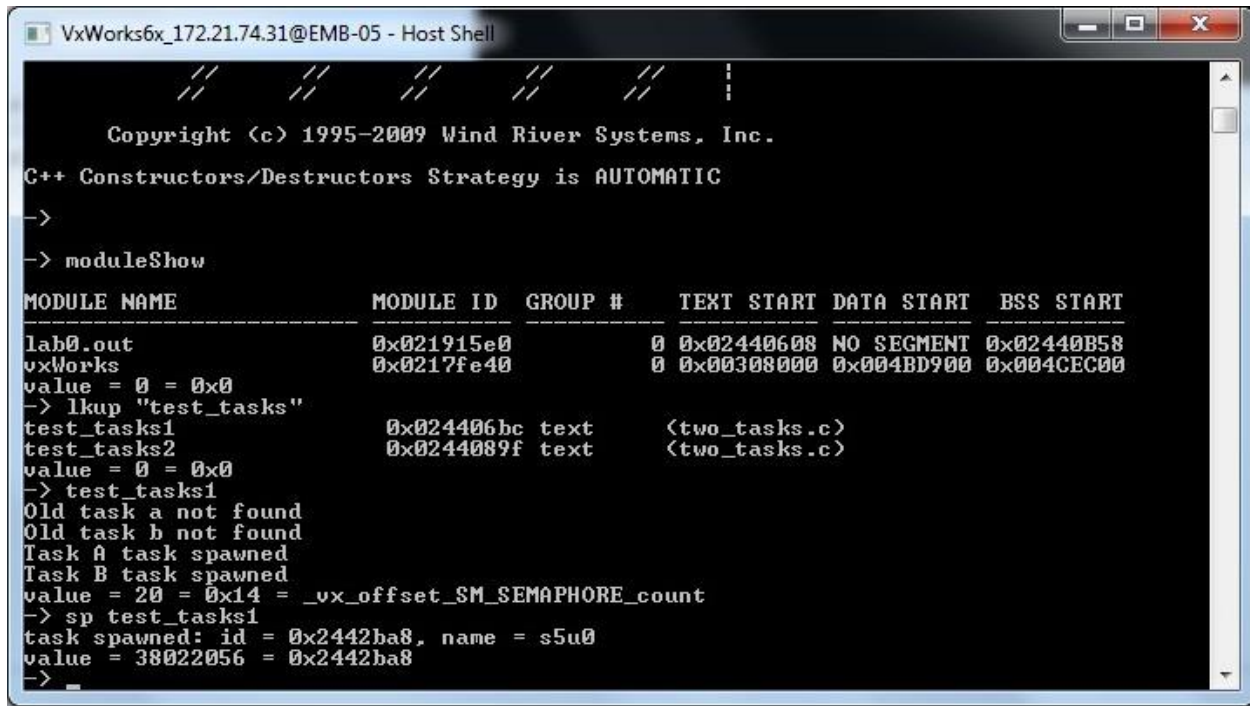


Q1).



```
VxWorks6x_172.21.74.31@EMB-05 - Host Shell

// // // // //
Copyright (c) 1995-2009 Wind River Systems, Inc.

C++ Constructors/Destructors Strategy is AUTOMATIC

->
-> moduleShow

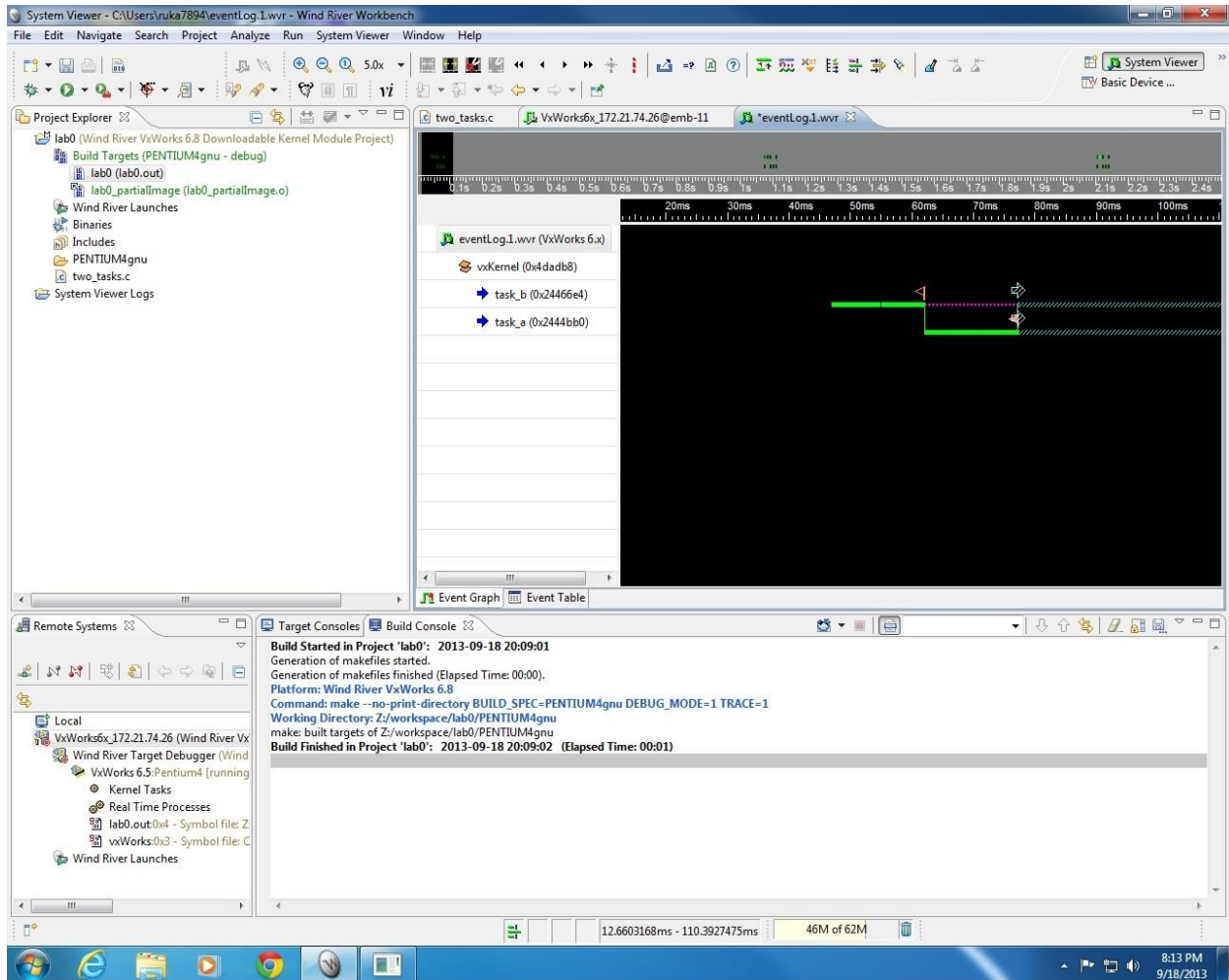
MODULE NAME          MODULE ID  GROUP #   TEXT START  DATA START  BSS START
-----
lab0.out             0x021915e0
vxWorks              0x0217fe40      0 0x02440608 NO SEGMENT 0x02440B58
