Problem detection in real-time systems by trace analysis

Mathieu Côté
Laboratoire DORSAL

mathieu.cote@polymtl.ca

POLYTECHNIQUE
MONTRÉAL

TRACE
COMPASS

Outline

- Introduction
- Literature review
- Approach
  - Modeling
  - Problems
  - Analysis
- Results
- Conclusion

Introduction : definition

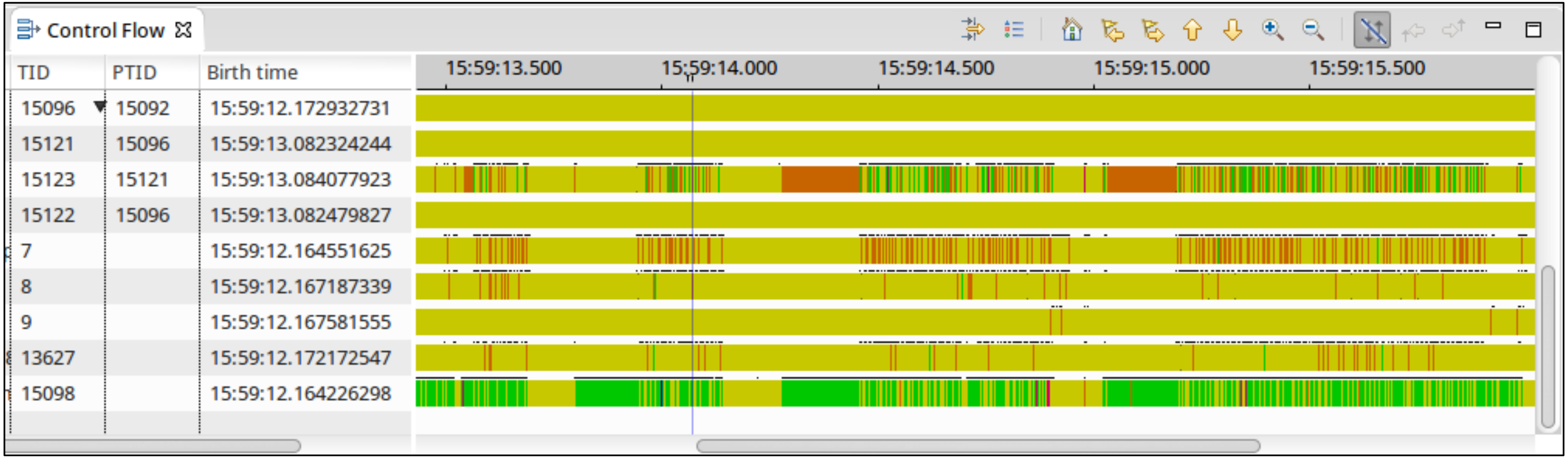
- Real-time task : execution time, deadline, period (optional)
- Execution : periodic, sporadic
- Hard/soft real-time

PREEMPT_RT

- Priority inheritance for mutex in kernel
- Reduce non-preemptive sections in kernel

Introduction : problematic

Music player trace in Trace Compass



TID	PTID	Birth time
15096	15092	15:59:12.172932731
15121	15096	15:59:13.082324244
15123	15121	15:59:13.084077923
15122	15096	15:59:13.082479827
7		15:59:12.164551625
8		15:59:12.167187339
9		15:59:12.167581555
13627		15:59:12.172172547
15098		15:59:12.164226298

Figure 1 : Multiple executions of an audio player

Introduction : problematic

Advantages of tracing real-time systems

- Low **overhead**
- Low **jitter**
- Access to **specific** information (priority, scheduling policy, etc.)

What is missing?

- Real-time **specific** user tools
- Show **useful** data

Introduction : goals

1. Develop a **model** to define real-time task **executions** in a trace
2. Identify common **problems** in real-time systems and useful **information** to analyze them
3. Develop a method to analyze the **trace segment** corresponding to an execution to identify if the execution presents a **problem**

Literature review

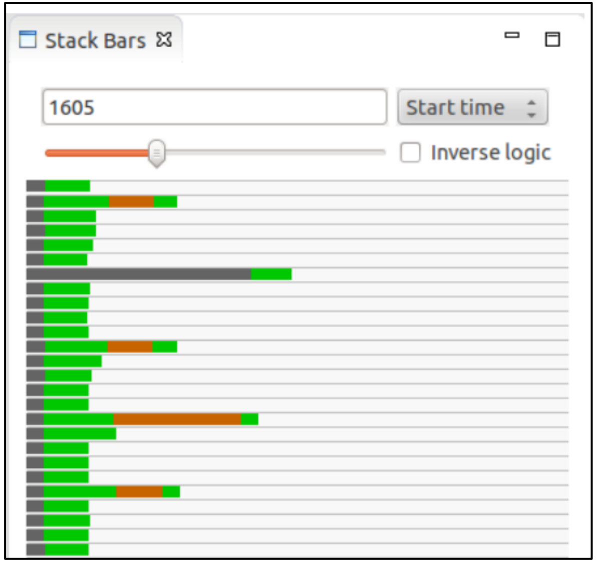
Linux low-latency tracing for multicore hard real-time systems
(Beamonte, 2013)

- LTTng-UST **modification** to reduce the added latency
- Demonstrated **low latency** tracing with LTTng

Literature review

Real-time Linux analysis using low-impact tracer (Rajotte, 2014)

