Lab 1: Group 6

Jordan Hutcheson

Walter Conway

Date: 02/13/14

Your report must:

A.) State whether your code works.
B.) Report/Analyze the results about the missed events (based on the filled table).
The quality of analysis and writing is critical to your grade.
C.) Address Part 3) “Code Analysis” (quality of writing (content and form) is of utmost importance)
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 he turnaround time?

PART A: State whether your code works.
The code that we wrote works. It is also available to view at the end of this report.

PART B: Report/Analyze the results about the missed events (based on the filled table).

NbrDevices	Lambda	Mu	A(%)Avg Missed Events	B(s) Avg Resp Time	C(s) Avg Turn Around Time
2	2	10	0%	0.017888	0.141611
2	2	30	14.070352%	0.115837	0.503532
2	2	60	29.648241%	0.282049	1.062917
2	2	90	45.728642%	0.544151	1.851571
4	2	10	0.501253%	0.010474	0.070144
4	2	30	4.260652%	0.099507	0.281448
4	2	60	14.536341%	0.270637	0.639912
4	2	90	28.388748%	0.429728	0.991665
8	4	10	0.125156%	0.010733	0.066509
8	4	30	1.251565%	0.102370	0.270114
8	6	60	6.700379%	0.561381	1.067083
8	6	90	18.101267%	1.408877	2.187743

Note: Analysis of results are covered more thoroughly in Part C.

PART C: Code Analysis:
A.)You must explain why your code misses events and how the number of misses is related to **lambda** and **Mu.**
Lambda – Events are generated on each device at an average inter-arrival (in seconds). Frequency of your events.
Mu – Each event requires an average service time (percentage of lambda). The required service time.
Explanation:

The reason why we are missing events is that when Mu is high this means that the required service time for each event will take longer. This correlates to the missed events as well as the low Lambda, which means each specified lambda that is when an event will occur seeing that we are giving low lambdas in respect of the high Mu specified this means that when we are servicing an event that has a high mu and a system that has a low lambda then the percentage of missed events will be higher. Since we do not have time to finish servicing an event before another one comes.

Explain also the variations of the response time and turn-around time.

Explanation of variations of the response time:

Response time = Time Stamp at First Response - Event Time Stamp.

The higher mu the longer it takes to respond to the next event because the previous event is still being serviced. As lambda stays stays the same As lambda increases the response time will decrease if mu stays the same.

Explanation of the variation of the turn-around time:

Turn-Around time = Time Stamp at End of Service – Event Time Stamp.

Mu is the most principle factor in both turn-around time and response time. Since this is true, the similar observations can be said for turn-around time as they were mentioned for response time.

B.)
What could be done to decrease the number of missed events?

The number of missed events can be decreased by either raising the lambda so the frequency of events coming in could be slowed down in respect of a high mu. Or lowering the mu so that the time it takes to service a particular event would be sped up so that the turn-around time would be cut down so we could have time to service other events. Or use interrupts. Or a Queue.

What could be done to improve the response time?
Improving the response time can be done by lowering mu so that serving each event would be faster. Also by implementing a different architecture like a function queue scheduling so that when events coming in would interrupt the service and en-queue the incoming event so that events would be in line to be serviced instead of missed. When we are using bit shifting right to get the device that needs to be serviced we are prioritizing more so to the lower devices, instead of equally sharing priority with all devices. So we are missing more of the higher devices than we are lower devices.

What could be done to improve the turn around time?
Improving the turn-around time can be done by lowering mu so that serving each event would be faster, which correlates to a faster turn-around time. We could also take out the calculation lines to improve efficiency of loop within control.