value = 0 = 0x0
-> lkup "test_tasks"
test_tasks1          0x024406bc text    <two_tasks.c>
test_tasks2          0x0244089f text    <two_tasks.c>
value = 0 = 0x0
-> test_tasks1
Old task a not found
Old task b not found
Task A task spawned
Task B task spawned
value = 20 = 0x14 = _vx_offset_SM_SEMAPHORE_count
-> sp test_tasks1
task spawned: id = 0x2442ba8, name = s5u0
value = 38022056 = 0x2442ba8
->
```

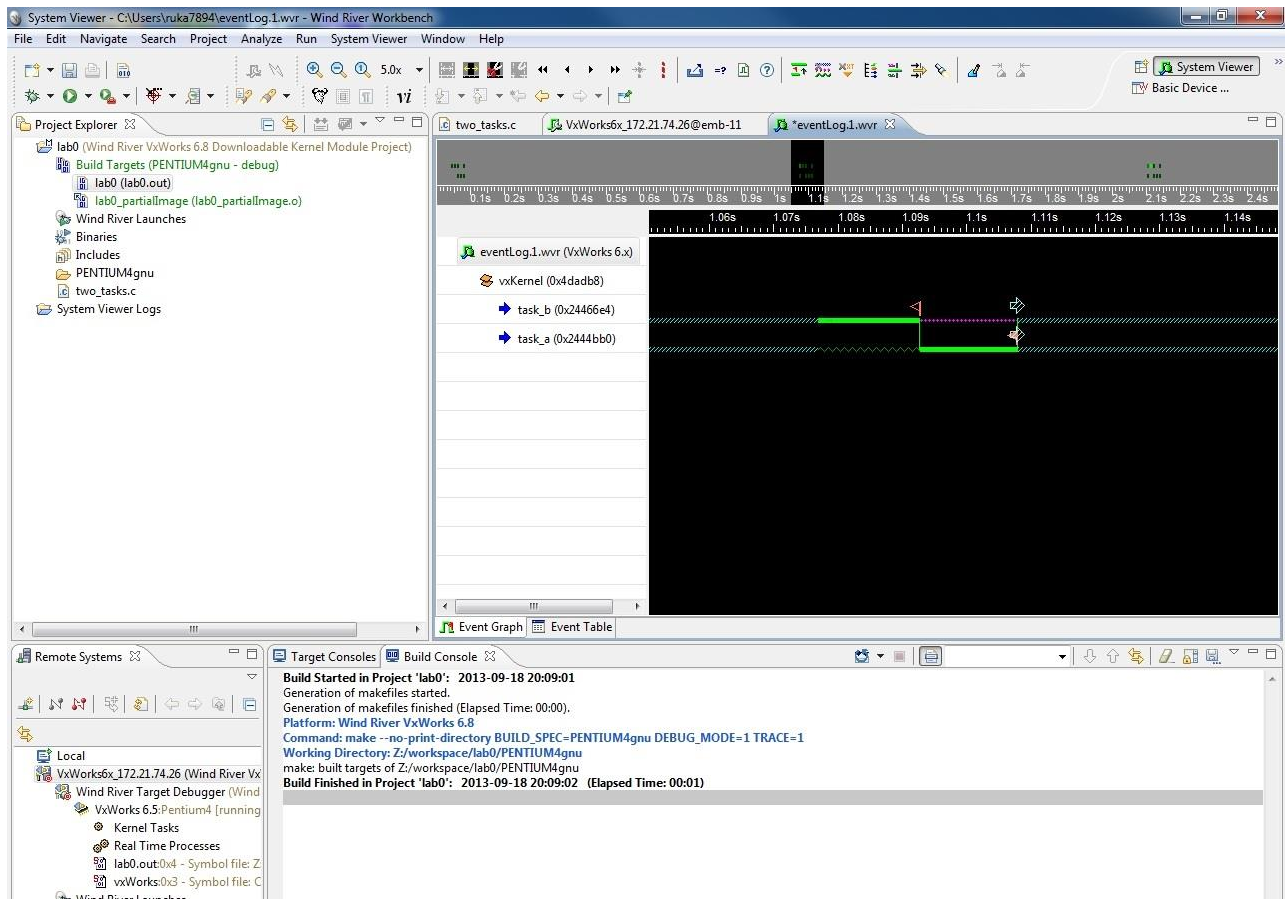
Q2).

