fork
Fork test at Tue Nov  8 18:42:39 2016

This is the main work cycle in process with pid=3090
This is the child work cycle in process with pid=3091
Closing main program... _
```

execve( )

77

```
int execve (
    const char *filename,
    char *const argv[ ],
    char *const envp[ ]);
```

filename: name of executable file (or script) to be executed
argv[]: list of arguments to be passed to executable file
(an array of strings. May use NULL)
envp[]: list of environment variables to be used
(an array of strings. May use NULL)
No return on success. -1 on failure

多种变体

78

- Used to call an executable program from a child process. 6 varieties available, depending on options needed:

	Path-name	file-name	Arg list	argv[ ]	environ	envp[ ]
EXECL	X		X		X	
EXECLP		X	X		X	
EXECLE	X		X			X
EXECV	X			X	X	
EXECVP		X		X	X	
EXECVE	X			X		X

示例

```
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
extern char **environ;
extern int errno;
pid_t pid;
using namespace std;

int main (int argc, char* argv[])  {
      FILE *fin;
      char lineBuf[128];
      char *progname= argv[1];
     char myname[15] = "Bob";
     char mynum[5] = "5";
      static int result;
      char *args[4];
   if (argc == 2)
     strcpy (progname, argv[1]);
   else      {
     cout << "Usage: " << argv[0] << " progname" << endl;
     exit(1);
   }
   pid = fork();
   if (pid==0)       {  //We are in the child process
   args[0] = progname;
   args[1] = (char *) &myname;
   args[2] = (char *) &mynum;
   args[3] = (char *) 0; // array must be null terminated
     if ( execv (progname, args) == -1)    {
       cout << "ERROR IN EXECVE " << endl;
       perror("execve");
     }
   }
   else  //We are in the parent process
   {
       cout << "Created new process with process ID " << pid << endl;
   sleep(1);
  }
   return 0;
}//main
```


子进程(hello.cpp)

```
#include <iostream>
using namespace std;

int main(int argc, char * argv[])
{
    int i, mynum;
    if (argc != 3)    {
        cout << "Usage: : " << argv[0] << " NAME NUMBER" << endl;
        return 0;    }
    cout << "Hello, World!" << endl;
    cout << "Passed in " << argc << " arguments" << endl;
    cout << "They are: " << endl;
    mynum = atoi(argv[2]);
    cout << "Program name is " << argv[0] << endl;
    cout << "Name argument is " << argv[1] << endl;
    cout << "The number is " << mynum << endl;
   return 0;
}
```

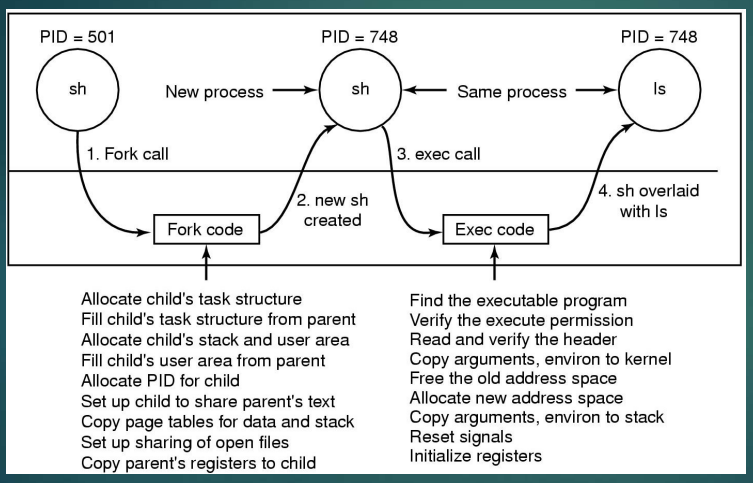
编译执行

```
[rcotter@kc-sce-450p2 cs431]$ g++ -o exec exec.cpp
[rcotter@kc-sce-450p2 cs431]$ g++ -o hello hello.cpp
[rcotter@kc-sce-450p2 cs431]$ ./exec hello

Created new process with process ID 2795
Hello, World!
Passed in 3 arguments
They are:
Program name is hello
Name argument is Bob
The number is 5
```


进程管理API

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

例说exec的实现



POSIX Threads

- 简称**Pthreads**
- POSIX Threads is an API defined by the standard POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)
- Implementations of the API are available on many Unix-like POSIX-conformant operating systems such as FreeBSD, NetBSD, OpenBSD, Linux, Mac OS X and Solaris, typically bundled as a library libpthread
- The Portable Operating System Interface (POSIX)

Pthread API

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

pthread_create()


```
#include <pthread.h>

int pthread_create (
    pthread_t *thread,  //variable to store returned thread ID
    pthread_attr_t *attr,  // thread attributes
    void * (*start_routine)(void *), //function to run
    void * arg);  //args to pass to new thread
```

参数attributes

87

- *May be set to **NULL** to use default attributes*
- **detachstate** //control whether the thread is created in the joinable state (default) or not.
- **schedpolicy** //select scheduling policy (default is regular, non-realtime scheduling)
- **schedparam** //set scheduling priority (default is 0)
- **inheritsched** //is sched policy to be inherited by child processes or threads? (default is no)

代码示例(silk.cpp)

88

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <pthread.h>
#include <stdlib.h>
#include <iostream>
#define LB_SIZE 128

using namespace std;
void * work (void * arg); // function prototype
int main (int argc, char * argv[]) {
    time_t now;
    pthread_t tid[3];
    int childno, childcount = 3, index, i, childid[3];
    void * ptReturn; // a pointer to the return value from a thread
    time (&now);
    cout << "Thread Test at " << ctime(&now) << endl;
    for (index = 0; index < childcount; index++) {
      childid[index] = index;
      if (( pthread_create(&tid[index], NULL, &work, (void *)&childid[index] )) > 0)
      {
        cout << "Couldn't create new thread!" << endl;
        exit(1);
      } else { //we're in main
```

```
        cout << "Thread " << index << " has tid " << tid[index] << endl;
        childno++;
      }
    }
    cout << "Now, wait for threads to finish ..." << endl;
    for (i = 0; i < childcount; i++)
    {
      pthread_join (tid[i], &ptReturn);
      cout << "Return value for " << tid[i] << " is: " << (long) ptReturn << endl;
    }
    cout << "Terminating Main program ....." << endl;
    return 0;
}

void * work(void * arg)
{
     int i, childnum; childnum = *(int *) arg; //recast as int pointer, then deref.
     for (i = 0; i < 3; i++)
     {
         cout<< "Loop "<< i << " of thread "<< childnum<<" work cycle"<< endl;
         sleep (1);
     }
     //pthread_exit ((void *) childnum);
     return ((void *)childnum);
}
```

