DJILALI BOUNAAMA KHEMIS MILIANA UNIVERSITY
Faculty of Material Sciences and Computer Science
DEPARTMENT OF COMPUTER SCIENCE

Jan 16th ,2025
3rd year L3-CS
T : 1h & 30mn

FINAL EXAME ()

OPERATING SYSTEMS *(A)*

ID: .....G:.....

NAME: .....

Exercice 1 : ( )

Etant donné le programme ci-dessous en supposant que le PID Shell est **1570**, le PID correspondant à ce programme est **1980** et que le système affecte des identificateurs séquentiels aux nouveaux processus.

```
/* Programme Exam_2.c */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main ()
{
        int i= 4 ;
        int j=1;
        int p ;

        p = fork() ;  i ++ ;
        if (p == 0) {
        int i = 1 ;
        p = fork () ; i += 2 ; j += 4 ;
                        if (p == 0) {
                           p = fork () ; j *= 3 ;

                         printf("processus=%d,i=%d; j=%d ;Proc.père=%d\n",getpid(),i,j,getppid());
                        return 0 ;
                        }
                }
                else     {        j=2;
                        p = fork () ;j--; i *= 5 ;
                }
                p = fork () ; i += 6 ; j *= 3 ;
                printf("processus=%d,i=%d;              j=%d;Proc.père=%d\n",getpid(),i,j, getppid()) ;
                return 0 ;
}
```

1. Dérouler le programme et compléter les informations suivantes. : **1**

Le nombre de processus générés dans le programme est : .....8.....

	Min	Max **1**
PID	1980	1987
i	3	31
j	3	15

```
C:\TP\testexam>Exam_2
processus=1980,i=31;     j=3;Proc.père=1570
processus=1981,i=11;     j=15;Proc.père=1980
processus=1982,i=31;     j=3;Proc.père=1980
processus=1983,i=31;     j=3;Proc.père=1980
processus=1984,i=3;      j=15 ;Proc.père=1981
processus=1985,i=31;     j=3;Proc.père=1982
processus=1986,i=11;     j=15;Proc.père=1981
processus=1987,i=3;      j=15 ;Proc.père=1984
```

Exercice 2 :

Consider N processes Pi and an independent process Pr, with the following schema: :

Process Pi:	**Independent Process Pr:**
Begin	**Begin**
PA ;	IA;
PB ;	IB;
End	**End**

(PA, PB, IA, and IB are blocks of instructions)

- The N processes Pi and the independent process Pr execute in parallel.
- Each process Pi executes the instruction block PA and then blocks.
- After completing the instruction block IA, the independent process Pr waits for all processes Pi to finish their respective PA blocks; it then proceeds to execute the IB block.
- Once the IB block is completed, the independent process Pr releases all blocked processes Pi, allowing them to continue their execution.

Q: Propose a synchronization schema for the processes Pi and the independent process Pr using semaphores.

Declarations :
SMaitre : semaphore (init to 0);
SP : semaphore (init to 0);
mutex : semaphore (init to 1);
i : entier (init to 0);
N : constant representing the number of processes « Pi ».

Processus Pi	**Processus Maître**
Begin	Begin
PA ;	MA ;
P(mutex) i := i + 1; V(mutex)	P(SP);
Si i == N alors V(SP) Finsi	MB ;
P(SMaitre);	V(SMaitre)
PB ;	End.
V(SMaitre)	
End.	

Exercise 3:

Consider two categories of activities: bakers and customers.

- **Bakers** bake pastries and place them in a shared display case called: **Pastry Display**.
- **Customers** purchase and consume the pastries placed in the display case.
- The display case has a limited capacity of **N** pastries.

The operation of these two categories of activities must satisfy the following constraints:

1. **Bakers** do not place more pastries when the display case is full.
2. **Customers** do not take pastries from the display case when it is empty.
3. **Only one person** (either a baker or a customer) can access the display case at a time.
4. Pastries must not be lost or consumed twice.

Q: Give the synchronization solution using monitors.

```
Program ProducersConsumers;
Const  N=…;
Type    object=….;

Monitor ProdCons;
Const  N=…;

Var      Buffer : Array [0…N-1] of object;
         nonEmpty , nonFull  : condition;
         in,out : integer
         Counter:0….N-1;

Procedure deposit (ob:object);
Begin
          If Counter=N then nonFull.wait;
          Buffer[in]:=ob;
          In:=in+1modN;
          Counter:=Counter+1;
          nonEmpty.signal;
End;

Procedure withdraw (var ob:object);
Begin
          If Counter=0 then nonEmpty.wait;
          ob:= Buffer[out];
          out:=out+1modN;
          Counter:=Counter-1;
          nonFull.signal;
End;


Begin
          Counter:=0;
          In:=0;
          Out:=0;
End;
```

```
Process Baker-I;
Var  objectproduce:object;
Begin
   Repeat
          Produce (objectproduce);
Call ProdCons. deposit (objectproduce);

    Until End= true;
```

```
Process Customer-j;
Var objectconsume: object;
Begin
Repeat
Call ProdCons.withdraw (objectconsume);

consume (objectconsume);
 Until Fin= true;
```

End ;	**End ;**

Begin
ParBegin
 	
Baker-1;Baker-2; Baker-3; …….; Baker-I;
Customer-1; Customer-2; Customer-3;……; Customer-j;
ParEnd;
End;	