Embedded Linux Device Drivers

Agenda

- Sequence Files
- Beaglebone Booting
- Yocto Project

Kernel debugging

- Different methods
  - Debugging by printing -- printk()
  - Debugging by querying -- /proc/seq_file, ioctl()
  - Debugging by watching -- strace command
  - Debugging by kernel OOPs and Hangs -- OOPS messages
  - Debugging by external tools -- Kdb, KGdb, ...

Proc File System

- "procfs" is a pseudo file system i.e. no persistent storage is associated with the file system.
- It is considered as "window" to the kernel i.e. it is used to monior activities and data structures inside the kernel.
- Typically they are used to make kernel space information available to user space application in most simplified way. The device drivers/kernel components use them as debugging feature.
- Few important proc files:
  - /proc/cpuinfo
  - /proc/meminfo, /proc/buddyinfo, /proc/slabinfo
  - /proc/interrupts
- Some of the proc files are also used to set kernel config.
  - /proc/sys/kernel/sched_child_runs_first
  - /proc/sys/vm/overcommit_memory

- In older kernels, proc files are implemented as kernel buffer; however programmer was responsible to keep track of page sized buffers. In other words, the output of proc should be restricted within a single page (4 kb) or programmer should keep track of page buffers in order to generate larger output.
- Newer kernel versions, has simplified access to the proc files using concept of "sequence files". The sequence files present a particular sequence of data to the user. This mechanism frees programmer from tracking page buffer size. Now programmer is only responsible for iterating some data structure and displaying its contents.
- Sequence files are represented with "struct seq_file". Any data written into this file (using helper functions seq_printf(), seq_putc() and seq_puts()) will be visible to the end user as proc file contents.

Implementing Sequence files

- step 1: Create entry in proc file system using kernel api proc_create(). This takes filename as an arg & also file_operations/proc_ops struct as another arg. The file_operations/proc_ops struct stores predefined kernel functions i.e. seq_read(), seq_lseek() & seq_release(). However, open operation is replaced by an user-defined function.
- step 2: The user-defined open() operation, simply calls pre-defined seq_open() with sequence file operations struct seq_operations
- step 3. Programmer should define these seq_operations i.e. .start, .stop, .next, .show.
- step 4: Implement .start operation:
  - Called when seq file is opened.
  - args: struct seq_file *s, loff_t *pos
    - 1st arg: is rarely needed into this operation.
    - 2nd arg: "pos" is programmer defined position (not byte based position). For just opened file, it is 0.
  - Programmer is expected to return the first object to be displayed from this method (return type is void*).
- step 5: Implement .next operation:
  - Called when progressing seq file reading.
  - args: struct seq_file *s, void *v, loff_t *pos
    - 1st arg: is rarely needed into this operation.
    - 2nd arg: current displayed object. Programmer can use it to go to next object in list.
    - 3rd arg: "pos" must be incremenet in (programmer-defined way).
  - Programmer is expected to return the object to be displayed from this methd. If end of data list is reached, return NULL.
- step 6. Implement .stop operation:
  - Called when sequence ends i.e. when next() operation returns NULL.
  - It is used to release resources, if any resource is allocated in start operation.

- Immediately after .stop(), once again .start operation is invoked (with current pos i.e. pos!=0). In that case start operation is expected to return NULL.
- step 7. Implement .show operation:
  - Called to display current object after non-NULL return from start/next operation.
  - args: struct seq_file *s, void *v
    - arg1: seq_file -> to display, contents are added in this file using seq_printf(), etc.
    - arg2: object returned from start/next operation. Use this object to get contents to be displayed.
- step 8. At the end (typically in module_exit()), remove proc file entry using kernel api remove_proc_entry().

Beaglebone Booting

- Refer Yocto Docs