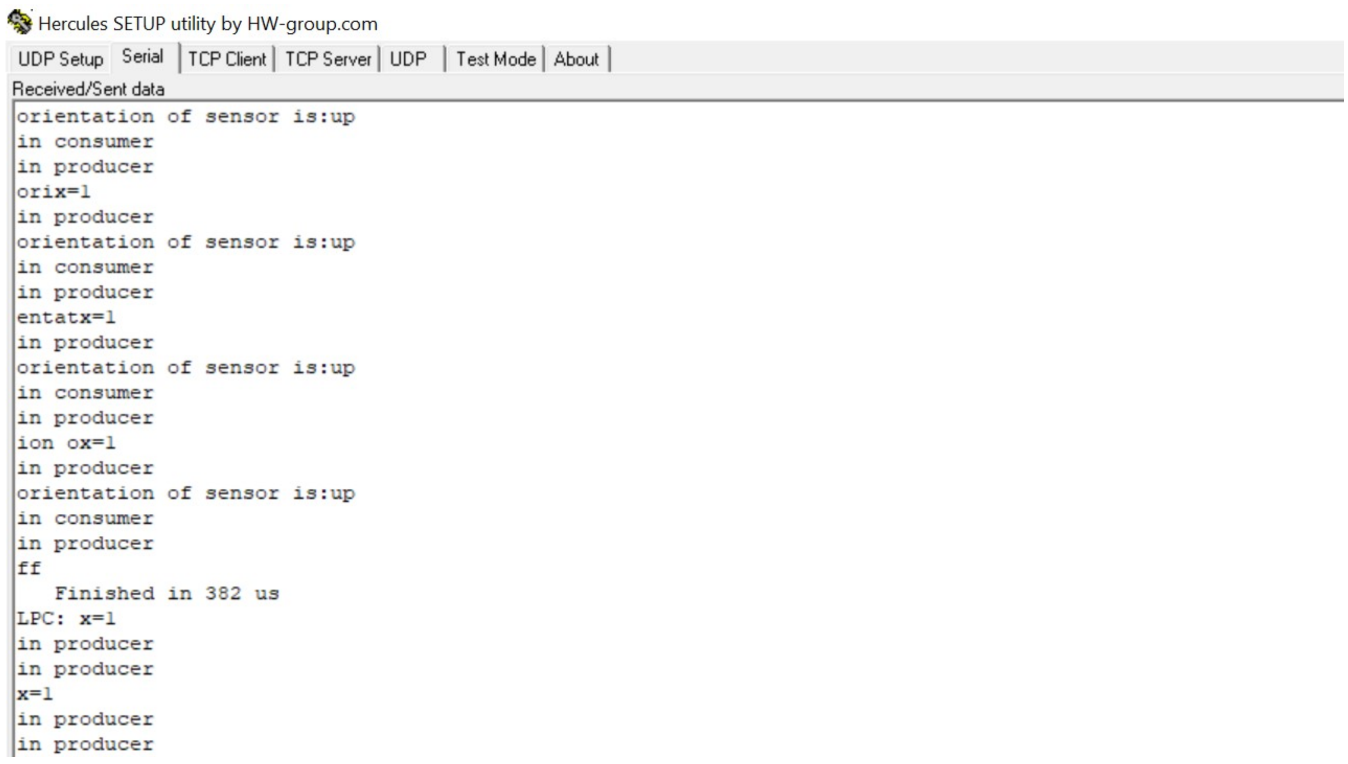
Kaustubh S. Jawalekar SJSU Id :- 012410944