- Recreate the task states using kernel events
- Compare executions of a task
- Limitations
  - Model
    - Threads need to have different priorities
    - Fixed
  - Analysis
    - Manual
    - Some statistics



1605

Start time

☐ Inverse logic

Figure 2 : Original stackbars view

Modeling

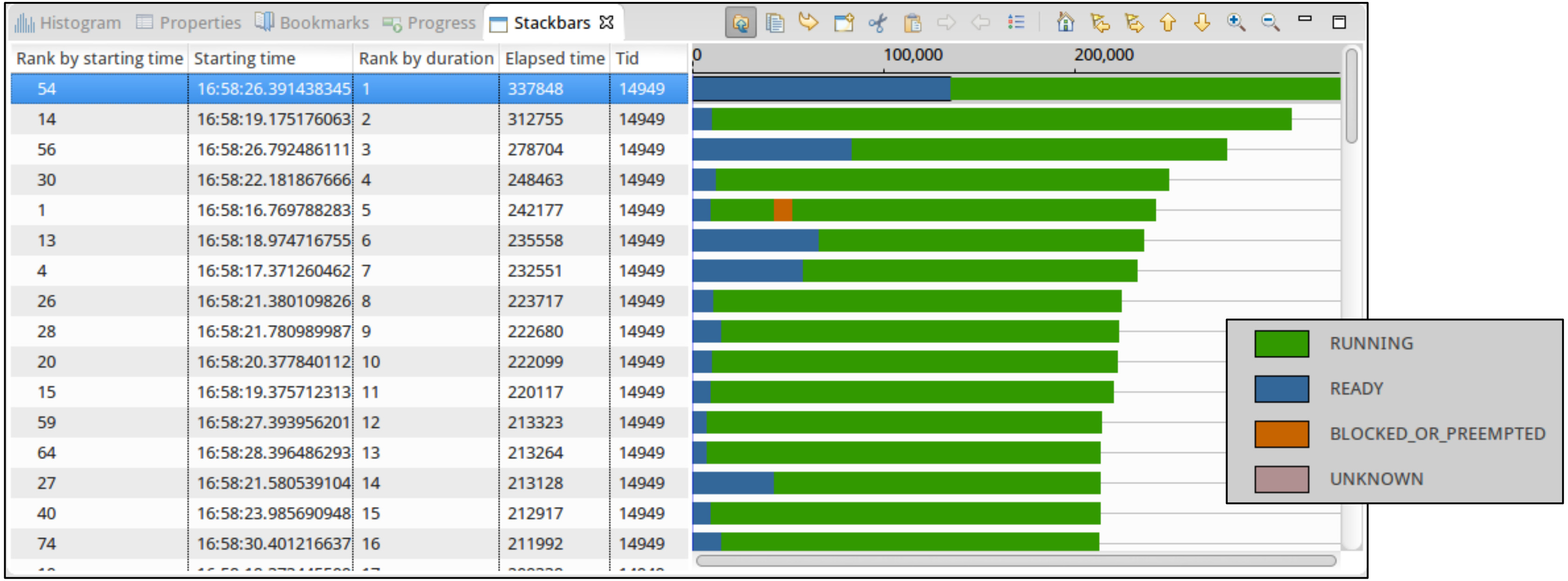
Introduction
Literature
Modeling
Problems
Analysis
Results
Conclusion

Advantage of using only kernel events

- No need to modify the application source code to add tracepoints manually

Modeling : view

Stackbars view in Trace Compass



Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
54	16:58:26.391438345	1	337848	14949
14	16:58:19.175176063	2	312755	14949
56	16:58:26.792486111	3	278704	14949
30	16:58:22.181867666	4	248463	14949
1	16:58:16.769788283	5	242177	14949
13	16:58:18.974716755	6	235558	14949
4	16:58:17.371260462	7	232551	14949
26	16:58:21.380109826	8	223717	14949
28	16:58:21.780989987	9	222680	14949
20	16:58:20.377840112	10	222099	14949
15	16:58:19.375712313	11	220117	14949
59	16:58:27.393956201	12	213323	14949
64	16:58:28.396486293	13	213264	14949
27	16:58:21.580539104	14	213128	14949
40	16:58:23.985690948	15	212917	14949
74	16:58:30.401216637	16	211992	14949
[illegible]	[illegible]	17	[illegible]	14949

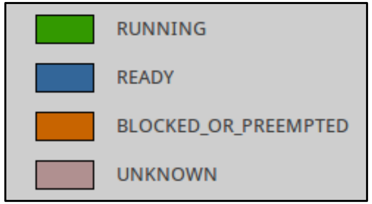
🟩	RUNNING
🟦	READY
🟧	BLOCKED_OR_PREEMPTED
🟫	UNKNOWN

Figure 3 : Stackbars view

Modeling : view

States in *Stackbars* view

- Running : in userspace or in system calls
- Ready : between sched_wakeup and sched_switch
- Blocked or preempted : when you are still in a task execution but are scheduled out



