Ingénieur systéme multi-coeurs et multiprocesseurs

COMPTE RENDU

Réalisé par :

Aya Youssef

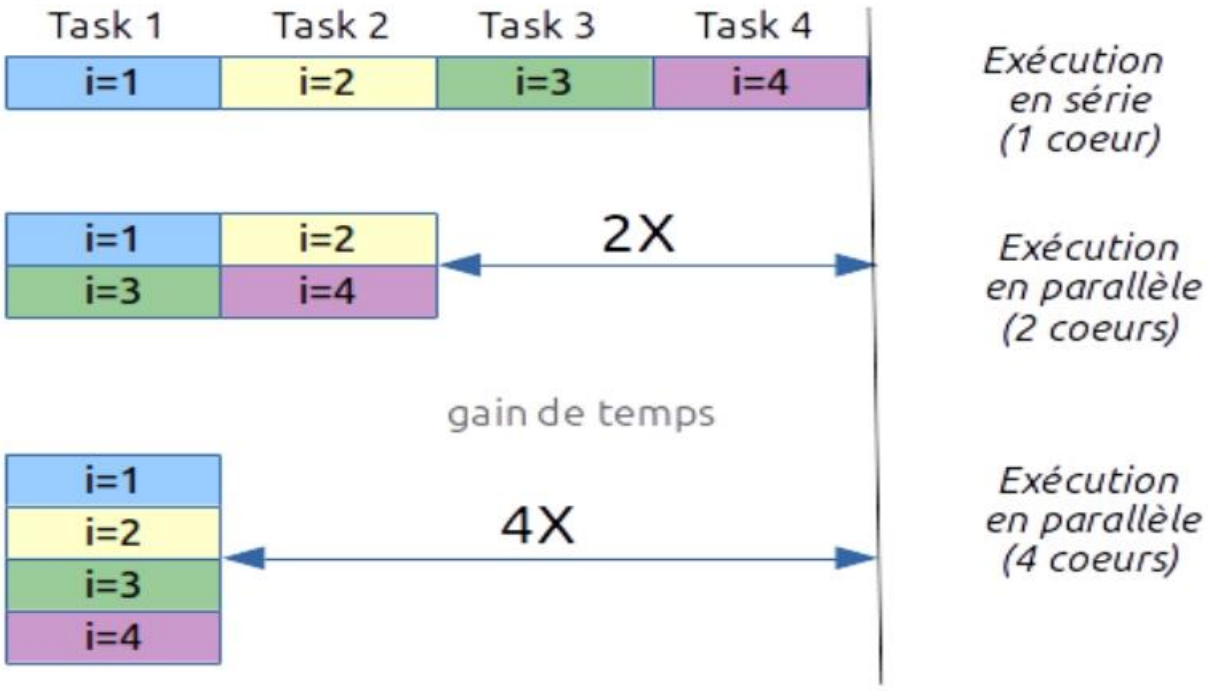
4éme année génie informatique G2.2

Introduction :

