TTY layer – here lies daemons

Greg Kroah-Hartman

gregkh@linuxfoundation.org
git.sr.ht/~gregkh/presentation-tty

Kernel Recipes

The TTY demystified

https://www.linusakesson.net/programming/tty/

“Good luck fixing the tty layer” – Thomas Gleixner

https://lwn.net/Articles/938236/

tty layer is why we are here

tty vs. line disciplines vs. serial port

tty vs. line disciplines vs. serial port

- We are going to ignore consoles, they are “magic”

tty

- Char device
  - userspace open/read/write/ioctl/close
- Assign a line discipline to a tty
- ioctls are line-discipline-specific

Line discipline

- Protocol to be talked on the tty
- ~20 different ones
  - Normal (n_tty), gps, slip, CAN, ham radio, ppp, videophone modem control, etc...
  - "internal serial port connection"
- Works for any tty connection / any tty device

“serial port” – tty driver

- “hardware” to look like a tty device
- serial port subsystem (uarts)
- USB-serial subsystem (fake and real uarts)
- ptys (pseudo-terminals)
- ISDN devices
- s390 devices
- ~40 different ones

ttyprintk.c

- Userspace write to show up in kernel log
- 200 lines of code
- One way “userspace → kernel”
- Good example codebase

struct tty_struct

- 1 kref
- 4 mutexes
  - One emulates old BKL
- 1 rw_semaphore
- 1 spinlock

struct tty_struct – cont.

- 2 wait queues
- 2 work structs

struct tty_struct – cont.

- 2 internal structs
  - Each have a spinlock
  - Properly padded for 64bit store on ALPHA

struct tty_struct – cont.

```
/* size: 656, cachelines: 11, members: 37 */
/* sum members: 654, holes: 1, sum holes: 2 */
/* forced alignments: 2 */
/* last cacheline: 16 bytes */
```

struct tty_driver

- 36 function callbacks

struct tty_port

- 1 spinlock
- 2 mutexes
- 1 kref
- 2 wait queues

struct uart_port

- 27 function callbacks
- 1 spinlock
- 1 global spinlock for any serial line change

struct usb_serial_driver

- 38 function callbacks

struct usb_serial_device

- 1 `struct tty_port`
- 1 spinlock
- 1 `struct device`

tty_write()

- Can be called in lots of odd ways
  - console magic
- Iterator fun!

```
static ssize_t tty_write(struct kiocb *, struct iov_iter *);
```

iterator_tty_write()

- Start of the real work
- `tty_write_lock()`

tty_write_lock()

- `mutex_trylock()`
- Failed? `mutex_lock_interruptable()`
- Failed? Return restart error

iterator_tty_write()

- Check buffer size
  - Too small – allocate more data!

    `kvmalloc()` / `kvfree()`

    - Non-deterministic mess #1

iterator_tty_write()

- `copy_from_iter()`
- Pass data to line discipline

iterator_tty_write()

```
/* FIXME! Have Al check this! */
 if (ret != size)
         iov_iter_revert(from, size-ret);
```

iterator_tty_write()

```
/* FIXME! Have Al check this! */
 if (ret != size)
         iov_iter_revert(from, size-ret);
…
 if (signal_pending(current))
         break;
 cond_resched();
```

iterator_tty_write()

- Keep looping until add data send to lower layer

iterator_tty_write()

- `tty_update_time()`

tty_update_time()

- `ktime_get_real_seconds()`
- Grab a spinlock
- Iterate over all open file descriptors for the tty
- Change the timestamp if within 8 seconds
- Release spinlock

iterator_tty_write()

- tty_write_unlock()

n_tty write

- Loop over all data given to us:
- `down_read()`
- Process pending echo chars (how many?)

n_tty write

- Loop over all data given to us:
- `down_read()`
- Process pending echo chars (how many?)
  - Non-deterministic mess #2!

n_tty write

- Loop over all data given to us:
- `down_read()`
- Process pending echo chars (how many?)
- Add wait queue
- Pending signals?
  - abort

n_tty write

- Process unknown amount of output blocks

n_tty write

- Process unknown amount of output blocks
  - Non deterministic mess #3

n_tty write

- Process unknown amount of output blocks
- `mutex_lock()`
  - tty driver write call
- `mutex_unlock()`
- `up_read()`

n_tty write

- Wake up waitqueue
- `down_read()`
- Back to top of loop

n_tty write

- Wake up waitqueue
- `down_read()`
- Back to top of loop if more data to send
- Remove wait queue
- `up_read()`

tty driver write

- Serial port write

Serial port write

- `uart_port_lock()`
- `memcpy()` data to local buffer
  - Only `PAGE_SIZE` big
- UART send