🟩 RUNNING
🟦 READY
🟧 BLOCKED_OR_PREEMPTED
▨ UNKNOWN

Figure 4 : Stackbars view legend

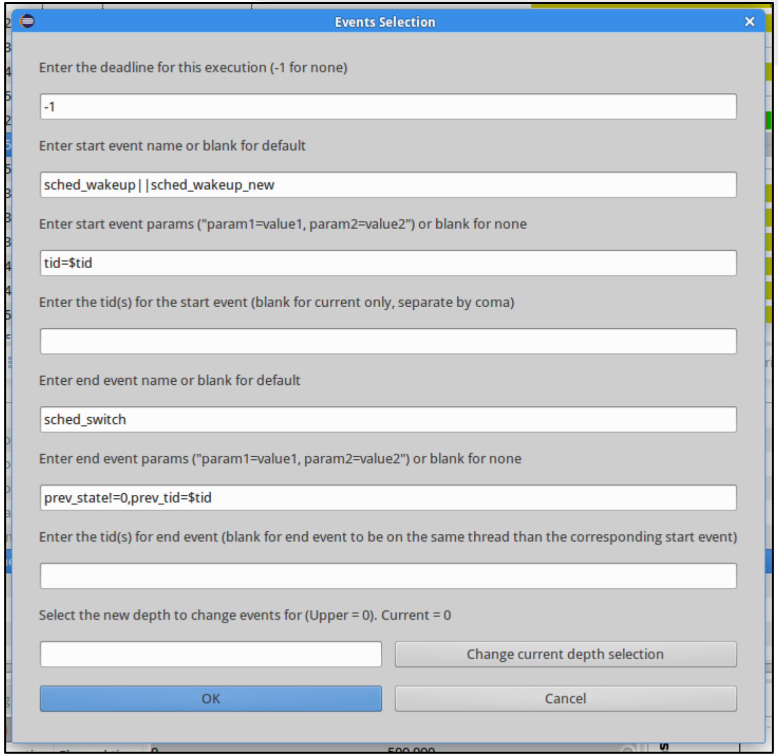
Modeling

- Identify executions automatically and then let the users choose between some valid models
  - Estimate the number of executions
  - Find the longest subsequence repeated at least *n* times
  - Difficulties :
    - Execution time
    - Too many possible resulting models

Modeling : method

State machine

- User identifies :
  - an execution or
  - events that defined the start and the end (name, parameters with operations, etc.)



Enter the deadline for this execution (-1 for none)

-1

Enter start event name or blank for default

sched_wakeup||sched_wakeup_new

Enter start event params ("param1=value1, param2=value2") or blank for none

tid=$tid

Enter the tid(s) for the start event (blank for current only, separate by coma)

Enter end event name or blank for default

sched_switch

Enter end event params ("param1=value1, param2=value2") or blank for none

prev_state!=0,prev_tid=$tid

Enter the tid(s) for end event (blank for end event to be on the same thread than the corresponding start event)

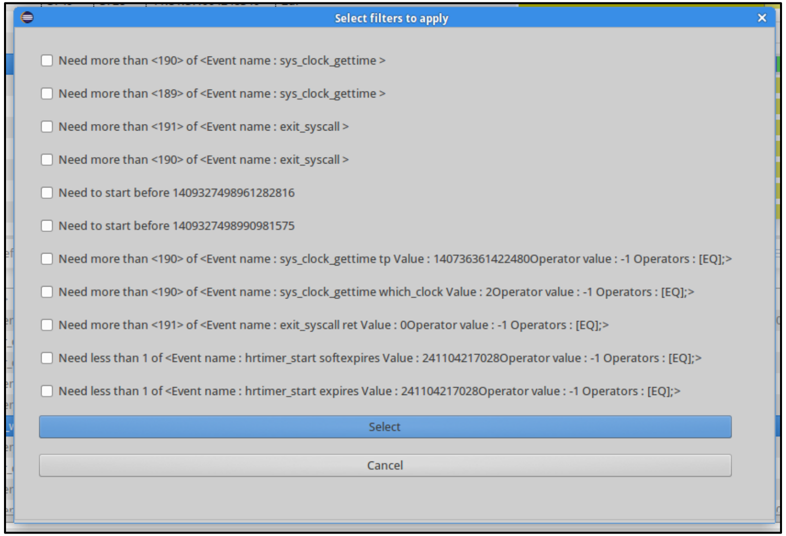
Select the new depth to change events for (Upper = 0). Current = 0

Figure 5 : Dialog to define model

Modeling : method

State machine

- Remove execution
- Add execution
- Define an execution as invalid and recalculate
  - Will suggest some modifications to the model based on differences between valid and invalid executions
  - The user can select the ones he wants to apply



☐ Need more than <190> of <Event name : sys_clock_gettime >

☐ Need more than <189> of <Event name : sys_clock_gettime >

☐ Need more than <191> of <Event name : exit_syscall >

☐ Need more than <190> of <Event name : exit_syscall >

☐ Need to start before 1409327498961282816

☐ Need to start before 1409327498990981575

☐ Need more than <190> of <Event name : sys_clock_gettime tp Value : 140736361422480Operator value : -1 Operators : [EQ];>

☐ Need more than <190> of <Event name : sys_clock_gettime which_clock Value : 2Operator value : -1 Operators : [EQ];>

☐ Need more than <191> of <Event name : exit_syscall ret Value : 0Operator value : -1 Operators : [EQ];>

☐ Need less than 1 of <Event name : hrtimer_start softexpires Value : 241104217028Operator value : -1 Operators : [EQ];>

