Simple Operating System

Tong Wu - z3417604
Lewis Lee - z3415068

October 15, 2014

1 Clock Driver

1.1 High Level Description

The clock driver is implemented using the EPIT1 and EPIT2 timers and implements the following functions;

- **int** start_timer(**seL4_CPtr** interrupt_ep1, **seL4_CPtr** interrupt_ep2)
  This is called to initialize both EPIT1 and EPIT2 timers. It takes in two **seL4_CPtr**s and sets them as interrupt endpoints for EPIT1 and EPIT2 respectively. If the timers are already running (ie start_timer had been called before), then they will be stopped and restarted. **CLOCK_R_OK** will be returned on success, otherwise **CLOCK_R_FAIL** will be returned.

- **uint32_t** register_timer(**uint64_t** delay, **timer_callback_t** callback, **void** *data)
  This is called to set a timer which will expire after a given delay while calling a given callback function. If timers are not initialized, **CLOCK_R_UINT** will be returned, **CLOCK_R_FAIL** will be returned if the maximum number of timers is reached (this is currently 50, this can be changed in clock.h). **CLOCK_R_OK** will be returned on success.

- **int** remove_timer(**uint32_t** id)
  This is an internal function which when called, removes the timer from the priority queue.

- **int** EPIT1_interrupt(**void**)
  This function should normally be called when there is a timer interrupt for **EPIT1** (used for heartbeats). When called, it will update the current time and reset the underflow flag (so the timer will continue to interrupt). It then acknowledges the interrupt handler. If **EPIT1** is not initialized, **CLOCK_R_UINT** will be returned, otherwise **CLOCK_R_OK** will be returned.

- **int** EPIT2_interrupt(**void**)
  This function should normally be called when there is a timer interrupt for **EPIT2**

(used for callback timers). When called, it will call the callback function for the current timer in queue, then removes the timer from queue using **remove_timer()**. If **EPIT2** is not initialized, **CLOCK_R_UINT** will be returned, otherwise **CLOCK_R_OK** will be returned.

- **timestamp_t** time_stamp(**void**)
  This function prints out the current time after **start_timer()** was called. It checks the underflow flag in case it missed an interrupt and adds additional time if necessary. Then it adds the time elapsed since the last interrupt to the current time and returns the current time in microseconds (64-bit unsigned integer). If **EPIT1** is not initialized, **CLOCK_R_UINT** will be returned.

- **int** stop_timer(**void**)
  This function can be called to stop all current callback timers and to do a soft reset on both **EPIT1** and **EPIT2**. **CLOCK_R_UINT** will be returned if the timers are not initialized, otherwise **CLOCK_R_OK** will be returned.