CSE 438: EMBEDDED SYSTEMS PROGRAMMING

ASSIGNMENT 4

EVENT HANDLING AND SIGNALLING IN LINUX

AUTHORS
TEAM 5
RAMA KUMAR KANA SUNDARA 1213347614
SHARATH RENJIT NAIK 1213340750

CONTENT

PROBLEM 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 pending task, 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."

This description of the signal delivery mechanism is certainly OS dependent. In task 2 of the assignment, you are requested to develop a program or several pieces of program to test what the Linux's signal facility does precisely and to show the time that a signal handler associated to a thread gets executed in the following conditions:
a. The thread is runnable (but not running, i.e. the running thread has a higher priority).
b. The thread is blocked by a semaphore (i.e. sema_wait() is called).
c. The thread is delayed (i.e., nanosleep() is called).

To show the time that a signal handler is invoked, you are required to use "trace_cmd" to collect events from the Linux internal tracer ftrace. The traced records can then be viewed via a GUI front end kernelshark in a Linux host machine. A report should be compiled that includes kernelshark's report images to illuminate the execution of signal handlers.

TASK 1

The thread is runnable but not running due to the presence of a higher priority thread

Kernelshark Image

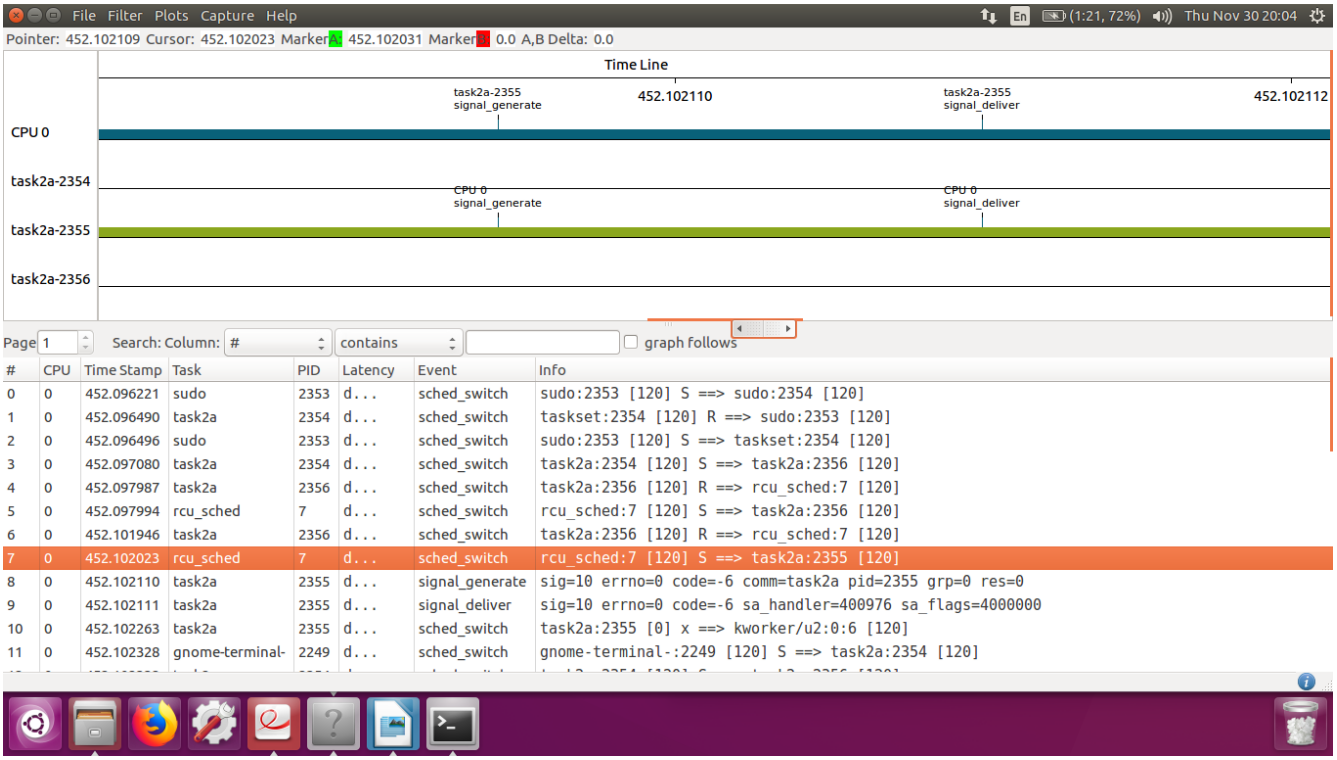


Figure 1 The thread is runnable but not running due to the presence of a higher priority thread