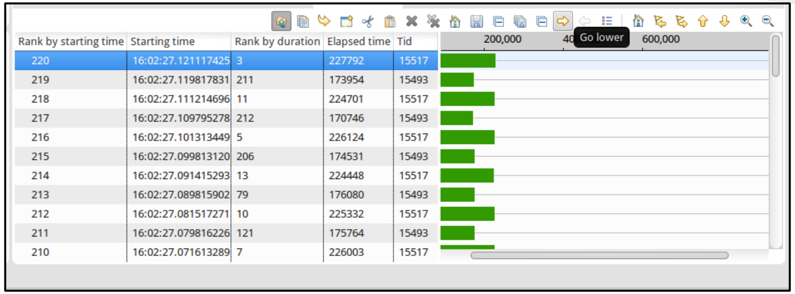
☐ Need less than 1 of <Event name : hrtimer_start expires Value : 241104217028Operator value : -1 Operators : [EQ];>

Figure 6 : Dialog to select modifications to apply

Modeling : method

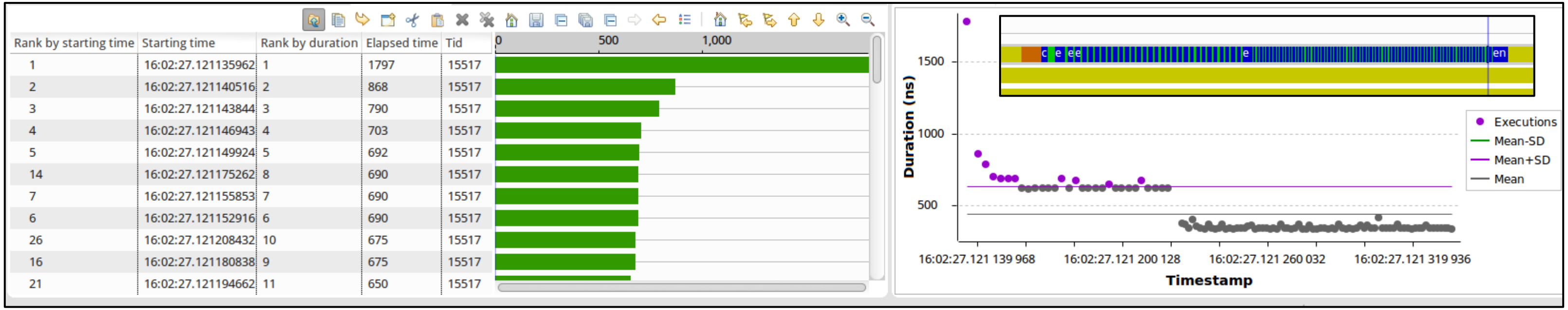
State machine

- Supports
  - Thread pool
  - Nested executions



Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
220	16:02:27.121117425	3	227792	15517
219	16:02:27.119817831	211	173954	15493
218	16:02:27.111214696	11	224701	15517
217	16:02:27.109795278	212	170746	15493
216	16:02:27.101313449	5	226124	15517
215	16:02:27.099813120	206	174531	15493
214	16:02:27.091415293	13	224448	15517
213	16:02:27.089815902	79	176080	15493
212	16:02:27.081517271	10	225332	15517
211	16:02:27.079816226	121	175764	15493
210	16:02:27.071613289	7	226003	15517

Figure 7 : Task on multiple threads



Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
1	16:02:27.121135962	1	1797	15517
2	16:02:27.121140516	2	868	15517
3	16:02:27.121143844	3	790	15517
4	16:02:27.121146943	4	703	15517
5	16:02:27.121149924	5	692	15517
14	16:02:27.121175262	8	690	15517
7	16:02:27.121155853	7	690	15517
6	16:02:27.121152916	6	690	15517
26	16:02:27.121208432	10	675	15517
16	16:02:27.121180838	9	675	15517
21	16:02:27.121194662	11	650	15517

Figure 8 : Nested executions

Specific informations

Scheduling policies

- Normal
  - SCHED_OTHER : standard
  - SCHED_BATCH
  - SCHED_IDLE
- Real-time
  - SCHED_FIFO
  - SCHED_RR : with time quantum
  - SCHED_DEADLINE : Global Earliest Deadline First, highest user controllable priority

Specific informations

Scheduling policies

- SCHED_FIFO and SCHED_RR
  - A deadline can be missed even if there was a valid scheduling to respect all deadlines
- SCHED_DEADLINE
  - No deadline will be missed if there is a valid scheduling

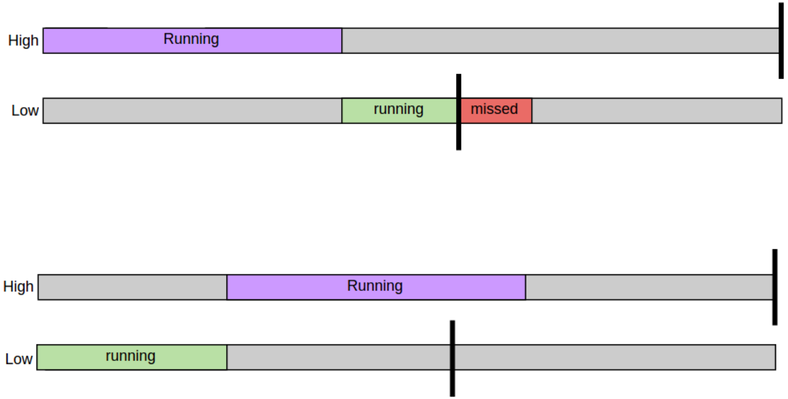


Figure 9 : Deadline missed

Specific informations

Scheduling policies

- SCHED_FIFO and SCHED_RR
  - The highest priority task will always execute if it is able to
- SCHED_DEADLINE
  - If there is a missed deadline, it can be on a highest priority task (for the user, because there is no priority set)

