Operating systems and concurrency B04

David Kendall

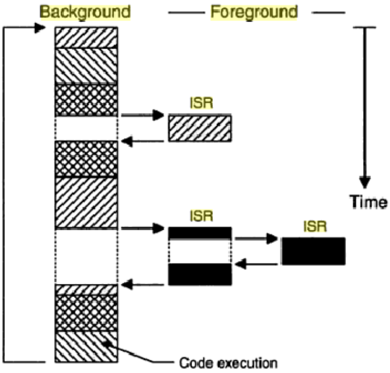
Northumbria University

Introduction

- Multi-tasking operating system services
- μC/OS-II (uC/OS-II)
- Task management
- Delay

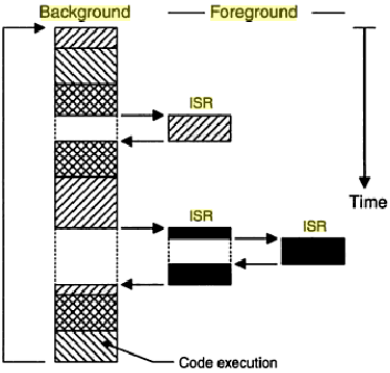
Foreground/background tasks

- Simple multitasking
- Super loop calls functions for computation (background)
- Interrupt service routines (ISRs) handle asynchronous events (foreground)



Foreground/background tasks

- Simple multitasking
- Super loop calls functions for computation (background)
- Interrupt service routines (ISRs) handle asynchronous events (foreground)



Problem

Model with only one computation task is not flexible enough

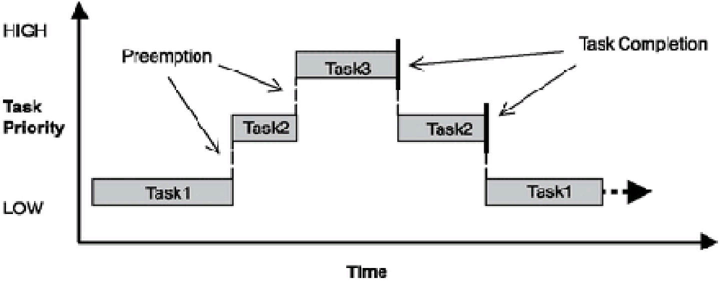
Multitasking

- Easier to structure the application as a set of concurrent tasks rather than as a single program or as foreground/background tasks: each task is responsible for some well-defined part of the system's overall behaviour
- But only the illusion of concurrency - the OS switches quickly between tasks, executing some instructions from one task before moving on to another task
- switching from one text to another requires a context switch
- deciding which task to switch to requires a scheduling algorithm

Scheduling

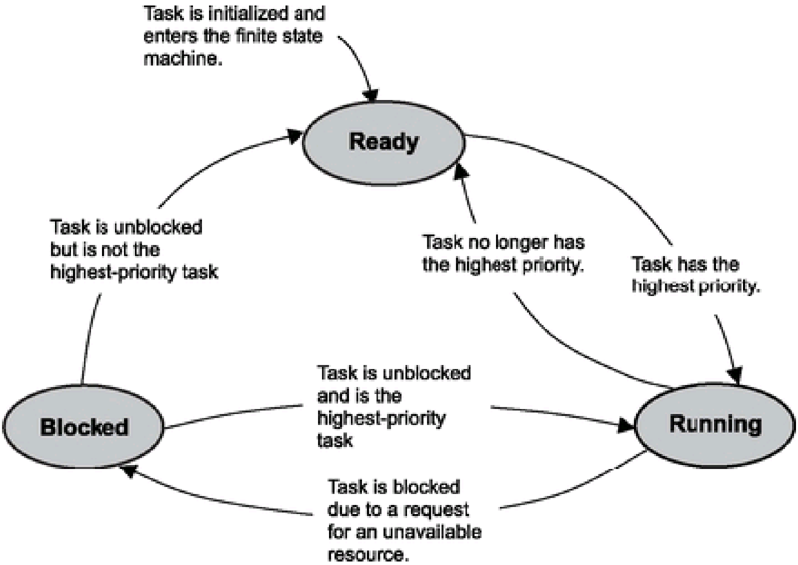
- Deciding when one task should stop executing and which one should begin next is a scheduling problem
- Two main approaches to scheduling:
  - Preemptive scheduling
    - Task is forced to yield the CPU
    - Round robin
    - Priority-based
  - Non-preemptive (cooperative) scheduling
    - Task voluntarily yields the CPU and signals the next task to begin

