ASSIGNMENT 4

REPORT

CSE 438
Embedded System Programming

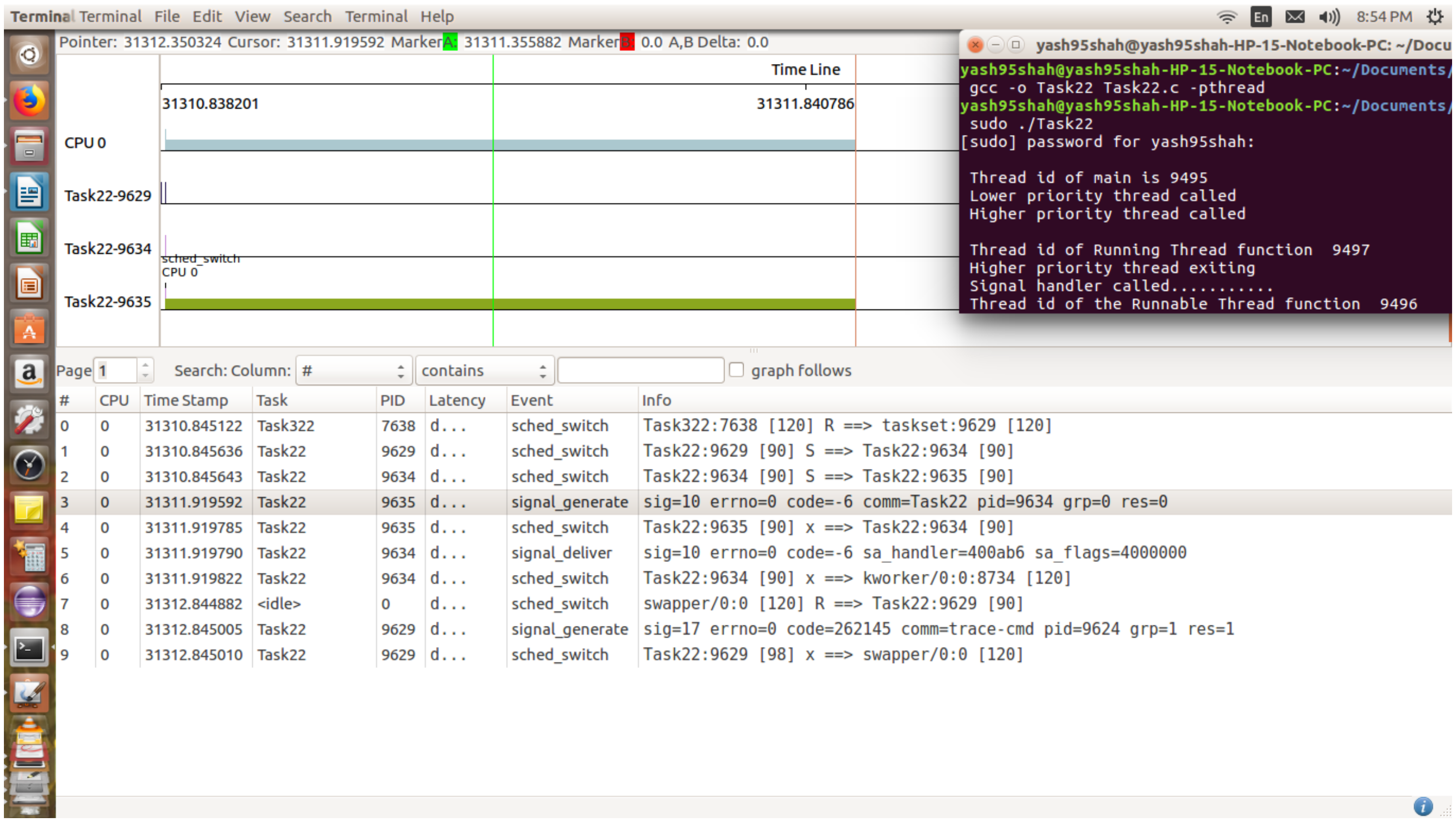
Yash Shah
ASU Id 1213315179

Assignment Statement: In vxWorks' Kernel API Reference Manual, it is stated that “If a task is pended (for instance, by waiting for a semaphore to become available) and a signal is sent to the task for which the task has a handler installed, then the handler will run before the semaphore is taken. When the handler returns, the task will go back to being pended (waiting for the semaphore). If there was a timeout used for the pend, then the original value will be used again when the task returns from the signal handler and goes back to being pended. If the handler alters the execution path, via a call to longjmp( ) for example, and does not return then the task does not go back to being pended.”

A signal handler associated with a thread gets executed in the following three conditions.
1) The thread is runnable(I.e running thread has a higher priority)
2)The thread is blocked by a semaphore
3)The thread is delayed (nanosleep() is used.)

Here, I used the command sudo trace-cmd record -e sched_switch -e signal taskset 0x01 ./Task2X to execute this and create a trace.dat file. We observe the results on the KernelShark.

Thread is runnable (running thread has a higher priority)- PART1



Pointer: 31312.350324 Cursor: 31311.919592 Marker A: 31311.355882 Marker B: 0.0 A,B Delta: 0.0

```
yash95shah@yash95shah-HP-15-Notebook-PC:~/Docu
yash95shah@yash95shah-HP-15-Notebook-PC:~/Documents/
 gcc -o Task22 Task22.c -pthread
yash95shah@yash95shah-HP-15-Notebook-PC:~/Documents/
 sudo ./Task22
[sudo] password for yash95shah:

 Thread id of main is 9495
 Lower priority thread called
 Higher priority thread called

 Thread id of Running Thread function  9497
 Higher priority thread exiting
 Signal handler called..........
 Thread id of the Runnable Thread function  9496
```

