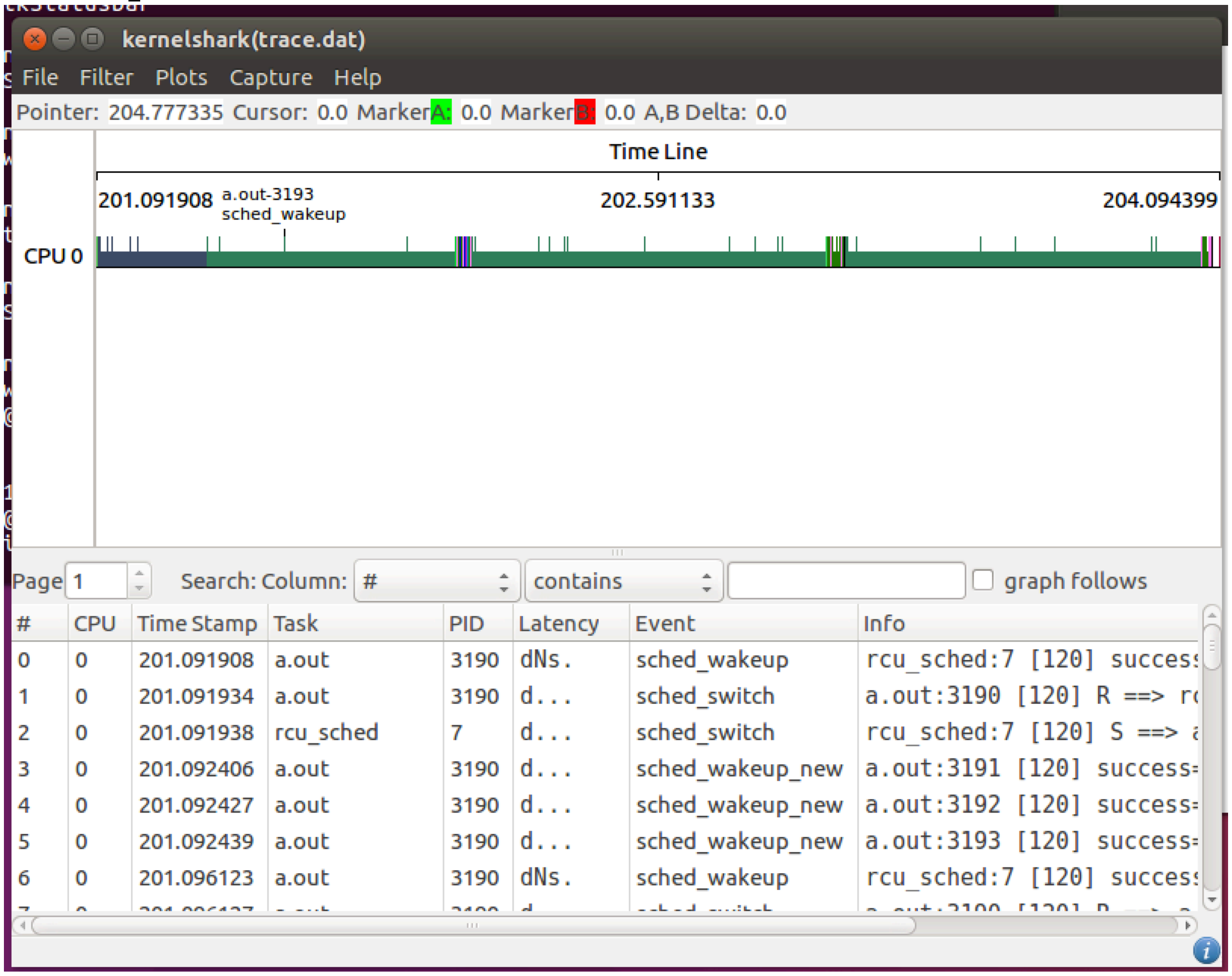
1. SCHED_FIFO



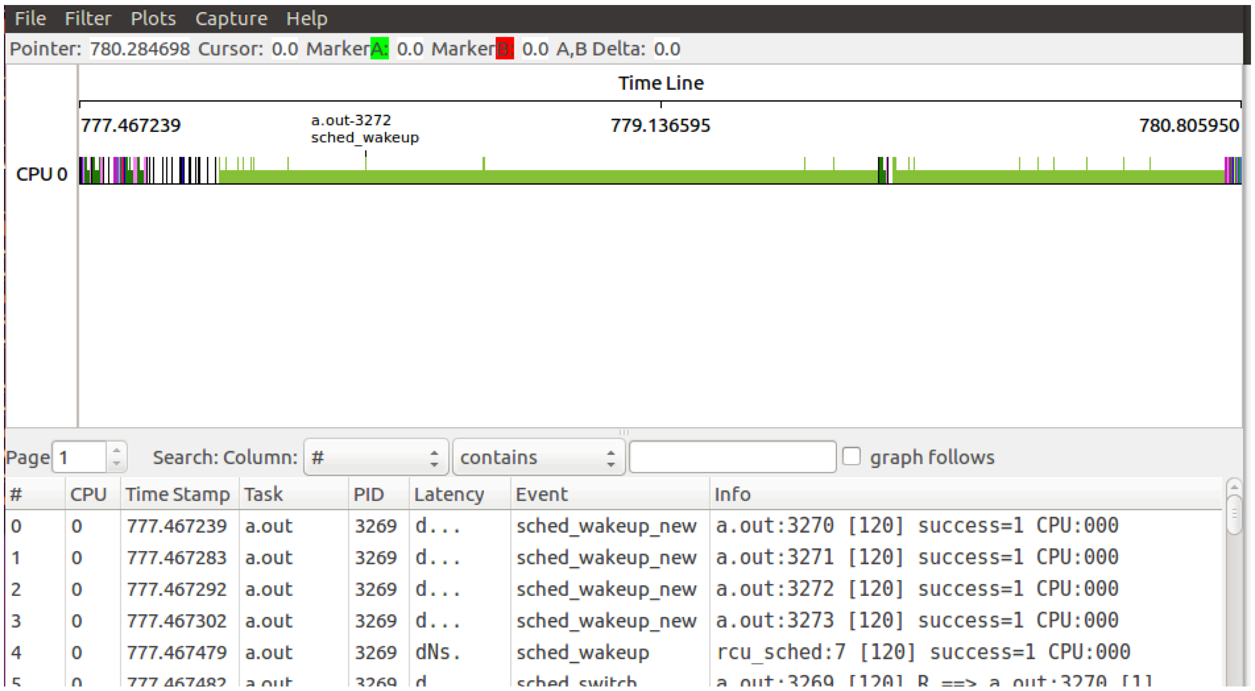
Pointer: 204.777335 Cursor: 0.0 MarkerA: 0.0 MarkerB: 0.0 A,B Delta: 0.0

201.091908 202.591133 204.094399

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	201.091908	a.out	3190	dNs.	sched_wakeup	rcu_sched:7 [120] succes
1	0	201.091934	a.out	3190	d...	sched_switch	a.out:3190 [120] R ==> r
2	0	201.091938	rcu_sched	7	d...	sched_switch	rcu_sched:7 [120] S ==> a
3	0	201.092406	a.out	3190	d...	sched_wakeup_new	a.out:3191 [120] success=
4	0	201.092427	a.out	3190	d...	sched_wakeup_new	a.out:3192 [120] success=
5	0	201.092439	a.out	3190	d...	sched_wakeup_new	a.out:3193 [120] success=
6	0	201.096123	a.out	3190	dNs.	sched_wakeup	rcu_sched:7 [120] succes