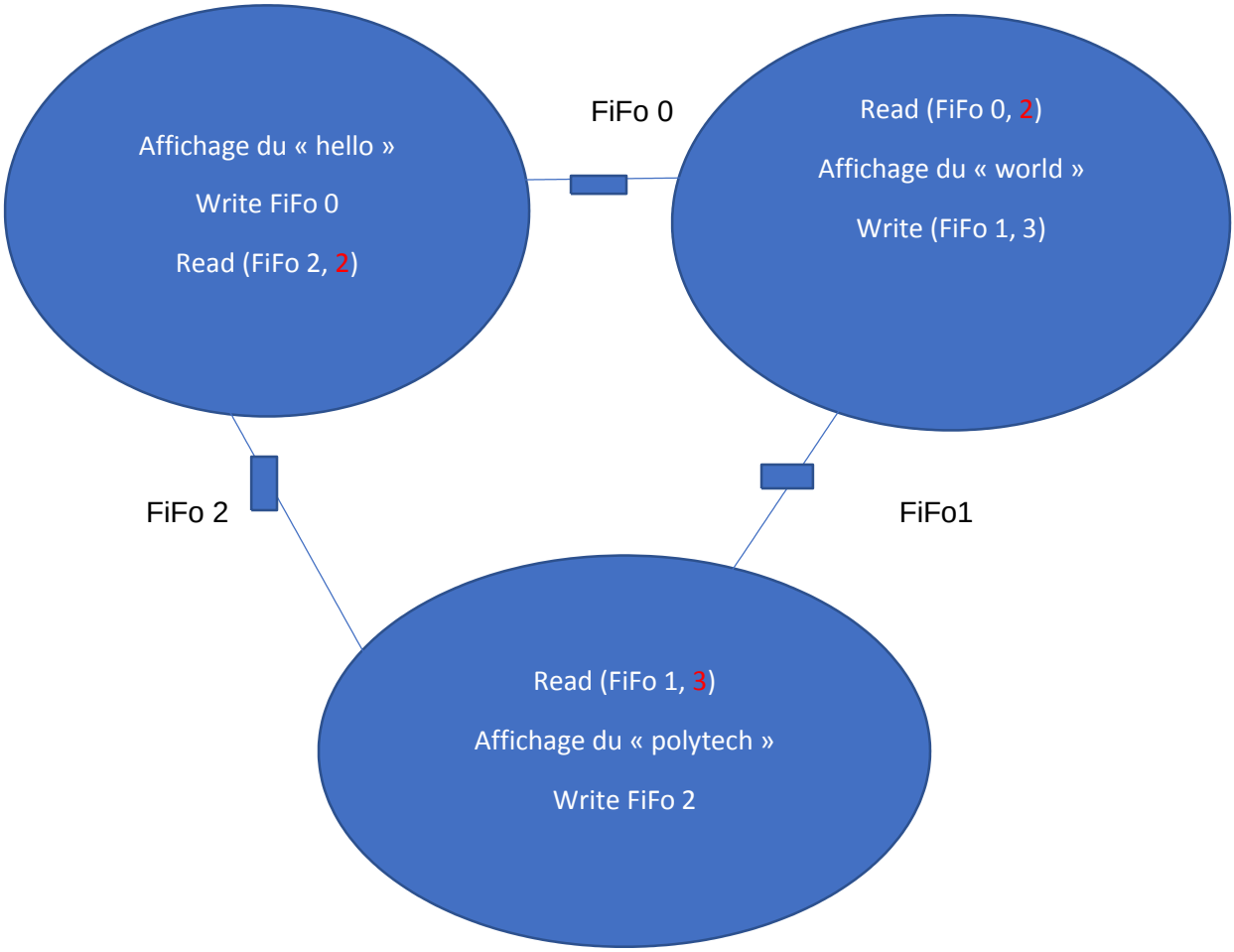
Plusieurs formes de parallélismes sont exploitables dans les systèmes informatiques. Les ordinateurs multiprocesseurs permettent un parallélisme de tâches, où un processus peut être exécuté sur chaque processeur. On obtient ainsi une plus grande puissance de calcul qu'avec un ordinateur uniprocessor, qui peut être utilisée soit pour plusieurs programmes qui disposeraient chacun d'un processeur, soit pour des programmes spécialement conçus, qui sont capables de répartir leurs calculs sur les différents processeurs.

Cette technologie a été utilisée pour des supercalculateurs, mais elle peut aussi l'être pour s'affranchir des limites de la montée en fréquence des processeurs : de nombreux processeurs actuels sont dits multicœurs, et embarquent en fait plusieurs uniprocessors sur une même puce.

Exemple du systéme parralleéle :



TP1 : MULTI-TACHES



Taches T	Hello	World	Polytech
T0	Affichage du “hello”	bloquée	bloquée
T1	Write fifo0	bloquée	bloquée
T2	bloquée	Read fifo0	bloquée
T3	bloquée	Affichage du “world”	bloquée
T4	bloquée	Write fifo1	bloquée
T5	bloquée	bloquée	Read fifo1
T6	bloquée	bloquée	Affichage du “polytech”
T7	bloquée	bloquée	Write fifo2
T8	Read fifo2	bloquée	bloquée

Code du fichier “***hello.c” :***

```
#include <srl.h>
#include "hello_proto.h"

FUNC(hello)
{
int a[1];

        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_log_printf(NONE, "tâche1 : Hello...\n");
        srl_mwmr_write(fifo0, &(a[0]), 4);
        srl_mwmr_read(fifo2, &(a[0]), 4);
        }
```

Code du fichier “***hello.task” :***

```
TaskModel(
        'hello',
        ports = { 'fifo0' : MwmrOutput(4), 'fifo2' : MwmrInput(4) },
        impls = [
                SwTask('hello',
                        stack_size = 2048,
                        sources = ['hello.c'])
        ] )
```

Code du fichier “***world.c” :***

```c
#include <srl.h>
#include "world_proto.h"

FUNC(world)
{
int a[1];
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);


        srl_mwmr_read(fifo0, &(a[0]), 4);
        srl_log_printf(NONE, "tâche2 : ... world\n");
        srl_mwmr_write(fifo1, &(a[0]), 4);

}
```

Code du fichier “***world.task” :***

```
TaskModel(
        'world',
        ports = { 'fifo0' : MwmrInput(4), 'fifo1' : MwmrOutput(4)},
        impls = [
                SwTask('world',
                        stack_size = 2048,
                        sources = ['world.c'])

        ] )
```

