MutekH startup and initializations

This page describes how the various parts of the kernel are initialized when the operating system starts.

Modular startup process

The MutekH statup process consists in calling many intialization functions for all components of the system. Depending on the current build configuration, different modules and features are enabled at compile time. This make the set of relevant initialization functions change with the current configuration.

The initialization order of the various parts of the kernel is important because some components do depend on other components which need to be initialized first. The proper order may also be different depending on some configuration parameters like the target architecture.

Rather than defining the initialization order directly, the MutekH build system generates a list of intialization function calls sorted in the right order based on some expressed ordering constraints. This allows the developper of a software component to insert its intialization function calls at the right time relative to intialization of other components.

The kernel initialization stages are arranged in a hierarchical manner. Internal nodes of this intialization tree define the main steps of startup process whereas leaf nodes actually define the function calls which must take place. Ordering constraints are expressed along with the hierarchy using intialization tokens in the build system configuration files.

Initialization functions

The whole startup process is split in two main groups of intialization tokens:

- Intializations which take place on the bootstrap processor while other processors are waiting.
- Intializations which take place at the same time on all processors.

The bootstrap processor usally starts execution in assembly code in the `cpu_boot` function. It then jumps to the `mutekh_startup` function which contains function calls generated by the build system for the `INIT_MUTEKH_STARTUP` initialization token.

On multiprocessors platforms other processors enter the `mutekh_startup_smp` function when instructed to do so by the bootstrap processor. This function contains function calls corresponding to the `INIT_SMP` token which is actually the same as the second half of `INIT_MUTEKH_STARTUP`.

Initialization tokens which are part of `INIT_SMP` group can call the `mutekh_startup_smp_barrier` function to ensure that all processors have reached this same point in code. If an intialization must be performed on a single processor in this intialization group, the result of the `cpu_isbootstrap` function must be tested.

Startup related functions are declared in the mutek/startup.h header file.

Main intialization steps

The first part of the startup process executed on the boostrap processor only includes the following main steps (when relevant):

- initialization of memory sections `.bss` and `.data`.

- initialization of the early output console.
- initialization memory and page allocators.
- initialization and enumeration of devices present in the platform, including processors.
- initialization of the associated device drivers.

The second part of the startup process which is executed by all processors includes:

- initialization of each processor registers
- initialization of the scheduler
- initialization of other libraries and software components
- initialization of the application
- enter the scheduler loop

Below is an sample herarchical view of the main intialization tokens as output by the `make listinit` command with a multiprocessor build configuration (as described on the BuildSystem page).

```
* INIT_MUTEKH_STARTUP
   * INIT_BOOTSTRAP
       * INIT_MEMORY
           * INIT_MEMORY_SECTIONS
               * ....
           * INIT_MUTEK_EARLY_CONSOLE
               * ....
           * INIT_MUTEK_MEMALLOC
               * ....
       * INIT_DEVICE
           * INIT_DEVICE_TREE
           * INIT_DEVICE_ENUM
               * ....
           * INIT_DEVICE_DRIVERS
       * INIT_SMP_STARTUP_BARRIER
       * INIT_START_CPUS
           * ....
   * INIT_SMP
       * INIT_CPUS
           * INIT_DEVICE_CPU_REGS
           * INIT_MUTEK_FAULT_HANDLER
       * INIT_MUTEK_SCHEDULER_INIT
       * INIT_MUTEK_CONSOLE
       * INIT_LIBRARIES
           * ....
       * INIT_APPLICATION
       * INIT_MUTEK_SCHEDULER_START
```

The `INIT_APPLICATION` step calls the `app_start` function which must be defined in the application source code.

Here is an example intialization function calls order retained by the build system:

```
INIT_MUTEKH_STARTUP (init):
    INIT_SOCLIB_BSS                    soclib_bss_section_init()
    INIT_SOCLIB_DATA                   soclib_data_section_init()
    INIT_SOCLIB_EARLY_CONSOLE          soclib_early_console_init()
    INIT_SOCLIB_MEM_ALLOC              soclib_mem_init()
    INIT_DEVICE_TREE                   device_tree_init()
    INIT_SOCLIB_FDT                    soclib_fdt_init()
    INIT_DEVICE_DRIVERS                libdevice_drivers_init()
    INIT_SMP_STARTUP_BARRIER           mutek_startup_barrier_init()
    INIT_SOCLIB_START_CPUS             soclib_start_cpus()
    INIT_SOCLIB_SMP_WAIT_BOOTSTRAP     soclib_smp_wait_bootstrap()
    INIT_DEVICE_CPU_REGS               libdevice_cpu_regs_initsmp()
```

`INIT_MUTEK_FAULT_HANDLER`	`mutek_fault_initsmp()`
`INIT_MUTEK_SCHEDULER_INIT`	`mutek_scheduler_initsmp()`
`INIT_MUTEK_CONSOLE`	`mutek_console_initsmp()`
`INIT_APPLICATION`	`mutek_app_initsmp()`
`INIT_MUTEK_SCHEDULER_START`	`mutek_scheduler_start()`