Assignment 2

- you can obtain the assignment 2 files from here ⟨`ftp://floppsie.comp.glam.ac.uk/pub/notes/osi/timers.tar`⟩.
- they are extracted via:
- ```
  tar xvf timers.tar
  ```
- you should then change directory into timers via:
- ```
  cd timers
  ```
- to build the assignment type:
- ```
  make c
  make
  ```
- you only ever need to type `make c` once

Assignment 2

- assignment is to implement a timer facility
- you must implement the following functions:
  - `Sleep`
  - `ArmEvent`
  - `Timer`

Example of using the TimerHandler

- here is a simple example of use for the TimerHandler
- two processes are created and sleep for 10 and 60 seconds, then print a message
- the process sleeping for 60 seconds can be woken up before the 60 second have expired
- ```
  void TenSeconds (void)
  {
    OnOrOff OldInts = SYSTEM_TurnInterrupts(On);

    while (1) {
      TimerHandler_Sleep(
          10*TimerHandler_TicksPerSecond);
      Debug_DebugString("..10..");
    }
  }
  ```


SixtySeconds

- ```
  static Event *timeout;

  void SixtySeconds (void)
  {
    OnOrOff OldInts = SYSTEM_TurnInterrupts(On);

    while (1) {
      timeout = TimerHandler_ArmEvent(
                  60*TicksPerSecond);
      if (TimerHandler_WaitOn(timeout))
        Debug_DebugString("..been cancelled..");
      else
        Debug_DebugString("..60 seconds alarm..");
    }
  }
  ```
  ```

Cancelling an event

- is achieved by another process executing this:

```
if (TimerHandler_Cancel(timeout))
  Debug_DebugString("have cancelled timeout");
else
  Debug_DebugString("not cancelled - expired");
```


TimerHandler module

- provides a simple set of timer functions microkernel
- provides the Executive with a basic round robin scheduler
- sets a timer device to issue an interrupt 100 times per second as defined by this constant in TimerHandler.h

```
#define TimerHandler_TicksPerSecond  100
```


TimerHandler

- exports an opaque or hidden type, Event, upon which all timer actions are held

```
/*
   GetTicks - returns the number of ticks
              since boottime.
*/

EXTERN int TimerHandler_GetTicks(void);
```

```
/*
   Sleep - suspends the current process for
           a time, t. The time is measured
           in ticks.
*/

EXTERN void TimerHandler_Sleep(int t);
```


ArmEvent

- initializes an event, e, to occur at time, t
  - the time, t, is measured in ticks
  - the event is NOT placed onto the event queue

```
EXTERN Event *TimerHandler_ArmEvent(int t);
```

WaitOn

- places event, e, onto the event queue and then the calling process suspends
  - it is resumed up by either the event expiring or the event, e, being cancelled
  - TRUE is returned if the event was cancelled
  - FALSE is returned if the event expires

```
EXTERN int TimerHandler_WaitOn(Event *e);
```


Cancel

- cancels the event, e, on the event queue and makes the appropriate process runnable again
  - TRUE is returned if the event was cancelled
  - FALSE is returned is the event was not found or no process was waiting on this event.

```
EXTERN int TimerHandler_Cancel(Event *e);
```

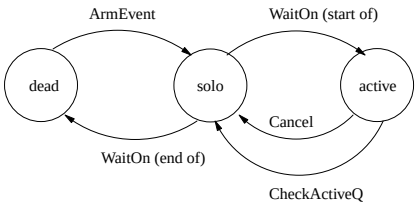

ReArmEvent

- removes an event, e, from the event queue.
  - a new time is given to this event and it is then re-inserted onto the event queue in the correct place
  - TRUE is returned if this occurred FALSE is returned if the event was not found

```
EXTERN int
   TimerHandler_ReArmEvent(Event *e, int t);
```


TimerHandler.c

- state transition diagram for the events within this module can be described as follows:



KeyboardLEDs Module

- used to switch
  - caps lock, scroll lock and num lock LEDs on and off
- this code turns the scroll LED on
- ```
  KeyBoardLEDs_SwitchScroll(TRUE);
  ```
- and this code turns the scroll LED off
- ```
  KeyBoardLEDs_SwitchScroll(FALSE);
  ```