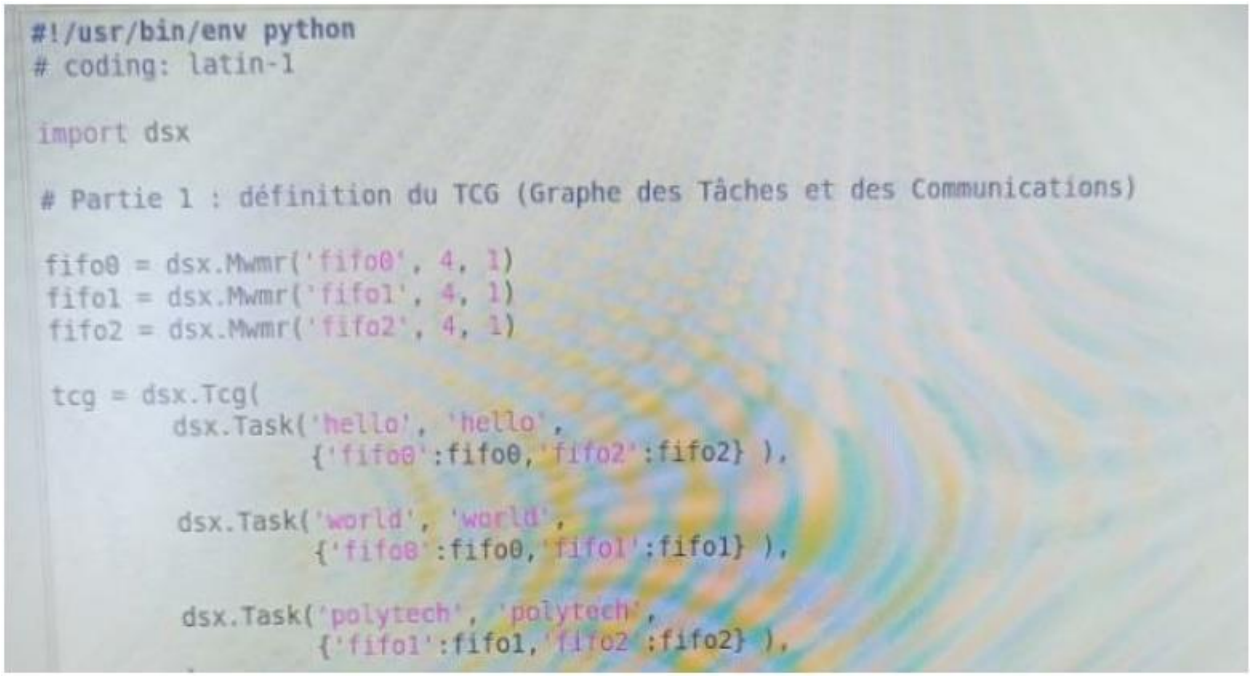
Code du fichier “***polytech.c” :***

```c
#include <srl.h>
#include "polytech_proto.h"

FUNC(polytech)
{
int a[1];
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_mwmr_read(fifo1, &(a[0]), 4);
        srl_log_printf(NONE, "tâche3 : ... polytech\n\n");
        srl_mwmr_write(fifo2, &(a[0]), 4);
```

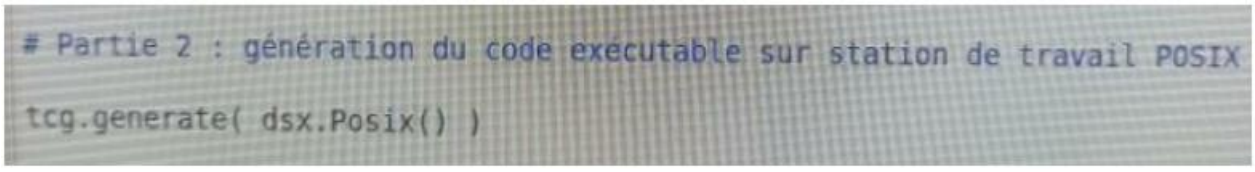
Code du fichier “***polytech.task” :***

```
TaskModel(
        'polytech',
        ports = { 'fifo1' : MwmrInput(4), 'fifo2' : MwmrOutput(4)},
        impls = [
                SwTask('polytech',
                        stack_size = 2048,
                        sources = ['polytech.c'])

        ] )
```

Code du fichier python ‘’hello.py’’ :



```
#!/usr/bin/env python
# coding: latin-1

import dsx

# Partie 1 : définition du TCG (Graphe des Tâches et des Communications)

fifo0 = dsx.Mwmr('fifo0', 4, 1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 4, 1)

tcg = dsx.Tcg(
        dsx.Task('hello', 'hello',
                 {'fifo0':fifo0,'fifo2':fifo2} ),

        dsx.Task('world', 'world',
                 {'fifo0':fifo0,'fifo1':fifo1} ),

        dsx.Task('polytech', 'polytech',
                 {'fifo1':fifo1,'fifo2':fifo2} ),
```