Date:- 03/20/2018 **Assignment – 7:-Producer-Consumer using Queue** CMPE-244

Main:-

```
#include "FreeRTOS.h"
#include "task.h"
#include "tasks.hpp"
#include "uart0_min.h"
#include "LPC17xx.h"
#include "labgpio.hpp"
#include "stdio.h"
#include "semphr.h"
#include "utilities.h"
#include "printf_lib.h"
#include "labspi.hpp"
#include "labadc.hpp"
#include "mutex"
#include "labUART.hpp"
#include "io.hpp"
#include "string.h"

/*Assignment 7 - Producer-consumer */

QueueHandle_t q;
TaskHandle_t pTask;
TaskHandle_t cTask;
CommandProcessor cp;

enum orientation_t  //enumeration for orientation
{
    invalid,
    up,
    down,
    left,
    right
};

int acc_in()    //accepts accelerometer input
{
    int x=0;
    if(AS.getZ()>1000  && AS.getX()>-100 && AS.getX()<100 &&
AS.getY()>-100 && AS.getY()<100)
        x=1;
    else if(AS.getZ()<0  && AS.getX()>-100 && AS.getX()<100 &&
AS.getY()>-100 && AS.getY()<100)
        x=2;
    else if(AS.getY()<0  && AS.getX()<0 && AS.getZ()>-100 &&
AS.getZ()<150)
        x=3;
    else if(AS.getZ()>0  && AS.getX()<250 && AS.getX()<0 &&
AS.getY()>800 && AS.getY()<1200)
        x=4;
    else
        x=0;
//    printf("x:%d\n",AS.getX());
//    printf("y:%d\n",AS.getY());
//    printf("z:%d\n",AS.getZ());
    return x;
}

void printp()   //producer task print
{
    printf("in producer\n");
}

void printc()   //consumer task print
{
    printf("in consumer\n");
}

void printop(int x) //print output orientation
{
    if(x==1)
        printf("orientation of sensor is:up\n");
    else if(x==2)
        printf("orientation of sensor is:down\n");
    else if(x==3)
        printf("orientation of sensor is:left\n");
    else if(x==4)
        printf("orientation of sensor is:right\n");
    else
        printf("invalid position\n");
}

CMD_HANDLER_FUNC(orientationCmd)    //Definition of the macro to
define command handler function
{
    // You can use FreeRTOS API or the wrapper resume() or suspend()
methods
    if (cmdParams == "on")
    {
        vTaskResume(cTask);
    }
    else
    {
        vTaskSuspend(cTask);
    }
    return true;
}

void producer(void *p) /* LOW priority */
{
  int x = 0;
  while (1)
  {
    x = acc_in();
    printf("x=%d\n",x);
    printp();
    // This xQueueSend() will internally switch context over to the
"consumer" task
    // because it is higher priority than this "producer" task
    // Then, when the consumer task sleeps, we will resume out of
xQueueSend()
    // and go over to the next line
    xQueueSend(q, &x, 0);
    printp();
    vTaskDelay(1000);
  }
}

void consumer(void *p) /* HIGH priority */
{
  int x;
  while(1)
  {
    printc();
    xQueueReceive(q, &x, portMAX_DELAY);
    printop(x);
  }
}

int main()
{
  q = xQueueCreate(10, sizeof(int));
  scheduler_add_task(new terminalTask(3));
  xTaskCreate(producer,"p_task",1024,NULL,1,&pTask);
  xTaskCreate(consumer,"c_task",1024,NULL,2,&cTask);
  scheduler_start();
  vTaskStartScheduler();
  return 0;
}
```



```
orientation of sensor is:up
in consumer
in producer
orix=1
in producer
orientation of sensor is:up
in consumer
in producer
entatx=1
in producer
orientation of sensor is:up
in consumer
in producer
ion ox=1
in producer
orientation of sensor is:up
in consumer
in producer
ff
   Finished in 382 us
LPC: x=1
in producer
in producer
x=1
in producer
in producer
```

Explanation:-

Consumer task at higher priority:-

In above code, the consumer task is at higher priority than producer task. Hence, it will run before the producer task.However, the task receives an item if any item produced on the queue handle. The xQueueReceive() function doesn’t find any item produced on the queue handle and hence it makes task sleep until any item is produced on the queue handle.

As soon as the consumer task sleeps, producer task wakes up and produces an item on the queue handle through xQueueSend() function. The item produced on queue handle wakes up consumer task which is at higher priority and waiting for the item to be produced on the queue handle.

The consumer task receives item, completes the task function and again sleeps for 1 second as per vTaskDelay(1000). His resumes producer task, it functions the rest of the task and sleeps.

Both the tasks at same priority:-

If both the tasks were at same priority, producer task would have been executed first and produce the item on the queue handle. It would have been functioned further to complete the whole task function and then sleep at vTaskDelay(1000);

Then consumer task would have woken up, received queued item on the queue handle, and functioned the rest of the task effectively sleeping on delay function.

Hence, just because the consumer task is at higher priority, it is waiting for an item on queue handle and as soon as the producer produces item on queue handle, it functions.

This is a great application of queues in inter task communication.

Q.1) What if you use ZERO block time while sending an item to the queue, will that make any difference?

Ans. :-

The blocking time of Queue send function is used in the case of full queue. If queue is full and blocking time is over, the queue won't be producing an item on the handle.

In our program, the item on queue handle is getting consumed as soon as it is produced. Hence, the queue in this case will never be full.

Hence, blocking time 0 or 1000 won't make any difference in this case.