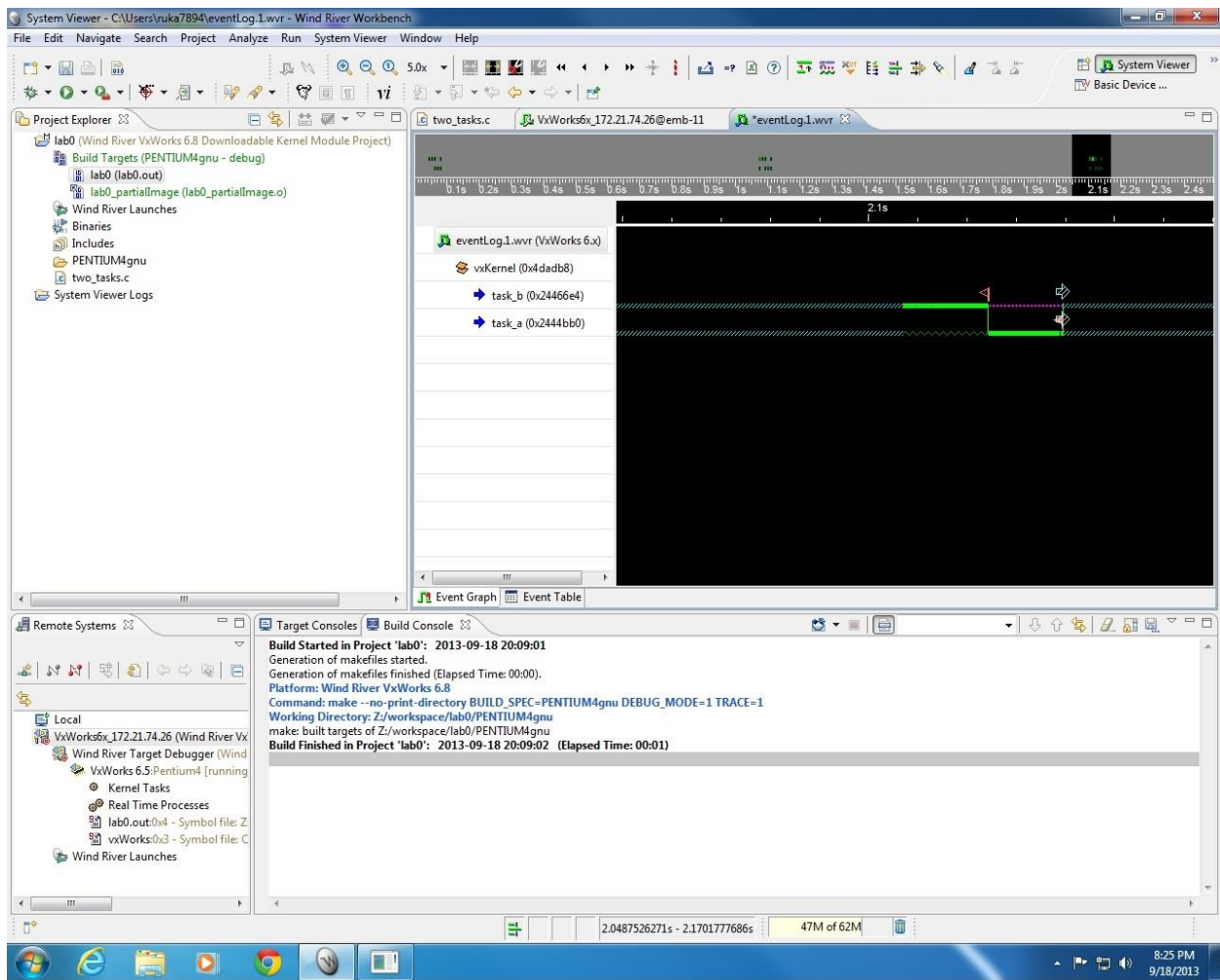
The function test_tasks1 and test_tasks2 initially set the system clock rate to 1000 ticks per second. then the old tasks task_a and task_b are deleted. Then a binary semaphore is created after which the tasks task_a and task_b are spawned. In test_tasks1(), task_b has a higher priority and hence executes first. The task_b then performs semTake which empties the semaphore after which task_b enters the pending state. As this happens, task_a starts execution and on completing the execution, it performs a semGive and at this point both the tasks are delayed. When the delay period is over, task_b returns execution while task_a enters ready state.