89

编译执行

90

```
ge@gewubox:~/work/silk$ g++ -o silk silk.cpp -lpthread
ge@gewubox:~/work/silk$ ./silk
Thread Test at Mon Nov  7 19:44:48 2016

Thread 0 has tid 3074472768
Thread 1 has tid 3066080064
Thread 2 has tid 3057687360
Now, wait for threads to finish ...
Loop 0 of thread 2 work cycle
Loop 0 of thread 1 work cycle
Loop 0 of thread 0 work cycle
Loop 1 of thread 2 work cycle
Loop 1 of thread 1 work cycle
Loop 1 of thread 0 work cycle
Loop 2 of thread 2 work cycle
Loop 2 of thread 1 work cycle
Loop 2 of thread 0 work cycle
Return value for 3074472768 is: 0
Return value for 3066080064 is: 1
Return value for 3057687360 is: 2
Terminating Main program .......
ge@gewubox:~/work/silk$
```

90

链接库

91

- Only basic libraries included by default.
- Other libraries (math, pthread, etc.) must be explicitly linked.
- Libraries included in /usr/lib
- File name format is "libxxx.a" where xxx is library name
  - Math library: libm.a use "-lm"
  - POSIX threads library: libpthread.a use "-lpthread"

互斥量

```
(gdb) pt  struct __pthread_mutex_s
type = struct __pthread_mutex_s {
    int __lock;
    unsigned int __count;
    int __owner;
    unsigned int __nusers;
    int __kind;
    short __spins;
    short __elision;
    __pthread_list_t __list;
}
```

```
{__lock = 2, __count = 0, __owner = 6138, __nusers = 1, __kind = 0, __spins = 0, __elision = 0, __list =
{__prev = 0x0, __next = 0x0}}
```

等待互斥量

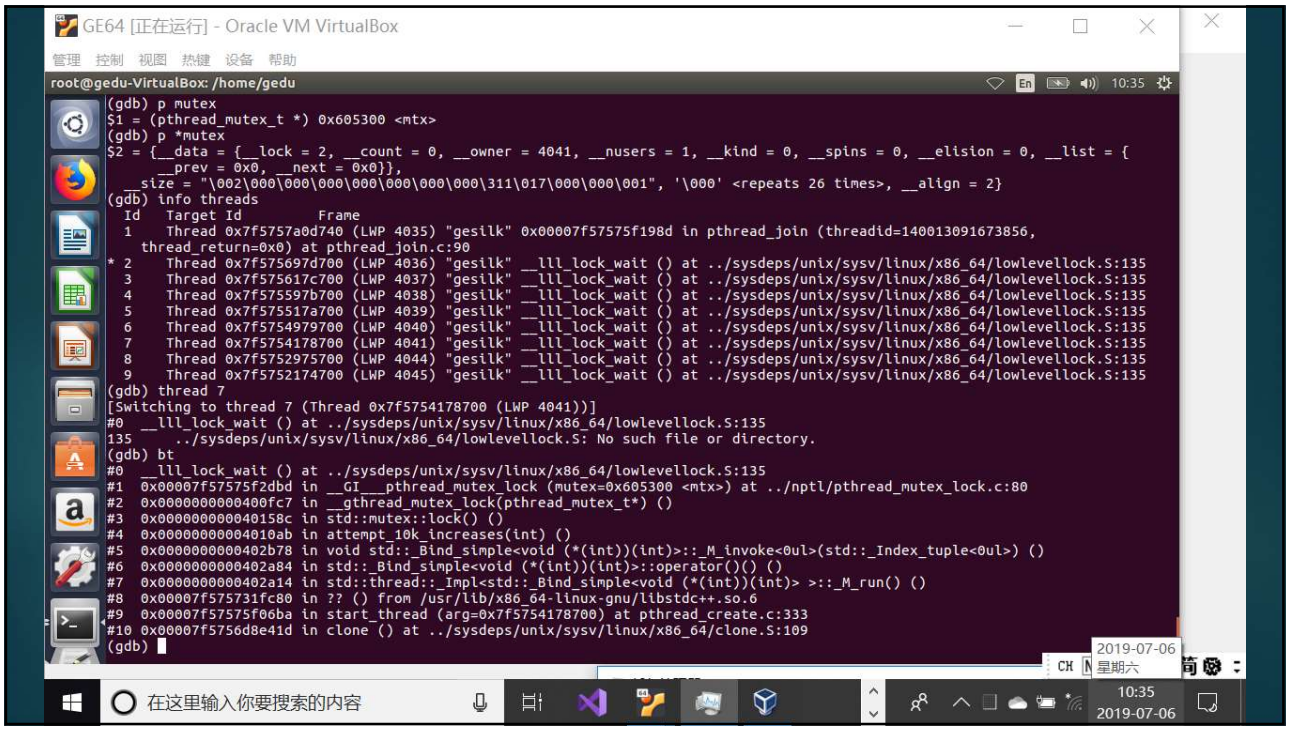
```
[0]kdb> btp 2481
Stack traceback for pid 2481
0xffff88003b93ab80     2481     2326  0    0   S  0xffff88003b93b5c0  gesilk
 ffff8800359a7c38 ffff88003936b800 ffff88003d2f4880 ffff88003b93ab80
 0000000000605300 ffff8800359a8000 ffffc900001ce280 ffff88003b93ab80
 0000000000000000 ffff8800359a7ce8 ffff8800359a7c50 ffffffff81891d25
Call Trace:
 [<ffffffff81891d25>] schedule+0x35/0x80
 [<ffffffff811106f64>] futex_wait_queue_me+0xc4/0x120
 [<ffffffff81107bc6>] futex_wait+0x116/0x280
 [<ffffffff81109ac7>] do_futex+0x237/0xb40
 [<ffffffff8110a451>] SyS_futex+0x81/0x180
 [<ffffffff810aaef6>] ? finish_task_switch+0x76/0x1f0
 [<ffffffff81896536>] entry_SYSCALL_64_fastpath+0x1e/0xa8
```

