**Header: Grade: FOR /20 DES /40 EXP /30 ORI /10 / TOT \_100**

*Lab # 3*

*CS 5390 Summer 2017, Date of Submission: 07/21/17*

*Students Names: Adeel Malik (amalik@utep.edu)*

*Instructor: Yadira Jacquez*

**Section 1: Effort: 12 hours**

- Planning and preparation: 1 hours

- Experiment: 7 hours (on simulator)

- Report writing: 1 hours

**Section 2: Objectives**

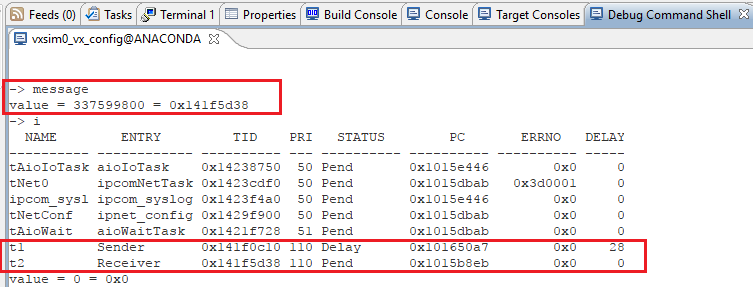
The objective of this experiment is to understand how message queues work and how tasks can send and receive messages to queues.

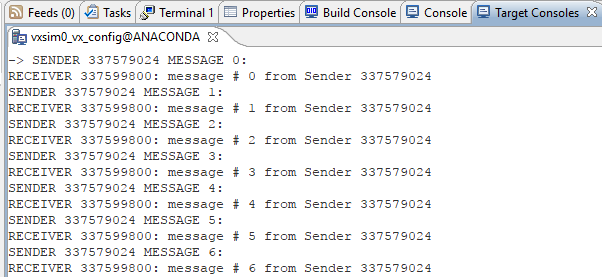
# Section 3: Procedures and Results

# Part A:

**A1.** Execute the function message from command line. Record and observe the output.

**Answer:** Running the command message, creates a maximum capacity FIFO message queue “mqId” and creates two tasks t1 Sender and t2 Receiver. One task sends message and the other receives it and its on going…

