**CG2271 Real Time Operating Systems**

**Lab 3 - Real Time Architectures II**

**Answer Book**

|  |  |
| --- | --- |
| Name: Vu Ngoc Quang | Matric Number: A0099878W |
| Name: Prachi Singhal | Matric Number: A0103527N |

Question 1 (2 marks)

The numbers that were enqueued appear in reverse order. The sequence of enqueuing was from zero to nine. However, the numbers appear from nine to zero on the serial monitor.

Question 2 (3 marks)

No, the numbers do not appear in the same order as they were enqueued. The priority for the largest number is the highest.

for(int i=0; i<QLEN; i++)

enq(queue, (void \*)i, QLEN-i-1);

As seen from the code above, for example, i = 9, the priority will be QLEN-9-1=0 (highest priority ). Therefore, 9 outputs first.

Question 3 ( 4 marks)

The function prototype accepts a void pointer. However, in the main setup the input variable i is of type integer. Therefore, we need to convert the type integer to type void pointer. We use (void \*) to do so.

Question 4 (4 marks)

The function returns the type void pointer. However, we are interested to obtain the value at the memory place that the pointer is pointing to. Hence, we need to typecast the void pointer to integer (the data type of the variable “val”). It is necessary to obtain the integer values and not the memory address of the place where the pointer is pointing to.

Question 5 (3 marks)

Switch bounce happens when a mechanical switch is closed and the metal contact connects the circuit. Usually switches take a few microseconds to a few milliseconds to completely close. This is a problem to electronic systems because the digital logic bounces back and forth low-to-high-to-low-... until the switch settles down. During this time, the electronic system may possibly interpret two different signals. Hence, it is important to prevent switch bouncing.

Question 6 (6 marks)

Debounce means that the system checks the input twice in a short period of time to make sure that the button is definitely pressed. This double checking ensures that the correct input is registered with the system. Without debouncing, pressing the button once may be interpreted by the system as multiple presses. Hence, debounce solves the switch bouncing problem.

Question 7 (5 marks)

“Enqueue” is a function that takes in three parameters and returns a void type pointer. Enqueue inputs a function pointer, which points to the function to be enqueued. The position in the queue is determined by the priority. The first parameter specifies the queue the function pointers have to be queued in. The function pointers are sorted according to priority before they are enqueued in the queue.

The “dequeue” function returns a void type pointer and inputs one parameter. It takes the queue that has to be dequeued and outputs the void pointer to the function with highest priority.

Question 8 (7 marks)

const int INT0\_PRIORITY = 0; //priority of INT0

const int INT1\_PRIORITY = 1; //priority of INT1

The LED connected to pin 6 has a higher priority than the one connected to pin 7, hence LED at pin 6 has a smaller number.

typedef void (\*funcptr)(void);

funcptr fp;

void int0ISR()

{

if (debounce(&int0time)) {

enq(queue, (void \*)int0task, INT0\_PRIORITY);

}

}

void int1ISR()

{

if(debounce(&int1time)) {

enq(queue, (void \*)int1task, INT1\_PRIORITY);

}

}

void setup()

{

queue=makeQueue();

pinMode(7, OUTPUT);

pinMode(6, OUTPUT);

attachInterrupt(INT0, int0ISR, RISING);

attachInterrupt(INT1, int1ISR, RISING);

Serial.begin(9600);

}

void loop()

{

if (qlen(queue) > 0) {

fp = (funcptr) deq(queue);

fp();

}

}

To implement function queue scheduling, which helps us to prioritise tasks as we wish, first we declare an instance *fp* of data type *void (\*funcptr)(void)* so that subsequent functions can be referred to by using this instance.

When an interrupt is issued, it does two things; it checks whether the input is a successful debounce as well as if it is then it will add the function pointer to the queue. The function queue takes in the function based on priority. Hence, if the priority of the red LED is higher than that of the yellow LED, and we press button for yellow, red, yellow, red, the sequence of light will be yellow, red, red and yellow. This is because the dequeue function only dequeues based on priority.