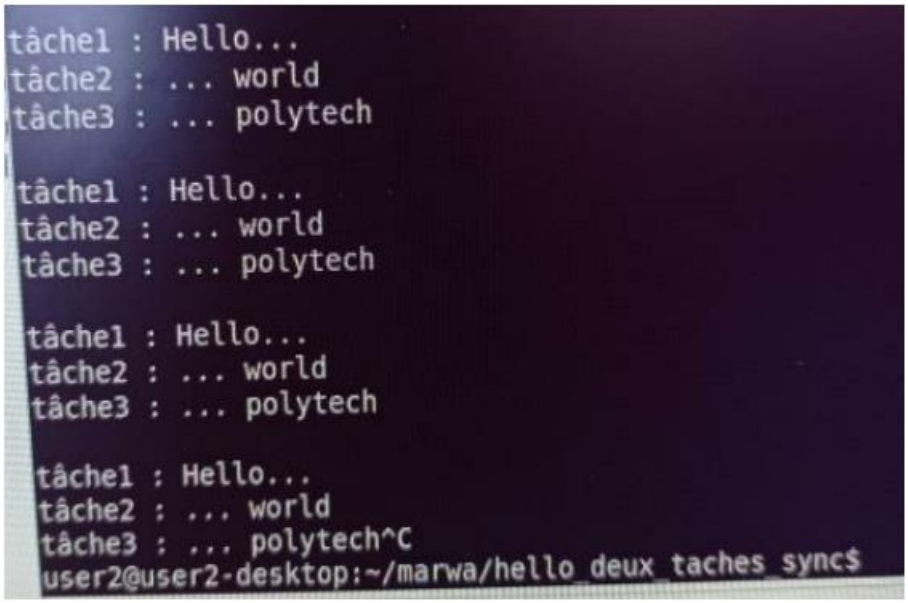


```
# Partie 2 : génération du code executable sur station de travail POSIX

tcg.generate( dsx.Posix() )
```

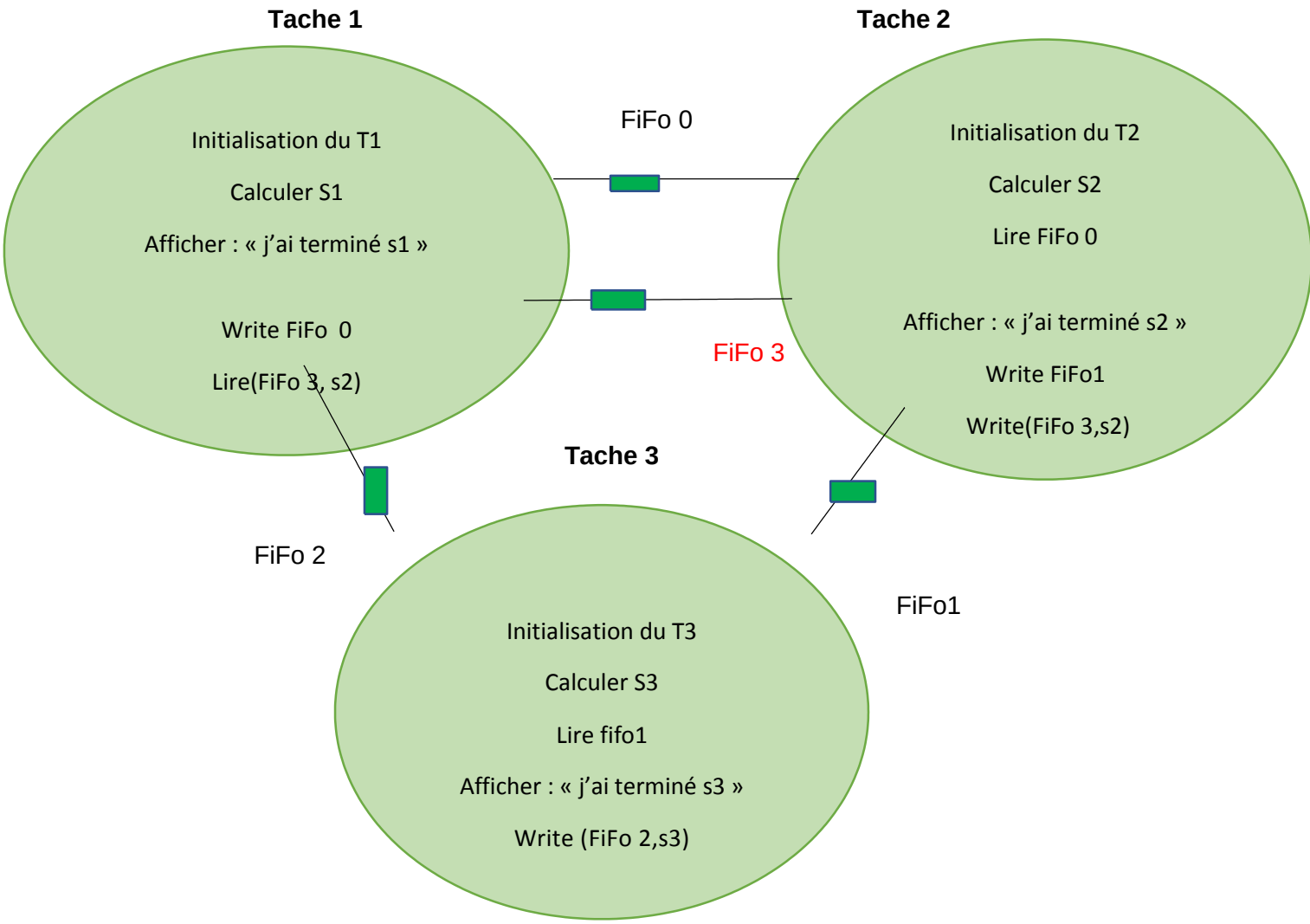
Exécution

On fait l’exécution avec ‘’ ***./exe.posix*** ‘’