**Host shell:**

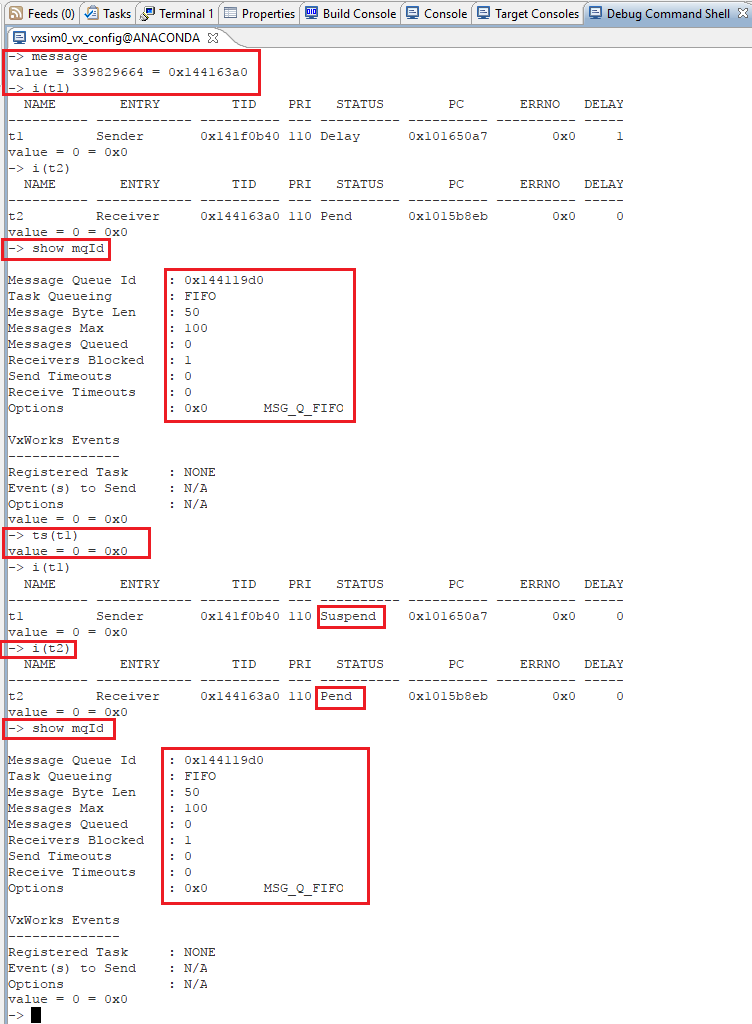
**Target console:** 

**A2.** Check the running tasks:

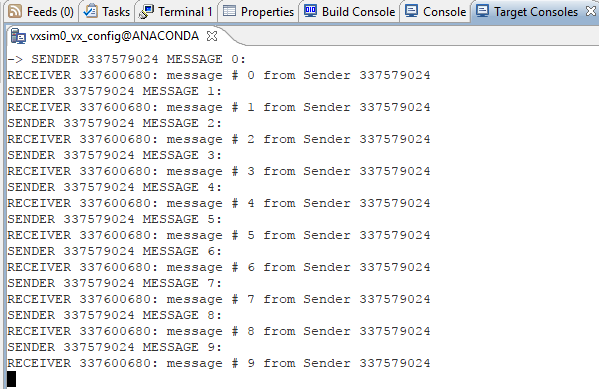
1. Suspend (and later resume) the sender and receiver tasks independently. Check the status of message queue (show mqId). Record the output and explain what happened.

**Answer:** Suspending t1 (Sender) stops message sending and the t2 (Receiver) goes to pending and waits for a message till sender again sends it. The printing loop pauses on the Console. Status of message queue however, remains the same.

