## COMP 3510 Embedded Systems Development

Lab Assignment 2

By Group (2 students at most per group)

(Due Date on Syllabus/Canvas)

**IMPORTANT: First, Use common.h and associated files from Lab 2, not from previous lab. Second, embedded systems are often limited in RAM.**

1. *Your code will be* ***REMOTELY*** *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 every other machine.*
2. *A late submission will get a 50% penalty if submitted after 9:00pm on the due day. After 9:00pm the next day, you cannot submit the lab.*
3. *One submission per group.*
4. *Writing and presentation of your report are considered to grade your lab. Your conclusions* ***must be supported*** *by the data/measurements you collect.*
5. *The quality of your code will be evaluated.*

**Objectives**:

The objectives of this lab is to initiate to the issue of **timely** monitoring a system without missing events. **The differences between Lab 1 and Lab 2 are highlighted in blue.**

**Programming Assignment (Turned in by one group mate)**

**I**t is assumed that now 1) you have an engineering Unix account, 2) you can edit C programs, 2) you can compile them, and 3) you can execute them. You can use any personal computer or computing lab to remotely access the Engineering Unix machines.

**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. For this, you must complete the following two routines: ***Control()*** and ***BookKeeping().*** The ***Control()*** routine **monitors** and **handles** events occurring on the events. The function BookKeeping() routine must print out at the end the data of interest: **FOR EACH DEVICE,** the number of missed events, the average response time, and the average turnaround time, and FOR ALL DEVICES, the average number of missed events, the average response time, and the average turnaround time.

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

Control System

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

Device(s)

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

The instructor wrote the code that randomly generates events from one to **31** 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 1) minimize the number of missed events, minimize the response time, and if possible the turnaround time).

**Flags**

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*** (*ith* means bit with weight *i* (least significant bit has weight *0*) 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). Later Device 1 gets an event and 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 ***BufferLastEvent[i],* and the device generates an interrupt that runs the routine interrupt handler *InterruptRoutineHandlerDevice (IRHD).*** **DO NOT CALL IRHD yourself.** 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 **urgently** display this event (you may use the function ***DisplayEvent*** in the file ***common.h* provided** ) and you **must** call **as soon as possible** the function ***Server(Event \*whichEvent)*** to process your event. The ***response time*** is equal *t-When* where *t* is the time just because you call Display and *When* is the the event’s timestamp. The turnaround time is *tf-When* where *tf* is the time just after the call Server() returns.

**We assume that the devices generate interrupts**: whenever an event occurs, the device raises a bit flag in the variable ***Flags,*** stores the last event in the buffer, **and generates an interrupt that runs the routine interrupt handler *InterruptRoutineHandlerDevice***. Recall that 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/when appropriate.**

Your code must process events such that you miss as few events without forking processes or threads: we assume that there is no real time OS support. You will be provided three files: ***common.h***, ***devices2.o (different from Lab 1)***, and ***lab2.c***. You are not allowed to modify the file ***common.h*** or the ***main*** function in the ***lab2.c*** file. In the file lab2.c, you must develop your code ***INSIDE*** the function ***Control().*** You may add global variables or routines (functions, methods) in the file ***lab2.c***. To compile your program,

you must type: ***cc-o lab2 devices2.o lab2.c -lm***

where ***devices2.o*** is the object file that emulates devices generating events, ***lab2***  is the executable, and ***lab2.c*** is your source file

**You must write the routine BookKeeping() in order to count the number of missed events, the turnaround time, and the average around time. This will allow you to keep a more precise number of missed events.**

**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.**

**YOU CAN create new variables, new types, new routines/functions …. in your lab2.c source.**

2) **Code Evaluation:**

1. Compile your code with “*cc –o lab2 devices2.o lab2.c -lm*”.
2. Execute your code with “lab2 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 |  |  |  |
| 4 | 2 | 30 |  |  |  |
| 4 | 2 | 60 |  |  |  |
| 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. Compare the values (missed events, average response time, and average turnaround time) for Lab2 versus Lab1 ones. Build plots to compare Lab 2 results against Lab 1’s. **Comment** and **explain** the differences.

b) Open question (needs some reading/research): what should we do to minimize the average turnaround time?

**Part I:**

**What to turn in?**

1. **Electronic copy** of your report and the C source code lab2.c. These two files must be submitted separately (not in a zipped folder). **A penalty of 10 points will be applied if these instructions are not followed.**
2. Your report must:
   1. state whether your code works
   2. report/plot/analyse the results about the missed events (fill out the table above), **the average response time, and the average turnaround time. Contrast lab2 values with those measured in Lab1**. 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)

**Getting Started**

1. Add some statement in the interrupt Handler routine (e.g., DisplayEvent..)
2. compile the code I provided you: ***cc-o lab2 devices2.o lab2.c –lm***
3. Execute the code with two devices: ./lab2 2 1 10 0
4. Observe the variable ***Flags*** printed out on the screen
5. Get a feel of the values ***Flags*** takes.
6. Stop the execution with CTRL-C and see what you read.
7. Execute with 4 devices: ./lab2 4 1 10 0
8. Observe the variable ***Flags*** printed out on the screen
9. Get a feel of the values ***Flags*** takes.
10. Stop and execute: ./lab2 2 1 10 **1**
11. Now, with the parameter 1 (highlighted in red), you should see events generated
12. Stop with CTRL-C…..
13. “Play” with code ***Control()*** in lab2.c for detecting all events, then try to process them..

**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 any other machine.*
2. *A late submission will get a 50% penalty if submitted after 9:00pm on the due day. After 9:00pm the next day, you cannot submit the lab.*
3. *One submission per group.*
4. *Writing and presentation of your report are considered to grade your lab. Your conclusions* ***must be supported*** *by the data/measurements you collect. The quality of your code will be evaluated.*