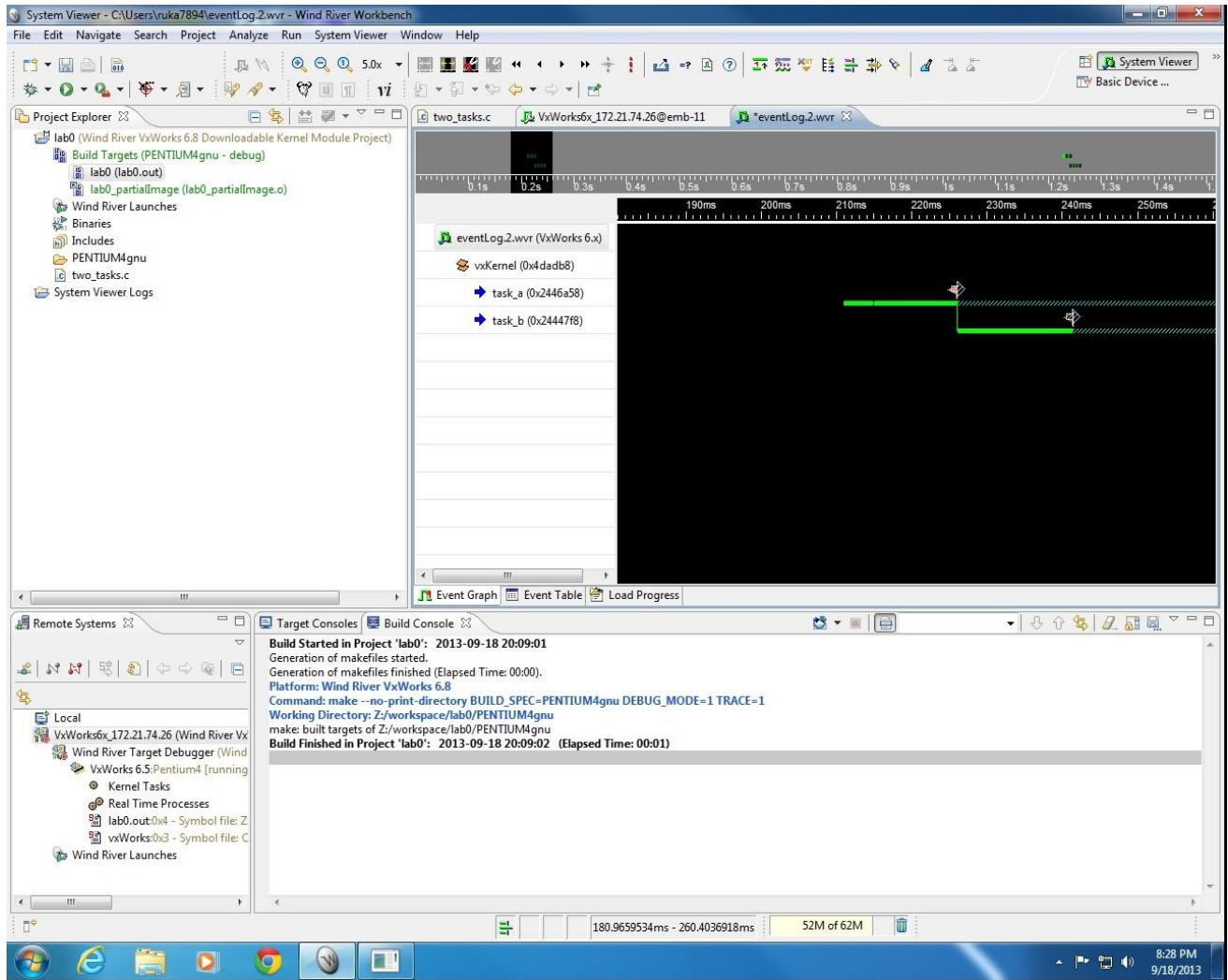
In test_tasks2, task_a has higher priority and hence executes first. After execution it performs a semGive and enters a delayed state. At this task_b starts execution and after execution performs semTake to empty the semaphore and enters the delayed state.