**Host shell:**

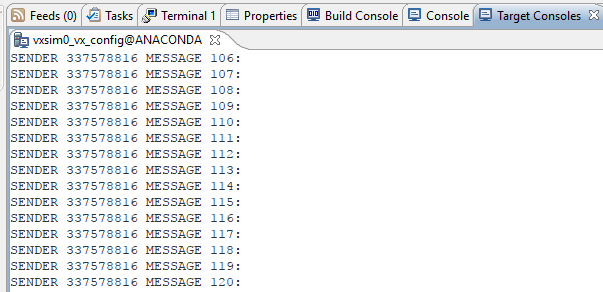


**Target console:**

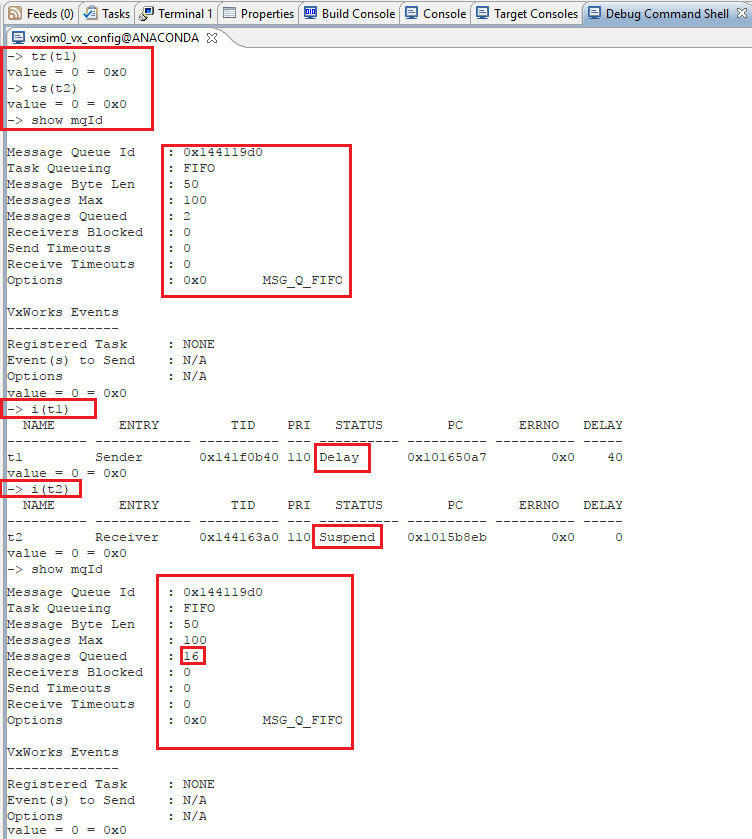


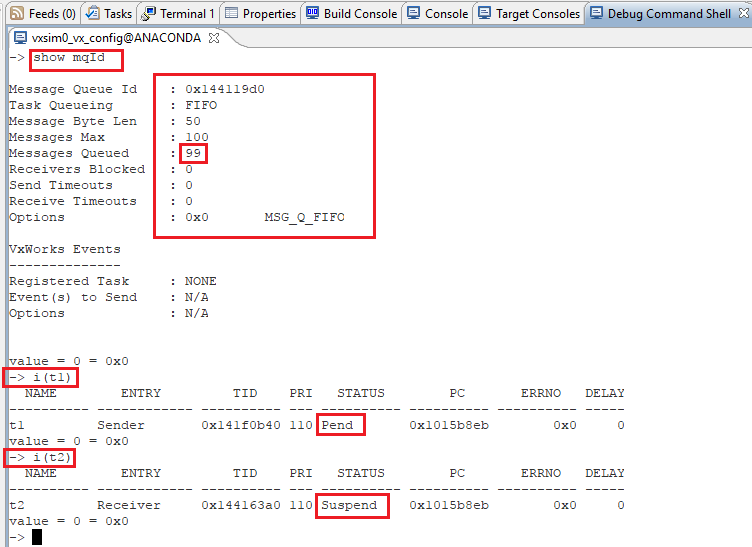
Resuming t1 (sender) and suspending t2 (receiver) stops the receiver task from receiving the messages and the sender keeps on sending messages till the queue is full, in this case when the queue will have 99 messages and then t1 (sender) also gets to pending or blocked state.

**Target console:**

****

**Host Shell:**

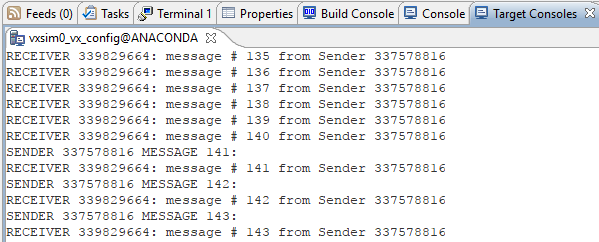




**b.** Suspend (and later resume) both tasks observing the tasks status. Check the status of message queue (show mqId). Record the output and explain what happened.

**Answer:** As soon as t1 (sender) is blocked t2 (receiver) goes to pending state since there is nothing to receive. When t2 (receiver) is blocked, t2 (sender) keeps on sending till the message queue is full and then sender also gets blocked. As soon as the receiver resumes, it gets all messages from the message queue and sender starts sending again. Images above and below explain the same since I have already observed the tasks along with the queue as well.

**Target console:**

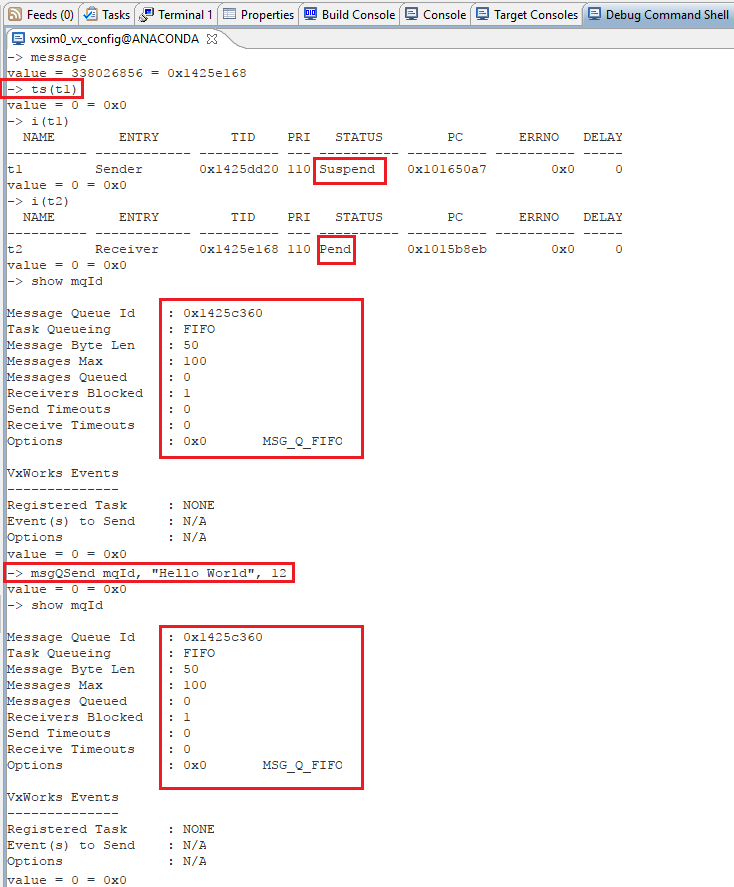


**A3.** With the sender(s) task(s) suspended and the receiver(s) pending, send a message from the shell command line. Show the command you used to send the message. How have message queue changed?