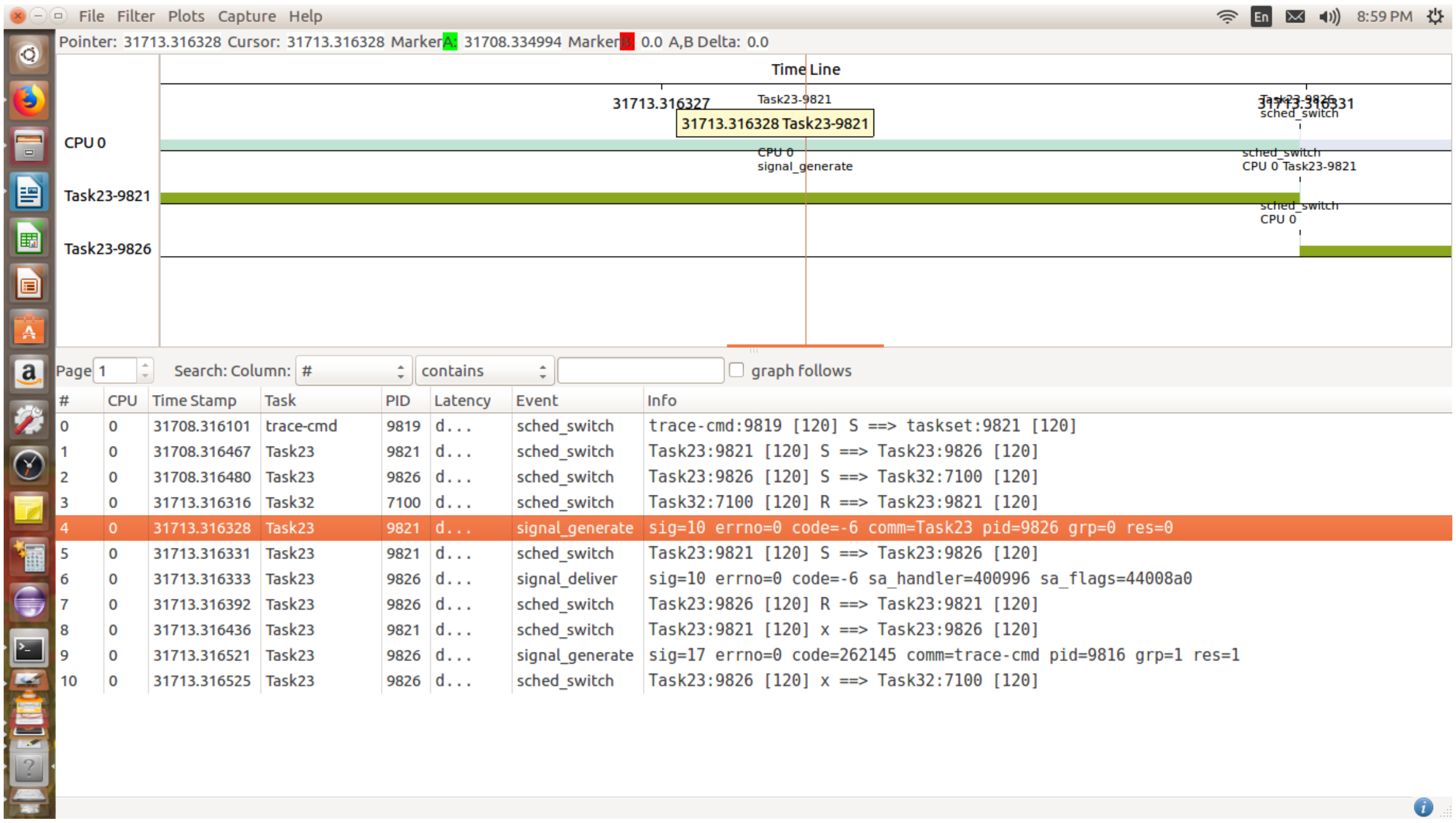
#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	31310.845122	Task322	7638	d...	sched_switch	Task322:7638 [120] R ==> taskset:9629 [120]
1	0	31310.845636	Task22	9629	d...	sched_switch	Task22:9629 [90] S ==> Task22:9634 [90]
2	0	31310.845643	Task22	9634	d...	sched_switch	Task22:9634 [90] S ==> Task22:9635 [90]
3	0	31311.919592	Task22	9635	d...	signal_generate	sig=10 errno=0 code=-6 comm=Task22 pid=9634 grp=0 res=0
4	0	31311.919785	Task22	9635	d...	sched_switch	Task22:9635 [90] x ==> Task22:9634 [90]
5	0	31311.919790	Task22	9634	d...	signal_deliver	sig=10 errno=0 code=-6 sa_handler=400ab6 sa_flags=4000000
6	0	31311.919822	Task22	9634	d...	sched_switch	Task22:9634 [90] x ==> kworker/0:0:8734 [120]
7	0	31312.844882	<idle>	0	d...	sched_switch	swapper/0:0 [120] R ==> Task22:9629 [90]
8	0	31312.845005	Task22	9629	d...	signal_generate	sig=17 errno=0 code=262145 comm=trace-cmd pid=9624 grp=1 res=1
9	0	31312.845010	Task22	9629	d...	sched_switch	Task22:9629 [98] x ==> swapper/0:0 [120]

EXPLANATION:

Both threads are created with a given priority. When they are called, the higher priority thread will be executed first after which the signal would be sent and signal handler would be called. Only then will the lower priority thread be executed.

The signal generate is having the time stamp of 31311.91992 and the signal deliver with the time_stamp of 31311.919790.