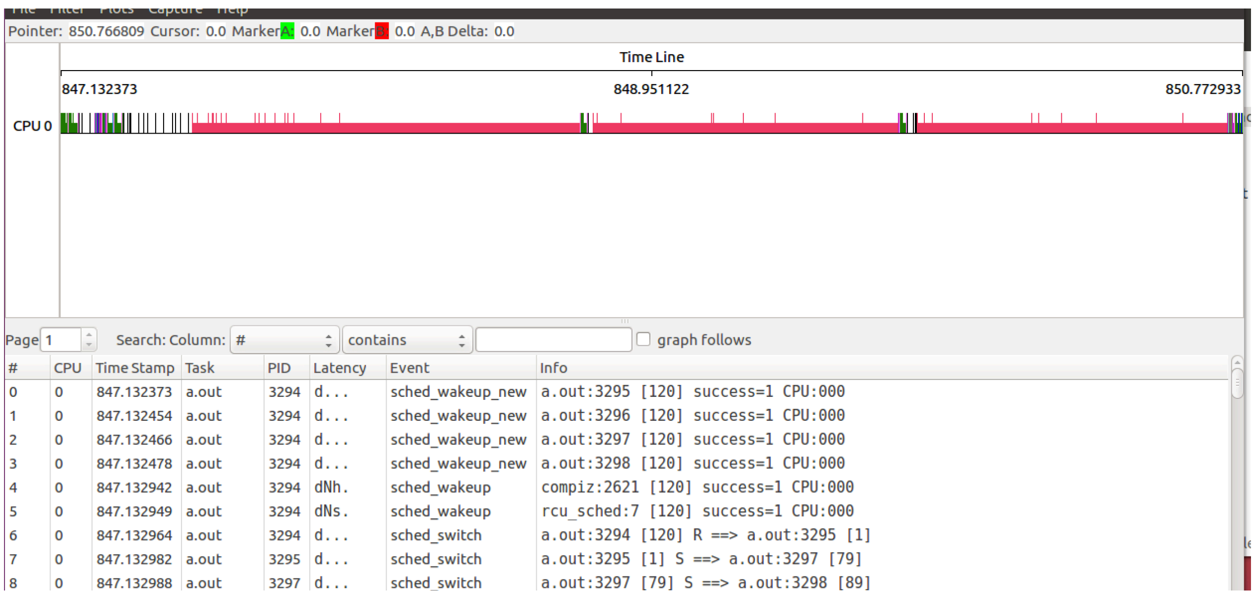
The kernel shark image shown above proves that the scheduled tasks are running as first-in-first-out. This is done with two periodic tasks. Both of the tasks have the same priority and run for the same period but the number of iterations in the second task is much larger. (i.e. P 20 1000 1000000, P 20 1000 1000000000). The blue segment above indicates the first task which takes much less time to run because it has a smaller number of iterations. It is the first task to finish and is also the first task in the list. The green segment above is the second task in the list has a much larger number of iterations and runs for the second part of the time. It is the last task to finish This shows that this program uses first-in-first-out as the scheduling policy for the threads created per task. 1 CPU was used in this instance to prove that priority scheduling was done (otherwise they would both run at the same time).

2. PI Enabled vs. Non PI Enabled Mutexes



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	777.467239	a.out	3269	d...	sched_wakeup_new	a.out:3270 [120] success=1 CPU:000
1	0	777.467283	a.out	3269	d...	sched_wakeup_new	a.out:3271 [120] success=1 CPU:000
2	0	777.467292	a.out	3269	d...	sched_wakeup_new	a.out:3272 [120] success=1 CPU:000
3	0	777.467302	a.out	3269	d...	sched_wakeup_new	a.out:3273 [120] success=1 CPU:000
4	0	777.467479	a.out	3269	dNs.	sched_wakeup	rcu_sched:7 [120] success=1 CPU:000
5	0	777.467482	a.out	3269	d	sched_switch	a.out:3269 [120] R ==> a.out:3270 [1]

This first kernel shark image shown above is using PI mutexes. The light green task above shows that all three tasks are run and there is only one short break, where nothing is being run, due to a resource being blocked within the task of lower priority. This break is due to the fact that the PTHREAD_PRIO_INHERIT protocol of the mutex does not always apply perfectly when it implemented within the Linux system. While there is still idle time, the PI mutex is still much better at handling priority inversion than a non PI mutex.


File Filter Plots Capture Help
Pointer: 850.766809 Cursor: 0.0 Marker A: 0.0 Marker B: 0.0 A,B Delta: 0.0
Time Line
847.132373
848.951122
850.772933
CPU 0
Page 1
Search: Column: #
contains
graph follows

#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	847.132373	a.out	3294	d...	sched_wakeup_new	a.out:3295 [120] success=1 CPU:000
1	0	847.132454	a.out	3294	d...	sched_wakeup_new	a.out:3296 [120] success=1 CPU:000
2	0	847.132466	a.out	3294	d...	sched_wakeup_new	a.out:3297 [120] success=1 CPU:000
3	0	847.132478	a.out	3294	d...	sched_wakeup_new	a.out:3298 [120] success=1 CPU:000
4	0	847.132942	a.out	3294	dNh.	sched_wakeup	compiz:2621 [120] success=1 CPU:000
5	0	847.132949	a.out	3294	dNs.	sched_wakeup	rcu_sched:7 [120] success=1 CPU:000
6	0	847.132964	a.out	3294	d...	sched_switch	a.out:3294 [120] R ==> a.out:3295 [1]
7	0	847.132982	a.out	3295	d...	sched_switch	a.out:3295 [1] S ==> a.out:3297 [79]
8	0	847.132988	a.out	3297	d...	sched_switch	a.out:3297 [79] S ==> a.out:3298 [89]

This second kernel shark image shows the same three tasks run without the PI mutex protocol as an attribute. As shown, there is an extra break where tasks are idle and the program is not being performed as one of the resources is being blocked by a task with a lower priority. The PI mutex definitely decreases the chance of this happening.