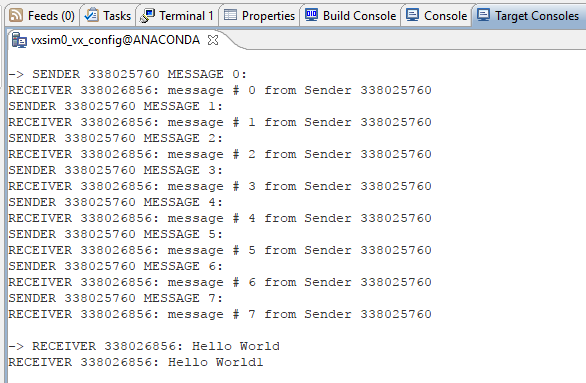
**Answer:** When sender is suspended and receiver is pending, I sent a message using the command

“**msgQSend mqId, "Hello World", 12**” and the receiver immediately receives the message and displays it. The message queue however, remains the same.

**Host shell:**



**Target console:**

****

**A4.** Resume the Sender task and suspend the Receiver task. Show how to receive a message from the shell. Create an “urgent” message on the message queue. Repeat your command above to receive the message from the shell (HINT: you will need to create a buffer from the shell to store the received message). Explain and comment on the results?

**Answer:**

Executed the following commands:

message //running the program

ts(t1) //suspending t1 (sender)

ts(t2) //suspending t2 (receiver)

tr(t1) //resuming t1 (sender)

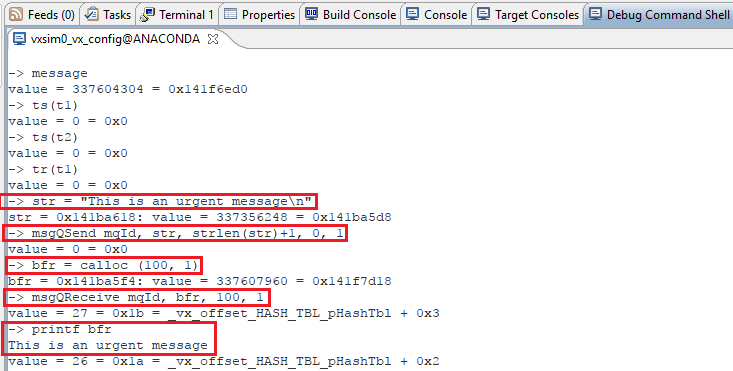
str = “This is an Urgent Message\n” //urgent message str

msgQSend mqId, str, strlen(str)+1, 0, **1** //sending as an urgent message (last parameter 1 is priority)

bfr = calloc (100, 1) //creating receive buffer

msgQReceive mqId, bfr, 100, 1 //reading the queue

printf bfr //displaying the result

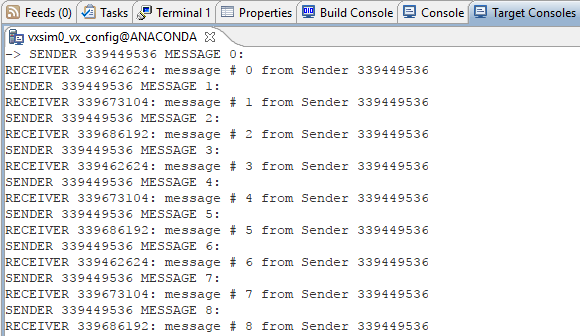
**Host shell:**

**A5.** Experiment with spawning 2-3 receiver tasks and/or 2-3 new sender tasks. Analyze the output, task status and the queue status while suspending/resuming the tasks. Check the status of the message queue while experimenting as in the point above. Describe explicitly how do you do that and what did you learned.

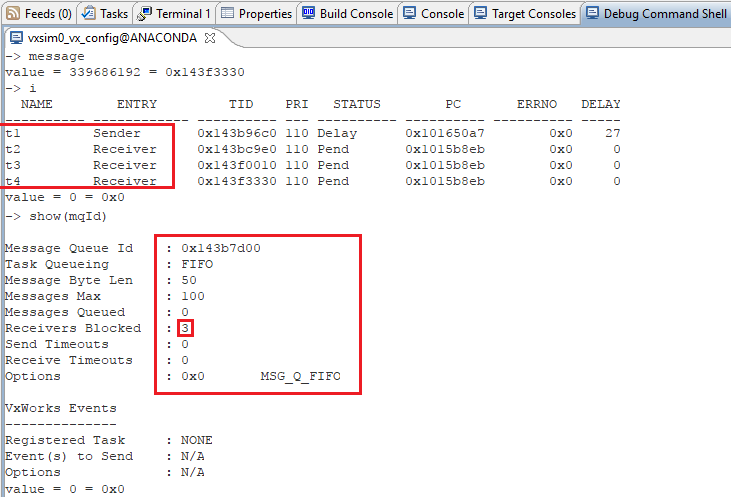
**Answer:**

When we have 2 receiving tasks, both of them receive message one by one from the sender task. When we have 3 receiving tasks, they receive one by one, one after another. Queue status shows that the receivers blocked by the queue are 3.