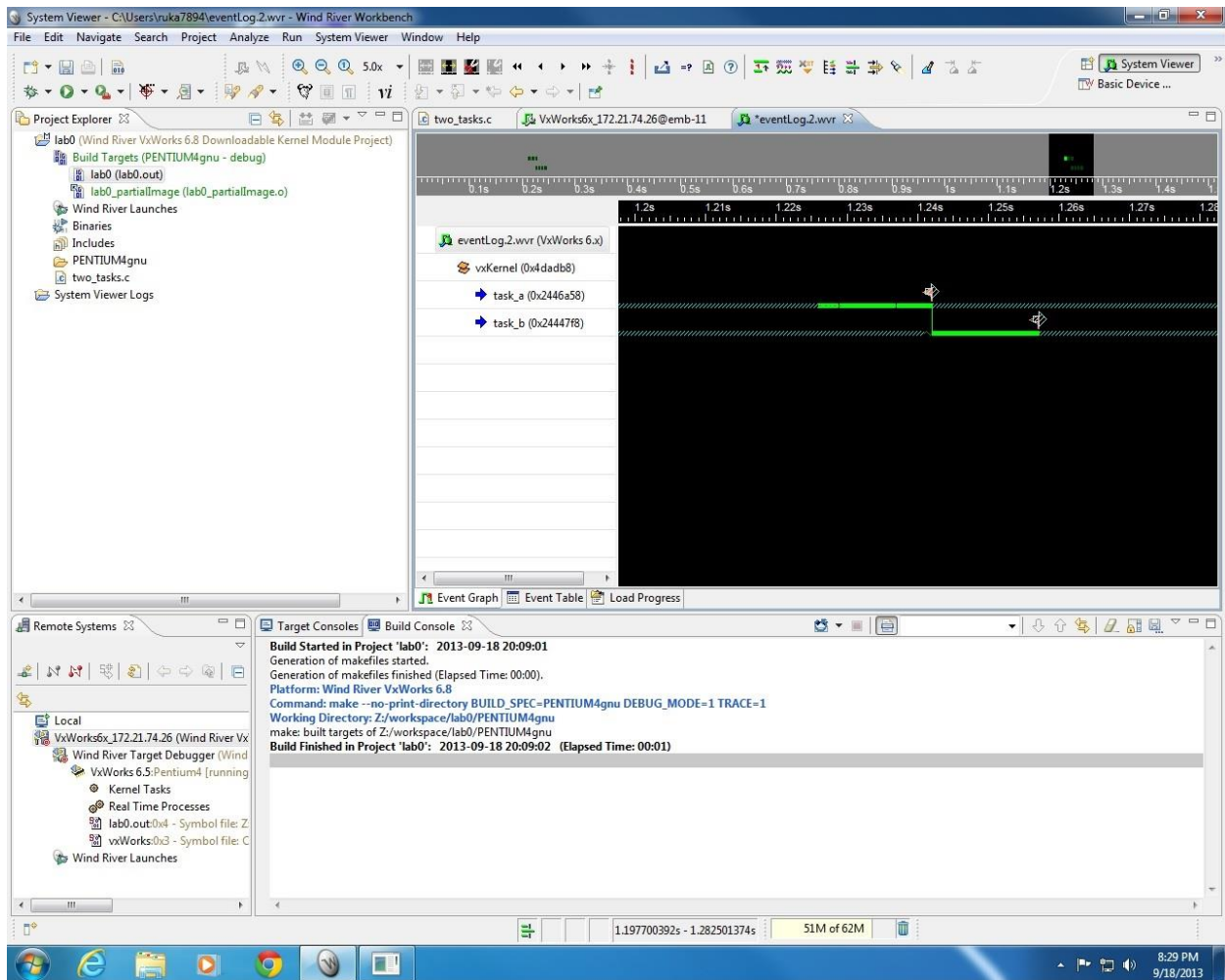
Q3).



