CG2271 Real Time Operating Systems
Lab 5

1. Introduction

In this lab we will look at queues in ArdOS, as well as how to implement barriers using semaphores. In particular we will look at how multiple tasks writing to the Serial port can cause problems, and how queues can help solve these problems.

The Serial class and ArdOS do not work correctly together on Atmel Studio 6 possibly due to a conflict in the gcc-avr versions, so for this lab we will use the Arduino IDE instead.

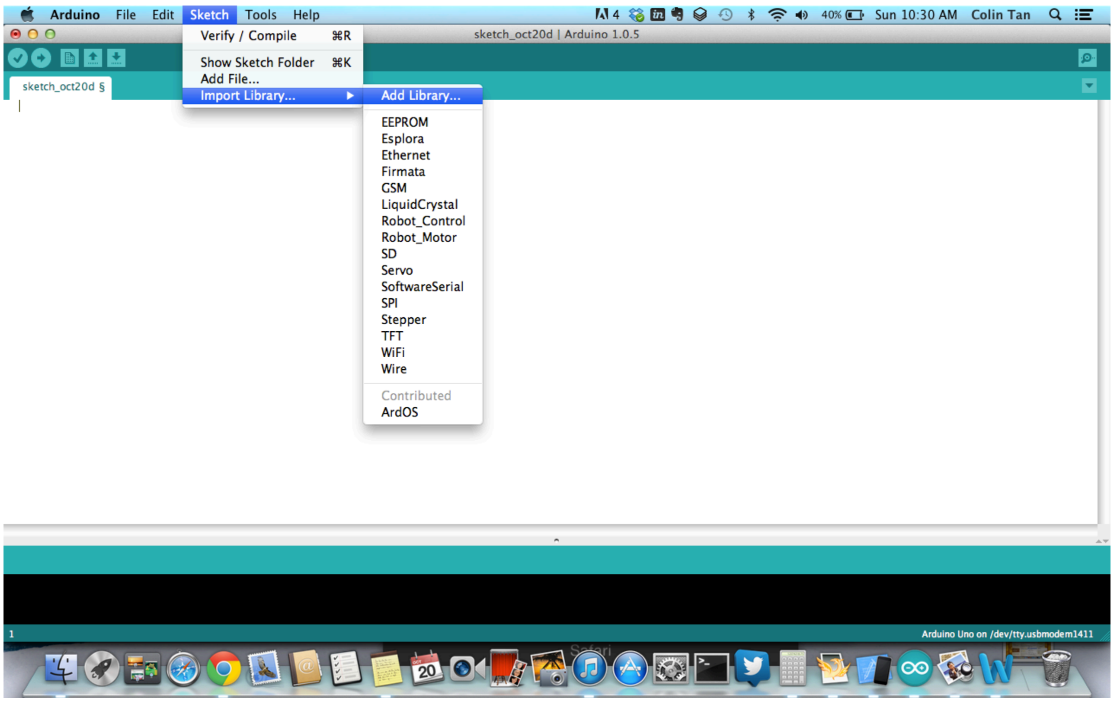
Download and install the Arduino IDE from http://www.arduino.cc if you have not already done so.

There is also an important bugfix that was made to ArdOS. Please download the latest copy of ArdOS (v0.9b) from http://www.bitbucket.org/ctank/ardos-ide/downloads.

Note: If you had previously imported ArdOS into Arduino, you will need to manually delete the old library. You can find this at Documents->Arduino->Libraries->ArdOS on the Mac Finder, or at \My Documents\Arduino\Libraries\ArdOS on Windows machines.

2. Using ArdOS with Arduino IDE

Using ArdOS with Arduino IDE is much simpler than with Atmel Studio 6. To install the ArdOS library, click Sketch->Import Library->Add Library:



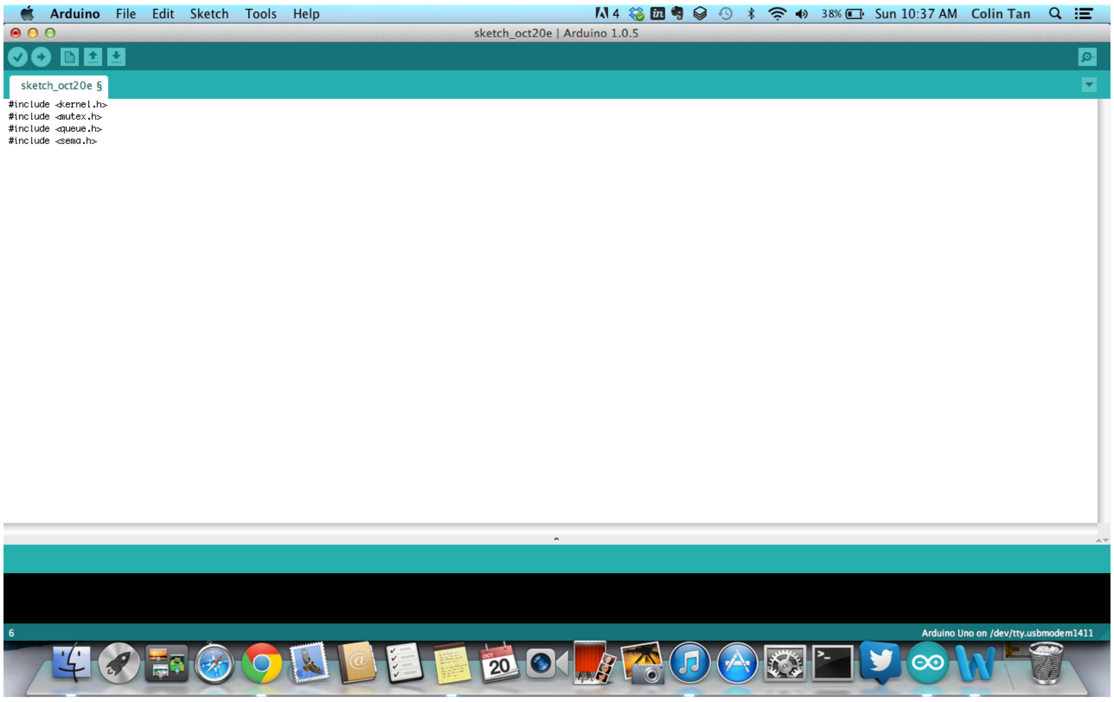
Navigate to the ZIP file that you downloaded containing the ArdOS codes, and select “Choose” to import the library.

When you choose Sketch->Import Library, you should now see “ArdOS” listed under the “Contributed” section.

3. Multiple Tasks Writing to the Serial Port

To create an Arduino project (called a “sketch” in Arduino), click File->New. This will automatically get you a new “sketch”. To include ArdOS into your sketch, click Sketch->Import Library->ArdOS.

You will automatically see the ArdOS headers being included for you:



```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>
```

Arduino Uno on /dev/tty.usbmodem1411

Now key in the following program (exclude the #include statements on top, which were done for you by the IDE):

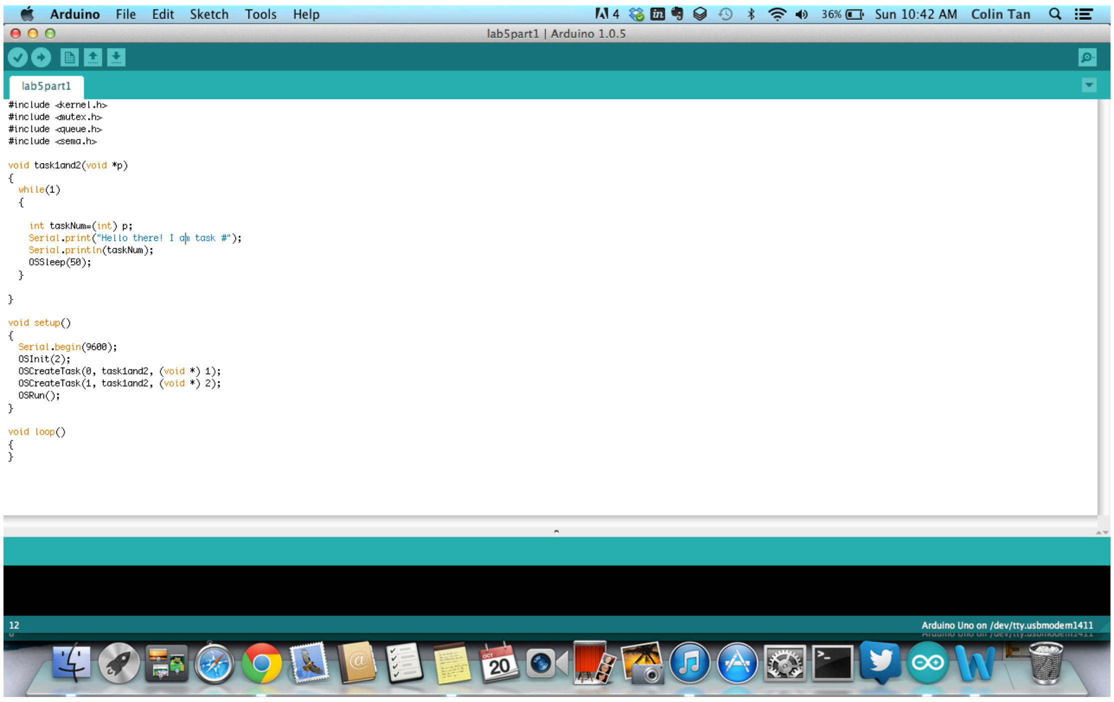
```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>

void task1and2(void *p)
{
 while(1)
 {

   int taskNum=(int) p;
   Serial.print("Task ");
   Serial.println(taskNum);
   OSSleep(5);
  }

}

void setup()
{
  Serial.begin(9600);
  OSInit(2);
  OSCreateTask(0, task1and2, (void *) 1);
  OSCreateTask(1, task1and2, (void *) 2);
  OSRun();
}

void loop()
{
}
```