UART send

- `pm_runtime_get()`
- 8250 send

8250 send

- Tweak pm flags again
- Read LSR from hardware

8250 send

- Tweak pm flags again
- Read LSR from hardware
  - Non-deterministic mess #4

8250 send

- Tweak pm flags again
- Read LSR from hardware
- Write bytes to uart (one at a time? DMA?)

8250 send

- Tweak pm flags again
- Read LSR from hardware
- Write bytes to uart (one at a time? DMA?)
  - Non-deterministic mess #X

8250 send

- Tweak pm flags again
- Read LSR from hardware
- Write bytes to uart (one at a time? DMA?)
- UART port unlock

Recieve data from hardware

- `tty_insert_flip_char()` / `tty_insert_flip_string()`
- `tty_flip_buffer_push()`

tty_insert_flip_*()

- Have enough memory?
  - No, allocate more (can not fail, wait forever)
  - Max buffer ~1Mb
  - No driver checks this, if no readers are there, data will drop on the floor
- Loop to copy all data to buffer
  - TTY_BUFFER_PAGE

tty_flip_buffer_push()

- smb_store_release()
- Wake up workqueue

Sometime later

tty_flip_buffer_push() workqueue

- Buffer lock (one per port)
- atomic_read()
- smb_load_acquire() X 2
- Line discipline receive_buff()
- cond_resched() if needed
- Loop until all data flushed
- Buffer unlock

n_tty receive_buff()

- down_read()
- smp_load_aquire()
- Copy data into ldisc buffer
- smb_load_release()
- Wake up ldisc read waitqueue
- up_read()

`tty_read()`

- No locks!
- 64 bytes on the stack
- Line discipline read()
- `copy_to_user_buffer()`
- `memset()` stack buffer to `0`

n_tty read()

- `down_read()`
- `smb_load_aquire()`
- `memcpy()` from flip buffer to stack buffer
- Adjust pointers
- `tty_audit_add_data()`
- `up_read()` sometime later...

tty_audit_add_data()

- Allocate buffer

`tty_audit_add_data()`

- Allocate buffer
  - Can sleep!

tty_audit_add_data()

- Allocate buffer
- mutex_lock()
- memcpy()
- Write to audit log

`tty_audit_add_data()`

- Allocate buffer
- `mutex_lock()`
- `memcpy()`
- Write to audit log
  - We don’t have enough time....

tty_audit_add_data()

- Allocate buffer
- mutex_lock()
- memcpy()
- Write to audit log
- mutex_unlock()

tty layer – the bad

- It’s complicated
- Too flexible
- Too many entry/exit points
- Lots of opportunities to sleep
- Non-deterministic in so many places
- UARTS are complex and dumb

tty layer – the good

- It is fast
- It is flexible
- It supports all hardware
- It is why Linux has succeeded

tty layer – how to fix it

- printk changes to add “simple” console callbacks

  `https://lwn.net/Articles/909980/`

tty layer – how to fix it

- printk changes to add “simple” console callbacks

  `https://lwn.net/Articles/909980/`

- Call it from non-realtime userspace tasks

tty layer – how to fix it

- printk changes to add “simple” console callbacks

  `https://lwn.net/Articles/909980/`

- Call it from non-realtime userspace tasks
- Don’t enable auditing!

tty layer – how to fix it

- printk changes to add “simple” console callbacks

  https://lwn.net/Articles/909980/

- Call it from non-realtime userspace tasks
- Don’t enable auditing!
- Don’t use it!

tty layer – how to fix it

- printk changes to add “simple” console callbacks

  https://lwn.net/Articles/909980/
- Call it from non-realtime userspace tasks
- Don’t enable auditing!
- Don’t use it!
  - raw_uart.c
    - read/write ringbuffer, no line control changes

tty layer – how to fix it

- patches welcome!

tty layer – leave it alone

git.sr.ht/~gregkh/presentation-tty