Fixed-priority preemptive scheduling



- We focus on fixed-priority premptive scheduling

A task and its execution states


Task is initialized and enters the finite state machine.
Ready
Task is unblocked but is not the highest-priority task
Task no longer has the highest priority.
Task has the highest priority.
Blocked
Task is unblocked and is the highest-priority task
Running
Task is blocked due to a request for an unavailable resource.

uC/OS-II: A small operating system

- Main features:
  - Multi-tasking
  - Preemptive
- Other features:
  - Predictable
  - Robust and reliable
  - Standards-compliant
  - Portable
  - Scalable
  - Source code available

uC/OS-II Services

- Task management
- Delay management

uC/OS-II Services

- Task management
- Delay management
- Semaphores
- Mutual exclusion semaphores
- Event flags
- Message mailboxes
- Message queues
- Memory management
- Timers
- Miscellaneous

uC/OS-II Services

- Task management
- Delay management
- Semaphores
- Mutual exclusion semaphores
- Event flags
- Message mailboxes
- Message queues
- Memory management
- Timers
- Miscellaneous

See uC/OS-II Quick Reference

Tasks behaviour

- The behaviour of a task is defined by a C function that:
  1. never terminates
  2. blocks repeatedly

Example of task behaviour definition

```
static void appTaskConnectLed(void *pdata) {
  while (true) {
    OSTimeDlyHMSM(0,0,0,500);
    ledToggle(USB_CONNECT_LED);
  }
}
```

Tasks: other requirements

Priority

- Used for fixed-priority pre-emptive scheduling
- a number between 0 and `OS_LOWEST_PRIO`
- low number ⇒ high priority
- high number ⇒ low priority
- OS reserves priorities 0 to 3 and `OS_LOWEST_PRIO` - 3 to `OS_LOWEST_PRIO`
- Advice: define an enumeration of task priority constants, starting at priority level 4.
- Example

```
enum {
  APP_TASK_BUTTONS_LED = 4,
  APP_TASK_LINK_LED_PRIO,
  APP_TASK_CONNECT_LED_PRIO
};
```

Tasks: other requirements

Stack

- Each task needs its own data area (stack) for storing
  - context
  - local variables
- Example stack definition

```
enum {APP_TASK_CONNECT_STK_SIZE = 256};
static OS_STK appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE];
```

Tasks: other requirements

Stack

- Each task needs its own data area (stack) for storing
  - context
  - local variables
- Example stack definition

```
enum {APP_TASK_CONNECT_STK_SIZE = 256};
static OS_STK appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE];
```

User data

- Optionally tasks can be given access to user data when they are created
- We will not use this feature in this module
- Advice: always specify this as `(void *)0` when creating a task

Task creation

- A task is created using the OS function

```
INT8U OSTaskCreate(
          void (*task)(void *pdata), /* function for the task   */
          void *pdata,               /* user data for function  */
          OS_STK *ptos,              /* pointer to top of stack */
          INT8U priority             /* task priority           */
);
```

Task creation

- A task is created using the OS function

```
INT8U OSTaskCreate(
        void (*task)(void *pdata),  /* function for the task   */
        void *pdata,                /* user data for function  */
        OS_STK *ptos,               /* pointer to top of stack */
        INT8U priority              /* task priority           */
);
```