Some things to note:

i. The loop() function is empty but required.
ii. There is no main function. Arduino will provide one for you.

Now click Tools->Board and ensure that Arduino UNO is selected, then click Tools->Serial Port to ensure that the correct serial port is selected (presumably you have already connected the board to your PC).

Click File->Save and save your sketch as "lab5part1". Now to run the sketch, click the round button with the arrow in it. This would be the second button from the left, next to the round button with the tick in it.


Arduino File Edit Sketch Tools Help
Sun 10:42 AM Colin Tan
lab5part1 | Arduino 1.0.5
lab5part1
12
Arduino Uno on /dev/tty.usbmodem1411

```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>

void task1and2(void *p)
{
  while(1)
  {

    int taskNum=(int) p;
    Serial.print("Hello there! I am task #");
    Serial.println(taskNum);
    OSSleep(50);
  }

}

void setup()
{
  Serial.begin(9600);
  OSInit(2);
  OSCreateTask(0, task1and2, (void *) 1);
  OSCreateTask(1, task1and2, (void *) 2);
  OSRun();
}

void loop()
{
}
```

To see the output of this program you need to launch the Serial Monitor. Click "SHIFT-CMD-M" on the Mac or "CTRL-SHIFT-M" on the PC to launch it. Then answer the questions below:

what is void1 and void2

Question 1 (6 marks)

You will notice that tasks 1 and 2 share exactly the same task code. That is, both tasks are created from the same function "task1and2".

Explain why it is possible to create two different tasks using the same function. (Hint: They are given different context saving space when the tasks are created)

Question 2 (3 marks)

By inspecting the code given above, describe what you EXPECT this program to write to the serial port.

Question 3 (5 marks)

Start the Serial Monitor in the Arduino IDE (or use any other terminal program that can monitor the serial port) to monitor the program's output. You will see that the output is not what you expected and is instead garbled quite badly.

Explain why you think the output is all messed up. (Hint: The serial port actually takes more than 2.5 ms to send all the data in the buffers, which is the average time between sends from two tasks sending at 5ms intervals each).

3. Using a Single Task to Regulate Serial Output

We will now centralize serial output to a single task called serialPrint. Tasks 1 and 2 will send what they want to say to serialPrint, who will print it.

Create a new Arduino sketch, and as before click Sketch->Import Library->ArdOS to bring in the ArdOS library, then key in the following code (again note that the #includes at the start have already been inserted for you by Arduino)

```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>

#define QLEN  64
int buffer[QLEN];
OSQueue queue;

void serialPrint(void *p)
{
 int x;
 while(1)
 {
  x=OSDequeue(&queue);
  Serial.print("Task ");
  Serial.println(x);
  OSSleep(50);
 }
}

void task1and2(void *p)
{
 int taskNum=(int) p;
 while(1)
 {
  OSEnqueue(taskNum, &queue);
  OSSleep(5);
 }
}


void setup()
{
 Serial.begin(9600);
 OSInit(3);
 OSCreateQueue(buffer, QLEN, &queue);
 OSCreateTask(0, serialPrint, NULL);
 OSCreateTask(1, task1and2, (void *) 1);
```

```
  OSCreateTask(2, task1and2, (void *) 2);
  OSRun();
}

void loop()
{
}
```

Save this sketch as lab5part2, upload it to the Arduino and answer these questions:

Question 4 (5 marks)

Describe what OSCreateQueue, OSEnqueue and OSDequeue do, and what parameters they take.

Describe also how ArdOS queues are different from the queues you learnt in your data structures course.

the queue stores the task or the task number?

Question 5 (3 marks)

task number

Using either the Arduino IDE's Serial Monitor or a terminal program capable of monitoring the serial port, describe the output of this program now.

Question 6 (5 marks)

You will see that this program now (sort of) works properly. However you will notice that serialPrint uses a OSSleep(50) call to space out calls to Serial.print and Serial.println to prevent the outputs from being garbled.