**Target console:**



**Host shell:**



**Adding new receiving task**

if((receiverId2 = taskSpawn("t3",110,0x100,2000,(FUNCPTR)Receiver,0,0,0,0,0,0,0, 0,0,0)) == ERROR)

printf("taskSpawn taskThree failed\n");

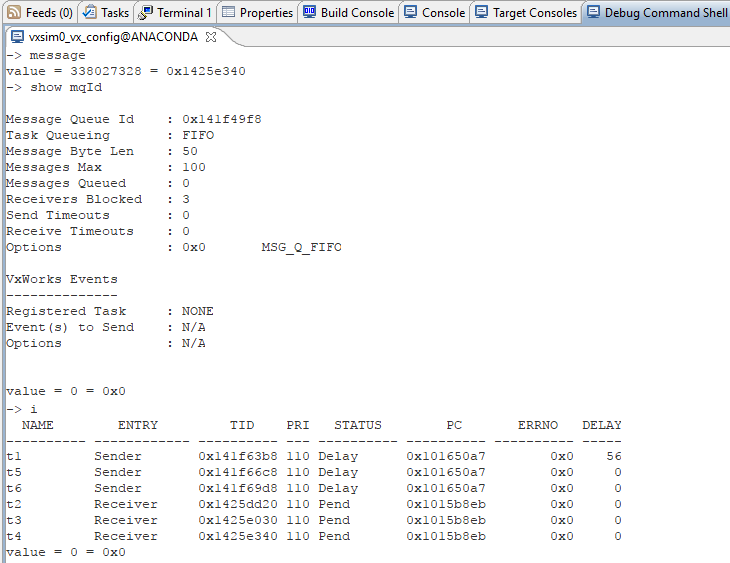
**Adding new sender task**

if((senderId2 = taskSpawn("t5",110,0x100,2000,(FUNCPTR)Sender,0,0,0,0,0,0,0, 0,0,0)) == ERROR)

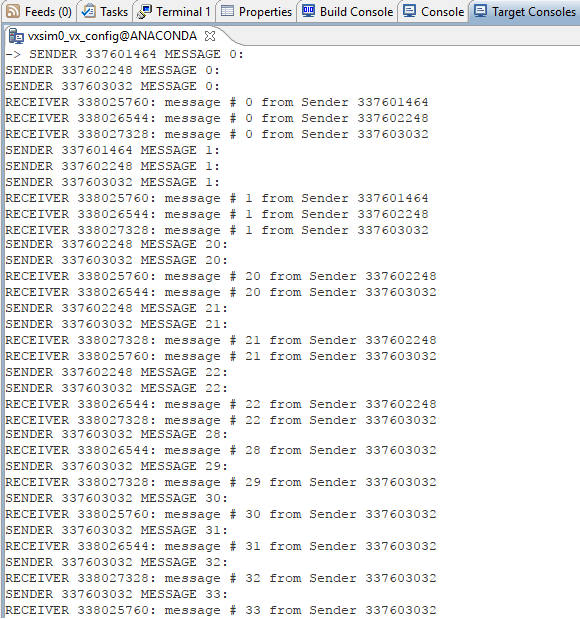
printf("taskSpawn taskFive failed\n");

When new sender task is added, sender tasks send messages one after another, and receiving tasks also receive messages one after another. 2 Sender tasks send messages first and 2 of the receiving receive on FIFO basis. When one more sender task is added, 3 sender tasks send message one after another and then 3 receiving tasks receive messages on after another. When a sender task is suspended then the receiving tasks receive messages by turns.

Host shell:



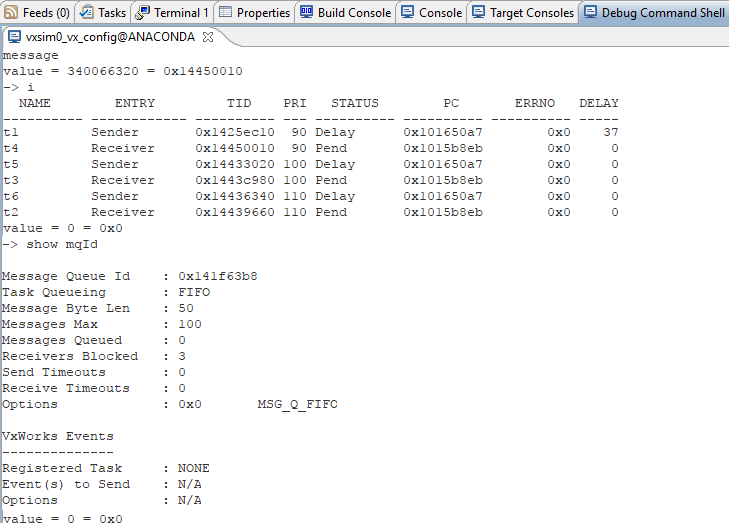
Target console:

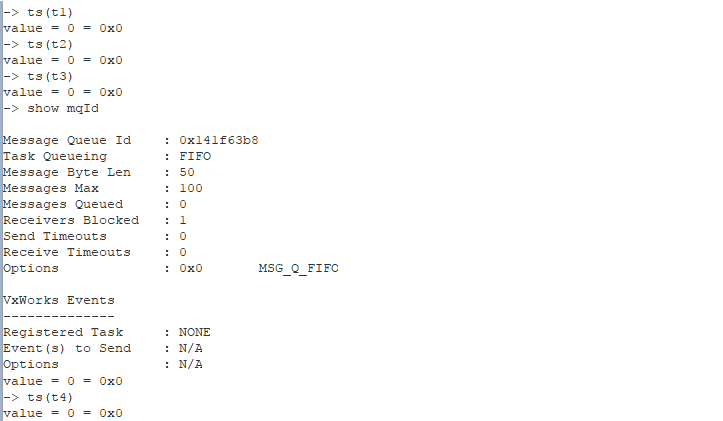


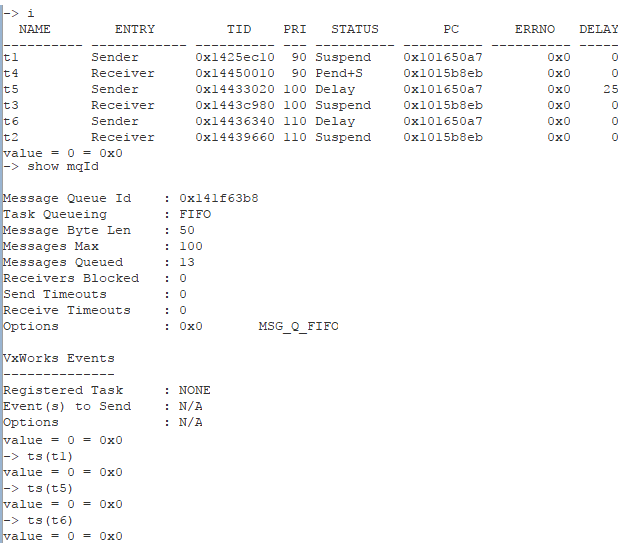
Note: The above results are in the case of same priority levels for each task. When the priority is changed, then the sending and receiving pattern also changes based on the priority, high priority task sends and receives task before the other tasks.

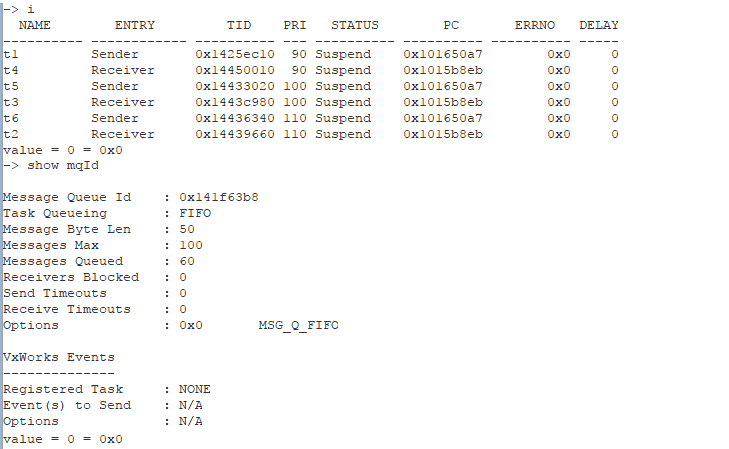
When receiving tasks are blocked, the queue status reduces number of blocked tasks. When all receiving tasks are blocked the queue starts filling towards max capacity. As soon as receiving tasks are released, they receive messages from queue.