93

死锁和调试

```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7fdfb01a8700 (LWP 6133))]
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
135     ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S: No such file or directory.
(gdb) bt
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
#1  0x00007fdfb0e1ddbd in __GI___pthread_mutex_lock (mutex=0x605300 <mtx>) at ../nptl/pthread_mutex_lock.c:80
#2  0x00000000000400fc7 in __gthread_mutex_lock(pthread_mutex_t*) ()
#3  0x000000000040158c in std::mutex::lock() ()
#4  0x00000000004010ab in attempt_10k_increases(int) ()
#5  0x0000000000402b78 in void std::_Bind_simple<void (*(int))(int)>::_M_invoke<0ul>(std::_Index_tuple<0ul>) ()
#6  0x0000000000402a84 in std::_Bind_simple<void (*(int))(int)>::operator()() ()
#7  0x0000000000402a14 in std::thread::_Impl<std::_Bind_simple<void (*(int))(int)> >::_M_run() ()
#8  0x00007fdfb0b4ac80 in ?? () from /usr/lib/x86_64-linux-gnu/libstdc++.so.6
#9  0x00007fdfb0e1b6ba in start_thread (arg=0x7fdfb01a8700) at pthread_create.c:333
#10 0x00007fdfb05b941d in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:109
(gdb) frame 2
#2  0x00000000000400fc7 in __gthread_mutex_lock(pthread_mutex_t*) ()
(gdb) info args
No symbol table info available.
(gdb) frame 1
#1  0x00007fdfb0e1ddbd in __GI___pthread_mutex_lock (mutex=0x605300 <mtx>) at ../nptl/pthread_mutex_lock.c:80
80      ../nptl/pthread_mutex_lock.c: No such file or directory.
(gdb) info args
mutex = 0x605300 <mtx>
(gdb) p mutex
$1 = (pthread_mutex_t *) 0x605300 <mtx>
(gdb) p *mutex
$2 = {__data = {__lock = 2, __count = 0, __owner = 6138, __nusers = 1, __kind = 0, __spins = 0, __elision = 0, __list = {__prev = 0x0, __next = 0x0}},
  __size = "\002\000\000\000\000\000\000\000\372\027\000\000\001", '\000' <repeats 26 times>, __align = 2}
```

94



创建本土线程("Native Threads")

96

```
#include <sched.h>

pid = __clone(
  int (*fn) (void *arg), //pointer to function
  void *child_stack, //pointer to a new stack
  int flags, //various flags (see next slide)
  void *arg) //argument passed to function

A new stack must be created for the cloned task
  (process or thread).
```

The Clone System Call

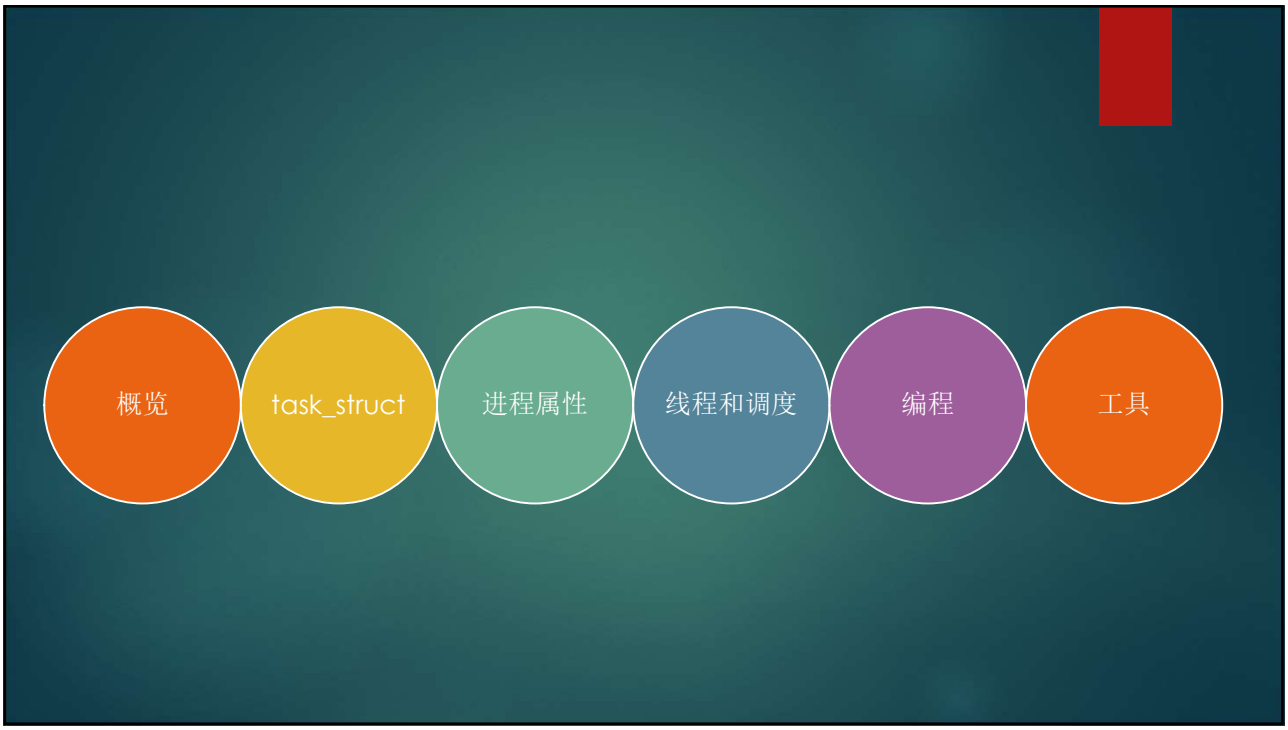
Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