Q.2) What is the purpose of the block time during xQueueReceive() ?
Ans.:-

xQueueReceive() function checks if any item is available on queue handle for the blocking time specified in the argument. The function waits for the blocking time stated and if any item is produced on the handle during that duration, it will receive the item. If it doesn't find any item on the handle throughout that duration, it won't receive anything.

Hence, if we provide potMAX_delay as blocking time, the function will wait till any item is produced on the queue handle.

If blocking time provided is 0, the function checks the handle once and moves on to function the further task.

Terminal.cpp

```
#include <stdio.h>          // printf

#include <string.h>         // memset() etc.


#include "FreeRTOS.h"

#include "semphr.h"


#include "uart0.hpp"        // Interrupt driven UART0 driver

#include "nrf_stream.hpp"
```

```
#include "lpc_sys.h"        // Set input/output char functions

#include "utilities.h"      // PRINT_EXECUTION_SPEED()

#include "handlers.hpp"     // Command-line handlers


#include "file_logger.h"

#include "io.hpp"           // Board IO

#include "tasks.hpp"


#include "c_tlm_var.h"

#include "c_tlm_comp.h"

#include "c_tlm_stream.h"

#include "c_tlm_binary.h"




#define MAX_COMMANDLINE_INPUT   128               ///< Max characters
for command-line input

#define CMD_TIMEOUT_DISK_VARS   (2 * 60 * 1000)  ///< Disk variables
are saved if no command comes in for this duration




terminalTask::terminalTask(uint8_t priority) :

        scheduler_task("terminal", 1024*4, priority),

        mCmdIface(2), /* 2 interfaces can be added without memory
reallocation */
```

```
        mCmdProc(24), /* 24 commands can be added without memory
reallocation */

        mCommandCount(0), mDiskTlmSize(0), mpBinaryDiskTlm(NULL),

        mCmdTimer(CMD_TIMEOUT_DISK_VARS)

{

    /* Nothing to do */

}


bool terminalTask::regTlm(void)

{

    #if SYS_CFG_ENABLE_TLM

    return
(TLM_REG_VAR(tlm_component_get_by_name(SYS_CFG_DEBUG_TLM_NAME),
mCommandCount, tlm_uint) &&


TLM_REG_VAR(tlm_component_get_by_name(SYS_CFG_DEBUG_TLM_NAME),
mDiskTlmSize, tlm_uint));

    #else

    return true;

    #endif

}


bool terminalTask::taskEntry()

{

    /* remoteTask() creates shared object in its init(), so we can
get it now */

    CommandProcessor &cp = mCmdProc;


    // System information handlers
    cp.addHandler(taskListHandler, "info",    "Task/CPU Info.  Use
'info 200' to get CPU during 200ms");

    cp.addHandler(memInfoHandler,  "meminfo", "See memory info");

    cp.addHandler(healthHandler,   "health",  "Output system
health");

    cp.addHandler(timeHandler,     "time",    "'time' to view time.
'time set MM DD YYYY HH MM SS Wday' to set time");



    // File I/O handlers:

    cp.addHandler(catHandler,    "cat",   "Read a file.  Ex: 'cat
0:file.txt' or "

                                          "'cat 0:file.txt -noprint'
to test if file can be read");

    cp.addHandler(cpHandler,     "cp",    "Copy files from/to
Flash/SD Card.  Ex: 'cp 0:file.txt 1:file.txt'");

    cp.addHandler(dcpHandler,    "dcp",   "Copy all files of a
directory to another directory.  Ex: 'dcp 0:src 1:dst'");

    cp.addHandler(lsHandler,     "ls",    "Use 'ls 0:' for Flash, or
'ls 1:' for SD Card");

    cp.addHandler(mkdirHandler,  "mkdir", "Create a directory. Ex:
'mkdir test'");

    cp.addHandler(mvHandler,     "mv",    "Rename a file. Ex: 'rm
0:file.txt 0:new.txt'");

    cp.addHandler(newFileHandler,"nf",    "Write a new file. Ex: 'nf
<file.txt>");

    cp.addHandler(rmHandler,     "rm",    "Remove a file. Ex: 'rm
0:file.txt'");



    // Misc. handlers

    cp.addHandler(i2cIoHandler,   "i2c",   "'i2c read 0x01 0x02
<count>' : Reads <count> registers of device 0x01 starting from
0x02\n"
```

```
                                                    "'i2c write 0x01 0x02
0x03'   : Writes 0x03 to device 0x01, reg 0x02\n"

                                                    "'i2c discover' :
Discovers all I2C devices on the BUS");

#if TERMINAL_USE_CAN_BUS_HANDLER

    CMD_HANDLER_FUNC(canBusHandler);

    cp.addHandler(canBusHandler,  "canbus", "'canbus init' :
initialize CAN-1\n"

                                                     "'canbus filter <id>' :
