April 5, 2018

RTOS

Lab 1

...

School:	ECE Paris
Class:	RTOS
Author:	Julia DESMAZES
	Andreas ORTALDA
	Nicolas VERHELST

Contents

1 The protected object Resource

1.1 Object mutual exclusion in Ada

The ada language allows us to create an objects howms private data will be updated only under mutual exclusion. This implicitly creats an acces barrier around our object that is evaluated on eatch call to our protected instance. If one of our protected writing tasks is beeing called then the barrier is evaluated to false and our call is blocked until the writing call is finished. At that point our barrier evaluation becomes true and we can acces the ressource for modification.
We use the **protected** keyword to enforce this behaviour for our subprograms on our protected entities.

Listing 1: chrt man page

```
Protected object
```

function to job_with_cpu_time_returned is declared outside our protected section so calling this function directly on our object will not result on a mutualy exclusif call. Yet, calling this exact same function in Lock will result in a mutulay excusive execution as Lock is declared as *protected* and it modifies our private ressource.

2 The concurrent programs Shared_2 and Shared_3

2.1 Shared_2

2.1.1 Multi-core

The first hight priority task is missed, this is because the ressource is being held by the lower priority task. Because of the Immediate Ceiling Priority protocole for ressource sharing the lower priority task has the same priority as our higher priority task thus the task can't be preempted and the highre priority task's deadline is missed.

2.1.2 Mono-core

At the beging of the program because our higher priority task begins it's execution run's, holds the ressource for 100ms and meats it's deadline. During the 200ms laxity before the next periode our second lower priority task starts it's execution. At the moment this tasks enters it's critical section because of our ICP protocol it is attributed an equal priority to our first task. As we are under fifo, when our first task is ready to resume execution it is unable to get preempt our second task and has to wait until the end of that task's execution to start running as it is in a critical section during all it's 500ms. When it does eventualy start running 300ms have elapsed since it's release and as the task only has a laxity of 200ms it is undertandable that it misses it's deadline.

2.2 Shared_3

Her the higher priority task never misses a deadline, as it doesn't share any resources with the other tasks it's remains the higher priority task and can preempt the other tasks during there executions.
The intresting section here is between the medium *MT* an low priority tasks *LT*: we observe that the fist instance of MT and LT respect there deadline yet the second realease of MT misses it.
Because HT stayes at a higher priority MT and LT can only be executed in a 100ms window every 200ms. During the fist occurence of this window MT executs it's entire task and meats it's deadline. So when the second window occurs MT has aleady finished it's execution and LT is called and enters it's critical section: it's priority is risen to equal the priority of MT. Thus during the third window, because both LT and MT now have the same prioity and we are under fifo, LT continues it's execution and again during the forth window before meating it's deadline just in time. Now when MT start's it's execution it has already missed it's deadline.

References

[1] The Ada 95 Protected Object, 2002, "**link: http://www.iuma.ulpgc.es/users/jmiranda/gnat-rts/node25.htm**", *Source to an indepth complementary explanation to help better undersand the protected object in Ada.*