NOM POUPA

Prénom Adrien

Promo M1 2018

Date 11/04/17

POUPA Adrien
M1 - 2016

20 1

MATIÈRE Real Time

1) Course question

1. Real time system: System that must respond to time constraints and handle tasks as soon as possible in a given window before a deadline.
It is used for systems that have major time constraints ("électronique embarquée") 1

2. Hard real time means that a window should never ever be missed no matter what. It is for critical systems where a failure can be fatal (eg: avionics)
Soft real time: a window can be missed if necessary, it is bad but not very severe. (eg: videogames, video conference) 1

3. Déterminisme: all cases ~~have~~ been taken into account

Reliability : the system will ~~not~~ fail and miss a window
Fiabilité

Prédictabilité : the system will respond always the same for a given input

problems ?

4. With a less efficient processor, tasks will take more time with a more efficient processor, tasks will execute more quickly.

5. When sending a burst of interruptions on a UNIX system, it can miss some of them. why ? 0,5
There is no "real" solution because the system will always have a hardware limitation to the number of interruptions it can handle simultaneously.

6. WCET : Worst Case of Execution Time. 1
Useful to know what is the maximum time a task can take to complete.

7. Priority inversion : if a resource A | acquires takes a lock on a shared resource, then is pre empted by a resource B with a higher priority but the resource B wants to acquire the shared resource locked by A.

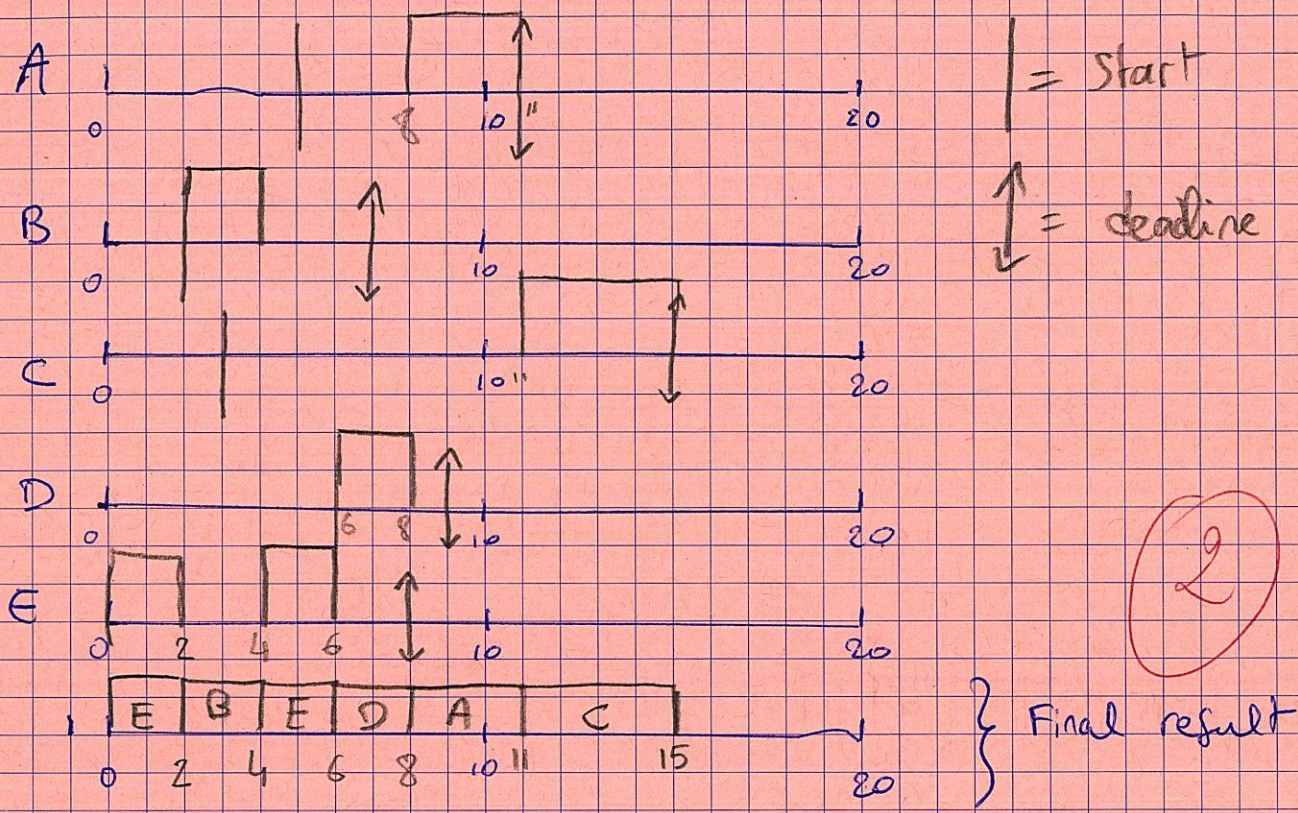
To avoid this, we use priority inheritance protocol : priority of resource A is changed to the priority of resource B so that A can complete before B.