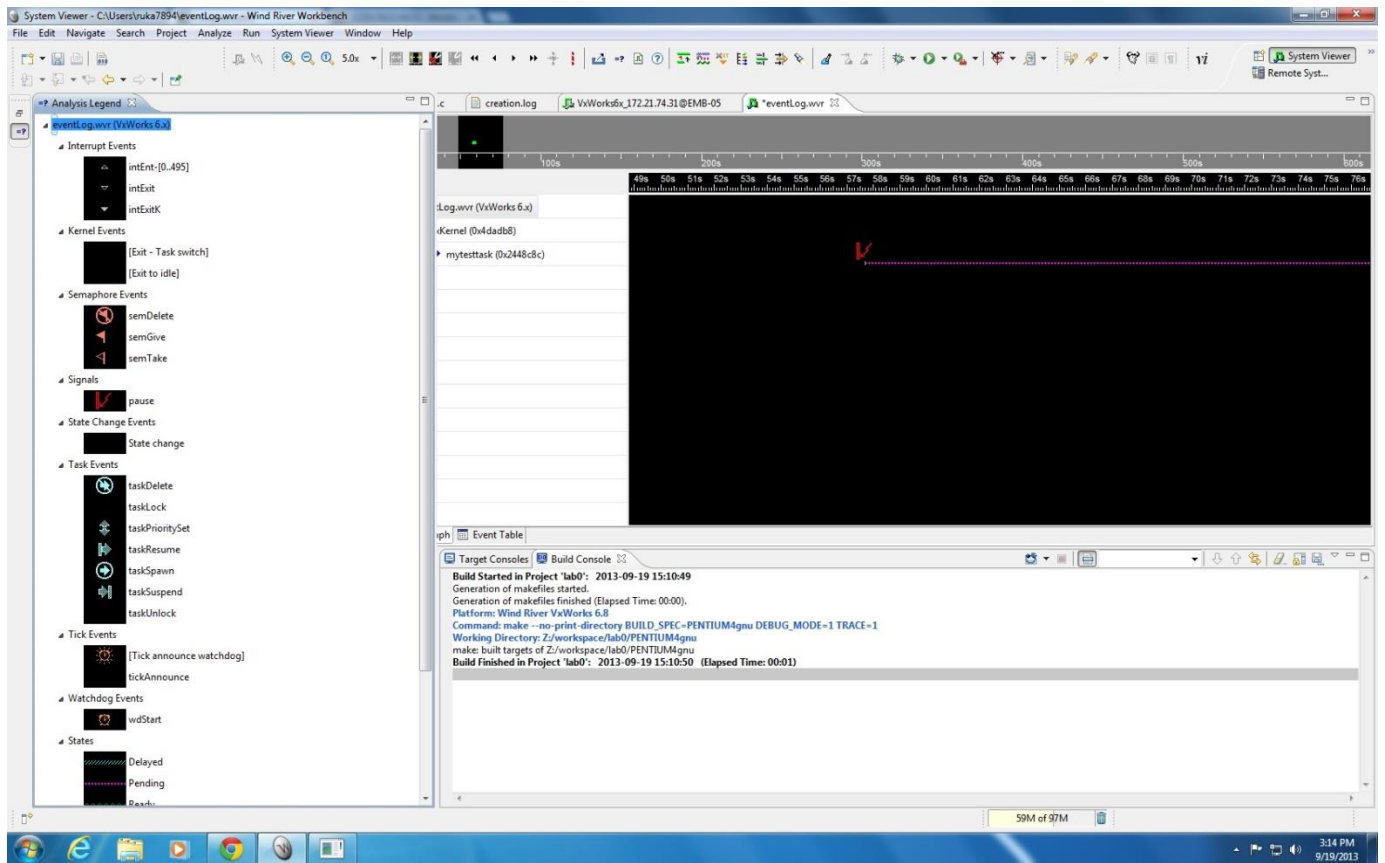








Q4).