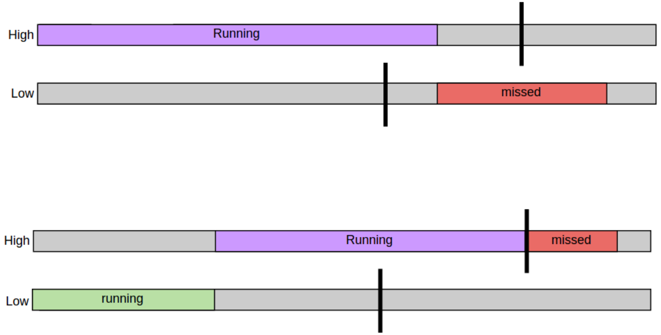


Figure 10 : Highest priority

Specific informations

Events to track to get *policy* :

sched_setscheduler, sched_setparam, sched_setattr

Additional events to track to get *priority* :

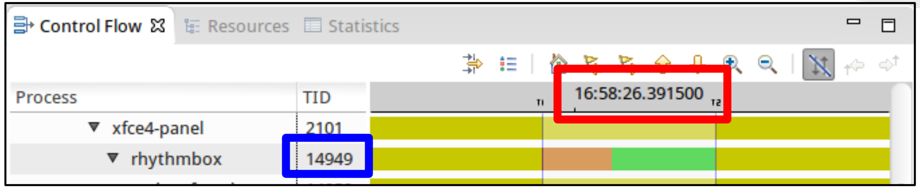
setprority, sched_pi_setprio, sched_switch

Events to track to get *cpus_allowed*:

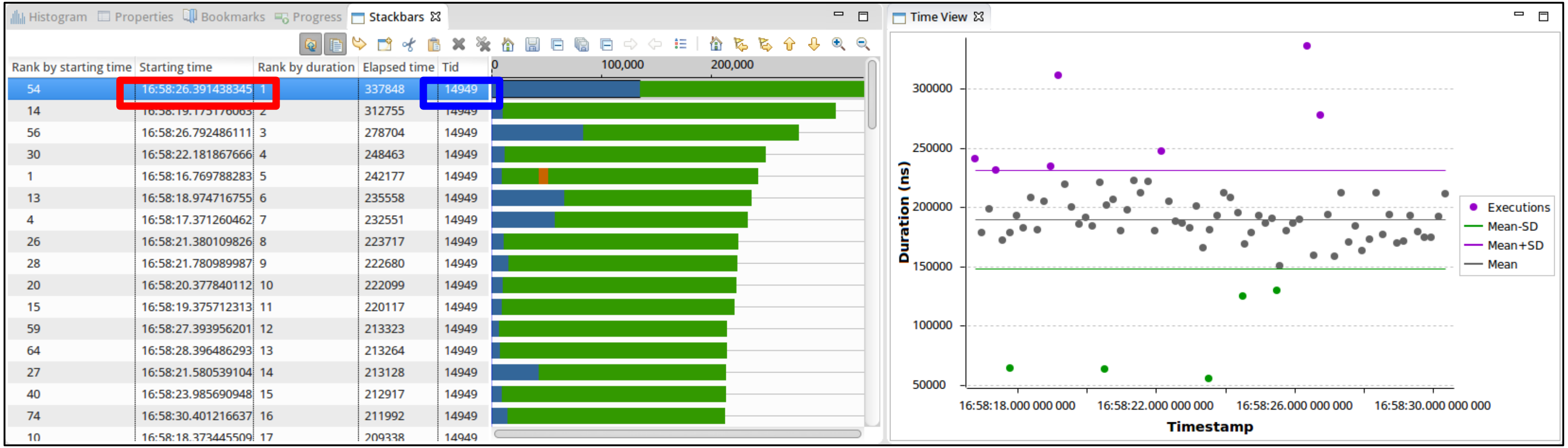
sched_setaffinity, need to add some

Results : views

- View of duration by starting timestamp
- Synced with other views



Process	TID
▼ xfce4-panel	2101
▼ rhythmbox	14949

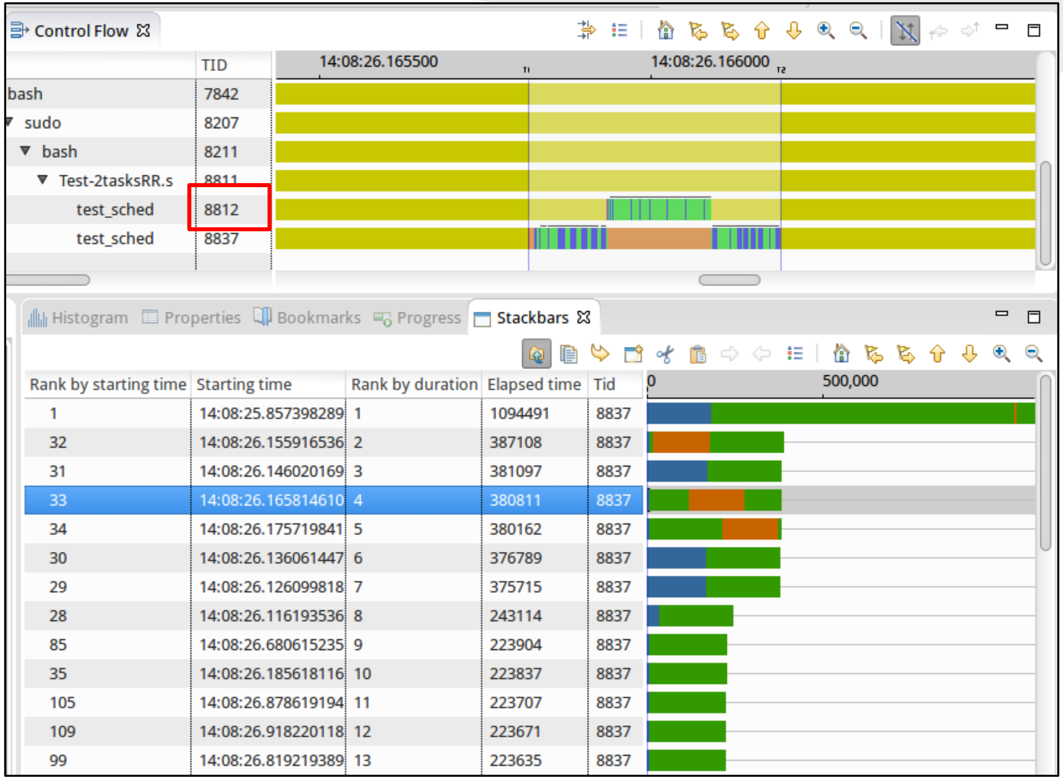


Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
54	16:58:26.391438345	1	337848	14949
14	16:58:19.175176063	2	312755	14949
56	16:58:26.792486111	3	278704	14949
30	16:58:22.181867666	4	248463	14949
1	16:58:16.769788283	5	242177	14949
13	16:58:18.974716755	6	235558	14949
4	16:58:17.371260462	7	232551	14949
26	16:58:21.380109826	8	223717	14949
28	16:58:21.780989987	9	222680	14949
20	16:58:20.377840112	10	222099	14949
15	16:58:19.375712313	11	220117	14949
59	16:58:27.393956201	12	213323	14949
64	16:58:28.396486293	13	213264	14949
27	16:58:21.580539104	14	213128	14949
40	16:58:23.985690948	15	212917	14949
74	16:58:30.401216637	16	211992	14949
10	16:58:18.373445509	17	209338	14949

Figure 11 : Stackbars view and stackbars time view

Results : periodic conflict

```
Analysis for the thread : [8837,.
/test_sched]

Priority : -49 from time : 14:08:
26.155926228

Policy : SCHED_RR

---

The analysed thread was preempted
from time : 14:08:26.155935758 for
: 160916

---

This thread was running when
[8837,./test_sched] was preempted.

First time : 14:08:26.155935758

Thread ID : 8812

Duration : 160917

Priority : -50

Policy : SCHED_RR
```



	TID
bash	7842
sudo	8207
bash	8211
Test-2tasksRR.s	8811
test_sched	8812
test_sched	8837

Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
1	14:08:25.857398289	1	1094491	8837
32	14:08:26.155916536	2	387108	8837
31	14:08:26.146020169	3	381097	8837
33	14:08:26.165814610	4	380811	8837
34	14:08:26.175719841	5	380162	8837
30	14:08:26.136061447	6	376789	8837
29	14:08:26.126099818	7	375715	8837
28	14:08:26.116193536	8	243114	8837
85	14:08:26.680615235	9	223904	8837
35	14:08:26.185618116	10	223837	8837
105	14:08:26.878619194	11	223707	8837
109	14:08:26.918220118	12	223671	8837
99	14:08:26.819219389	13	223635	8837

Figure 12 : Periodic conflict

Results : priority inversion

The high priority task is blocked by the low priority task that is preempted because the medium priority task is running

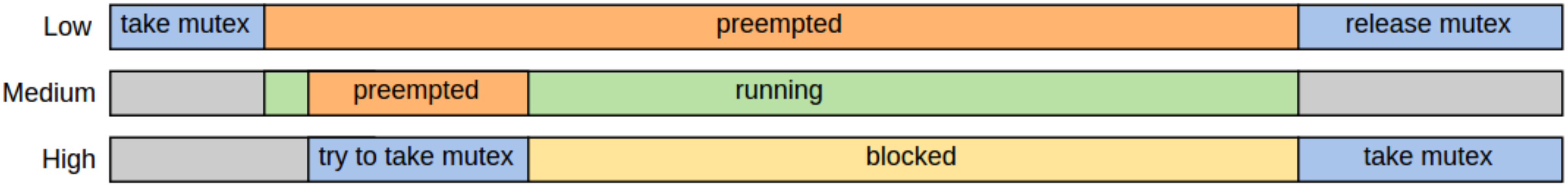


Figure 13 : Priority inversion

Results : priority inversion

Priority ceiling protocol

- Better if the high priority task accesses the resource more often than the low priority task, because it is faster and has fewer context switches, but it can give an unnecessary high priority to the lower task

