Single thread Scheduler
All processes called once each sample

```c
void main(void) {
    init_routines();
    done = 0;
    while (!done) {
        perform_process1(); // Highest priority process
        perform_process2();
        perform_process3(); // Lowest priority
        wait_for_sample_period(); // waste time here waiting for sample period to expire
    }
}
```

What if the three processes have different sample rates? With a single thread scheduler the following would work but all sample rates would have to be a multiple of the fastest rate.

```c
void main(void) {
    init_routines();
    done = 0;
    samplecnt = 0
    while (!done) {
        perform_process1(); // Call every sample
        if ((samplecnt%5)==0) perform_process2(); // Call every 5 samples
        if ((samplecnt%12)==0) perform_process3(); // Call every 12 samples
        wait_for_sample_period(); // waste time here waiting for sample period to expire
        samplecnt = samplecnt+1;
    }
}
```

Here the problem becomes that all processes (in this case all three) need to complete in the time of one sample period. For example at samplecnt = 60 all three processes are called during a single sample period. What if process 3 takes longer than a single sample rate?
int samplecnt = 0 // Global integer

void main(void) {
    init_routines();
done = 0;
while (!done) {
if ((samplecnt%12)==0) {
    //Call every 12 samples
    perform_process3();
}
}
}

The while loop inside main() now becomes the low priority processing loop. Also called the “background” loop. Process 1 and Process 2 have the highest priority. When either timer 0 or timer 1 counts down to 0, the DSP’s hardware automatically stops the current code running in the background loop and jumps to the function specified in the Interrupt Vector Table, in this case timer0_isr() or timer1_isr(). When the DSP is done running the instructions in the corresponding interrupt service routine (ISR), the DSP’s hardware automatically returns and continues processing where it left off in the background loop.

If timer 0 and timer 1 timeout at exactly the same time, timer 0 has the highest priority so its code will run first to completion and then timer 1’s code will be executed. If a timer interrupt occurs while the other timer’s interrupt service routine code is running, the running code continues to completion and then the other timer’s code is executed.
Process #4
Occurs 0.001
Duration 0.00015

Process #3
Occurs 0.002
Duration 0.00025

Process #2
Occurs 0.004
Duration 0.001

Process #1
Occurs 0.007
Duration 0.002

Idle Loop
Example Time Load Graph

Units in milliseconds

Process #2 - Highest Priority - Occurs every 2ms - Duration 0.5ms
Process #1 - Occurs every 5ms - Duration 2ms
OMAPL138 SOM from LogicPD

- 300Mhz TMS320C6748 Floating Point DSP
- 300Mhz ARM9 General Purpose Processor.
- 128Mb External RAM/8Mb Flash
- EMAC (Ethernet)
- UART/SPI/I2C/McBSP
- SD Card Interface
- CMOS Camera Interface
SYS/BIOS

SYS/BIOS Startup Code

Main()

DSP/BIOS Initialization Code

HARDWARE Interrupts

1. HWI / Timer
   - Decreasing Priority

SOFTWARE Interrupts

2. SWI
   - Clock
   - Decreasing Priority

Task Manager

3. TSK
   - SEM
   - QUE / MBX

Lowest Priority Task

IDLE Loop Functions

IDLE

TASK 1

TASK 2

IDLE

Functions
### TSK Level

Arrows in the TSK and between the TSK and HWI Levels indicate direction of multi-thread communication.

User Defined Receive TSK waits for a new character by suspending (or blocking) itself until the semaphore, SEM_UART1RecChar_rdy is set active by HWI 4’s function. The new character is then read from the Queue, UART1RecCharQueue. After receiving this character, the task loops back to the beginning of its code and again blocks itself to wait for the next character to be sent.

Semaphore: SEM_UART1RecChar_rdy
Queue: UART1RecCharQueue

### HWI Level

**PIE Interrupt 9.1**
Function: RXAINT_recv_ready
Receives an interrupt when there is a new character received.

**PIE Interrupt 9.2**
Function: TXAINT_data_sent
SCIA set in UART mode send one character at a time until all characters in TX buffer have been transmitted. Then Post SEM_SendStrmsg_done

Global char Array: txbuffer
Semaphore: SEM_SendStrmsg_done

Read SCI registers

Write to SCI registers

### Hardware Level

Arrows in the Hardware Level indicate actual signals and their Direction.

DSP’s SCIA

Serial Port

TX

RX

Wireless Modem or other UART Devices

RS-232 standard serial port. (The standard serial port on the back of your PC.)

User Created Clocks, SWIs or TSKs call for example SendWireless(…) to send a message to the PC. SendWireless, places the message in the Queue, SendStrmsgQueue, and the Activates the Semaphore, SEM_SendStrmsg_rdy.

Semaphore: SEM_SendStrmsg_rdy
Queue: SendStrmsgQueue

### SYS/BIOS Example

User Created Clocks, SWIs or TSKs call for example SendWireless(…) to send a message to the PC. SendWireless, places the message in the Queue, SendStrmsgQueue, and the Activates the Semaphore, SEM_SendStrmsg_rdy.