**Host shell:**

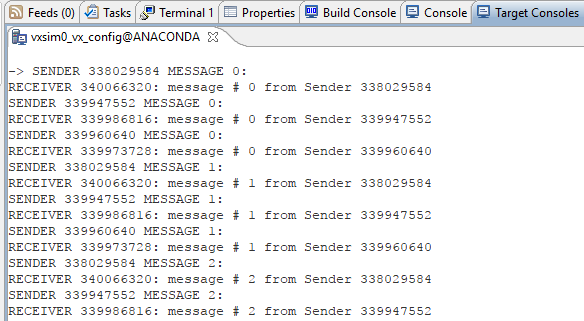








**Target console:**



# Part B:

**B1.** Write a new program to implement Client-Server scenario with three Clients sending messages to a Server. Have the three Clients each send messages at a different rate (20, 40, 60 and ticks). The message shall be a string with the client identifier and a message number (e.g. "clientId-#"). The Server shall respond to each client with the same string appending "received at - <timestamp>" (where the <timestamp> is the time of getting the message). Each message shall be displayed three times: (a) upon creation by the client, (b) upon receiving by the server and addition of the timestamp by the server (c) upon receiving back by the client (adding second timestamp to identify the time of receiving the message back to the client). Include the code, excerpts of output, and explain the results of executing your program.

**NOTE1:** Consider spawning three threads and use only one client function. You may consider using semaphores if you need any synchronization or mutual exclusion. Also think about thread priorities. Describe how the program runs to prove that it works as specified.

**NOTE2:** See appendix for help with the timestamp implementation.

**Answer:** I have added message receiving section in the else part of “Sender” method of the code provided and renamed it as client and a sending section in the “Receive” method’s else section and renamed it as server. On top of these, I have added a mutex semaphore with WAIT\_FOREVER to ensure message delivery. Additionally, I have formatted the message format as requested to clearly show the message synchronization. Message task with lower ticks repeats quickly then the other tasks.

The code is as below:

#include <vxWorks.h> /\* Always include this as the first thing in every program \*/

#include <stdio.h> /\* Always include this as the first thing in every program \*/

#include <unistd.h>

#include <time.h> /\* we use clock\_gettime \*/

#include <taskLib.h> /\* we use tasks \*/

#include <sysLib.h> /\* we use sysClk... \*/

#include <semLib.h> /\* we use semaphores \*/

#include <logLib.h> /\* we use logMsg rather than printf \*/

#include <msgQLib.h> /\* we use message queues \*/

/\* function prototypes \*/

void Client(int);

void Server(void);

/\* defines \*/

#define MAX\_MESSAGES 100

#define MSG\_SIZE 50

/\* globals \*/

MSG\_Q\_ID mqId;

SEM\_ID semMQ;

struct timespec timer;

int sender1, sender2, sender3, server\_task;

/\* Function to create semaphore and message queue \*/

void createSemQueue (){

mqId = msgQCreate(MAX\_MESSAGES, MSG\_SIZE, MSG\_Q\_PRIORITY);//Max capacity priority message que

semMQ = semBCreate(0, 1); //Binary semaphore FIFO and Full

}

/\* Run program \*/

void message(){

createSemQueue();

timer.tv\_sec = 0;

clock\_settime(CLOCK\_REALTIME, &timer);

//Client one at 20 ticks

if((sender1 = taskSpawn("ClientOne", 110, 0x100, 2000, (FUNCPTR)Client, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0)) == ERROR)

printf("taskSpawn ClientOne failed\n");

//Client two at 40 ticks

if((sender2 = taskSpawn("ClientTwo", 110, 0x100, 2000, (FUNCPTR)Client, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0)) == ERROR)

printf("taskSpawn ClientTwo failed\n");

//Client three at 60 ticks

if((sender3 = taskSpawn("ClientThree", 110, 0x100, 2000, (FUNCPTR)Client, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0)) == ERROR)

printf("taskSpawn ClientThree failed\n");

if((server\_task = taskSpawn("ServerOne", 110, 0x100, 2000, (FUNCPTR)Server, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)) == ERROR)

printf("taskSpawn Server failed\n");

}

/\* Client 1 task that writes to the message queue \*/

void Client(int delayTicks){

int i = 0;

while (1) {

char message[MSG\_SIZE];

semTake(semMQ, WAIT\_FOREVER);

sprintf(message, "Message#%d from Client#%d",i, taskIdSelf());

printf("Client#%d, Message: %d:\n",taskIdSelf(), i++); /\* print what is sent \*/

if(msgQSend(mqId, message, MSG\_SIZE, WAIT\_FOREVER, MSG\_PRI\_NORMAL) == ERROR){

printf("msgQSend failed on Client#%d.\n",taskIdSelf());

} else {

taskDelay(10);//delay to let the other tasks run and wait for incoming message

if (msgQReceive(mqId, message, MSG\_SIZE, WAIT\_FOREVER) == ERROR) {

printf("msgQReceive failed on Client#%d.\n", taskIdSelf());

