Lab1 Rasbian RT

Julia Desmazes

March 8, 2018

1 Latency test

1.1 Supported schedualing policies

1.2 Cyclictest

Cyclictest is a programme included in the rt_test suite to measure the latency of an particular environement.
It creats a defined number of threads that it wakes up at set intervals and then measures the latency between
the moment the thread was expected to wake-up and the actual time the action took place.
Reading the -h guide we can learn that -t allows us to set the number of threads we wish to create and -p sets
the priority.

threads -t More precisly if no parameter is give the number of threads will be equal to the number of cpus
and the execution of thows threads will be ballanced equlay[1] between they s different cpu's. As we are currently
on a 4 cpu system if we wish to create 4 threads leaving it to the default option would be correct. But if we
didn't specify the -t option then only 1 thread would have been created.

Listing 1: cyclictest -h
```
-t         --threads           one thread per available processor
-t [NUM] --threads=NUM     number of threads:
without NUM, threads = max_cpus
without -t default = 1
```

priorities -p In linux priority ranges are fixed by the schedualing algorythme used. by default cyclictest runs
SCHED_FIFO[2] and it's priorities will range from 0 to 99. Here priorities are reversed with 99 beeing the highest
priority and 0 the lowest.

Listing 2: cyclictest -h
```
-p PRIO  --prio=PRIO        priority of highest prio thread
```

1.3 Test

1.3.1 SHED_OTHER

Listing 3: preempt-rt kernel
```
^Cpi@raspberrypi:~/rt-tests$ sudo ./cyclictest --policy=other -t -n
policy: other/other: loadavg: 0.01 0.03 0.03 1/150 1324

T: 0 ( 1229) P: 0 I:1000 C:1242270 Min:      21 Act:    71 Avg:    69 Max:    2541
T: 1 ( 1230) P: 0 I:1500 C: 828181 Min:      21 Act:    67 Avg:    67 Max:    1991
T: 2 ( 1231) P: 0 I:2000 C: 621135 Min:      21 Act:    68 Avg:    68 Max:    4476
T: 3 ( 1232) P: 0 I:2500 C: 496909 Min:      21 Act:    67 Avg:    69 Max:    3180
```

---

[1]under the best possible conditions
[2]See patch for fix https://www.spinics.net/lists/linux-rt-users/msg05449.html

```
sudo ./cyclictest -t -n

policy: other/other: loadavg: 0.31 0.09 0.02 1/116 702

T: 0 (  695) P: 0 I:1000 C:1220752 Min:     15 Act:   66 Avg:   66 Max:    528
T: 1 (  696) P: 0 I:1500 C: 813835 Min:     15 Act:   65 Avg:   64 Max:    490
T: 2 (  697) P: 0 I:2000 C: 610376 Min:     16 Act:   66 Avg:   67 Max:   1354
T: 3 (  698) P: 0 I:2500 C: 488301 Min:     15 Act:   64 Avg:   65 Max:    545
```

1.3.2 SHED_FIFO

Figure 1: SHED_FIFO volontary

Figure 2: SHED_FIFO preempt-rl

1.3.3 SHED_RR

Figure 3: SHED_RR volontary

Figure 4: SHED_RR preempt-rl

1.4 Cyclictest + Hackbench

1.4.1 Hackbench under SHED_OTHER and priority 20

Figure 5: Hackbench under SHED_OTHER and priority 20

1.4.2 Volontary

Figure 6: Volontary fifo with hackbench

Figure 7: Volontary rr with hackbench

1.4.3 Preempt-rt

Listing 5: Preempt fifo with hackbench

```
pi@raspberrypi:~/rt-tests$ sudo ./cyclictest --policy=fifo -t -n -p99
policy: fifo: loadavg: 130.65 51.20 18.86 85/551 1404

T: 0 ( 1396) P:99 I:1000 C: 106248 Min:        7 Act:    26 Avg:    20 Max:
78
T: 1 ( 1397) P:99 I:1500 C:  70832 Min:        7 Act:    18 Avg:    20 Max:
73
T: 2 ( 1398) P:99 I:2000 C:  53124 Min:        7 Act:    24 Avg:    20 Max:
73
T: 3 ( 1399) P:99 I:2500 C:  42499 Min:        9 Act:    22 Avg:    21 Max:
74
```

