Lab 2: Group 6

Jordan Hutcheson

Walter Conway

Date: 3/18/2014

Your report must:

A.) State whether your code works.

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

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.) Open question (needs some reading/research): What should we do to minimize the average 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).), the average response time, and the average turnaround time. Contrast lab2 values with those measured in Lab1.

Lab 1 | Lab 2

NbrDevices	Lambda	Mu	A(%)Avg Missed Events		B(s) Avg Resp Time		C(s) Avg Turn Around Time	
2	2	10	0%	0.000000%	0.017888	0.001811	0.141611	0.141564
2	2	30	14.070352%	0.000000%	0.115837	0.001731	0.503532	0.635252
2	2	60	29.648241%	0.000000%	0.282049	0.001673	1.062917	1.949625
2	2	90	45.728642%	21.025640%	0.544151	0.002129	1.851571	3.181273
4	2	10	0.501253%	0.000000%	0.010474	0.003337	0.070144	0.076101
4	2	30	4.260652%	0.000000%	0.099507	0.002528	0.281448	0.309809
4	2	60	14.536341%	0.000000%	0.270637	0.002017	0.639912	0.943348
4	2	90	28.388748%	10.899182%	0.429728	0.002202	0.991665	2.409190
8	4	10	0.125156%	0.000000%	0.010733	0.003980	0.066509	0.067517
8	4	30	1.251565%	0.000000%	0.102370	0.004003	0.270114	0.271854
8	6	60	6.700379%	0.000000%	0.561381	0.005877	1.067083	1.435643
8	6	90	18.101267%	0.000000%	1.408877	0.006068	2.187743	6.085079

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:

As mu increases, the number of missed events remains largely unaffected until it comes across the 90 mu mark. At this point the amount of time that it takes to service is overwhelming when compared to the frequency of the events being generated. So, the service essentially takes so long that the devices overwrite and we cannot have them processed.

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 response time remains largely unaffected and you don't really see any huge fluctuations. From our results as mu changes the response time increases slightly.

Explanation of the variation of the turn-around time:

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

As mu increases, the turn-around time increases exponentially. It's fairly drastic in the way it jumps from say 1.4-6 with mu values of 60-90.

Compare the values (missed events, average response time, and average turnaround time) for Lab2 in comparison to Lab1.
Comment and explain the differences.

Lab2 vs Lab1

Average Missed Events:

Lab1:

As mu doubled our missed events would double throughout all of Lab1. The round robin implementation didn't cope well with the longer service times. Such that when the system is servicing an event and an event occurred while we were still serving an event then we would miss that event.

Lab2:

As mu increased with Lab2, the number of missed events largely didn't even occur on almost all of our tests. The number of missed events was drastically reduced. The only time we did experience missed events were when mu was at its highest values. Yet, even when it did occur, it was still more than half the number of missed events in Lab1. Oddly though we didn't experience any missed events when testing our 8 6 90 test.

Average Response Time:

Lab1:

Our response time for lab one suffered much worse with increasing values of Mu than Lab 2. The response time increased as the mu increased. Almost

every time mu doubled, the response time would double.

Lab2:

The response time for our project is greatly decreased because of the quick response by the IRH to quickly grab the value and en-queue them into our queue to await their processing in the Control() function. It's reduction is quite drastic.

Average Turnaround Time:

Lab1:

The average turnaround time was best in Lab 1. This is simply due to the way in which the events were addressed. We used round robin which service each event as they came in and if an event occurred while processing then that event was missed. But this way the turn-around time would be better since a device is not waiting around to be processed it is just missed if it occurs while another one is being serviced.

Lab2:

The average turnaround time was much long since when an event occurred simultaneously while another event is being serviced then that event is en-queued and when the system has time to service that event it is then de-queued and processed. In this way this allows better response time overall, but long turn-around time.

B.)

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

We could possibly try to implement priorities with the task code or a priorities with a focus on interrupts. We could also try to use a Real Time Operating System.