The thread is blocked by a semaphore

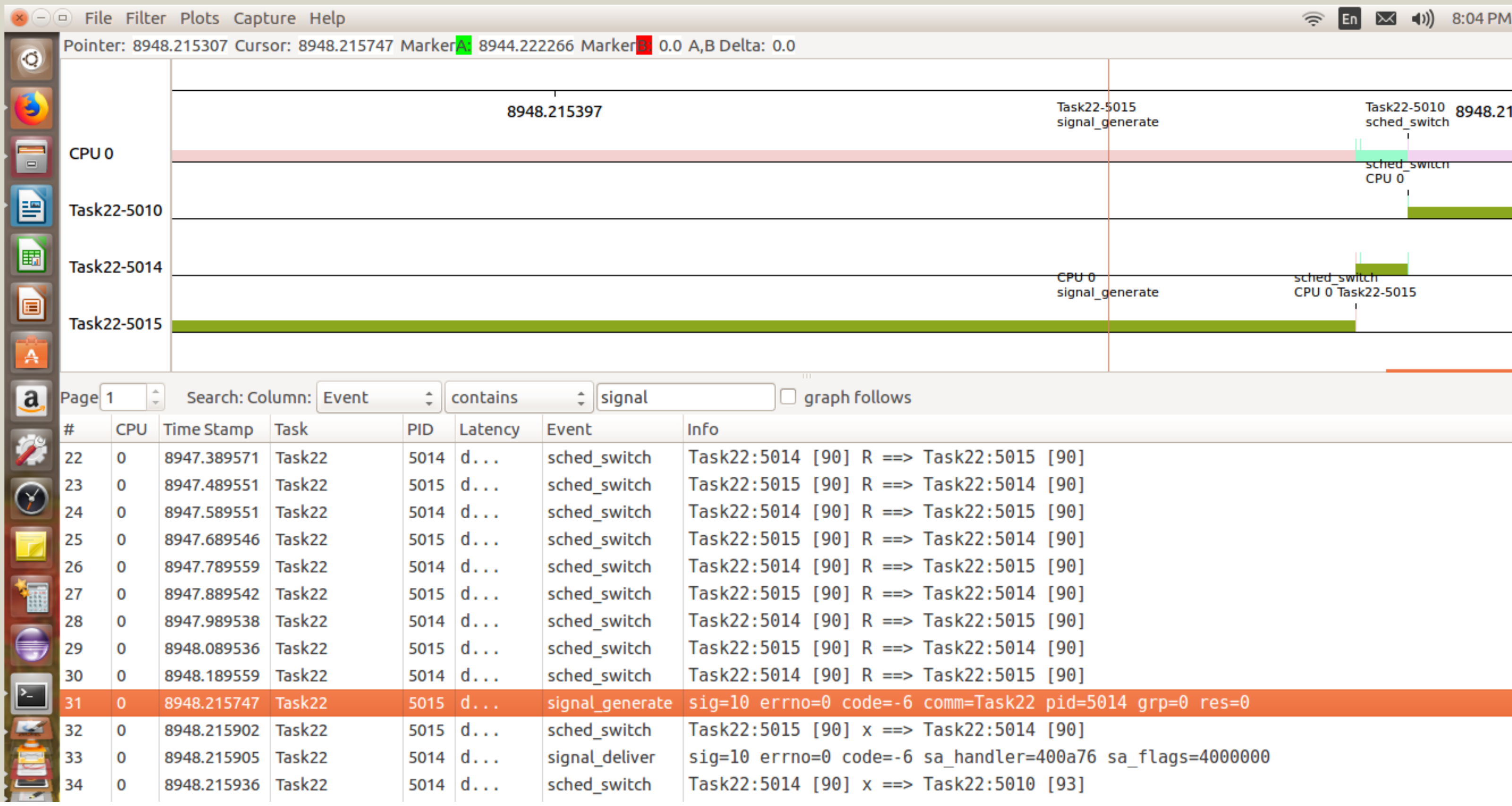


Pointer: 31713.316328 Cursor: 31713.316328 MarkerA: 31708.334994 MarkerB: 0.0 A,B Delta: 0.0