Listing 6: Preempt rr with hackbench

```
pi@raspberrypi:~/rt-tests$ sudo ./cyclictest --policy=rr -t -n -p99
policy: rr: loadavg: 133.77 95.30 43.45 100/551 1454

T: 0 ( 1450) P:99 I:1000 C:  70877 Min:        8 Act:    20 Avg:    20 Max:
91
T: 1 ( 1451) P:99 I:1500 C:  47251 Min:        8 Act:    20 Avg:    21 Max:
83
T: 2 ( 1452) P:99 I:2000 C:  35438 Min:        9 Act:    23 Avg:    21 Max:
69
```

```
T: 3 ( 1453) P:99 I:2500 C:  28350 Min:       9 Act:   21 Avg:   21 Max:
92
```

1.4.4 Hackbench under RT scheduling policies and priority 49

Figure 8: Hackbench with shed_fifo and priority 49

Figure 9: Preempt-rt with fifo

Figure 10: Preempt-rt with fifo

SHED_FIFO

Listing 7: Hackbench with shed_rr and priority 49

```
pi@raspberrypi:~/rt-tests$ sudo chrt --rr 49 ./hackbench -l 1000000 -g 1q
Each sender will pass 1000000 messages of 100 bytes
```


Figure 11: Preempt-rt with fifo

Listing 8: Volontary kernel

```
sudo ./cyclictest --policy=rr -t -n -p99
policy: rr: loadavg: 29.39 27.96 14.37 32/161 848
```

```
T: 0 (  832) P:99 I:1000 C: 338191 Min:      8 Act:   18 Avg:   67 Max:   49660
T: 1 (  833) P:99 I:1500 C: 225480 Min:      8 Act:   54 Avg:   96 Max:   49563
T: 2 (  834) P:99 I:2000 C: 169312 Min:      8 Act:   18 Avg:  118 Max:   49511
T: 3 (  835) P:99 I:2500 C: 135316 Min:      9 Act:   19 Avg:  144 Max:   47964
```

2 ADA concurrent programs

2.1 Program 1

```
with Ada.Text_IO;
use Ada.Text_IO;
procedure Main is
--decleration of tasks
   task A;
   task B;
   task body A is
begin
loop
Put_Line("A");
Put_Line("B");
delay 1.0;
end loop;
end A;
   task body B is
begin
loop
Put_Line("C");
Put_Line("D");
delay 1.0;
end loop;
end B;
--creation of 2 instances of task
--main task
begin
null;
end Main;
```

Figure 12: Program 1

code

Observations The A,B,C,D are printed in order throwout the execution with there not beeing any apparent modification of that ordered sequence.

2.2 Program 2

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Real_Time;

with Ada.Calendar;


procedure Main2 is
--decleration of tasks
    task type A;
    task type B;
    task body A is
use Ada.Calendar;
begin
loop
delay 0.1 ;
Put_Line("A");
Put_Line("B");

end loop;
end A;
    task body B is
--declare usage
use Ada.Real_Time;
--variables of task
Wait_Time: Ada.Real_Time.Time;
Interval :constant Ada.Real_Time.Time_Span := Ada.Real_Time.Milliseconds (100);
begin
loop
Wait_Time := Clock+Interval;
delay until Wait_Time ;
Put_Line("C");
Put_Line("D");

end loop;
end B;
--creation of 2 instances of tasknatm

type A_ptr is access A;
type B_ptr is access B;

AA : A_ptr;
BB : B_ptr;
--main task
begin
AA := new A;
BB := new B;
end Main2;
```

Figure 13: Program 2

code

Observations In this test we can observe that even thow both treads are confirured to sleep for the same 100ms time interval they eventually end up off sync. We can attribute this to the diffrence between the way time is treated in the *delay* and *delay until*.

delay   This main diffrence is that *delay* uses an *approximate relative time delay*. It measured the time from a specific refrence point . If this refrence point si deisplaced in time by the process beeing preempted for example this introduces a delay of *at least* the amount of time specified,not the exact specified task. This introdoced local drift (delay) can be cumulated resulting in this unsynchronisation. If we wanted delay to wait for an absolute time then the task would have to run uninterrupted by any other task whitch is not the case here.

delay until   This statement sheduales for an absolute wake-up task, this works by making the task not schedualable until the interval time has elapsed. Her again there is no guaranty on the actual time the task will be executed if for example the ressource required to handle that task is not available (used by a higher priority task , ect ...) the task will still eventually stall.