Add 29-bit ID fitler\n"

                                                     "'canbus tx <msg id>
<len> <byte0> <byte1> ...' : Send CAN Message\n"

                                                     "'canbus rx <timeout in
ms>' : Receive a CAN message\n"

                                                     "'canbus registers' :
See some of CAN BUS registers");

#endif



    cp.addHandler(storageHandler,  "storage",  "Parameters: 'format
sd', 'format flash', 'mount sd', 'mount flash'");

    cp.addHandler(rebootHandler,   "reboot",   "Reboots the
system");

    cp.addHandler(logHandler,      "log",      "'log <hello>': log
an info message\n"

                                                        "'log flush'  : flush
the logs\n"

                                                        "'log status' : get
status of the logger\n"

                                                        "'log enable print
debug/info/warn/error' : Enables logger calls to printf\n"

                                                        "'log disable print
debug/info/warn/error': Disables logger calls to printf\n"
```

```
                                            );

    cp.addHandler(learnIrHandler,  "learn",    "Begin to learn IR
codes for numbers 0-9");

    cp.addHandler(wirelessHandler, "wireless", "Use 'wireless' to
see the nested commands");


    /* Firmware upgrade handlers

     * Please read "netload_readme.txt" at ref_and_datasheets
directory.

     */

    CMD_HANDLER_FUNC(getFileHandler);

    CMD_HANDLER_FUNC(flashProgHandler);

    CMD_HANDLER_FUNC(orientationCmd);


    cp.addHandler(getFileHandler,   "file",  "Get a file using
netload.exe or by using the following protocol:\n"

                                          "Write buffer: buffer
<offset> <num bytes> ...\n"

                                          "Write buffer to file:
commit <filename> <file offset> <num bytes from buffer>");

    cp.addHandler(flashProgHandler, "flash", "'flash <filename>'
Will flash CPU with this new binary file");


    cp.addHandler(orientationCmd,  "orientation", "Two options:
'orientation on' or 'orientation off'");


    #if (SYS_CFG_ENABLE_TLM)

    cp.addHandler(telemetryHandler, "telemetry", "Outputs registered
telemetry: "
```

```
                                                        "'telemetry save' :
Saves disk tel\n"
                                                        "'telemetry ascii'
: Prints all telemetry in human readable format\n"
                                                        "'telemetry <comp.
name> <name> <value>' to set a telemetry variable\n"
                                                        "'telemetry get
<comp. name> <name>' to get variable value\n");
    #endif


    // Initialize Interrupt driven version of getchar & putchar
    Uart0& uart0 = Uart0::getInstance();
    bool success = uart0.init(SYS_CFG_UART0_BPS, 32,
SYS_CFG_UART0_TXQ_SIZE);
    uart0.setReady(true);
    sys_set_inchar_func(uart0.getcharIntrDriven);
    sys_set_outchar_func(uart0.putcharIntrDriven);


    /* Add UART0 to command input/output */
    addCommandChannel(&uart0, true);


    #if TERMINAL_USE_NRF_WIRELESS
    do {
        NordicStream& nrf = NordicStream::getInstance();
        nrf.setReady(true);
        addCommandChannel(&nrf, false);
    } while(0);
    #endif
    #if SYS_CFG_ENABLE_TLM

    /* Telemetry should be registered at this point, so initialize
the binary

     * telemetry space that we periodically check to save data to
disk

     */

    tlm_component *disk =
tlm_component_get_by_name(SYS_CFG_DISK_TLM_NAME);

    mDiskTlmSize = tlm_binary_get_size_one(disk);

    mpBinaryDiskTlm = new char[mDiskTlmSize];

    if (success) {

        success = (NULL != mpBinaryDiskTlm);

    }


    /* Now update our copy of disk telemetry */

    tlm_binary_get_one(disk, mpBinaryDiskTlm);

    #endif


    /* Display "help" command on UART0 */

    STR_ON_STACK(help, 8);

    help = "help";

    mCmdProc.handleCommand(help, uart0);


    return success;

}


bool terminalTask::run(void* p)

{
    printf("LPC: ");
    cmdChan_t cmdChannel = getCommand();

    // If no command, try to save disk data (persistent variables)
    if (!cmdChannel.iodev) {
        if (saveDiskTlm()) {
            /* Disk variables saved to disk */