La sortie du programme précédent est la suivante :

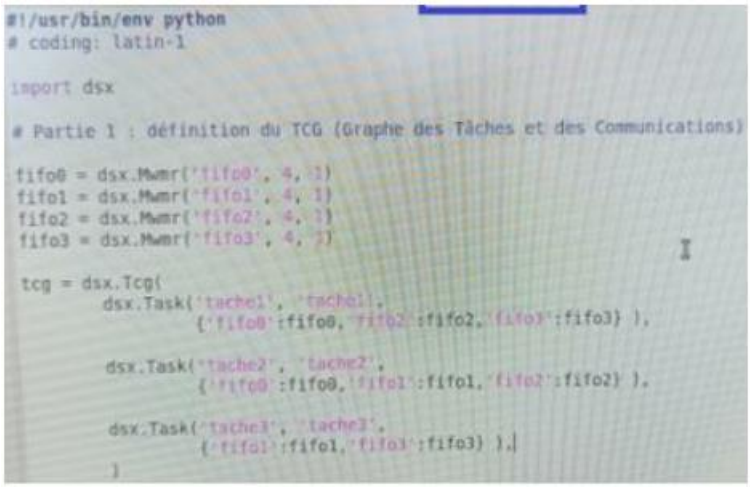


TP2: Calcul de Somme



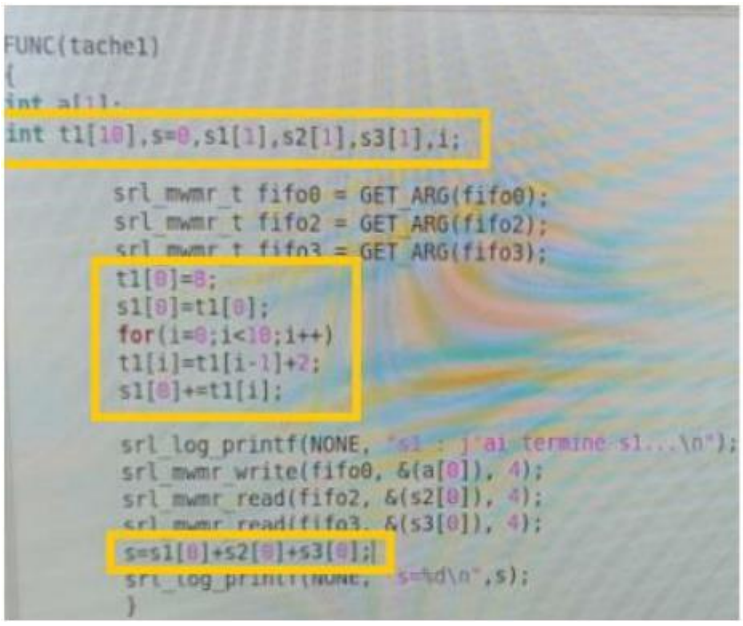
Taches T	Tache 1	Tache 2	Tache 3
T0	Initialisation T1	Initialisation T2	Initialisation T3
T1	Calculer S1	Calculer S2	Calculer S3
T2	Affichage "..S1"	bloquée	Bloquée
T3	Ecrire FiFo 0	bloquée	Bloquée
T4	bloquée	Lire fifo 0	Bloquée

Code du fichier “***somme.py” :***



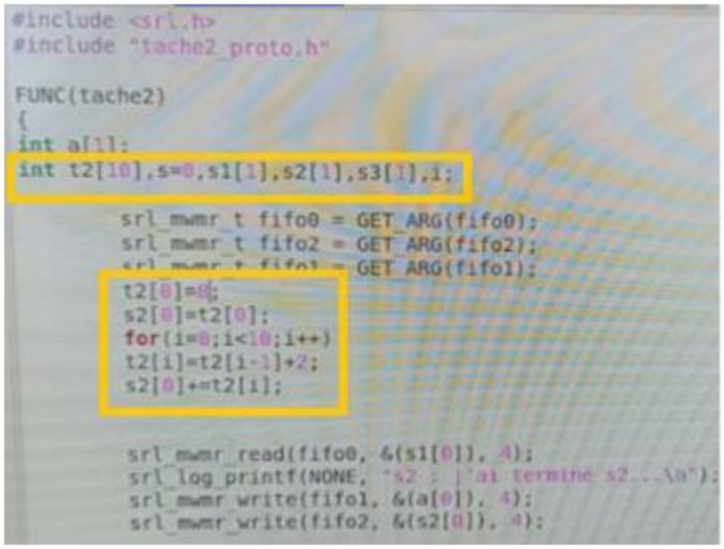
```
#!/usr/bin/env python
# coding: latin-1

import dsx

# Partie 1 : définition du TCG (Graphe des Tâches et des Communications)

fifo0 = dsx.Mwmr('fifo0', 4, 1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 4, 1)
fifo3 = dsx.Mwmr('fifo3', 4, 1)

tcg = dsx.Tcg(
        dsx.Task('tache1', 'tache1',
                 {'fifo0':fifo0,'fifo2':fifo2,'fifo3':fifo3} ),

        dsx.Task('tache2', 'tache2',
                 {'fifo0':fifo0,'fifo1':fifo1,'fifo2':fifo2} ),

        dsx.Task('tache3', 'tache3',
                 {'fifo1':fifo1,'fifo3':fifo3} ),
        )
```