Source Code lab1.c

```
/*****************************************************************************\
* Laboratory Exercises COMP 3510                                              *
* Author: Jordan Hutcheson                                                    *
* Author: Walter Conway                                                       *
* Date  : March 18, 2013                                                      *
\*****************************************************************************/
/*****************************************************************************\
*                        Global system headers                                *
\*****************************************************************************/
#include "common.h"
/*****************************************************************************\
*                        Global data types                                    *
\*****************************************************************************/
/*****************************************************************************\
*                        Global definitions                                   *
\*****************************************************************************/
/*****************************************************************************\
*                       Global data structures                                *
\*****************************************************************************/
typedef struct node
{
        Event *event;
        float beginServiceTime;
        struct node *ptr;
} node_t;
struct queue{
        node_t * tail;
        node_t * head;
        int size;
};
/*****************************************************************************\
*                           Global data                                       *
\*****************************************************************************/
struct queue que;
float totalServiceTime = 0 ;
float totalResponseTime = 0;
float totalTurnAroundTime = 0;
int numberOfDevices = 0;
int totalEvents = 0;
int totalServicedEvents = 0;
int totalMissedEvents = 0;
int totalGeneratedEvents = 0;

int nextEventThatShouldBeServiced[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);
void InterruptRoutineHandlerDevice(void);
void BookKeeping();

Event *headelement(void);
void enq(Event *eventAddr);
Event *deq(void);
void empty(void);
void display(void);
int queuesize(void);
void create(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                                              *
* START CONTROL ROUTINE                                                       *
```

```
\*****************************************************************************/
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 (written by students)        *
 \*****************************************************************************/
void Control(void){
//Local Variables
create();
Event *currentEvent;
float turnAroundTime = 0;
int deviceNumber = 0;
int eventNumber = 0;
float eventTime = 0;
float begunTime = 0;
float endServiceTime = 0;
        while (1) {
                while(queuesize() != 0) {
                        //Storing current event's information to reference
                        {
                                currentEvent = deq();
                                eventNumber = currentEvent->EventID;
                                eventTime = currentEvent->When;
                                deviceNumber = currentEvent->DeviceID;
                        }
                        //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((nextEventThatShouldBeServiced[deviceNumber]) != eventNumber) {
                                totalMissedEvents += eventNumber - (nextEventThatShouldBeServiced[deviceNumber]);
                        }
                        //This Displays the event and records the begining service time.
                        //This also services the event and records the end of service time.
                        {
                                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.
                        {
                                totalServicedEvents++;
                                nextEventThatShouldBeServiced[deviceNumber] = (eventNumber) + 1;
                        }
                        //Calculation being done to determine the response time,
                        //total response time, turn around time and total turn around time.
                        {
                                turnAroundTime = endServiceTime - (eventTime);
                                totalTurnAroundTime += turnAroundTime;
                        }
                        //This finds out how many devices there are
                        //throughout the life cycle of the program.
                        if(deviceNumber > numberOfDevices) {
                                numberOfDevices = deviceNumber;
                        }
                }
        }
}
/*****************************************************************************\
* Input : None                                                              *
* Output: None                                                              *
* Function: This routine is run whenever an event occurs on a device        *
*         The id of the device is encoded in the variable flag              *
\*****************************************************************************/
void InterruptRoutineHandlerDevice(void){
totalGeneratedEvents++;
Event *currEvent;
float startTime = 0;
```

```
float eventTime = 0;
int tempFlags = Flags;
Flags = 0;
int count = 0;
        while(tempFlags != 0){
                if(tempFlags & 1){
                        currEvent = &BufferLastEvent[count];
                        eventTime = currEvent->When;
                        startTime = Now();
                        DisplayEvent('A'+count, currEvent);
                        enq(currEvent);
                        fprintf(stderr,"size of queue: %d\n",queuesize());
                        totalResponseTime += startTime - eventTime;
                }
        tempFlags >>= 1;
        count++;
        }
}
/*************************************************************************\
* 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){
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);
}
/* Create an empty queue */
void create()
{
        que.size = 0;
        que.head = NULL;
        que.tail = NULL;
}
/* Returns queue size */
int queuesize()
{
  return que.size;
}
/* Enqueing the queue */
void enq(Event *eventAddr)
{
        if (que.tail == NULL)
   {
      que.tail = (struct node *)malloc(1*sizeof(struct node));
      que.tail->ptr = NULL;
                que.tail->event = eventAddr;
                que.head = que.tail;
   }
   else
   {
                node_t *temp;
      temp=(struct node *)malloc(1*sizeof(struct node));
                que.tail->ptr = temp;
                temp->event = eventAddr;
      temp->ptr = NULL;
                que.tail = temp;
   }
        que.size++;
}
/* Displaying the queue elements */
void display()
{
        node_t *front1;
        front1 = que.head;

        if ((front1 == NULL) && (que.tail == NULL))
```

```
   {
      printf("Queue is empty");
      return;
   }
   while (front1 != que.tail)
   {
            DisplayEvent('Z',front1->event);
      front1 = front1->ptr;
   }
   if (front1 == que.tail)
     DisplayEvent('Z',front1->event);
}

/* Dequeing the queue */
Event *deq()
{
            node_t *front1;
            Event *tempEvent = NULL;
            front1 = que.head;
   if (front1 == NULL)
   {
      printf("\n Error: Trying to display elements from empty queue");
      return tempEvent;
   }
   else
      if (front1->ptr != NULL)
      {
         front1 = front1->ptr;
                                    tempEvent = que.head->event;
                                    free(que.head);
                                    que.head = front1;
      }
      else
      {
                                    tempEvent = que.head->event;
                                    free(que.head);
                                    que.head = NULL;
                                    que.tail = NULL;
      }
                        que.size--;
                        return tempEvent;
}
/* Returns the front element of queue */
Event *headelement()
{
            if ((que.head != NULL) && (que.tail != NULL))
                        return(que.head->event);
   else
      return 0;
}
/* Display if queue is empty or not */
void empty()
{
            if ((que.head == NULL) && (que.tail == NULL))
      printf("\n Queue empty");
   else
     printf("Queue not empty");
}
```