} else {

clock\_gettime(CLOCK\_REALTIME, &timer);

printf("Client#%d received Server reply at %d Seconds and reply is: %s\n", taskIdSelf(), timer.tv\_sec, message);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

}

}

semGive(semMQ);

taskDelay(delayTicks);

}

}

/\* tasks that reads from the message queue \*/

void Server(){

while(1){

char message[MSG\_SIZE];

if (msgQReceive(mqId, message, MSG\_SIZE, WAIT\_FOREVER) == ERROR) {

printf("msgQReceive in Server failed\n");

} else {

clock\_gettime(CLOCK\_REALTIME, &timer);

printf("Server received: %s at %d Seconds.\n", message, timer.tv\_sec);

if(msgQSend(mqId, message, MSG\_SIZE, WAIT\_FOREVER, MSG\_PRI\_NORMAL) == ERROR){

printf("Server: Unable to SEND message.\n", taskIdSelf());

}

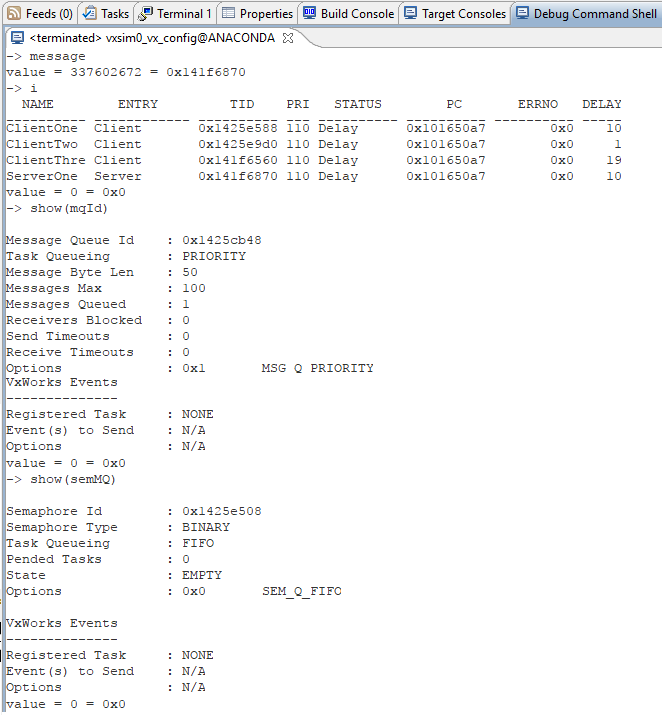
}

taskDelay(20);//Delay added to let the other tasks run…

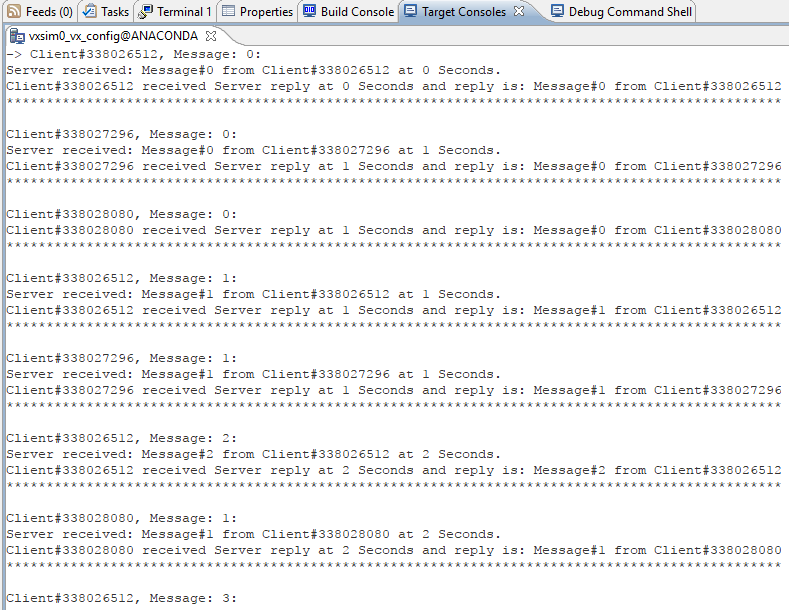
}

}

**Host shell:**



**Target console:**



**Section 4: Observations, Comments, and Lessons Learned**

I have learned about message queues, how they work, how to send and receive messages from message queues. I have used a binary semaphore to ensure message delivery between tasks. However, the server acknowledgment is missing randomly in between task synchronization. Due to shortage of time, I am unable to troubleshoot this part, otherwise program works fine.