Ollscoil na hÉireann, Gaillimh
National University of Ireland, Galway

GX 695

Semester II Examinations, 2002/2003
Front Page Template

Exam Code(s)	4BP121
Exam(s)	4th Electronic and Computer Engineering
Module Code(s)	EE422
Module(s)	Embedded Systems Software
Paper No.	2
Repeat Paper	Special Paper
External Examiner(s)	Professor S. McLaughlin
Internal Examiner(s)	Professor D.J. Wilcox
	Dr. P. Corcoran
Instructions:	Section A: Answer **all** questions Section B: Answer **three** questions
Duration	2hrs
No. of Answer books	
Requirements:	
Handout	
MCQ	
Statistical Tables	
Graph Paper	
Log Graph Paper	
Other Material	
No. of Pages	7
Department(s)	Department of Electronic Engineering

Section A

Attempt all questions in this section – 25 marks; 2.5 marks per question.
(NB: where applicable you must include rough-work calculations to obtain full marks for these questions.)

A1. Which of the following services would you not expect to find offered in a conventional RTOS kernel?

(a) Interrupt handling
(b) Event handling
(c) Semaphores
(d) Intertask messaging
(e) Task management
(f) none of the above

A2. What minimum ROM and RAM requirements would meet the needs of a basic embedded TCP/IP stack?

(a) 1k ROM, 128 bytes RAM
(b) 4k ROM, 256 bytes RAM
(c) 8k ROM, 512k RAM
(d) 16k ROM, 1k RAM
(e) 16k ROM, 4k RAM
(f) none of the above

A3. Which of the following is not a valid use of a semaphore?

(a) Intertask signalling
(b) Protect shared data
(c) Intertask bulk data transfer
(d) Signal that an ISR is active
(e) Control access to a hardware resource
(f) none of the above

A4. Which of the following would be most suitable for passing a single JPEG image between two tasks?

(a) A semaphore
(b) A mailbox
(c) A queue
(d) A pipe
(e) A task control block
(f) none of the above

A5. When a low level task blocks, but retains access rights to some RTOS resources and thus prevents another task with hard real-time deadlines from completing successfully, this is known as:

(a) Resource contention
(b) Functional queue scheduling
(c) Kernel deadlock
(d) A shared data conflict
(e) Priority inversion
(f) none of the above

A6. Which of the following is not a valid method for intertask communication?

(a) A semaphore
(b) A mailbox
(c) A queue
(d) A pipe
(e) An event
(f) none of the above

A7. Which of the following would be most suitable for passing an MJPEG image stream between two tasks?

(a) A semaphore
(b) A mailbox
(c) A queue
(d) A pipe
(e) A task control block
(f) none of the above

A8. Which embedded software architecture has the best response time for running task code?

(a) Round-robin
(b) Round-robin with interrupts
(c) Function-queue scheduling
(d) Co-operative RTOS
(e) Pre-emptive RTOS
(f) none of the above

A9. What is the simplest embedded software architecture that is suited to implement a system where interrupts routines need to be handled in priority order, but all task code may be handled at a single equal priority?

(a) Round-robin
(b) Round-robin with interrupts
(c) Function-queue scheduling
(d) Co-operative RTOS
(e) Pre-emptive RTOS
(f) none of the above

A10. Which of the following is not a valid task state for a pre-emptive RTOS?

(a) Blocking
(b) Ready
(c) Interrupted
(d) Running
(e) Pre-empted
(f) none of the above

Section B

Attempt 3 from 6 questions in this section – 75 marks; 25 marks per question.

(NB: Candidates should note that marks may be lost if answers are not presented in a neat and orderly manner)

B 1. (a) What is interrupt nesting? What are the requirements for it to operate correctly? Use sketches, diagrams and code segments to illustrate your explanation. Give one example of a key component of an embedded software design where the use of interrupt nesting is a requirement.

[8 marks]

(b) Explain the *shared-data* problem which occurs between the application code and the interrupt handling code of an embedded system. Illustrate your discussion with example C-code.

[8 marks]

(c) Describe three different methods to solve the shared data problem. What are the advantages/disadvantages of each? Illustrate your discussion with example C-code.

[9 marks]

B 2. (a) What are the 4 principle categories of software architecture employed in embedded systems? Explain the basic functionality of each, using code segments and diagrams, where appropriate, to illustrate your answer. Compare each of these architectures under the headings of (i) availability of priority levels, (ii) worst-case response times, (iii) stability of response with code changes and (iv) simplicity.

[12 marks]

(b) Design an API for a simple RTOS which includes simple messaging, task priority, and allows a subroutine to be called from within a task. Detail the task control block structure and briefly explain the functionality of each of the API functions. Provide 8051 code to implement the "NextTask" or *scheduler* function and explain its operation through comments, flowcharts, or otherwise.

[13 marks]

B 3. (a) Describe four different approaches to implementing an embedded TCP/IP network stack.