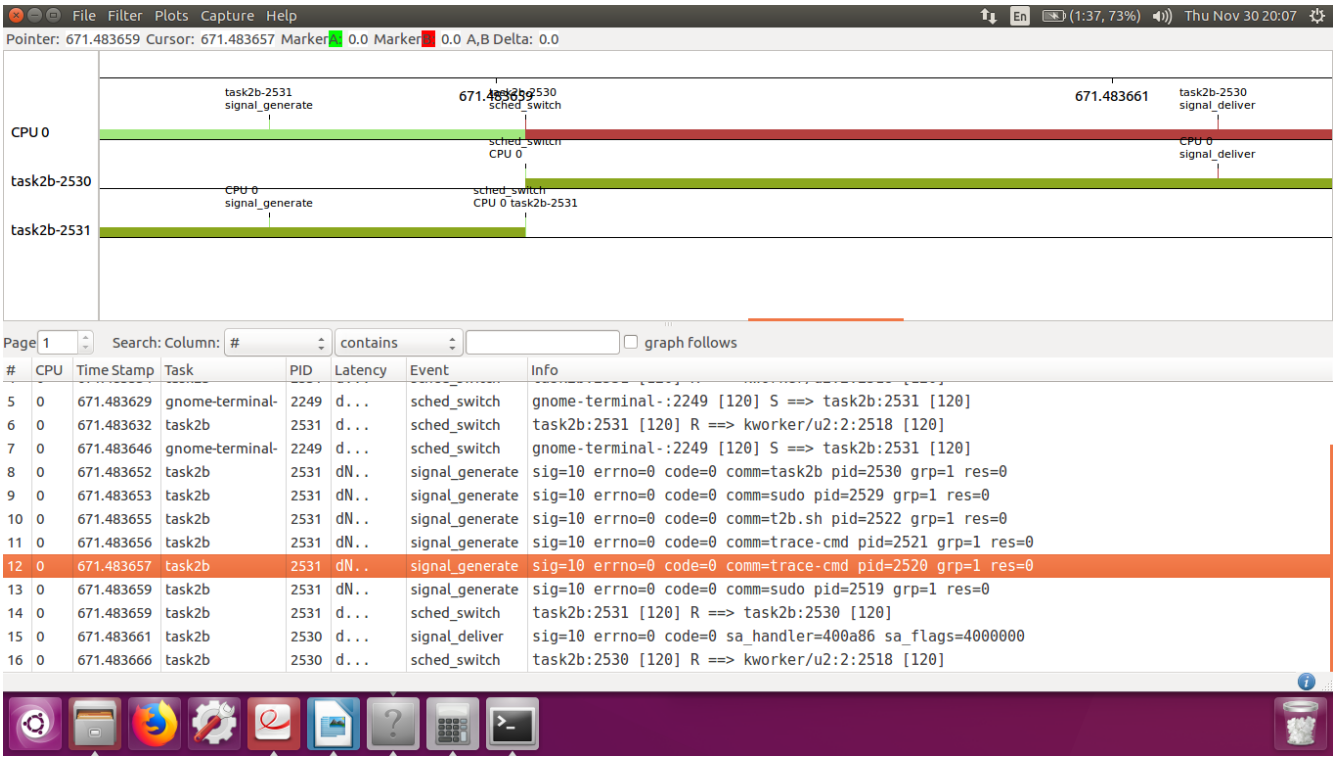
Analysis

In this part, a thread is runnable, but it is unable to execute due the execution of a higher priority thread running in the background.
The signal_generate is given at 452.102110 but the signal_deliver occurs at 452.102111, which is a delay of 1 microseconds.

TASK 2

The thread is blocked by a semaphore

Kernelshark Image



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
5	0	671.483629	gnome-terminal-	2249	d...	sched_switch	gnome-terminal-:2249 [120] S ==> task2b:2531 [120]
6	0	671.483632	task2b	2531	d...	sched_switch	task2b:2531 [120] R ==> kworker/u2:2:2518 [120]
7	0	671.483646	gnome-terminal-	2249	d...	sched_switch	gnome-terminal-:2249 [120] S ==> task2b:2531 [120]
8	0	671.483652	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=task2b pid=2530 grp=1 res=0
9	0	671.483653	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=sudo pid=2529 grp=1 res=0
10	0	671.483655	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=t2b.sh pid=2522 grp=1 res=0
11	0	671.483656	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=trace-cmd pid=2521 grp=1 res=0
12	0	671.483657	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=trace-cmd pid=2520 grp=1 res=0
13	0	671.483659	task2b	2531	dN..	signal_generate	sig=10 errno=0 code=0 comm=sudo pid=2519 grp=1 res=0
14	0	671.483659	task2b	2531	d...	sched_switch	task2b:2531 [120] R ==> task2b:2530 [120]
15	0	671.483661	task2b	2530	d...	signal_deliver	sig=10 errno=0 code=0 sa_handler=400a86 sa_flags=4000000
16	0	671.483666	task2b	2530	d...	sched_switch	task2b:2530 [120] R ==> kworker/u2:2:2518 [120]