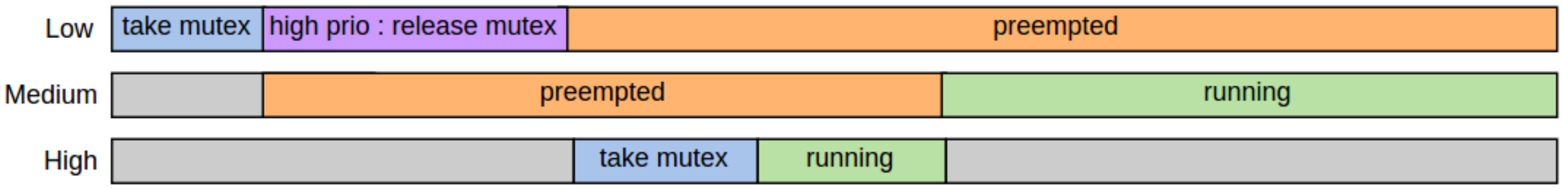
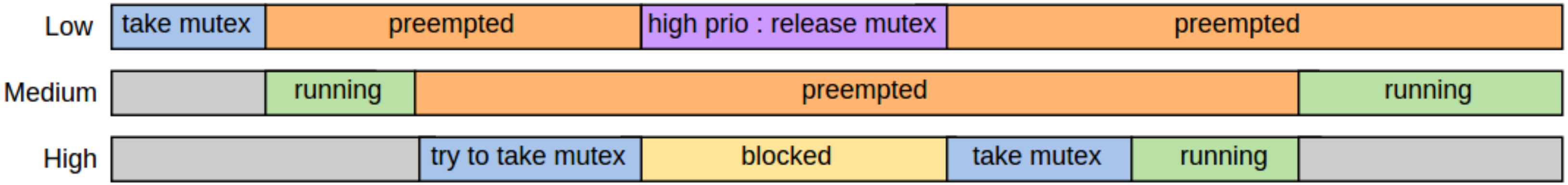


Figure 14 : Priority ceiling protocol

Results : priority inversion

Priority inheritance

- Better if the low priority task accesses the resource more often



Low: take mutex | preempted | high prio : release mutex | preempted

Medium: running | preempted | running

High: try to take mutex | blocked | take mutex | running

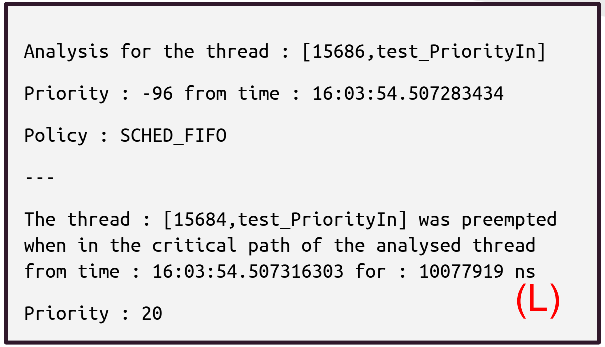
Figure 15 : Priority inheritance

Results : priority inversion

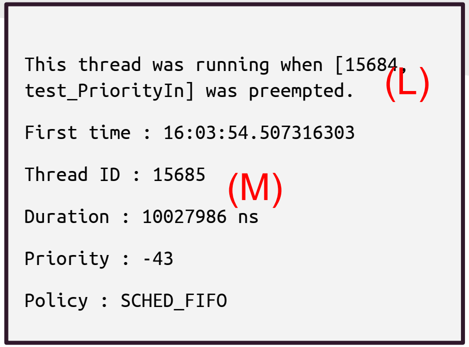
Tid:15684 -> low priority

Tid:15685 -> medium priority

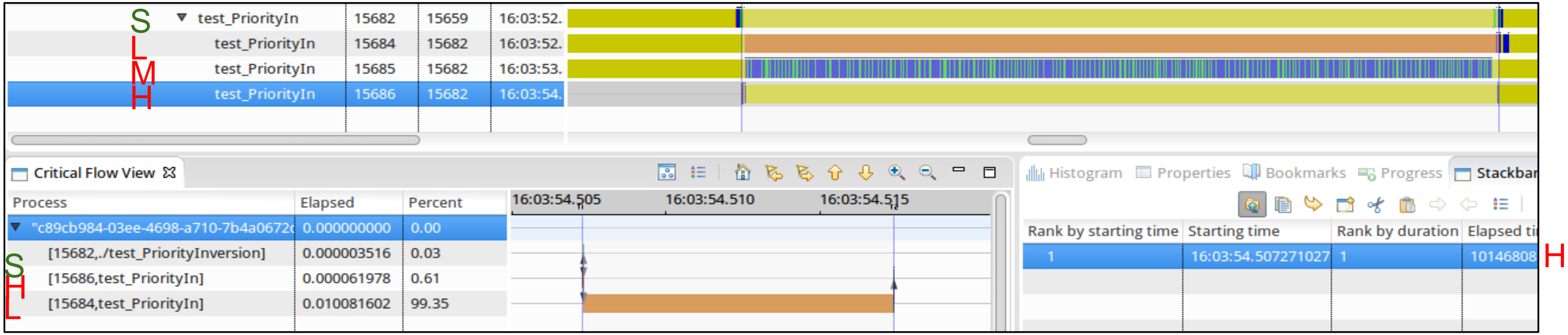
Tid:15686 -> high priority



```
Analysis for the thread : [15686,test_PriorityIn]

Priority : -96 from time : 16:03:54.507283434

Policy : SCHED_FIFO

---

The thread : [15684,test_PriorityIn] was preempted
when in the critical path of the analysed thread
from time : 16:03:54.507316303 for : 10077919 ns

Priority : 20
```



```
This thread was running when [15684,
test_PriorityIn] was preempted.

First time : 16:03:54.507316303

Thread ID : 15685

Duration : 10027986 ns

Priority : -43

Policy : SCHED_FIFO
```



▼ test_PriorityIn	15682	15659	16:03:52.
test_PriorityIn	15684	15682	16:03:52.
test_PriorityIn	15685	15682	16:03:53.
test_PriorityIn	15686	15682	16:03:54.