[8 marks]

(b) Compare and contrast each of these approaches under the headings of (i) relative cost, (ii) development time & effort, (iii) memory requirements and (iv) customizability and flexibility of the end-solution.
Which solution is best for implementing a low-cost 2400 baud dial-up solution? Why?
Which solution is best for implementing an Ethernet based network appliance? Why?

[10 marks]

(c) On some RTOSs you can write two types of interrupt routines: *conforming routines*, which inform the RTOS when they enter and exit by using semaphores, and *non-conforming routines* which do not. Explain, using short code segments, how a *conforming* interrupt routine might be implemented. What are the advantages of a *non-conforming* interrupt routine? What are its disadvantages?

[7 marks]

B 4. **(a)** In the context of an RTOS explain, using diagrams and/or code segments where appropriate, what is meant by: (i) a task, (ii) the scheduler; (iii) the dispatcher; (iv) a semaphore; (v) a task control block and (vi) a pipe.

[9 marks]

(b) What is a pre-emptive RTOS? Sketch the state transition diagram for tasks in such an RTOS and explain the various pre-emption mechanisms.

[8 marks]

(c) In an RTOS based embedded system the shared data problem can occur between tasks which share global memory. *Fig 1* shows some C-code which is susceptible to the shared-data problem. Explain how errors can occur in this code. How could you modify this code, using a semaphore, to solve the shared data problem? Explain using code segments where appropriate.

Comment on the statement: "In a non-pre-emptive RTOS, tasks cannot interrupt one another and therefore there are no data-sharing problems between tasks". Do you agree or disagree?

[8 marks]

```
struct {
  long lTankLevel;
  long lTimeUpdated;
} tankdata[MAX_TANKS];

void vRespondToButton(void)
{ /* high priority task */
  int i;
  while (TRUE) {
    !! Block until button pressed
    i = !! ID of button pressed
    !! output tank level, timestamp
  }
}
```

```
void vCalculateTankLevels(void)
{ /* low priority task */
  int i = 0;
  while (TRUE) {
    !! read float levels in task i
    !! do bunches of calculations
    /* store result */
    tankdata[i].lTimeUpdated =
      !! current time
    tankdata[i].lTankLevel =
      !! result of long calculation
    !! pick next tank to handle, etc.
  }
}
```

Fig 1:.Code to implement a push-button user interface and a real-time scan of tank levels using two RTOS tasks which both access the shared data array containing the tank-level data.

B 5. **(a)** Deign a state-machine dialog to model the higher level functionality of a digital camera. The dialog should feature an initialization state which allows the camera to be set into one of the following states: (i) Normal-Record-Mode; (ii) Normal-Playback-Mode; (iii) Normal-Setup-Mode and (iv) Normal-PC-Mode; and (v) Defaults-State . Where appropriate each of this states can enter into a "Menu" state which allows the properties of the parent state to be changed. A brief description of the functionality of each state is given below:

(i) In normal recording mode the camera should be able to take pictures, maintain a picture count, manage zoom functionality and to switch between a passive viewfinder and a live real-time image on its LCD screen.

(ii) Normal playback mode should allow a user to navigate recorded pictures in a linear manner, and ideally offer some picture zoom and non-linear navigation functionality.

(iii) Normal setup mode should offer menus to configure the camera's parameters.

(iv) Normal PC mode sets the camera as a slave to a PC over the USB port. In this mode the camera only needs to respond to a *mode-change* event.

(v) The defaults state handles mode changes and other major *exception* events including an unformatted data card, lens cap still on camera, no data card in camera, low battery power and no USB connection.

The camera user interface has a shutter button, a four-way rocker switch, a rotary mode-select dial and a variety of simple push switches.

[25 marks]

B 6. (a) Explain the concept of re-entrant code. What is required for code to be re-entrant? Is the following function re-entrant? Explain.

```
int cErrors;
void vCountErrors (int cNewErrors)
{
        cErrors += cNewErrors;        }
```

[8 marks]

(b) The following routines are called by several tasks, but they do not function as intended. Can you explain why and devise, and explain, a solution for this problem:

```
static int iRecordCount;

void increment_records (int iCount)
{
        OSSemPend (SEMAPHORE_PLUS);
        iRecordCount += iCount;
}
```

```
void decrement_records (int iCount)
{
        iRecordCount -= iCount;
        OSSemPost (SEMAPHORE_MINUS);
}
```

[8 marks]

(c) The following routines are called by several tasks, but they do not function as intended. Can you explain why and devise, and explain, a solution for this problem:

```
static int iValue;

int iFixValue (int iParam)
{
        int iTemp

        iTemp = iValue;
        iTemp += iParam * 17;

        if (iTemp > 4922)
                iTemp - iParam;
        iValue = iTemp;

        iParam = iTemp +179;
        if (iParam < 2000)
                return 1;
        else
                return 0;
}
```

[9 marks]