Figure 10-9. Bits in the sharing_flags bitmap.


```c
#define _GNU_SOURCE
#include <sched.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>

#define LB_SIZE 128
int flag;
void test (void *childnum);  //function prototype

int main (int argc, char * argv[])
{
    time_t now;
    char lineBuf[LB_SIZE];
    pid_t  pid;
    int childno = 1, mainnum = 0;
    void *csp, *tcsp;

    csp=(int *)malloc(8192*8); //create a new stack
    if(csp)
       tcsp=csp+8192*8; //move pointer to the end of the stack
    else
       exit(errno);
```

```
    time (&now);
    printf ("Clone Test run on %s\n\n", ctime(&now));
    flag = 0;
    childno = 1;
    if (( pid = __clone( (void *)&test, tcsp, CLONE_VM,
            (void *)&childno )) < 0)
    {
        printf("Couldn't create new thread!\n");
        exit(1);
    }
    else { //we're in main
        while (flag == 0) ;
        printf("Just created thread %d\n", pid);
    }
    test(&mainnum);
    printf ("Main program is now shutting down\n\n");
    return 0;
}

void test(void * arg)   {
    int  childnum;
    flag = 1;
    childnum = *(int *) arg;
    printf("Thread %d work cycle\n",childnum);
    sleep (3);
}
```

99



100

进程快照 - ps

```
PS(1)                                Linux User's Manual                                PS(1)

NAME
       ps - report a snapshot of the current processes.

SYNOPSIS
       ps [options]

DESCRIPTION
       ps displays information about a selection of the active processes. If you want a repetitive update of
       the selection and the displayed information, use top(1) instead.

       This version of ps accepts several kinds of options:
       1   UNIX options, which may be grouped and must be preceded by a dash.
       2   BSD options, which may be grouped and must not be used with a dash.
       3   GNU long options, which are preceded by two dashes.

       Options of different types may be freely mixed, but conflicts can appear. There are some synonymous
       options, which are functionally identical, due to the many standards and ps implementations that this ps
       is compatible with.

       Note that "ps -aux" is distinct from "ps aux". The POSIX and UNIX standards require that "ps -aux" print
       all processes owned by a user named "x", as well as printing all processes that would be selected by the
       -a option. If the user named "x" does not exist, this ps may interpret the command as "ps aux" instead
       and print a warning. This behavior is intended to aid in transitioning old scripts and habits. It is
       fragile, subject to change, and thus should not be relied upon.

       By default, ps selects all processes with the same effective user ID (euid=EUID) as the current user and
       associated with the same terminal as the invoker. It displays the process ID (pid=PID), the terminal
       associated with the process (tname=TTY), the cumulated CPU time in [DD-]hh:mm:ss format (time=TIME), and
       the executable name (ucmd=CMD). Output is unsorted by default.
```

101

三种选项

UNIX风格	• 必须一开始 • 可以组合
BSD风格	• 可以组合 • 必须不带-
GNU的长选项	• 必须两个-开始

102

示例

- 显示进程树:
  - Unix风格　　ps -ejH
  - BSD风格　　ps axjf
- 显示线程信息
  - Unix风格　　ps -eLf
  - BSD风格　　ps axms

103

选项分类

进程筛选
- 简单选择
- 列表选择

输出格式
- 选择列

输出控制
- 排序
- 屏幕宽度高度

杂项

104

进程筛选 – 简单选择

- -A 所有进程
- -a 除去与终端无关和session leader进程外的所有进程
- -e 与-A相同
- -N 反向选取，等同于—deselect

105

进程筛选 – 使用列表选择

- --pid pid列表
  - 比如ps –pid 123,456
  - 也可以使用p、-p、-123、123
- -C cmd 根据命令选择

```
gedu@gedu-VirtualBox:~$ ps 669 1921
  PID TTY      STAT   TIME COMMAND
  669 ?        Ssl    0:00 /usr/sbin/rsyslogd -n
 1921 ?        Sl     0:00 /usr/lib/unity-settings-daemon/unity-fallback-mount-helper
gedu@gedu-VirtualBox:~$ ps -669 -1921
  PID TTY      STAT   TIME COMMAND
  669 ?        Ssl    0:00 /usr/sbin/rsyslogd -n
gedu@gedu-VirtualBox:~$ ps -669 1921
  PID TTY      STAT   TIME COMMAND
  669 ?        Ssl    0:00 /usr/sbin/rsyslogd -n
 1921 ?        Sl     0:00 /usr/lib/unity-settings-daemon/unity-fallback-mount-helper
```

```
gedu@gedu-VirtualBox:~$ ps -C upowerd
  PID TTY          TIME CMD
 1280 ?        00:00:00 upowerd
```

106

输出格式选项

- -l 长格式
- -o/O/o format

```
gedu@gedu-VirtualBox:~$ ps o pid,ppid,cmd
  PID  PPID CMD
 2194  2187 bash
 2286  2194 gedit ps.txt
 3533  2194 ps o pid,ppid,cmd
```

- -c 调度器信息
- -f 完整格式
- -F 更加完整的格式
- -X 寄存器格式
  - 显示ESP，EIP等

```
gedu@gedu-VirtualBox:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
gedu        2194  2187  0 15:58 pts/2    00:00:00 bash
gedu        2286  2194  0 16:13 pts/2    00:00:25 gedit ps.txt
gedu        3572  2194  0 20:44 pts/2    00:00:00 ps -f
gedu@gedu-VirtualBox:~$ ps -F
UID          PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
gedu        2194  2187  0  6090  3648   0 15:58 pts/2    00:00:00 bash
gedu        2286  2194  0 164867 13820  0 16:13 pts/2    00:00:25 gedit ps.txt
gedu        3573  2194  0  9775  3216   0 20:44 pts/2    00:00:00 ps -F
```