Critical Flow View

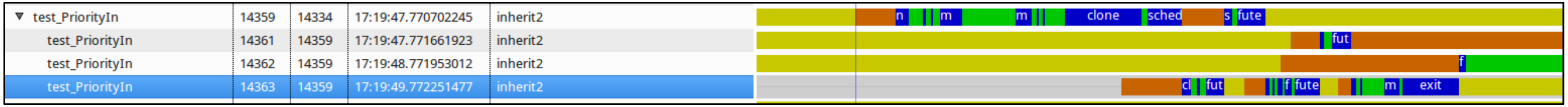
Process	Elapsed	Percent
▼ "c89cb984-03ee-4698-a710-7b4a0672(	0.000000000	0.00
[15682,./test_PriorityInversion]	0.000003516	0.03
[15686,test_PriorityIn]	0.000061978	0.61
[15684,test_PriorityIn]	0.010081602	99.35

Rank by starting time	Starting time	Rank by duration	Elapsed ti
1	16:03:54.507271027	1	10146808

Figure 16 : Priority inversion

Results : priority

Priority inheritance (PTHREAD_PRIO_INHERIT)



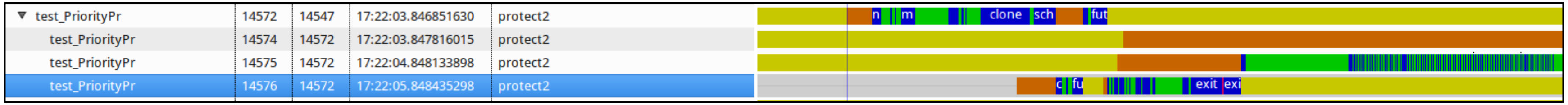
▼ test_PriorityIn	14359	14334	17:19:47.770702245	inherit2
test_PriorityIn	14361	14359	17:19:47.771661923	inherit2
test_PriorityIn	14362	14359	17:19:48.771953012	inherit2
test_PriorityIn	14363	14359	17:19:49.772251477	inherit2

Figure 17 : Priority inheritance protocol

Low priority temporarily set to the same priority as the high priority thread (-96) when high is blocked

Results : priority

Priority ceiling (PTHREAD_PRIO_PROTECT)



▼ test_PriorityPr	14572	14547	17:22:03.846851630	protect2
test_PriorityPr	14574	14572	17:22:03.847816015	protect2
test_PriorityPr	14575	14572	17:22:04.848133898	protect2
test_PriorityPr	14576	14572	17:22:05.848435298	protect2

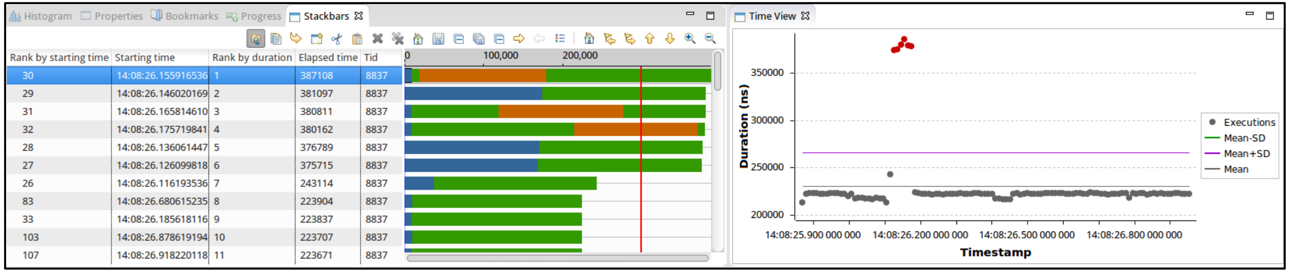
Figure 18 : Priority ceiling protocol

Low priority set to -96

Other results

Introduction
Literature
Modeling
Problems
Analysis
Results
Conclusion

- Deadline analysis
  - Tell which executions missed their deadlines
  - User input
  - Get it from events for SCHED_DEADLINE policy



Rank by starting time	Starting time	Rank by duration	Elapsed time	Tid
30	14:08:26.155916536	1	387108	8837
29	14:08:26.146020169	2	381097	8837
31	14:08:26.165814610	3	380811	8837
32	14:08:26.175719841	4	380162	8837
28	14:08:26.136061447	5	376789	8837
27	14:08:26.126099818	6	375715	8837
26	14:08:26.116193536	7	243114	8837
83	14:08:26.680615235	8	223904	8837
33	14:08:26.185618116	9	223837	8837
103	14:08:26.878619194	10	223707	8837
107	14:08:26.918220118	11	223671	8837

Figure 19 : Deadline

- Device blocked analysis

Conclusion

- Future work
  - Modeling
    - Instrument complex real-time application in user-space and for each task, validate if it is possible to model only with kernel events
  - Analysis
    - Validate with real bugs
    - Add new analysis
- Questions?