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	31708.316101	trace-cmd	9819	d...	sched_switch	trace-cmd:9819 [120] S ==> taskset:9821 [120]
1	0	31708.316467	Task23	9821	d...	sched_switch	Task23:9821 [120] S ==> Task23:9826 [120]
2	0	31708.316480	Task23	9826	d...	sched_switch	Task23:9826 [120] S ==> Task32:7100 [120]
3	0	31713.316316	Task32	7100	d...	sched_switch	Task32:7100 [120] R ==> Task23:9821 [120]
4	0	31713.316328	Task23	9821	d...	signal_generate	sig=10 errno=0 code=-6 comm=Task23 pid=9826 grp=0 res=0
5	0	31713.316331	Task23	9821	d...	sched_switch	Task23:9821 [120] S ==> Task23:9826 [120]
6	0	31713.316333	Task23	9826	d...	signal_deliver	sig=10 errno=0 code=-6 sa_handler=400996 sa_flags=44008a0
7	0	31713.316392	Task23	9826	d...	sched_switch	Task23:9826 [120] R ==> Task23:9821 [120]
8	0	31713.316436	Task23	9821	d...	sched_switch	Task23:9821 [120] x ==> Task23:9826 [120]
9	0	31713.316521	Task23	9826	d...	signal_generate	sig=17 errno=0 code=262145 comm=trace-cmd pid=9816 grp=1 res=1
10	0	31713.316525	Task23	9826	d...	sched_switch	Task23:9826 [120] x ==> Task32:7100 [120]

Here the main thread makes use of a semaphore. It attempts to get a semaphore lock, which is not free. So it gets blocked. It is in blocking state until the signal is generated by another thread is delivered to it, to interrupt it. After the signal has been delivered to the main thread, the thread continues to execute even before acquiring the lock for the same.

Thread is delayed



Pointer: 8948.215307 Cursor: 8948.215747 MarkerA: 8944.222266 MarkerB: 0.0 A,B Delta: 0.0

Page 1 Search: Column: Event contains signal

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
22	0	8947.389571	Task22	5014	d...	sched_switch	Task22:5014 [90] R ==> Task22:5015 [90]
23	0	8947.489551	Task22	5015	d...	sched_switch	Task22:5015 [90] R ==> Task22:5014 [90]
24	0	8947.589551	Task22	5014	d...	sched_switch	Task22:5014 [90] R ==> Task22:5015 [90]
25	0	8947.689546	Task22	5015	d...	sched_switch	Task22:5015 [90] R ==> Task22:5014 [90]
26	0	8947.789559	Task22	5014	d...	sched_switch	Task22:5014 [90] R ==> Task22:5015 [90]
27	0	8947.889542	Task22	5015	d...	sched_switch	Task22:5015 [90] R ==> Task22:5014 [90]
28	0	8947.989538	Task22	5014	d...	sched_switch	Task22:5014 [90] R ==> Task22:5015 [90]
29	0	8948.089536	Task22	5015	d...	sched_switch	Task22:5015 [90] R ==> Task22:5014 [90]
30	0	8948.189559	Task22	5014	d...	sched_switch	Task22:5014 [90] R ==> Task22:5015 [90]
31	0	8948.215747	Task22	5015	d...	signal_generate	sig=10 errno=0 code=-6 comm=Task22 pid=5014 grp=0 res=0
32	0	8948.215902	Task22	5015	d...	sched_switch	Task22:5015 [90] x ==> Task22:5014 [90]
33	0	8948.215905	Task22	5014	d...	signal_deliver	sig=10 errno=0 code=-6 sa_handler=400a76 sa_flags=4000000
34	0	8948.215936	Task22	5014	d...	sched_switch	Task22:5014 [90] x ==> Task22:5010 [93]

Here, the signal is generated by the thread and delivered to main thread. The main has delayed itself by going into sleep by calling nanasleep(). The main thread wakes up as soon asit receives interrupt signal generated by the thread. This signal after being delivered to main thread,
leaves the main thread in the awake conditions and the threads sleep for the remaining time and then
starts to execute the code.