Figure 2 The thread is blocked by a semaphore

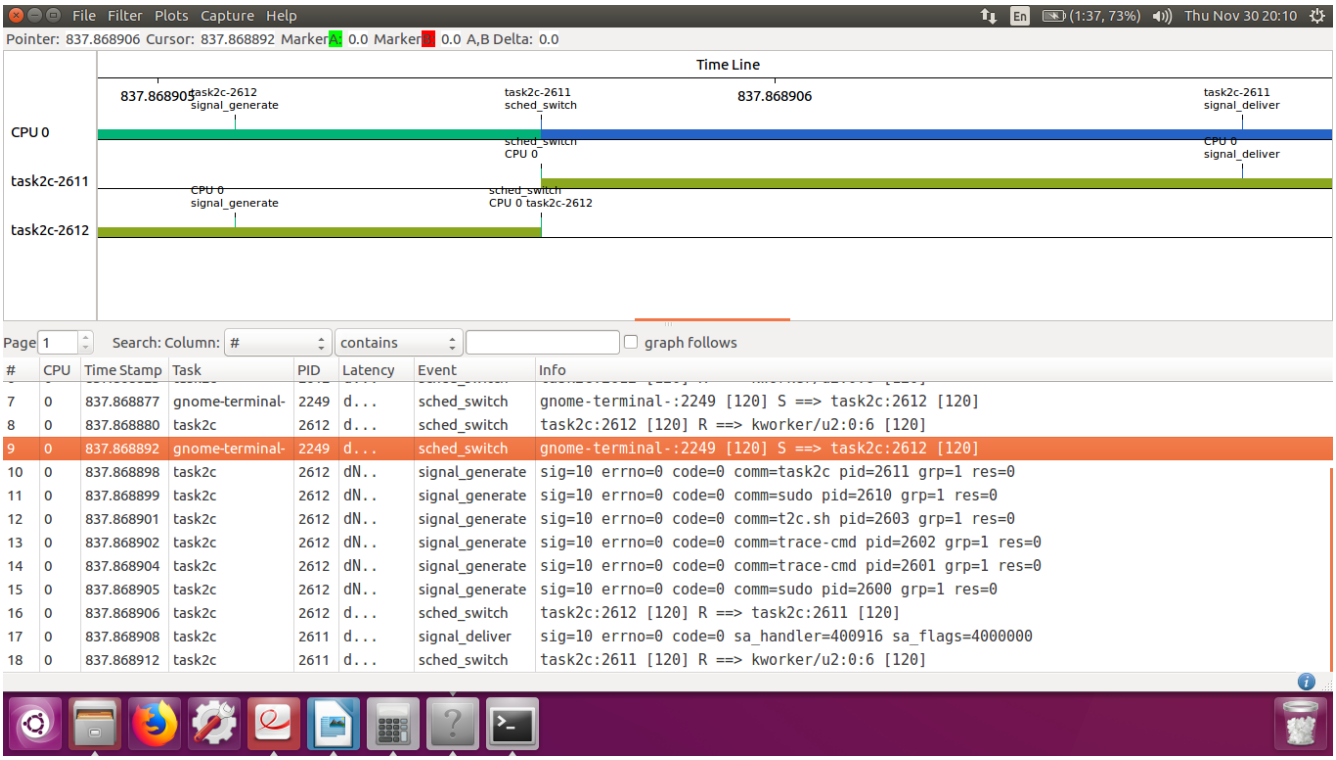
Analysis

This part of the assignment involves checking for a thread that has been waiting for a semaphore to be freed.
The signal_generate has been sent at 671.483657 and the signal_deliver occurs at 671.483661 which results in delay of 6 microsecond.

TASK 3

The thread is delayed by a sleep function call

Kernelshark Image



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
7	0	837.868877	gnome-terminal-	2249	d...	sched_switch	gnome-terminal-:2249 [120] S ==> task2c:2612 [120]
8	0	837.868880	task2c	2612	d...	sched_switch	task2c:2612 [120] R ==> kworker/u2:0:6 [120]
9	0	837.868892	gnome-terminal-	2249	d...	sched_switch	gnome-terminal-:2249 [120] S ==> task2c:2612 [120]
10	0	837.868898	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=task2c pid=2611 grp=1 res=0
11	0	837.868899	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=sudo pid=2610 grp=1 res=0
12	0	837.868901	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=t2c.sh pid=2603 grp=1 res=0
13	0	837.868902	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=trace-cmd pid=2602 grp=1 res=0
14	0	837.868904	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=trace-cmd pid=2601 grp=1 res=0
15	0	837.868905	task2c	2612	dN..	signal_generate	sig=10 errno=0 code=0 comm=sudo pid=2600 grp=1 res=0
16	0	837.868906	task2c	2612	d...	sched_switch	task2c:2612 [120] R ==> task2c:2611 [120]
17	0	837.868908	task2c	2611	d...	signal_deliver	sig=10 errno=0 code=0 sa_handler=400916 sa_flags=4000000
18	0	837.868912	task2c	2611	d...	sched_switch	task2c:2611 [120] R ==> kworker/u2:0:6 [120]

Figure 3 The thread is delayed by a sleep function call

Analysis

The last part of the assignment involves the delaying of a thread with a nanosleep function. The signal_generate here occurs at 837.868905 and the signal_deliver occurs at 837.868908 which is a resultant delay of 3 microsecond.