107

输出控制

- --sort 排序
- --width n 指定屏幕宽度
- --forest 进程树

```
gedu@gedu-VirtualBox:~$ ps -e -o pid,cmd,stime --forest
  PID CMD                         STIME
    2 [kthreadd]                  15:54
    4  \_ [kworker/0:0H]          15:54
    6  \_ [mm_percpu_wq]          15:54
    7  \_ [ksoftirqd/0]           15:54
    8  \_ [rcu_sched]             15:54
    9  \_ [rcu_bh]                15:54
   10  \_ [migration/0]           15:54
   11  \_ [watchdog/0]            15:54
   12  \_ [cpuhp/0]               15:54
   13  \_ [kdevtmpfs]             15:54
   14  \_ [netns]                 15:54
   15  \_ [khungtaskd]            15:54
   16  \_ [oom_reaper]            15:54
   17  \_ [writeback]             15:54
   18  \_ [kcompactd0]            15:54
   19  \_ [ksmd]                  15:54
   20  \_ [khugepaged]            15:54
   21  \_ [crypto]                15:54
   22  \_ [kintegrityd]           15:54
   23  \_ [kblockd]               15:54
   24  \_ [ata_sff]               15:54
   25  \_ [md]                    15:54
   26  \_ [edac-poller]           15:54
   27  \_ [devfreq_wq]            15:54
   28  \_ [watchdogd]             15:54
   32  \_ [kauditd]               15:54
   33  \_ [kswapd0]               15:54
   34  \_ [ecryptfs-kthrea]       15:54
   76  \_ [kthrotld]              15:54
   77  \_ [acpi_thermal_pm]       15:54
   78  \_ [scsi_eh_0]             15:54
   79  \_ [scsi_tmf_0]            15:54
   80  \_ [scsi_eh_1]             15:54
   81  \_ [scsi_tmf_1]            15:54
   87  \_ [ipv6_addrconf]         15:54
```

108

进程状态符

D	uninterruptible sleep (usually IO)
R	running or runnable (on run queue)
S	interruptible sleep (waiting for an event to complete)
T	stopped by job control signal
t	stopped by debugger during the tracing
W	paging (not valid since the 2.6.xx kernel)
X	dead (should never be seen)
Z	defunct ("zombie") process, terminated but not reaped by its parent

109

$ ps -ef

```
ge         2609     1  0 19:18 ?        00:00:00 /usr/lib/telepathy/mission-control-5
ge         2613     1  0 19:18 ?        00:00:00 /usr/lib/gnome-online-accounts/goa-daemon
ge         2618  1845  0 19:18 ?        00:00:00 gnome-screensaver
ge         2628  1845  0 19:19 ?        00:00:00 update-notifier
root       2652     1  0 19:19 ?        00:00:00 /usr/bin/python /usr/lib/system-service/system-service-d
ge         2660     1  0 19:19 ?        00:00:04 /usr/bin/python /usr/bin/update-manager --no-focus-on-map
ge         2685  1845  0 19:20 ?        00:00:00 /usr/lib/deja-dup/deja-dup/deja-dup-monitor
ge         2691  2504  0 19:21 pts/0    00:00:21 gedit silk.c
ge         2702     1  0 19:22 ?        00:00:00 /usr/lib/gvfs/gvfsd-metadata
ge         2704     1  0 19:22 ?        00:00:00 deja-dup --prompt
postfix    2869  1403  0 19:48 ?        00:00:00 pickup -l -t fifo -u -c
root       3147     2  0 20:38 ?        00:00:00 [kworker/u2:0]
root       3152     2  0 20:42 ?        00:00:00 [kworker/0:0]
root       3170     2  0 20:43 ?        00:00:00 [kworker/u2:2]
root       3175     2  0 20:48 ?        00:00:00 [kworker/0:1]
root       3178     2  0 20:48 ?        00:00:00 [kworker/u2:1]
ge         3182  2504  0 20:51 pts/0    00:00:00 ps -ef
```

- -e to display all the processes.
- -f to display full format listing.

110

$ ps -e -o pid,args --forest

```
2425 /usr/bin/python /usr/lib/unity-lens-video/unity-lens-video
2426 /usr/lib/unity-lens-music/unity-music-daemon
2427 /usr/lib/unity-lens-files/unity-files-daemon
2464 /usr/bin/zeitgeist-daemon
2470 /usr/lib/zeitgeist/zeitgeist-fts
2478  \_ /bin/cat
2472 zeitgeist-datahub
2499 gnome-terminal
2503  \_ gnome-pty-helper
2504  \_ bash
2691      \_ gedit silk.c
3190      \_ ps -e -o pid,args --forest
2564 /usr/lib/unity-lens-music/unity-musicstore-daemon
2565 /usr/bin/python /usr/lib/unity-scope-video-remote/unity-scope-video-remote
2609 /usr/lib/telepathy/mission-control-5
2613 /usr/lib/gnome-online-accounts/goa-daemon
2652 /usr/bin/python /usr/lib/system-service/system-service-d
2660 /usr/bin/python /usr/bin/update-manager --no-focus-on-map
2702 /usr/lib/gvfs/gvfsd-metadata
2704 deja-dup --prompt
```

111

$ ps aux --sort pmem

```
ge        2369  0.0  1.2  60596  9412 ?        Sl   19:18   0:00 /usr/lib/indicator-printers/indicator-printers-se
ge        2424  0.0  1.2  73796  9516 ?        Sl   19:18   0:00 /usr/lib/unity-lens-applications/unity-applicatio
ge        2600  0.0  1.2  73344  9592 ?        Sl   19:18   0:00 telepathy-indicator
ge        2618  0.0  1.2  41600  9760 ?        Sl   19:18   0:00 gnome-screensaver
colord    1949  0.0  1.4  53900 11124 ?        Sl   19:18   0:00 /usr/lib/i386-linux-gnu/colord/colord
ge        1974  0.0  1.4  85504 11180 ?        Sl   19:18   0:00 bluetooth-applet
ge        2628  0.0  1.5  59788 11876 ?        Sl   19:19   0:00 update-notifier
ge        2425  0.0  1.5  86076 12120 ?        Sl   19:18   0:00 /usr/bin/python /usr/lib/unity-lens-video/unity-l
ge        2704  0.0  1.7 116884 13072 ?        SNl  19:22   0:00 deja-dup --prompt
ge        1947  0.0  1.7 147188 13132 ?        Sl   19:18   0:03 metacity
ge        1977  0.0  1.9 125224 14724 ?        Sl   19:18   0:00 nm-applet
ge        2565  0.0  2.0 100820 15708 ?        Sl   19:18   0:00 /usr/bin/python /usr/lib/unity-scope-video-remote
ge        1933  0.0  2.1 137548 16312 ?        Sl   19:18   0:01 /usr/lib/gnome-settings-daemon/gnome-settings-dae
ge        2499  0.1  2.1  91744 16664 ?        Sl   19:18   0:11 gnome-terminal
ge        2342  0.0  2.3  98664 18044 ?        Sl   19:18   0:01 /usr/lib/unity/unity-panel-service
ge        1975  0.0  2.9 133104 22924 ?        Sl   19:18   0:01 nautilus -n
ge        1965  0.0  3.3 137920 25320 ?        Sl   19:18   0:02 unity-2d-panel
ge        2691  0.3  3.9 104912 30264 pts/0    Sl   19:21   0:21 gedit silk.c
root       924  0.4  7.2 103940 55432 tty7     Ss+  18:08   0:51 /usr/bin/X :0 -auth /var/run/lightdm/root/:0 -nol
ge        2660  0.0  7.5 159168 57716 ?        SNl  19:19   0:04 /usr/bin/python /usr/bin/update-manager --no-focu
ge        1966  0.1  7.6 260144 58332 ?        Sl   19:18   0:10 unity-2d-shell
```