VxWorks6x_172.21.74.31@EMB-05 - Host Shell

Old mytesttask deleted

mytesttask task spawned

value = 24 = 0x18 = _vx_offset_RBUFF_INFO_TYPE_maxBufs

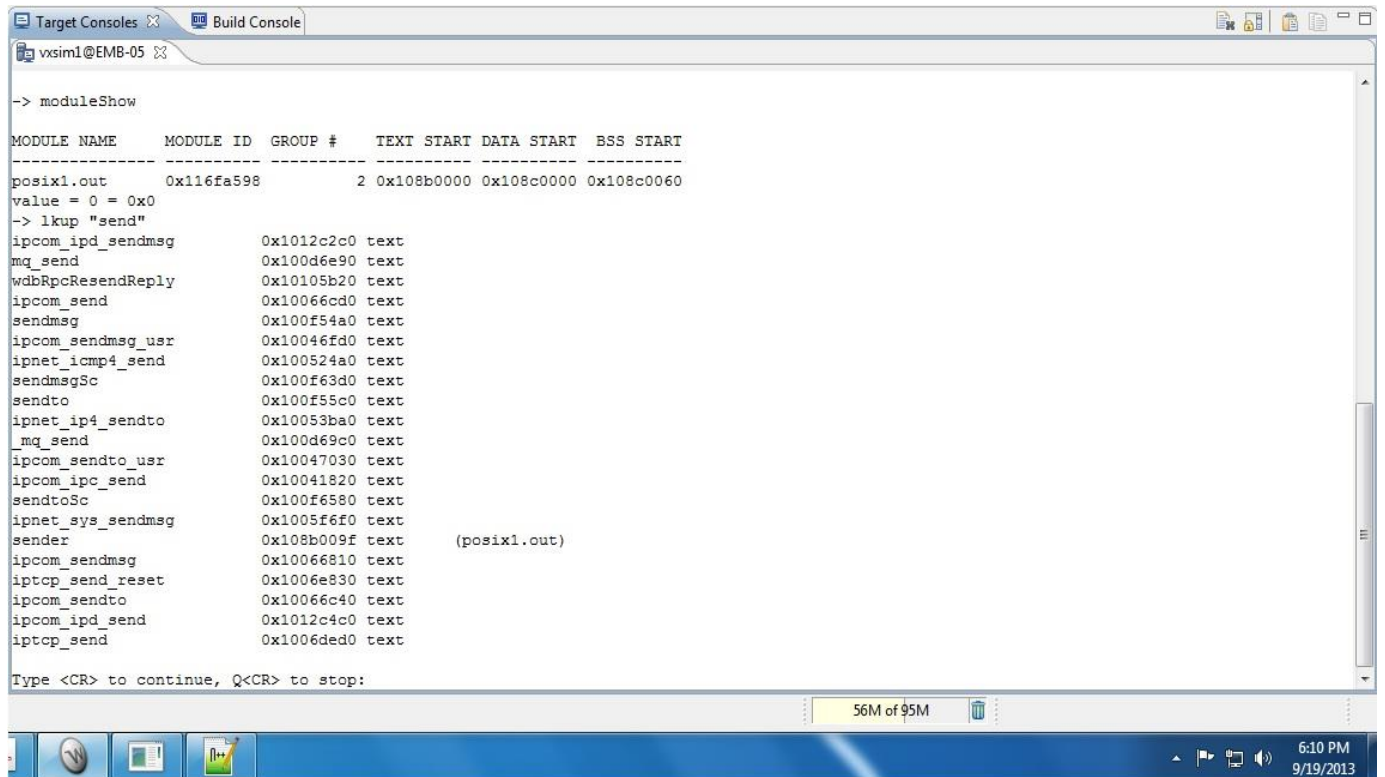
-> i

NAME	ENTRY	TID	PRI	STATUS	PC	ERRNO	DELAY
tJobTask	jobTask	0x242e2a0	0	Pend	0x4178bc	0x0	0
tExcTask	excTask	0x4eca24	0	Pend	0x4178bc	0x0	0
tLogTask	logTask	0x25ac228	0	Pend	0x415913	0x3d0001	0
tNbIoLog	nbIoLogServe	0x25ac5a0	0	Pend	0x4178bc	0x0	0
svFuncCall	funcCallWrap	0x170703c4	0	Pend	0x4178bc	0x0	0
svFuncCall	funcCallWrap	0x28d8790	0	Pend	0x4178bc	0x0	0
svFuncCall	funcCallWrap	0x28d84c8	0	Pend	0x4178bc	0x0	0
tShell10	shellTask	0x243achc	1	Pend	0x4178bc	0x0	0
tWdbTask	wdbTask	0x287d9b8	3	Ready	0x4178bc	0x3d0001	0
tBulkClnt	bulkClientTh	0x286d858	5	Pend	0x417ea2	0x0	0
tUfiClnt	ufiClientThr	0x287d370	5	Pend	0x417ea2	0x0	0
tErfTask	erfServiceTa	0x25b2020	10	Pend	0x417ea2	0xc4000a	0
tDevConn	vxbDevConnec	0x2428204	11	Suspend	0x41bba2	0x0	0
tAtaSvc0	ataXhdServic	0x25b2cb4	50	Pend	0x417ea2	0x0	0
tNetTask	netTask	0x2430010	50	Ready	0x41777e	0x0	0
ipcom_sysl	ipcom_syslog	0x2432c78	50	Pend	0x417ea2	0x0	0
ipnetd	ipnetd	0x2436970	50	Pend+T	0x4178bc	0x3d0004	0
mytesttask	mytesttask	0x244abb0	90	Pend	0x389a48	0x0	0
t1	threadHead	0x243890c	100	Pend	0x415913	0x0	0
BULK_CLASS	threadHead	0x2438ce8	100	Pend	0x415913	0x0	0
BULK_CLASS	threadHead	0x286d020	100	Pend	0x415913	0x0	0
CBI_UFI_CL	threadHead	0x286dc50	100	Pend	0x415913	0x0	0
CBI_UFI_CL	threadHead	0x287d020	100	Pend	0x415913	0x0	0
tWvRBufMgr	wvRBufMgr	0x2442d10	100	Pend	0x417ea2	0x0	0

value = 0 = 0x0

->

Q5).



The screenshot shows a debugger window with two tabs: 'Target Consoles' and 'Build Console'. The 'Build Console' tab is active, displaying the output of a 'moduleShow' command. The output lists various modules and their properties, including module names, IDs, group numbers, and start addresses for text, data, and BSS segments. The module 'posix1.out' is highlighted, and its properties are shown. Below the table, the command 'value = 0 = 0x0' is entered, followed by 'lkup "send"', which lists various system call symbols and their addresses. The status bar at the bottom indicates '56M of 95M' memory usage and the system time '6:10 PM 9/19/2013'.

```
-> moduleShow
```

MODULE NAME	MODULE ID	GROUP #	TEXT START	DATA START	BSS START
posix1.out	0x116fa598	2	0x108b0000	0x108c0000	0x108c0060

```
value = 0 = 0x0
```

```
-> lkup "send"
```