Explain why you think OSDequeue itself is insufficient for ensuring correct output on the Serial port (you can verify this yourself by commenting out the OSSleep(50) in serialPrint. You will see the output garbled again)

Question 7 (5 marks)

You will notice that after a while you only see "Task 1" on the screen. Explain why you think this happens.

huh?

3. Implementing Barriers using Semaphores

We will now take on something more challenging. Create a new sketch, import ArdOS, then key in the following code:

```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>

// Data structure for the barrier.
typedef struct OSBarrier
{
 int count;
 OSSema sema;
};

// Creates a new barrier.
// Parameters: count = # of tasks expected to reach the barrier
// barrier = pointer to barrier data structure
void OSCreateBarrier(unsigned int count, struct OSBarrier *barrier)
{
        // TO DO: IMPLEMENT THE CREATE BARRIER FUNCTION
}

// Informs barrier that tasks has reached it. barrier = pointer to
// barrier data structure
void OSReachBarrier(struct OSBarrier *barrier)
{
        // TO DO: IMPLEMENT REACH BARRIER
}

// DO NOT MODIFY ANYTHING BELOW THIS LINE

#define QLEN  8
int crossedBuffer[QLEN];
int reachedBuffer[QLEN];

OSQueue crossedQueue;
OSQueue reachedQueue;
struct OSBarrier barrier;

void task1(void *p)
{
 char crossed=0;
 while(1)
 {
  if(!crossed)
```

```
    {
      OSSleep(250);
      OSEnqueue(1, &reachedQueue);
      OSReachBarrier(&barrier);
      OSEnqueue(1, &crossedQueue);
      crossed=1;
    }
    else
      OSSleep(100);

  }
}

void task2(void *p)
{
  char crossed=0;
  while(1)
  {
    if(!crossed)
    {
      OSSleep(195);
      OSEnqueue(2, &reachedQueue);
      OSReachBarrier(&barrier);
      OSEnqueue(2, &crossedQueue);
      crossed=1;
    }
    else
      OSSleep(100);
  }

}

void task3(void *p)
{

  char crossed=0;

  while(1)
  {
    if(!crossed)
    {
      OSSleep(850);
      OSEnqueue(3, &reachedQueue);
      OSReachBarrier(&barrier);
      OSEnqueue(3, &crossedQueue);
      crossed=1;
    }
    else
      OSSleep(100);
```

```
 }

}

void reachTask(void *p)
{
 while(1)
 {
  unsigned pnum=OSDequeue(&reachedQueue);
  Serial.print(pnum);
  Serial.println(" has reached the barrier");
  OSSleep(50);
 }
}

void crossTask(void *p)
{
 while(1)
 {
  unsigned pnum=OSDequeue(&crossedQueue);
  Serial.print(pnum);
  Serial.println(" has crossed the barrier");
  OSSleep(50);
 }
}



void setup()
{
 Serial.begin(9600);
 Serial.println();
 Serial.println("------ NEW RUN -----");
 Serial.println();
 OSInit(5);
 OSCreateBarrier(3, &barrier);
 OSCreateQueue(crossedBuffer, QLEN, &crossedQueue);
 OSCreateQueue(reachedBuffer, QLEN, &reachedQueue);
 OSCreateTask(4, task1, NULL);
 OSCreateTask(2, task2, NULL);
 OSCreateTask(3, task3, NULL);
 OSCreateTask(0, reachTask, NULL);
 OSCreateTask(1, crossTask, NULL);
 OSRun();
}

void loop()
{
}
```

A structure called OSBarrier has been defined for you consisting of a counter and a semaphore. You are also provided with two functions OSCreateBarrier and OSReachBarrier, which you have to implement.

When you barrier is working correctly you should see the following on your Serial monitor when you press the RESET button on the Arduino UNO (the RESET button is a small brown button marked RESET)

```
----- NEW RUN -----

2 has reached the barrier
1 has reached the barrier
3 has reached the barrier
2 has crossed the barrier
3 has crossed the barrier
1 has crossed the barrier
```

Some hints:

i. Use the counter in OSBarrier to keep track of the number of tasks that have still NOT YET reached the barrier.
ii. If not all tasks have reached the barrier yet, wait on the semaphore.
iii. Each time OSGiveSema is called, it will release ONE task that is waiting for the semaphore.
iv. Properly done the code for both OSBarrier and OSCreateBarrier is very short, consisting of under 6 lines each.

Question 8 (8 marks)

Cut and paste and explain your code for OSCreateBarrier.

Question 9 (10 marks)

Cut and paste and explain your code for OSReachBarrier.

Question 10 (5 marks)
Notice that although the tasks reach the barrier in the order 2,1,3, when they cross their outputs are listed in the order 2,3,1 instead. Explain why. (Hint: Look at the priorities given in OSCreateTask).

Note that this section will work correctly only if you use ArdOS v0.9b.