**RTX Part 4 Document**

**rtx\_tezt.c  
  
void test1()**

* **Purpose –** This test case is to verify that set\_process\_priority(#,#) is working correctly.
* **Pseudocode –** Set the priority of process 1 to 1. Set the priority of process 2 to 2. Then, check the priority of process 2 using get\_process\_priority(#). If it is 2, output that test 1 passed to the debug terminal, and increment test\_results. If it is not 2, output that test 2 failed to the debug terminal.

**void test2()**

* **Purpose –** This test case is to verify that receive\_message() is working correctly.
* **Pseudocode –** Set a void pointer to the result of receive\_message(#). If msg\_type is equal to 99, the test passes. A pass message is output to the debug terminal and test\_results is incremented. If msg\_type does not equal 99, the test fails and a fail message is output to the debug terminal.  
    
  Next, check the value of test\_results to see how many tests passed. If test\_results equals 0, no test cases passed. This result is output to the debug terminal. If test\_results equals 1, one out of two test cases passed and this result is output to the debug terminal. Else, both test cases passed and a pass message is output to the debug terminal.  
    
  Call release\_processor().

**void test3()**

* **Purpose –** This function effectively unblocks test2() and sends it a message. This message is of type equal to 99. It gets run after test2() gets blocked when calling receive\_message().
* **Pseudocode –** Set a void pointer called msg to the result of request\_memory\_block(). Set msg\_type to msg offset by 8. Dereference msg\_type and set it to 99. Next, send a message w/ envelope msg to process 2.  
    
  Call release\_processor().

**rtx\_test\_stress**

**proc\_a()**

* **Purpose –** This is the first process in the stress test suite.
* **Pseudocode –** The %Z command gets registered to the keyboard decoder. A new s\_message structure named z is initialized its pointer is set to the result of request\_memory\_block(). z’s type is set to 2. z’s msg\_text is set to z + 4. strCopy() is called with Z being passed as an argument. This copies the message “Z” to the message block of s\_message z. Then send the message z to process 10 which is the KCD.  
    
  Inside an infinite while loop, receive message. s\_message p is set to receive\_message(0). A new void pointer called m is created, and is pointed at s\_message p. m is then incremented by 64. A char pointer called d is created, and is set to m. Check to see if the message was directed at process A from the UART. If d[0] is equal to ‘Z’, then the expected has occurred. The memory block is released and the process breaks out of the while loop. If not, the memory block is released and the loop continues.  
    
  Inside an infinite while loop, set up a message to send to process B. p is set to the result of request\_memory\_block(). p’s type is set to 4, which is type “count\_report”. Void pointer m is set to p. m is incremented by 64. int pointer d is set to m. d is dereferenced to equal num. Send the message pointed to by p to process B (p\_id 8). num is incremented by 1. release\_processor() is called.

**proc\_b()**

* **Purpose –** This is the second process in the stress test suite.
* **Pseudocode –** Create a void pointer called tmp. Looping infinitely, set tmp to the result of receive\_message(0). Send the message to process C (p\_id 9).

**proc\_c()**

* **Purpose –** This is the third process in the stress test suite.
* **Pseudocode –** Initialize the following structures and variables:
  + **s\_message\_queue** l\_msg\_queue
  + **s\_message\_queue\_item** l\_msg\_queue\_slots[NUM\_MEM\_BLKS]

Set the head and tail of l\_msg\_queue to 0, and set the number of slots in l\_msg\_queue to NUM\_MEM\_BLKS (50).

Using a for loop, iterate through l\_msg\_queue\_slots and set the data and next pointer at each index to 0. Initialize the structures s\_message \* p and s\_message \* q.

Looping infinitely, perform the following. Check if local queue is empty, and if it is empty wait for a message from the global queue. Check if the message type is of type “count\_report” (type == 4). Create a void pointer m and set it to p. Increment m by 64. Create a void pointer d and set it to m. Check if contents at msg\_data[0] are divisible by 20 by performing a modulus operation and checking for a 0 remainder. Send a message to the CRT (p\_id 11). To hibernate for 10 seconds, delay send itself a message with a delay of 10 seconds (10000ms). Inside an embedded infinite while loop, receive messages and check if the message is WakeUp10, signaling that hibernation is over. If a message is received that isn’t WakeUp10, add it to the local message queue to be dealt with later.  
  
Release memory block for p. Call release\_processor().