Linux Threads

Kiran Divekar

Nevis Networks, Pune

kirandivekar@gmail.com

www.geocities.com/kirandivekar/

GEEP

GEEKS OF PUNE

Abstract

- What are threads? User space
- Threads Basics
- Creation and termination: thread_info
- User space thread development
- Threads Debugging
- Thread Experience
- Kernel Threads
- Latest Kernel Developments

User space threads

- Threads are separate contexts of execution within the same process.
- Fork(), vfork(), clone() create a process
- Threads share the same global memory space, but they have different stacks.
- <linux/sched.h> task_struct

User space threads

- Process creation: clone().
- Kernel : do_fork
- Threads creation: pthread_create.
- Kernel : do_fork

Thread_info

```
Struct thread_info {
    struct task_struct *task ;
    unsigned long flags ;

    exec_domain ;
    cpu ;
    prempt_count ;
    add_limit ;
    supervisor_stack ;
}
```

Copy On Write [COW]

- Linux threads follow COW technique
- Process descriptor is duplicated if written.

- CLONE_VM
- CLONE_FILES
- CLONE_FS
- CLONE_SIGHAND

- Note: Child execs first. Why ?
- Shared page tables : kernel 2.7 ???

Threads Exit

- User space : exit / pthread_cancel
- Kernel : do_exit().

- Recycling pids as per MAX value

- Remember : thread_info
- Threads are scheduled.
- Why 2 separate structures ?

Thread Development

- Pthreads
  - Pthread_create, pthread_attributes.
  - Create a detached thread
- Basic Linuxthreads
  - Why thread-safe, reentrant ?
  - Example: errno, strtok
- TLS
- NPTL

Thread Local Storage

- The purpose of TLS is to give each thread access to a region of memory which is *not* shared with all other threads. It is stored just below the stack address.
  - `__thread int i;`
  - `static __thread char *p;`
- When the addr-of operator is applied to a thread-local variable, it returns the address of the current thread's instance of that variable at run time.

- /lib/tls/libc.so
- Example : can;t find TLS link

Disadvantages

- The manager thread control: performance
- Issues with SMP systems
- Requirement of pthread_exit
- Context switching delay
- N+2 threads
- POSIX incompatible

- NPTL

Native POSIX Thread Library

- No manager thread
- Kernel scheduled
- POSIX synchronization usign futex

- Per process signal handling than per thread
- ABI support
- Backword compatibility using LD_ASSUME_KERNEL

- getconf GNU_LIBPTHREAD_VERSION

Kernel Requirements

- Ingo Molnar: Early 2003
- Support for an arbitrary number of thread-specific data areas
- Removes threads per process limit
- Clone system call extended to optimize creation and fascilitate termination of threads [detached thread]
- Futex wakeup on thread Id [pthread_join]
- Signals sent per thread. Fatal signals terminate process
- /proc contains only process information

Thread Debugging

Fork Behaviour

- Default :
  - `set follow-fork-mode parent`
- Child breakpoint gets a SIGTRAP
- Separate gdb to debug child

Pthread behaviour

- gdb 6 & above
- info threads
- thread apply [all / threadno] cmd

Thread experience

- Separate open system call
- Crash analysis.
  - Kernel support for thread backtrace
  - Critical section analysis
- Thread scheduling
  - Unlocking thread wakes up one thread
- Blocking behaviour
  - e.g. system call in an embedded system
- Avoid asynchronous thread cancellation
  - It may be in critical section

Thread experience

- Timing issues
  - Socket descriptor manipulation
- Any more ?

Kernel execution

- Kernel Context
  - Asynchronous: Interrupt handlers
- Process [User] Context
  - Synchronous: Response to system calls
- Kernel threads
  - Similar to user space daemons.

Kernel threads

ps –ef

ID	PID	PPID	STIME	TIME	CMD
root	1	0	22:36	00:00:00	init [3]
root	2	1	22:36	00:00	[ksoftirqd/0]
root	3	1	22:36	00:00:00	[events/0]
root	38	3	22:36	00:00:00	[pdflush]
root	39	3	22:36	00:00:00	[pdflush]
root	29	1	22:36	00:00:00	[khubd]
root	695	1	22:36	00:00	[kjournald]
root	3914	1	22:37	00:00:00	[nfsd]
root	3915	1	22:37	00:00:00	[nfsd]
root	4066	4015	22:59	00:00:00	ps –ef

Kernel Threads

- The [ksoftirqd/0] kernel thread is an aid to implement soft IRQs
- The events/n threads (n = processor number) help implement work queues
- The pdflush kernel thread flushes dirty pages from the page cache
- The khubd thread, part of the Linux USB core, monitors the machine's USB hub
- The nfsd thread, monitoring network filesystems

Why Kernel Threads?

Candidates for a kernel thread because:

- It's a background task, since it has to wait for asynchronous events.
- It needs access to kernel data structures, since the actual detection of events must be done by other parts of the kernel.
- It has to invoke a user-mode helper program, which is a time consuming operation.

- Ex: monitor receive buffers of a network
- create_kthread ???

Kernel Thread How?

- Kernel thread Life Cycle:
  - Daemonize
  - Make init as my parent
  - Wait for event using wait queue.

- Event Generation
  - Kernel data structure monitoring
  - If the health is unsatisfactory, (eg. Buffers below low watermark), wake up the queue.

- When event comes, check for signal SIGKILL
  - Start kthread operation
  - Set state to TASK_RUNNING

Kernel threads…

- User space interaction

  SYSCTL : system control operations.
  - /proc/sys/kernel/modprobe
  - /proc/sys/kernel/hotplug

  Reference:

  ksoftirqd : kernel/softirq.c
  pdflush : mm/pdflush.c
  khubd drivers : usb/core/hub.c

Latest Kernel Development

- Bottom Halves: Interrupt Deferred routine
- Tasklet. Deferred work processing
- Task queues: Fixed. TQ_INTERRUPT
- Work queues: Event Handling

- Syslets
- Threadlets

Syslets…

- Syslets are small, simple, lightweight programs (consisting of system-calls, 'atoms') that the kernel can execute autonomously (and, not the least, asynchronously), without having to exit back into user-space.

- Makes use of cachemiss technique.

- There are open issues.

Threadlets…

- "Threadlets" are basically the user-space equivalent of syslets: small functions of execution that the kernel attempts to execute without scheduling. If the threadlet blocks, the kernel creates a real thread from it, and execution continues in that thread.
- The 'head' context (the context that never blocks) returns to the original function that called the threadlet.

Happy Kernel Hacking !!!

THANK YOU

GEEP

GEEKS OF PUNE