Code du fichier “***tache1.c” :***



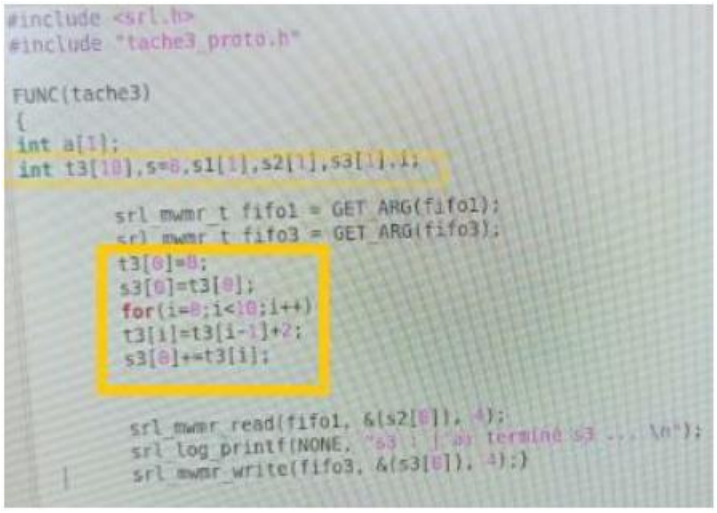
```
FUNC(tache1)
{
int a[1];
int t1[10],s=0,s1[1],s2[1],s3[1],i;

        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);
        srl_mwmr_t fifo3 = GET_ARG(fifo3);
        t1[0]=8;
        s1[0]=t1[0];
        for(i=0;i<10;i++)
        t1[i]=t1[i-1]+2;
        s1[0]+=t1[i];

        srl_log_printf(NONE, "s1 : j'ai termine s1...\n");
        srl_mwmr_write(fifo0, &(a[0]), 4);
        srl_mwmr_read(fifo2, &(s2[0]), 4);
        srl_mwmr_read(fifo3, &(s3[0]), 4);
        s=s1[0]+s2[0]+s3[0];
        srl_log_printf(NONE, "s=%d\n",s);
        }
```

Code du fichier “***tache2.c” :***



```
#include <srl.h>
#include "tache2_proto.h"

FUNC(tache2)
{
int a[1];
int t2[10],s=0,s1[1],s2[1],s3[1],i;

        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        t2[0]=0;
        s2[0]=t2[0];
        for(i=0;i<10;i++)
        t2[i]=t2[i-1]+2;
        s2[0]+=t2[i];

        srl_mwmr_read(fifo0, &(s1[0]), 4);
        srl_log_printf(NONE, "s2 : j'ai termine s2...\n");
        srl_mwmr_write(fifo1, &(a[0]), 4);
        srl_mwmr_write(fifo2, &(s2[0]), 4);
```