112

线程选项

H Show threads as if they were processes.

-L Show threads, possibly with LWP and NLWP columns.

m Show threads after processes.

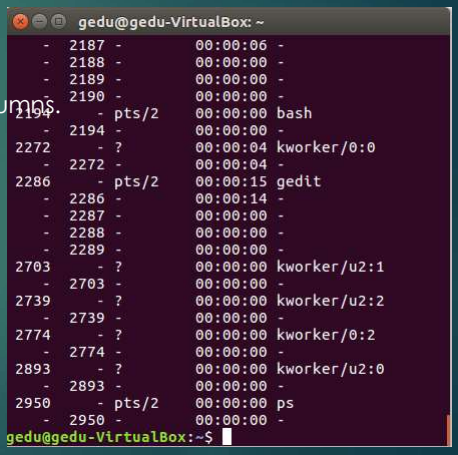
-m Show threads after processes.

-T Show threads, possibly with SPID column.

显示线程数：

ps -A -eo pid,nwchan,cmd,nlwp --sort nlwp

ps -A -m -eL



作者

ps was originally written by Branko Lankester ⟨lankeste@fwi.uva.nl⟩.
Michael K. Johnson ⟨johnsonm@redhat.com⟩ re-wrote it significantly to use the proc filesystem, changing a few things in the process.
Michael Shields ⟨mjshield@nyx.cs.du.edu⟩ added the pid-list feature.
Charles Blake ⟨cblake@bbn.com⟩ added multi-level sorting, the dirent-style library, the device name-to-number mmaped database, the approximate binary search directly on System.map, and many code and documentation cleanups.
David Mossberger-Tang wrote the generic BFD support for psupdate.
Albert Cahalan ⟨albert@users.sf.net⟩ rewrote ps for full Unix98 and BSD support, along with some ugly hacks for obsolete and foreign syntax.

pstree

```
gd@gdbox:~$ pstree -ps
init(1)─┬─NetworkManager(834)─┬─dhclient(878)
        │                     ├─dnsmasq(971)
        │                     ├─{NetworkManager}(872)
        │                     └─{NetworkManager}(879)
        ├─VBoxClient(1682)───VBoxClient(1683)───{VBoxClient}(1712)
        ├─VBoxClient(1692)───VBoxClient(1693)
        ├─VBoxClient(1697)───VBoxClient(1698)───{VBoxClient}(1704)
        ├─VBoxClient(1702)───VBoxClient(1703)─┬─{VBoxClient}(1707)
        │                                     └─{VBoxClient}(1708)
        ├─VBoxService(1162)─┬─{VBoxService}(1163)
        │                   ├─{VBoxService}(1164)
        │                   ├─{VBoxService}(1165)
        │                   ├─{VBoxService}(1166)
        │                   ├─{VBoxService}(1167)
        │                   ├─{VBoxService}(1170)
        │                   └─{VBoxService}(1171)
        ├─accounts-daemon(1367)───{accounts-daemon}(1368)
        ├─acpid(938)
        ├─atd(943)
        ├─avahi-daemon(741)───avahi-daemon(742)
        ├─bamfdaemon(2244)─┬─{bamfdaemon}(2245)
        │                  └─{bamfdaemon}(2246)
        ├─bluetoothd(723)
        ├─colord(1747)─┬─{colord}(1758)
        │              └─{colord}(2281)
        ├─console-kit-dae(1169)─┬─{console-kit-dae}(1174)
        │                       ├─{console-kit-dae}(1177)
        │                       ├─{console-kit-dae}(1178)
        │                       ├─{console-kit-dae}(1179)
        │                       ├─{console-kit-dae}(1180)
        │                       ├─{console-kit-dae}(1181)
        │                       ├─{console-kit-dae}(1182)
        │                       ├─{console-kit-dae}(1183)
        │                       ├─{console-kit-dae}(1184)
        │                       ├─{console-kit-dae}(1185)
        │                       ├─{console-kit-dae}(1186)
```