it was already the case ...

0,5

2) Scheduling

8. EDF. Priority = $\frac{1}{\text{deadline}}$ P(B) > P(E) > P(D) > P(A) > P(C)



9. LLF. Priority = $\frac{1}{\text{laxity}}$

We will compute the laxity whenever a task spawns:
t = 0, t = 2, t = 3, t = 5, t = 6

From t=0 to t=2, only E is available, so we execute it.

$\Pi_d(t=0)\ E = (8-0) - 4 = 4$

At t=2, B becomes available

$\Pi_d(t=2)\ E = (8-2) - 2 = 4$

$\Pi_d(t=2)\ B = (7-2) - 2 = 3$

We take B until t = 3

$\Pi_d(t=3)\ E = (8-3) - 2 = 3$

$\Pi_d(t=3)\ B = (7-3) - 1 = 3$

$\Pi_d(t=3)\ C = (15-3) - 4 = 8$

E and B are equivalent. We take E until 5

At $t=5$, A becomes available

Md ($t=5$) B = $(7-5)-1 = 1$

Md ($t=5$) A = $(11-5)-3 = 3$

Md ($t=5$) C = $(15-5)-4 = 6$

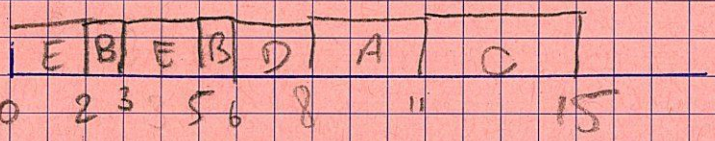
We take B until $t=6$. Then D becomes available

Md ($t=5$) A = $(11-6)-3 = 2$

Md ($t=5$) C = $(15-6)-4 = 5$

Md ($t=5$) D = $(9-6)-2 = 1$

We take D, A and C.



Result

2

ie. $W = \sum \frac{\text{Execution time}}{\text{Periode}}$

$W = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{1}{3} + \frac{5}{8} = \boxed{\frac{23}{24} < 1}$

0,5

1,5

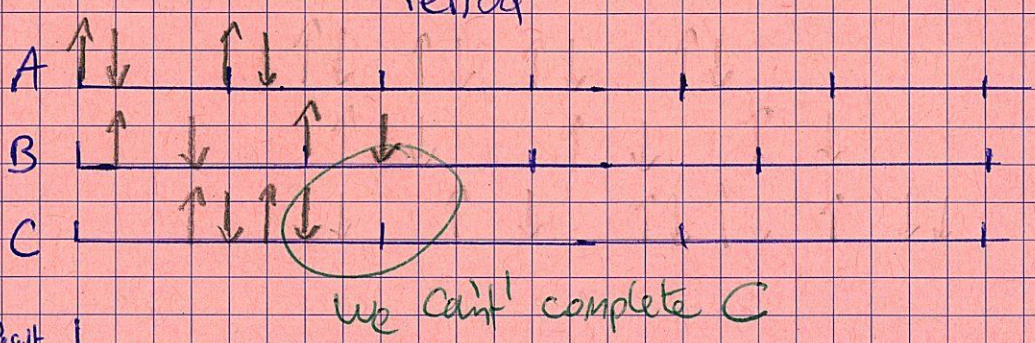
We have $W < 1$ (necessary condition)

but $W > U(n)$ (while $W < U$ is a sufficient condition)

⇒ RMS is not optimal but there can be a solution.