        }
        else {
            puts("");
        }
    }
    else {
        // Set our references to the IO channel and the command str
(just for covenience sake)
        CharDev& io = *(cmdChannel.iodev);
        str& cmd = *(cmdChannel.cmdstr);

        if (cmd.getLen() > 0)
        {
            PRINT_EXECUTION_SPEED()
            {
                ++mCommandCount;
                mCmdProc.handleCommand(cmd, io);

                /* Send special chars to indicate end of command
output
```

```
                 * Usually, serial terminals will ignore these chars
                 */
                const char endOfTx[] = TERMINAL_END_CHARS;
                for (unsigned i = 0; i < sizeof(endOfTx); i++) {
                    io.putChar(endOfTx[i]);
                }
            }

            cmd.clear();
            io.flush();
        }
    }

    return true;
}

bool terminalTask::saveDiskTlm(void)
{
    bool changed = false;

    #if SYS_CFG_ENABLE_TLM
    tlm_component *disk =
tlm_component_get_by_name(SYS_CFG_DISK_TLM_NAME);

    /* Size of telemetry shouldn't change */
    if (0 == mDiskTlmSize || mDiskTlmSize !=
tlm_binary_get_size_one(disk)) {
        return changed;
    }

    if (!tlm_binary_compare_one(disk, mpBinaryDiskTlm))
    {
        changed = true;
        puts("Disk variables changed...");

        FILE *file = fopen(SYS_CFG_DISK_TLM_NAME, "w");
        if (file) {
            // Only update variables if we could open the file
            tlm_binary_get_one(disk, mpBinaryDiskTlm);

            tlm_stream_one_file(disk, file);
            fclose(file);

            puts("Changes saved to disk...");
            LOG_SIMPLE_MSG("Disk variables saved to disk");
        }
    }
    #endif

    return changed;
}
```

```
void terminalTask::handleEchoAndBackspace(cmdChan_t *io, char
newChar)

{

    /* Pointers to reduce too many -> references */

    const bool echo = io->echo;

    str *pStr = io->cmdstr;

    CharDev *iodev = io->iodev;


    // Backspace 1 char @ terminal and erase last char of string

    if (echo && '\b' == newChar) {

        if(pStr->getLen() > 0) {

            iodev->put("\b ");

            pStr->eraseLast(1);

        }

        else {

            // Send "Alert" sound to terminal because we can't
backspace

            const char bellSound = '\a';

            newChar = bellSound;

        }

    }

    else if ('\n' != newChar && '\r' != newChar) {

        *(pStr) += newChar;

    }


    if (echo) {

        iodev->putChar(newChar);
    }
}

void terminalTask::addCommandChannel(CharDev *channel, bool echo)
{
    cmdChan_t input;
    input.iodev = channel;
    input.echo = echo;
    input.cmdstr = new str(MAX_COMMANDLINE_INPUT);
    mCmdIface += input;
}

terminalTask::cmdChan_t terminalTask::getCommand(void)
{
    if (0 == mCmdIface.size()) {
        vTaskDelayMs(1000);
        cmdChan_t noIface = { NULL, NULL };
        return noIface;
    }

    unsigned int idx = 0;
    cmdChan_t ret = mCmdIface[0];
    char c = 0;

    do
    {
        /* Get a single char from one of the input sources */
        const TickType_t ticksBefore = xTaskGetTickCount();
        bool gotChar = false;

        for (idx = 0; idx < mCmdIface.size(); idx++)
        {
            if (mCmdIface[idx].iodev->isReady() &&
mCmdIface[idx].iodev->getChar(&c, 0))
            {
                ret = mCmdIface[idx];
                handleEchoAndBackspace(&ret, c);

                mCmdTimer.reset();
                gotChar = true;
                break;
            }
        }

        /* If no interfaces are ready, we will not sleep in the
previous loop,
         * so we don't want to hog the CPU, so just delay here by
one tick
         */
        if (!gotChar && xTaskGetTickCount() == ticksBefore) {
            vTaskDelay(2);
        }

        /* Guard against command length too large */
        if (ret.cmdstr->getLen() >= ret.cmdstr->getCapacity() - 1) {
            break;
        }

        /* If no command and timer expires, then use this time to do
something else
         * and return back to the caller but set iodev to NULL
         */
        if (mCmdTimer.expired()) {
            mCmdTimer.reset();
            ret.iodev = NULL;
            break;
        }
    } while (c != '\n');

    return ret;
}
```