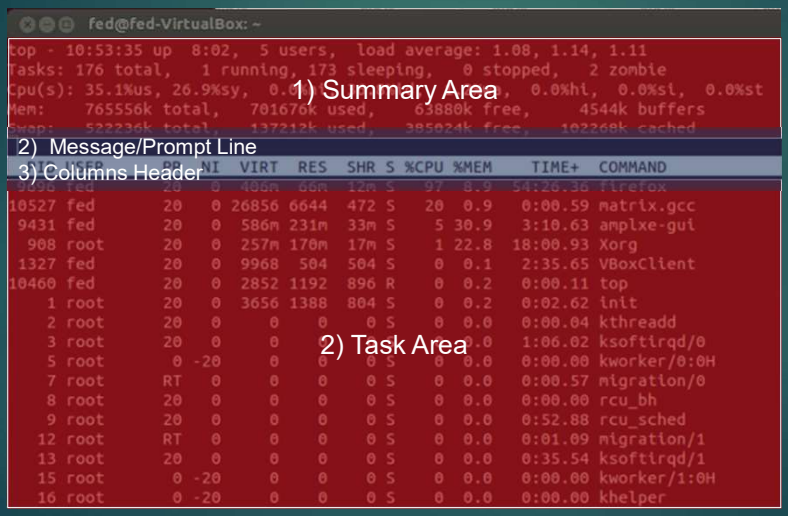
115

top

```
fed@fed-VirtualBox: ~
top - 10:53:35 up  8:02,  5 users,  load average: 1.08, 1.14, 1.11
Tasks: 176 total,   1 running, 173 sleeping,   0 stopped,   2 zombie
Cpu(s): 35.1%us, 26.9%sy,  0.0%ni, 36.8%id,  1.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    765556k total,   701676k used,    63880k free,     4544k buffers
Swap:   522236k total,   137212k used,   385024k free,   102268k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9696 fed       20   0  406m  66m  12m S   97  8.9  54:26.36 firefox
10527 fed       20   0 26856 6644  472 S   20  0.9   0:00.59 matrix.gcc
 9431 fed       20   0  586m 231m  33m S    5 30.9   3:10.63 amplxe-gui
  908 root      20   0  257m 170m  17m S    1 22.8  18:00.93 Xorg
 1327 fed       20   0  9968  504  504 S    0  0.1   2:35.65 VBoxClient
10460 fed       20   0  2852 1192  896 R    0  0.2   0:00.11 top
    1 root      20   0  3656 1388  804 S    0  0.2   0:02.62 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.04 kthreadd
    3 root      20   0     0    0    0 S    0  0.0   1:06.02 ksoftirqd/0
    5 root       0 -20     0    0    0 S    0  0.0   0:00.00 kworker/0:0H
    7 root      RT   0     0    0    0 S    0  0.0   0:00.57 migration/0
    8 root      20   0     0    0    0 S    0  0.0   0:00.00 rcu_bh
    9 root      20   0     0    0    0 S    0  0.0   0:52.88 rcu_sched
   12 root      RT   0     0    0    0 S    0  0.0   0:01.09 migration/1
   13 root      20   0     0    0    0 S    0  0.0   0:35.54 ksoftirqd/1
   15 root       0 -20     0    0    0 S    0  0.0   0:00.00 kworker/1:0H
   16 root       0 -20     0    0    0 S    0  0.0   0:00.00 khelper
```

116

信息块



任务队列信息

118

```
top - 10:53:35 up  8:02,  5 users,  load average: 1.08, 1.14, 1.11
```

- 系统当前时间
- 系统工作时间，格式为时:分
- 登录用户数
- 任务队列的平均长度，三个数值分别为 1分钟、5分钟、15分钟前到现在的平均值

进程信息

119

```
Tasks: 176 total,   1 running, 173 sleeping,   0 stopped,   2 zombie
```

- 总进程个数
- 正在运行进程个数
- 睡眠状态的进程个数
- 被调试或者停止的进程数
  - ptrace
- 僵尸进程数

CPU利用率

120

```
Cpu(s): 35.1%us, 26.9%sy,  0.0%ni, 36.8%id,  1.2%wa,  0.0%hi,  0.0%si,  0.0%st
```

- 用户空间的代码占用CPU的百分比
- 内核空间占用百分比
- 用户空间内改变过优先级的进程占用百分比
- Idle进程占用的百分比
- 等待输入输出的CPU时间百分比
- %hi(Hardware IRQ), %si(Software Interrupts)
- %st(Steal time) 是当 hypervisor 服务另一个虚拟处理器的时候，虚拟 CPU 等待真实CPU 的时间的百分比
- 对于多CPU系统，是所有CPU的总占用率

物理内存信息

121

```
Mem:    765556k total,   701676k used,    63880k free,     4544k buffers
```

- 总内存数
- 使用的内存数
- 空闲内存数量
- 用作内核缓存区的内存数量

121

虚拟内存信息

122

```
Swap:   522236k total,   137212k used,   385024k free,   102268k cached
```

- 交换区总大小
- 已经使用的大小
- 空闲量
- 缓冲量
  - 已经换回到物理内存的页在交换区的“影子”，如果这些页需要再换出时不需要写交换区，条件是没有被修改过

122

strace

- trace system calls and signals

```
ge@gewubox:~/work/silk$ strace ./silk
execve("./silk", ["./silk"], [/* 37 vars */]) = 0
brk(0)                                  = 0x8a11000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7700000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63110, ...}) = 0
mmap2(NULL, 63110, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76f0000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p[\0\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=124663, ...}) = 0
mmap2(NULL, 107008, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb76d5000
mmap2(0xb76ec000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16) =
0xb76ec000
mmap2(0xb76ee000, 4608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x
b76ee000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/usr/lib/i386-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\0]\4\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=905712, ...}) = 0
mmap2(NULL, 935336, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75f0000
mprotect(0xb76c8000, 4096, PROT_NONE)   = 0
mmap2(0xb76c9000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xd8) =
 0xb76c9000
```

123

strace -f ./fork

```
clone(Process 3147 attached
child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xb760b968)
 = 3147
[pid  3146] getpid()                    = 3146
[pid  3146] write(1, "This is the main work cycle in p"..., 53This is the main work cycle in p
rocess with pid=3146
) = 53
[pid  3146] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
[pid  3146] rt_sigaction(SIGCHLD, NULL, {SIG_DFL, [], 0}, 8) = 0
[pid  3146] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid  3146] nanosleep({1, 0},  <unfinished ...>
[pid  3147] write(1, "This is the child work cycle in "..., 54This is the child work cycle in
process with pid=3147
) = 54
[pid  3147] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
[pid  3147] rt_sigaction(SIGCHLD, NULL, {SIG_DFL, [], 0}, 8) = 0
[pid  3147] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid  3147] nanosleep({1, 0},  <unfinished ...>
[pid  3146] <... nanosleep resumed> 0xbfa9a068) = 0
[pid  3146] write(1, "Closing main program...\n", 24Closing main program...
) = 24
[pid  3146] exit_group(0)               = ?
<... nanosleep resumed> 0xbfa9a068)     = 0
exit_group(0)                           = ?
Process 3147 detached
```