To conclude, we must proceed to do the policy.

11. RMS Priority = $\frac{1}{\text{Period}}$. $P(A) > P(B) > P(C)$



1,5

NOM Poupa

Prénom Adrien

Promo M1 2018

Date 11/04/17

MATIÈRE Real Time

12. IF we increase the CPU to be 1 second Faster it will work.

13. Since $U \leq 1$, EDF and LCF are optimal (1)

14. IF K is set to its minimum ($K=1$), we will need 3 processors to complete in 7 time units.
So depending on K's value we need at least 3 processors.

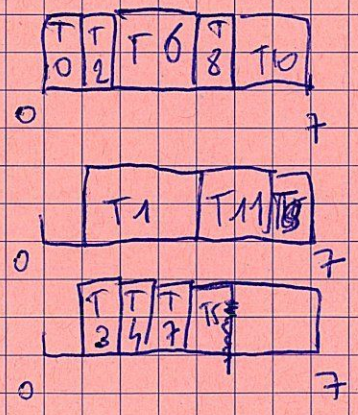
Because: total time = 21 (if $K=1$) (1)

$\frac{21}{7} = 3$.

ok

15.

	+ tot	+ tard	C
T0	0	14	1
T1	1	15	3
T2	1	14	1
T3	1	16	1
T4	2	17	1
T5	2	15	3
T6	2	16	2
T7	3	18	K(=1)
T8	5	18	1
T9	4	18	3
T10	5	19	2
T11	3	19	2



(0,5)

3) Synchronization

16. IF a thread starts to execute B after we have done P(sem1) in A, but not P(sem2) in A yet we have a deadlock
(ie: we have executed the first line of A & B in parallel.

17. ① Invert the order of PS:

A	B
P(sem1)	P(sem1)
P(sem2)	P(sem2)
...	...
V(sem2)	V(sem2)
V(sem1)	V(sem1)

② use a mutex where?

A	B
P(mutex)	P(mutex)
...	...
V(mutex)	V(mutex)

③ Use a private semaphore

P(se-)	V(se-)
P(se-)	V(se-)

18. Producer/Consumer problem ⇒ 2 semaphors

Free: number of free spaces (class)

occ: number of occupied spaces (class)

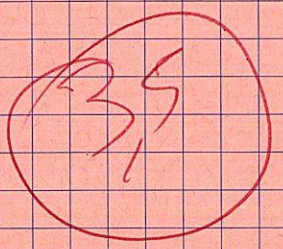
⇒ 1 mutex lib to prevent two writers to write simultaneasly

Reader

P(occ)

read_buffer()

V(free)



Writer

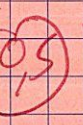
P(free)

P(lib)

write_buffer(data)

V(lib)

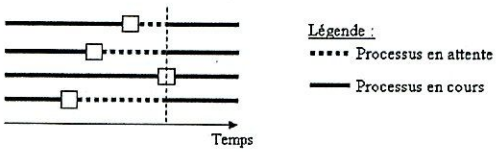
V(occ)

19. Copy is joined. ⇒ 

POUPA
Adrien

Exercice 3.3 : Synchronization barrier for N threads

A synchronization barrier ensures that several threads have reached a particular moment. Thus, there are several threads that run in parallel and they need to wait for the last of them to finish (one section of the code) before they can all move on. The figure below illustrates the operation of this synchronization barrier (or rendez-vous) for 4 threads.



Question 19. Write the code of a synchronization barrier mechanism between N threads using mutex and/or semaphores and the primitves P() and V() (as in the course). Fill the empty cells in the program given in figure ??.

Notes : – Note that some of the empty cells can be remain empty and some other(s) can contain multiple lines of code if needed ;
– La global variable « nb_process »represents the remaining number of threads to be synchronized (i.e. the number of threads that have not reached the barrier). It is initialized to N.

Sem : initialized at N

```
void barriere(void)
{
    int i = 0;
    P(Sem)
    nb_process--;
    
    if(nb_process == 0)
    {
        for(i=0 ; i<N ; i++)
        {
            V(Sem)
        }
        nb_process = N;
    }
    
}
```

(9)