University of KwaZulu-Natal School of Engineering

Discipline of Electrical, Electronic & Computer Engineering

Course: ENEL4RC – Real Time Computing

Main examination – paper one

Date: October 2025 **Duration**: 120 minutes **Marks**: 100

Programmable calculators are not allowed.
Clearly delete (strike out) any rough work that is in your answer book. The rough-work will not be marked.
Ensure that you write your student number and seat number on each answer book.

Section A (Basics scheduling – 35 marks)

Question one: **[15 marks]**

1 a) Draw the three-state, task state diagram. Label all states and transitions. [9]

Each of the following statements is not entirely correct. In each case, explain the fault or missing information.

1 b) When a task leaves the running state, it is called pre-emption. [3]

1 c) Pre-emption is always triggered by priority inversion or resource contention. [3]

Question two: **[20 marks]**

Consider the following task set. Note that all tasks arrive at t=0, and that deadlines are shorter than the task periods.

	c	p	d
A	3	10	7
B	2	5	4
C	2	10	8

You are required to schedule these tasks in part a) and b) of the question. In your scheduling diagram, clearly indicate and number the execution slots. In each slot indicate the running task, the front of the queue, and second in the queue. Also indicate all task-arrivals from below and all task-deadlines from above.
Note: Negative marking: -1 for every error or omission.

a) Draw the non-pre-emptive, static priority, DMA schedule for the above task set. Schedule the first 10 time slots. Does the DMA schedule succeed or fail? Explain your answer. [10]

b) Draw the pre-emptive, dynamic priority, EDF schedule for the above task set. Schedule the first 10 time slots. Does the EDF schedule succeed or fail? Explain your answer. [10]

Section B (Cyclic executive – 30 marks)

Question five: **[20 marks]**

Perform a full theoretical design of a Cyclic Executive for the following set of tasks. Compute the number of times each task runs in the Major cycle, and compute the number of idle slots in the major cycle. Show full working, indicate formulae, design choices, etc.

	c	*p*	*d*
A	2	12	12
B	2	6	6
C	1	4	4

Question six: **[10 marks]**

Draw the schedule of a feasible major-cycle for the above CE. Indicate all system clock ticks, all task arrivals, major and minor cycles. Assume that all tasks initially arrive at t=0. Remember that idle times must be attached to the end of a minor cycle.

Section C (Deadlock and bankers algorithm – 15 marks)

Question seven: **[5 marks]**

a) Explain the meaning of Coffman's "hold and wait" condition for deadlock. [2]

b) To cancel this "hold and wait" condition, what action must we take? Explain how this cancellation will resolve the deadlock, and discuss one negative consequence. Clearly explain the three points, namely "Action", "Resolution" and "Consequence" [3]

Question eight: **[10 marks]**

Given four processes that use three resource types, we wish to establish if deadlock can be avoided. After executing for some time, all resources have been allocated and the system status is indicated as follows:

	Allocation	Request/Need	Available
	R1 R2 R3	R1 R2 R3	R1 R2 R3
P1	1 0 0	1 0 0	1 0 1
P2	0 0 1	0 2 0	
P3	0 1 0	2 0 1	
P4	0 1 0	2 1 0	

From the point in time indicated above, does a safe sequence exist, or will we end up in deadlock? Use the resource allocation algorithm (from Banker's algorithm) to establish whether we have deadlock or a safe sequence exists. Show all steps. Begin at P1 and proceed sequentially to P4 and repeat as needed thereafter.

Section D (FreeRTOS – 20 marks)

Question nine: **[20 marks]**

Consider the following partial FreeRTOS application code and answer the questions that follow.

```
#include <Arduino_FreeRTOS.h>

void setup() {
 // Arbitrary application setup code goes here
  xTaskCreate(Task_1, "Task no. 1!", 128, NULL, 1, NULL);
  xTaskCreate(Task_2, "Task no. 2!", 128, NULL, 1, NULL);
}

void loop() {}

static void Task_1(void* pvParameters) {
  while (1) {
   Continuous_workload1(); // Arbitrary continuous workload for task 1 goes here
  }
}

static void Task_2(void* pvParameters) {
  while (1) {
   Continuous_workload2(); // Arbitrary continuous workload for task 2 goes here
  }
}
```

a) Explain what type of schedule the above code produces. [5]

b) The above tasks are continuous and not periodic. Periodic tasks, on the other hand, complete their workloads in finite time and leave the CPU voluntarily. How do periodic tasks leave the CPU in FreeRTOS? Explain the logic and the two common code options for leaving the CPU. [7]

c) Add a third Task that executes a brief finite workload, at a higher priority. Assume that the workload is defined in function Finite_workload3(), and that it executes rapidly in 2 clock ticks. This task must run its workload every 10 clock ticks. The modified setup() function is provided. Write the code for the new task. [8]

```
void setup() {
  xTaskCreate(Task_1, "Task no. 1!", 128, NULL, 1, NULL);
  xTaskCreate(Task_2, "Task no.  2!", 128, NULL, 1, NULL);
  xTaskCreate(Task_3, "Task no.  3!", 128, NULL, 3, NULL);
}
```