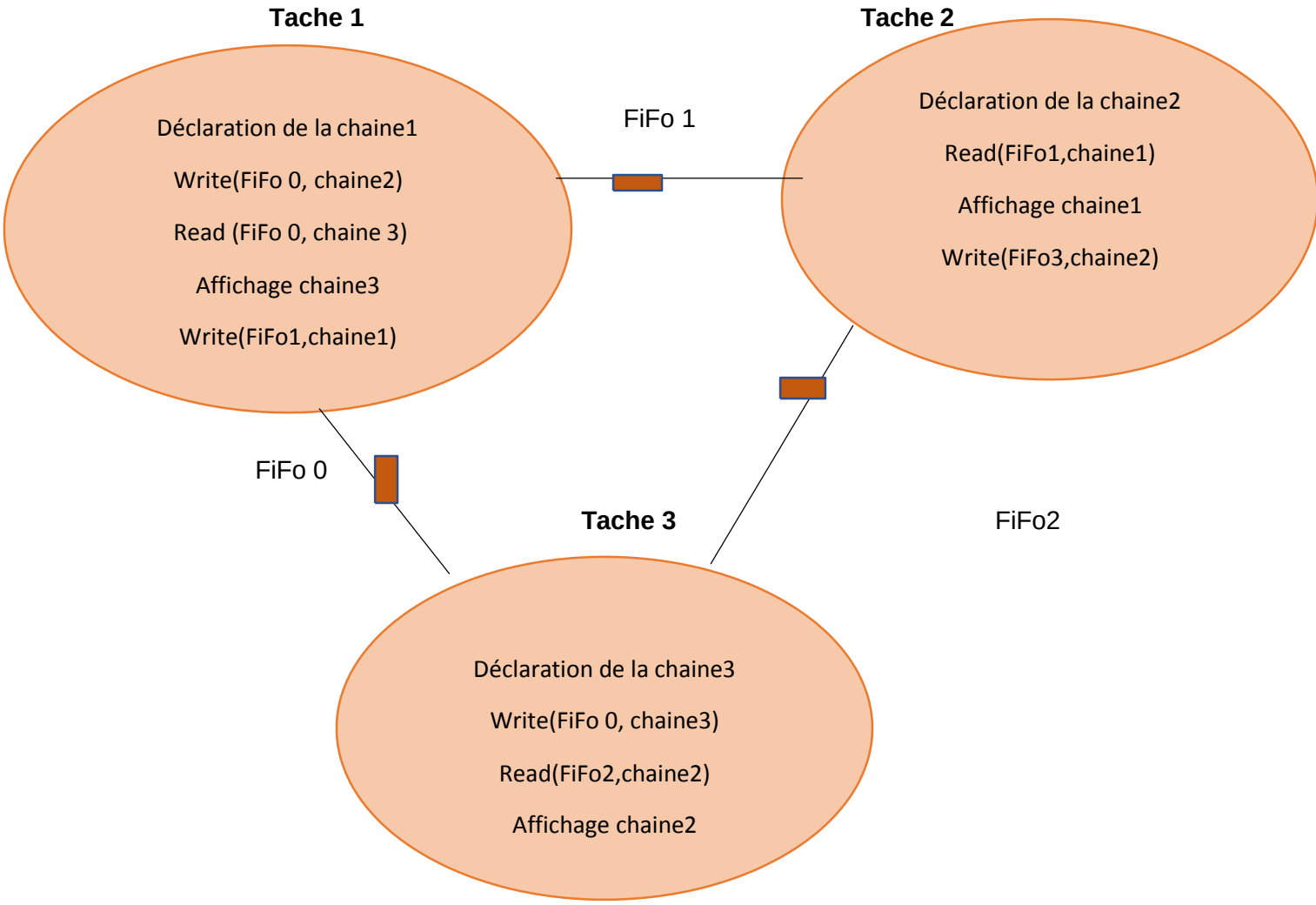
Code du fichier “***tache2.task” :***



```
#include <srl.h>
#include "tache3_proto.h"

FUNC(tache3)
{
int a[1];
int t3[10],s=0,s1[1],s2[1],s3[1],i;

        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo3 = GET_ARG(fifo3);
        t3[0]=0;
        s3[0]=t3[0];
        for(i=0;i<10;i++)
        t3[i]=t3[i-1]+2;
        s3[0]+=t3[i];

        srl_mwmr_read(fifo1, &(s2[0]), 4);
        srl_log_printf(NONE, "s3 : j'ai termine s3 ...\n");
        srl_mwmr_write(fifo3, &(s3[0]), 4);}
```

On fait l’exécution avec ‘’ ***./exe.posix*** ‘’

TP3 : Partage de données entre tâches



Taches T	Tache 1	Tache 2	Tache 3
T0	Déclaration de la chaine1	Déclaration de la chaine2	Déclaration de la chaine3
T1	bloquée	bloquée	Write(FiFo 0, chaine2)
T2	Read (FiFo 0, chaine 3)	bloquée	bloquée
T3	Affichage chaine3	bloquée	bloquée
T4	Write(FiFo1,chaine1)	bloquée	bloquée

T5	Déclaration de la chaine1	Read(FiFo1,chaine1)	bloquée
T6	bloquée	Affichage chaine1	

Ce TP contient les fichiers suivants :

Code « ***hello.c » :***

```
#include <srl.h>
#include "hello_proto.h"

FUNC(hello)
{
char *chaine1="world";
char *chaine2;
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);

        srl_mwmr_read(fifo0, &chaine2, 4);
        srl_log_printf(NONE, "tache 1: %s\n", chaine2);

        srl_mwmr_write(fifo1, &chaine1, 4);

}
```

Code « ***world.c » :***

```c
#include <srl.h>
#include "world_proto.h"

FUNC(world)
{
char *chaine="polytech";
char *ch2;
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_mwmr_read(fifo1, &ch2, 4);
        srl_log_printf(NONE, "tache 2 : %s\n", ch2);

        srl_mwmr_write(fifo2, &chaine, 8);
}
```

Code « ***poly.c » :***

```c
#include <srl.h>
#include "poly_proto.h"

FUNC(poly)
{
char *ch="hello";
char *ch2;
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        srl_mwmr_write(fifo0, &ch, 4);
        srl_mwmr_read(fifo2, &ch2, 8);

        srl_log_printf(NONE, "tache3 : %s\n", ch2);


}
```