Source Code lab1.c

```
/*****************************************************************************\
* Laboratory Exercises COMP 3510                                              *
* Author: Walter Conway                                                       *
* Author: Jordan Hutcheson                                                    *
* Date  : Feburary 13, 2014                                                   *
\*****************************************************************************/
/*****************************************************************************\
*                        Global system headers                                *
\*****************************************************************************/
#include "common.h"
/*****************************************************************************\
*                        Global data types                                    *
\*****************************************************************************/
/*****************************************************************************\
*                        Global definitions                                   *
\*****************************************************************************/
/*****************************************************************************\
*                       Global data structures                                *
\*****************************************************************************/
/*****************************************************************************\
*                            Global data                                      *
\*****************************************************************************/
float totalServiceTime = 0 ;
float totalResponseTime = 0;
float totalTurnAroundTime = 0;
int totalEvents = 0;
int totalServicedEvents = 0;
int totalMissedEvents = 0;
int totalMissedEventsByDevice[MAX_NUMBER_DEVICES]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int totalEventServicedByDevice[MAX_NUMBER_DEVICES] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

/*****************************************************************************\
*                          Function prototypes                                *
\*****************************************************************************/
void Control(void);
/*****************************************************************************\
* function: main()                                                            *
* usage:    Create an artificial environment for embedded systems. The parent *
*           process is the "control" process while children process will gene- *
*           generate events on devices                                        *
*******************************************************************************
* Inputs: ANSI flat C command line parameters                                 *
* Output: None                                                                *
*                                                                             *
* INITIALIZE PROGRAM ENVIRONMENT                                              *
* WHILE JOBS STILL RUNNING                                                    *
*    CREATE PROCESSES                                                         *
*    RUN LONG TERM SCHEDULER                                                  *
*    RUN SHORT TERM SCHEDULER                                                 *
*    DISPATCH NEXT PROCESS                                                    *
\*****************************************************************************/
int main (int argc, char **argv) {
  if (Initialization(argc,argv)){
    Control();
   }
} /* end of main function */
```

```
/***********************************************************************\
 * Input : none                                                          *
 * Output: None                                                          *
 * Function: Monitor Devices and process events                          *
 \***********************************************************************/
void Control(void){
//Local Variables
Event *currentEvent;
int deviceNumber = 0;
int eventNumber = 0;
int flagTemp = 0;
float beginServiceTime = 0;
float endServiceTime = 0;
float responseTime = 0;
float turnAroundTime = 0;
float eventTime = 0;
int nextEventServiced[MAX_NUMBER_DEVICES] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
while (1) {
		if(Flags != 0 ) {
			lagTemp = Flags;
			Flags = 0;
			deviceNumber = 0;
			while(flagTemp != 0) {
				if(flagTemp & 1) {
				   //Storing current event's information to reference
				{
					currentEvent = &BufferLastEvent[deviceNumber];
					eventNumber = currentEvent->EventID;
					eventTime = currentEvent->When;
				}
				//This finds the number of missed events since last serviced events of the particular device
				//and updates two variables: Total missed and total missed by device.
				if((nextEventServiced[deviceNumber]) != eventNumber) {
					totalMissedEvents += currentEvent->EventID - (nextEventServiced[deviceNumber]);
					totalMissedEventsByDevice[deviceNumber] +=
(eventNumber-(nextEventServiced[deviceNumber]));
				}

				//This Displays the event and records the begining service time.
				//This also services the event and records the end of service time.
				{
				beginServiceTime = Now();
				DisplayEvent('A'+ deviceNumber, currentEvent);
				Server(currentEvent);
				endServiceTime = Now();
				}

				//This updates the total number of serviced events
				//by device and overall as well as recording the next event
				//that should be service this helps in determining the amount
				//of services missed since last serviced event took place.
				{
					totalEventServicedByDevice[deviceNumber]++;
					totalServicedEvents++;
					nextEventServiced[deviceNumber] = (eventNumber) + 1;
				}

				//Calculation being done to determine the response time,
				//total response time, turn around time and total turn around time.
				{
					responseTime = beginServiceTime - (eventTime);
					totalResponseTime += responseTime;
					turnAroundTime = endServiceTime - (eventTime);
					totalTurnAroundTime += turnAroundTime;
				}
```

```
                        }

                        flagTemp >>= 1;
                        deviceNumber++;
                        } //End of if for flag temp
            } //End of If flag
} //End of While
} //End of Contro()

/*************************************************************************\
* Input : None                                                            *
* Output: None                                                            *
* Function: This must print out the number of Events buffered not yet     *
*          not yet processed (Server() function not yet called)           *
\*************************************************************************/
void BookKeeping(void){
printf("\n >>>>>> Done\n");
int totalGeneratedEvents = totalServicedEvents + totalMissedEvents;
fprintf(stderr,"Total Generated Events: %d\n", totalGeneratedEvents);
fprintf(stderr,"Total Unserviced Events: %d\n", totalMissedEvents);
fprintf(stderr,"Total Serviced Events: %d\n", totalServicedEvents);
fprintf(stderr,"Average percentage of missed events: %10.6f%\n", ((float)totalMissedEvents/totalGeneratedEvents)*100);
fprintf(stderr,"Average response time: %10.6f\n",totalResponseTime/totalServicedEvents);
fprintf(stderr,"Average turn around time: %10.6f\n",totalTurnAroundTime/totalServicedEvents);
}
```