124

$ strace php 2>&1 | grep php.ini

```
open("/usr/local/bin/php.ini", O_RDONLY) = -1 ENOENT (No such file or directory)
open("/usr/local/lib/php.ini", O_RDONLY) = 4
lstat64("/usr/local/lib/php.ini", {st_mode=S_IFLNK|0777, st_size=27, ...}) = 0
readlink("/usr/local/lib/php.ini", "/usr/local/Zend/etc/php.ini", 4096) = 27
lstat64("/usr/local/Zend/etc/php.ini", {st_mode=S_IFREG|0664, st_size=40971, ...}) = 0
```

- 监视读文件情况
- $ strace -e open,access 2>&1 | grep your-filename
  - Look for an open() or access() syscall that fails

125

调优用途

- # strace -c >/dev/null ls
  - 创建并监视新进程
- # strace -c -p 11084
  - 附加到已经运行的进程

% time	seconds	usecs/call	calls	errors	syscall
74.57	0.050856	1	50393	62	getdents64
25.36	0.017294	1	26202	986	openat
0.06	0.000038	0	25231		close
0.02	0.000015	0	25230		fstat64
0.00	0.000000	0	11		read
0.00	0.000000	0	5814		write
0.00	0.000000	0	36	23	open
0.00	0.000000	0	1		execve
0.00	0.000000	0	9	9	access
0.00	0.000000	0	30		brk
0.00	0.000000	0	3	3	ioctl
0.00	0.000000	0	4		munmap
0.00	0.000000	0	1		uname
0.00	0.000000	0	9		mprotect
0.00	0.000000	0	2		mremap
0.00	0.000000	0	2		rt_sigaction
0.00	0.000000	0	1		rt_sigprocmask
0.00	0.000000	0	1		getrlimit
0.00	0.000000	0	29		mmap2

126

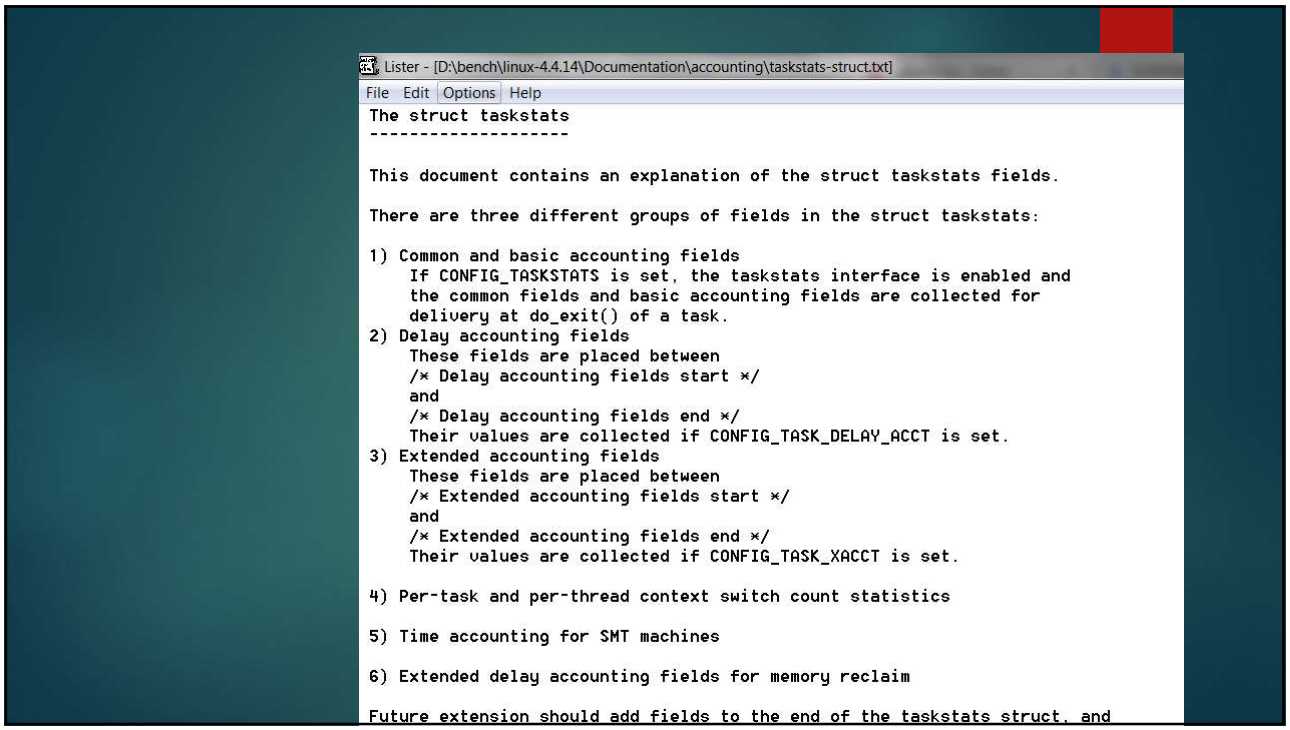
归纳

127

- 进程是线程的住所
- LINUX既使用task_struct来描述进程，又使用它描述线程
- Task_info和task_struct
- Linux支持多种方式创建进程和线程
  - Fork(): create a duplicate process
  - Exec*(): create a new process
  - Pthread_create(): create a POSIX thread
  - Clone(): Create a Linux native thread
- 观察进程信息: ps, pstree, top, strace

Q & A

张银奎


Lister - [D:\bench\linux-4.4.14\Documentation\accounting\taskstats-struct.txt]
File Edit Options Help
The struct taskstats

This document contains an explanation of the struct taskstats fields.
There are three different groups of fields in the struct taskstats:
1) Common and basic accounting fields
If CONFIG_TASKSTATS is set, the taskstats interface is enabled and
the common fields and basic accounting fields are collected for
delivery at do_exit() of a task.
2) Delay accounting fields
These fields are placed between
/* Delay accounting fields start */
and
/* Delay accounting fields end */
Their values are collected if CONFIG_TASK_DELAY_ACCT is set.
3) Extended accounting fields
These fields are placed between
/* Extended accounting fields start */
and
/* Extended accounting fields end */
Their values are collected if CONFIG_TASK_XACCT is set.
4) Per-task and per-thread context switch count statistics
5) Time accounting for SMT machines
6) Extended delay accounting fields for memory reclaim
Future extension should add fields to the end of the taskstats struct, and