The switch bouncing problem is resolved using the debounce function. This function takes in a pointer that indicates the last time when the switch was pressed. It compares this time to the current time since the program started. If the value of the input doesn’t change within 500 milliseconds, we can conclude that the switch has actually been pressed and hence, the debounce function returns a positive value.

The ISR checks for this value to be positive before proceeding to enqueue the input function, hence solving the debounce problem.

Question 9 (3 marks)

The LED at pin 7 connected to the button at INT0 flashes five times first this button was pressed first.

Question 10 (3 marks)

The LED at pin 6 connected to the button which is connected to INT1 will flash first. Although INT1 has a lower priority as compared to INT0, the LED at pin 6 still flashes five times before the LED at pin 7 flashes. This is because function queue scheduling is not pre-emptive. Therefore, the higher priority task has to wait for the current task to complete before it can be executed. The higher priority task does not interrupt the current task in the process. In this case, the LED which is connected to the higher priority has to wait for the LED which is connected to the lower priority to finish flashing five times before it can start to flash.

Question 11 (5 marks)

When we press the buttons in the order INT1, INT0 five times, the actual order of execution will be INT1, INT0, INT0, INT0, INT0, INT0, INT1, INT1, INT1, INT1 because INT1 has lower priority (the first one is INT1 as it is pressed first). This corresponds to the LED at pin 6 flashing five times, then the LED at pin 7 flashing 25 times and then the LED at pin 6 flashing 20 times. This follows the function queue scheduling as after the current task is completed, the rest of the tasks are executed based on priority. In this case, it is 1, 0, 0, 0, 0, 0, 1, 1, 1, 1.

Question 12 (5 marks)

The LED at pin 6 is supposed to flash five times in one second, which corresponds to a frequency of 5Hz. The LED at pin 7 is supposed to flash once in one second, which corresponds to a frequency of 1Hz.

We are trying to make the fast LED on and off five times in one second. Therefore, in one second, the LED is on for half a second and off for the next half second. This results in a total of 10 cycles. To implement this using timed loops, we will need to have a fast loop function which is called every second ( for the LED blinking 5 times a second ) and another slow loop function for the LED blinking ( once a second ) .

The slow loop will only execute the blinking of the LED after every 10 cycles. To implement this, we have a counter for the slow loop. For the first four counts (8 cycles), the counter is incremented by one. In the fifth count (9th and 10th cycle), the slow LED is executed to blink and the counter is reset.

For the fast loop, the condition to satisfy to enable the blinking of the LED is that the time should be greater than 99. Timed loops allows us manage the frequency of the blinking of the two LEDs as well as allow them to blink simultaneously.

#include <Arduino.h>

const int T = 100; //100ms per cycle

int fastLoopTimer = 0;

int slowLoopCounter = 0;

void togglePin6()

{

static char state=1;

if(state)

digitalWrite(6, HIGH);

else

digitalWrite(6, LOW);

state=!state;

}

void togglePin7()

{

static char state=1;

if(state)

digitalWrite(7, HIGH);

else

digitalWrite(7, LOW);

state=!state;

}

void setup()

{

pinMode(6, OUTPUT);

pinMode(7, OUTPUT);

}

//LED-pin 6 flashes 5 times a sec

void fastLoop()

{

togglePin6();

}

//LED-pin 7 flashes 1 time a sec

void slowLoop()

{

switch(slowLoopCounter) {

case 0:

slowLoopCounter++;

break;

case 1:

slowLoopCounter++;

break;

case 2:

slowLoopCounter++;

break;

case 3:

slowLoopCounter++;

break;

case 4:

slowLoopCounter = 0;

togglePin7();

break;

}

}

void loop()

{

int deltaMiliSeconds = millis() - fastLoopTimer;

if(deltaMiliSeconds > (T-1)) {

fastLoopTimer = millis();

fastLoop();

slowLoop();

}

}

int main()

{

init();

setup();

while(1)

{

loop();

if(serialEventRun)

serialEventRun();

}

}