Example

```
enum {APP_TASK_CONNECT_PRIO = 4};
enum {APP_TASK_CONNECT_STK_SIZE = 256};

static OS_STK appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE];

OSTaskCreate(appTaskConnectLed,
             (void *)0,
             (OS_STK *)&appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE - 1],
             APP_TASK_CONNECT_PRIO);
```

Task delay

- Often, a task will block itself by explicitly asking the OS to delay it for some period of time
- `void OSTimeDly(INT16U ticks);`
- Causes a context switch if `ticks` is between 1 and 65535
- If `ticks` is 0, `OSTimeDly()` returns immediately to caller
- On context switch uC/OS-II executes the next highest priority task
- Task that called `OSTimeDly()` will be made ready to run when the specified number of ticks elapses - actually runs when it becomes the highest priority ready task
- Resolution of the delay is between 0 and 1 tick
- Another task can cancel the delay by calling `OSTimeDlyResume()`

Task delay

- `OSTimeDly()` specifies delay in terms of a number of ticks
- Use `OSTimeDlyHMSM()` to specify delay in terms of Hours, Minutes, Seconds and Milliseconds
- Otherwise `OSTimeDlyHMSM()` behaves as `OSTimeDly()`

Complete example

```
#include <stdbool.h>
#include <ucos_ii.h>
#include <osutils.h>
#include <bsp.h>
#include <leds.h>

/*
***************************************************************************************
*                              APPLICATION TASK PRIORITIES
***************************************************************************************
*/

enum {
  APP_TASK_LINK_PRIO = 4,
  APP_TASK_CONNECT_PRIO
};

/*
***************************************************************************************
*                              APPLICATION TASK STACKS
***************************************************************************************
*/

enum {
  APP_TASK_LINK_STK_SIZE    = 256,
  APP_TASK_CONNECT_STK_SIZE = 256
};

static OS_STK appTaskLinkStk[APP_TASK_LINK_STK_SIZE];
static OS_STK appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE];
```

Complete example

```
/*
****************************************************************************************
*                               APPLICATION FUNCTION PROTOTYPES
****************************************************************************************
*/

static void appTaskLinkLed(void *pdata);
static void appTaskConnectLed(void *pdata);

/*
****************************************************************************************
*                               GLOBAL FUNCTION DEFINITIONS
****************************************************************************************
*/
int main() {

  /* Initialise the board support package and the OS */
  bspInit();
  OSInit();

  /* Create the tasks */
  OSTaskCreate(appTaskLinkLed,
               (void *)0,
               (OS_STK *)&appTaskLinkStk[APP_TASK_LINK_STK_SIZE - 1],
               APP_TASK_LINK_PRIO);

  OSTaskCreate(appTaskConnectLed,
               (void *)0,
               (OS_STK *)&appTaskConnectStk[APP_TASK_CONNECT_STK_SIZE - 1],
               APP_TASK_CONNECT_PRIO);
```

Complete example

```
  /* Start the OS */
  OSStart();

  /* Should never arrive here */
  return 0;
}
/*
***************************************************************************************
*                              APPLICATION TASK DEFINITIONS
***************************************************************************************
*/
static void appTaskLinkLed(void *pdata) {
  /* Start the OS ticker -- must be done in the highest priority task */
  osStartTick();

  /* Task main loop */
  while (true) {
    ledToggle(USB_LINK_LED);
    OSTimeDlyHMSM(0,0,0,500);
  }
}

static void appTaskConnectLed(void *pdata) {
  while (true) {
    OSTimeDlyHMSM(0,0,0,500);
    ledToggle(USB_CONNECT_LED);
  }
}
```

Acknowledgements

- Qing Li and Caroline Yao, Real-time concepts for embedded systems, CMP, 2003
- Jean Labrosse, MicroC/OS-II: The Real-Time Kernel, CMP, 2002