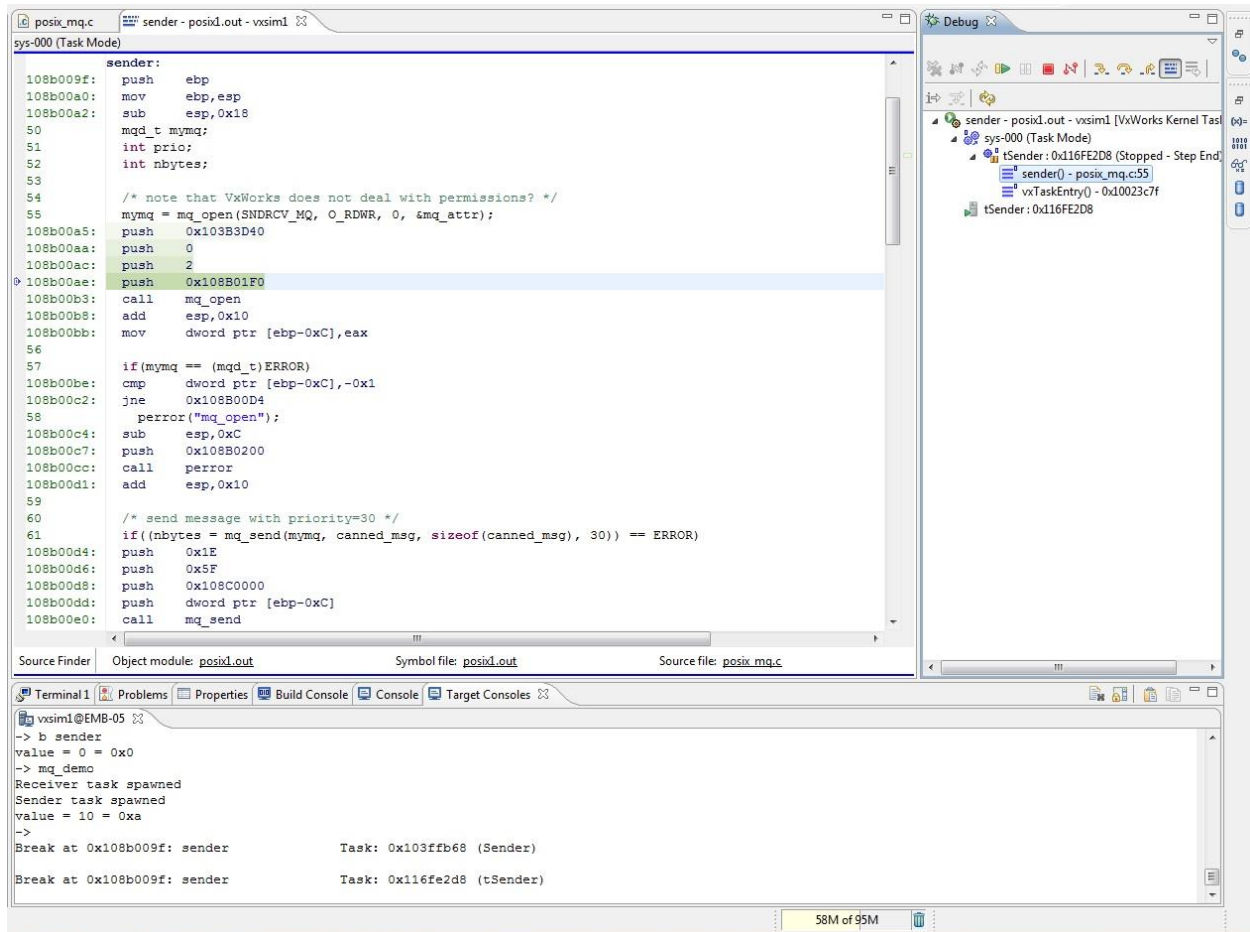
Symbol	Address	Type
ipcom_ipd_sendmsg	0x1012c2c0	text
mq_send	0x100d6e90	text
wdbRpcResendReply	0x10105b20	text
ipcom_send	0x10066cd0	text
sendmsg	0x100f54a0	text
ipcom_sendmsg_usr	0x10046fd0	text
ipnet_icmp4_send	0x100524a0	text
sendmsgSc	0x100f63d0	text
sendto	0x100f55c0	text
ipnet_ip4_sendto	0x10053ba0	text
_mq_send	0x100d69c0	text
ipcom_sendto_usr	0x10047030	text
ipcom_ipc_send	0x10041820	text
sendtoSc	0x100f6580	text
ipnet_sys_sendmsg	0x1005f6f0	text
sender	0x108b009f	text (posix1.out)
ipcom_sendmsg	0x10066810	text
iptcp_send_reset	0x1006e830	text
ipcom_sendto	0x10066c40	text
ipcom_ipd_send	0x1012c4c0	text
iptcp_send	0x1006ded0	text

```
Type <CR> to continue, Q<CR> to stop:
```

56M of 95M

6:10 PM 9/19/2013

Q6).



Ans: 7

Name: RUTVIJ KARKHANIS

Question 1:

a) Student understanding of Module Show

☐ Average ☒ Good ☐ Excellent

Question 2:

a) Understanding of Test_Task1()

☐ Average ☐ Good ☒ Excellent

b) Understanding of Test_Task2()

☐ Average ☐ Good ☒ Excellent

Question 3:

a) System Viewer Understanding.

☐ Average ☒ Good ☐ Excellent

Question 4:

a) Pause code implementation

☐ Average ☒ Good ☐ Excellent

b) Understanding of 'I' command

☐ Average ☒ Good ☐ Excellent

Question 5:

a) Creating the kernel Image and running mq_demo

☐ Average ☒ Good ☐ Excellent

Question 6:

a) Setting up break points

☒ Average ☐ Good ☐ Excellent

Comments:

Did not know taskspawn stack size.

fwethan
9/19/13