## COMP 3510 Embedded Systems Development

Lab Assignment 1

By Group

(Due Date on Syllabus/Canvas)

**IMPORTANT:**

1. *Your code will be tested and graded on the Engineering Unix (Tux) machines. If the code does not work on those machines, you do not get any credit even if your code works on all the other machines in the universe.*
2. *An incomplete submission (electronic or hard copy missing) is considered as late and will get a penalty of 50% per day. One submission per group no later than 10 mn after the start of the lecture*
3. *Writing and presentation of your report are considered to grade your lab.*
4. *The quality of your code will be evaluated.*

**Programming Assignment**

**First**, it is assumed that by January 23, 1) you have an engineering Unix account, 2) you can edit, 3) you can compile, and 4) you can execute C programs on the Unix (Tux) machines. You can use any personal computer or computing lab to remotely access the Engineering Unix machines.

**Second,** it is assumed that by January 23, you have your group partners.

**Objectives**:

**Look at “How to get started?” at the very end of the lab.**

This lab has three parts: 1) Write a code to control an embedded system, 2) evaluate your code, and 3) analyze your code.

The instructor developed a framework that allows the simulation of a system to monitor/control.

Control System

(**Control()**: Students code)

The instructor wrote code that randomly generates events from one to 32 devices. Events are generated on EACH device at an average inter-arrival (in seconds) ***lambda***. Each event requires an average service time ***mu*** (provided in percentage of *lambda*). Your objective is to detect ALL events and service them. *Depending on your software and hardware architecture, the frequency of your events (lambda), and the required service time (mu), you may miss events*. You must minimize the number of missed events.

Device(s)

(**devices.o**: instructor code)

1) **Control Program:** Write a program to control an embedded system with ***Number\_Devices*** devices. Events occur randomly on each device. When an event occurs on device ***i,*** the *ith* bit of the variable ***Flags*** (starting from the left) is raised. Example: suppose that Flags = 0 (0000 0000) and Device 3 generates an event, the variable *Flags* takes the value 8 = (0000 **1**000), then Device 1 gets an event, Flags becomes ((0000 **1**0**1**0). When an event occurs on Device ***i***, a flag is raised in the variable ***Flags*** and the corresponding event is stored in the buffer in ***BufferLastEvent[i].*** An event has the following structure:

typedef struct EventTag{

Identifier EventID;

Identifier DeviceID;

Timestamp When;

char msg[64];

int priority;

} Event;

When a device generates an event, you must display this event (you may use the function ***DisplayEvent*** in the file ***common.h***) and you **must** call the function ***Server(Event \*whichEvent)*** to process your event.

**We assume that the devices do not generate interrupts**: whenever an event occurs, the device raises a bit flag in the variable ***Flags*** and stores the last event in the buffer. There is storage for only one event on each device. **Note that devices raise the flags in the variable** *Flags***, but the function** *Server* **does not reset the flags in** *Flags***. It is your responsibility to reset the corresponding flag as appropriate.**

Your code must process events such that you miss as few as possible events without forking processes or threads: we assume that there is not real time OS support. You will be provided three files: ***common.h***, ***devices.o***, and ***lab1.c***. You are not allowed to modify the file ***common.h*** or the ***main*** function in the ***lab1.c*** file. In the file lab1.c, you must develop your code ***INSIDE*** the function ***Control().*** You may add global variables or routines (functions, methods) in the file lab1.c. To compile your program,

you must type: ***cc-o lab1 devices.o lab1.c -lm***

where

***devices.o*** is the object file that emulates devices generating events

***lab1***  is the executable.

***lab1.c*** is your source file

**YOU CANNOT MODIFY *common.h*.**

**YOU CAN read and write the variable *Flags. Flags* is the only variable you can modify.**

**YOU CAN ONLY READ the array *BufferLastEvent[i]*. Do not write on this array.**

2) **Code Evaluation:**

1. Compile your code with “*cc –o lab1 devices.o lab1.c -lm*”.
2. Execute your code with “lab1 NbrDevices lambda mu Show” where ***NbrDevices*** is the number of devices, ***lambda*** is the average interarrival (in seconds) of events on EACH device, ***mu*** is the average service time (provided in percentage of *lambda*/*NbreDevices*) of an event (duration of the function ***Server***()), ***Show*** is a flag (0 or 1) to display and dump events to files. ***The code generates about 100 events per device and stops.***
3. In order to evaluate your code, you must execute the program until it stops. You must “instrument” your code to collect the following values (A is the average percentage of missed events, B is the average response time, and C is the average turn around time):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NbrDevices | lambda | mu | A (%) | B (s) | C (s) |
| 2 | 2 | 10 |  |  |  |
| 2 | 2 | 30 |  |  |  |
| 2 | 2 | 60 |  |  |  |
| 2 | 2 | 90 |  |  |  |
| 4 | 2 | 10 | 2 / 400 = 0.5% | 0.004183 | 0.063068 |
| 4 | 2 | 30 | 14 / 400 = 3.5% | 0.094464 | 0.263507 |
| 4 | 2 | 60 | 58 / 400 = 14.5% | 0.222752 | 0.552498 |
| 4 | 2 | 90 |  |  |  |
| 8 | 4 | 10 |  |  |  |
| 8 | 4 | 30 |  |  |  |
| 8 | 6 | 60 |  |  |  |
| 8 | 6 | 90 |  |  |  |

1. For each combination, you must collect the number of missed events (events generated by the devices and that you did not process (serve)) for each device and compute the average number of missed events.

3) **Code Analysis:**

**a)** You must explain why your code misses events and how the number of misses is related to ***lambda*** and ***Mu***. Explain also the variations of the response time and turn-around time.

b) What could be done to decrease the number of missed events? What could be done to improve the response time? What could be done to improve the turnaround time?

**Get Started**

1. compile the code I provided you: ***cc-o lab1 devices.o lab1.c –lm***
2. Execute the code with two devices: ./lab1 2 1 10 0
3. Observe the variable ***Flags*** printed out on the screen
4. Get a feel of the values that the variable ***Flags*** takes.
5. Stop the execution with CTRL-C and see what you read.
6. Execute with 4 devices: ./lab1 4 1 10 0
7. Observe the variable ***Flags*** printed out on the screen
8. Get a feel of the values that the variable ***Flags*** takes.
9. Stop and execute: ./lab1 2 1 10 1
10. Now, you should see events generated
11. Stop with CTRL-C…..
12. “Play” with code ***Control()*** in lab1.c for detecting all events, then try to process them..

**What to turn in?**

1. **Hard copy** of your report and code for lab1.c (do not submit any other file than lab1.c)
2. **Electronic copy** of your report and code for lab1.c. These two files and your report must me put in a folder named lab1XX where XX is your group ID (on Canvas). Zip the folder and post it on Canvas. **A penalty of 10 points will be applied if these instructions are not followed.**
3. Your report must:
   1. state whether your code works
   2. report/analyze the results about the missed events (based on the filled table above). The quality of analysis and writing is critical to your grade.
   3. address Part 3) “Code Analysis” (quality of writing (content and form) is of utmost importance)

Good writing and presentation are expected.

**IMPORTANT:**

1. your code will be tested and graded on the Engineering Unix (Tux) machines. If the code does not work on those machines, you will not get any credit even if your code works on all the other machines in the universe.
2. An incomplete submission (electronic or hard copy missing) is considered as late and will get a penalty of 50% per day. Submit no later than 10 mn after start of class.