Code « ***hello.py » :***

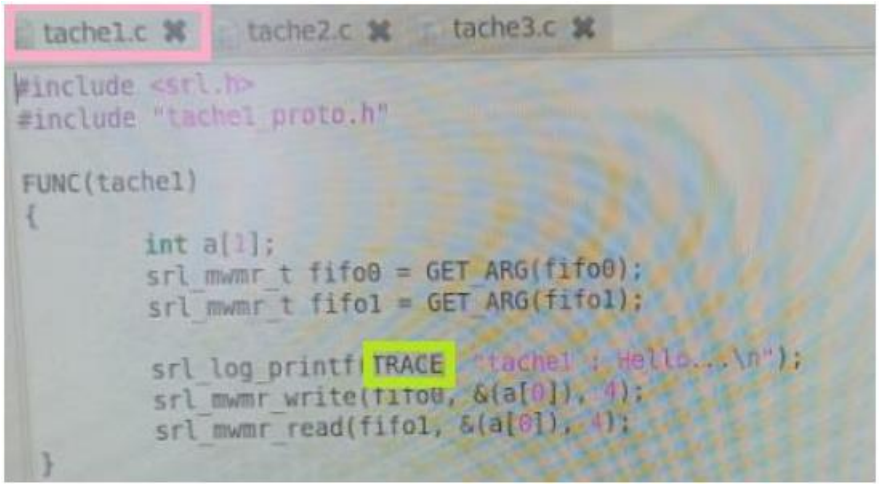
```python
# Partie 1 : définition du TCG (Graphe des Tâches et des Communications)

fifo0 = dsx.Mwmr('fifo0', 4, 1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 8, 1)

tcg = dsx.Tcg(
        dsx.Task('hello', 'hello',
                {'fifo0':fifo0,'fifo1':fifo1} ),

        dsx.Task('world', 'world',
                {'fifo1':fifo1,'fifo2':fifo2} ),

        dsx.Task('poly', 'poly',
                {'fifo0':fifo0,'fifo2':fifo2} )
        )

# Partie 2 : génération du code exécutable sur station de travail POSIX

tcg.generate( dsx.Posix() )
```

On fait l'exécution avec ‘’ ***./exe.muteks_hard*** ‘’

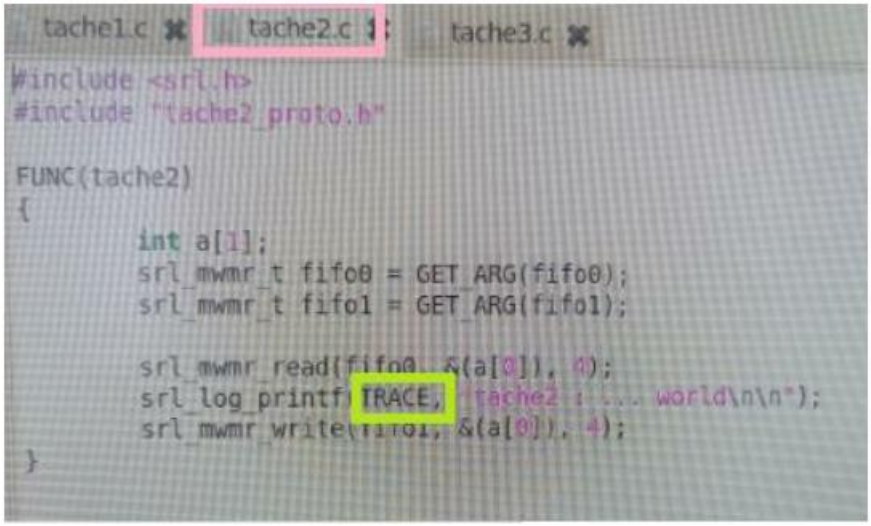

tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech
tache 1: hello
tache 2 : world
tache3 : polytech^C

TP4 : Multiprocesseur

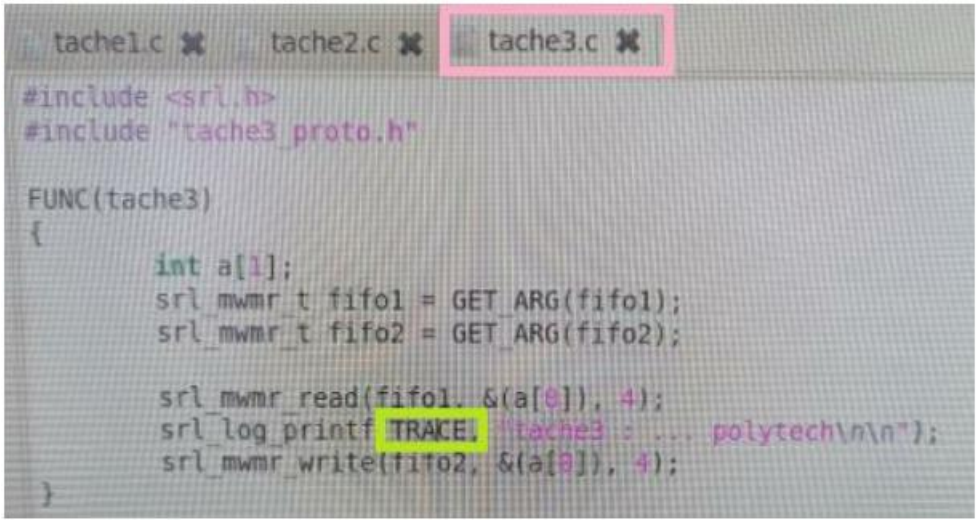
Code **« Tache1.c »:**



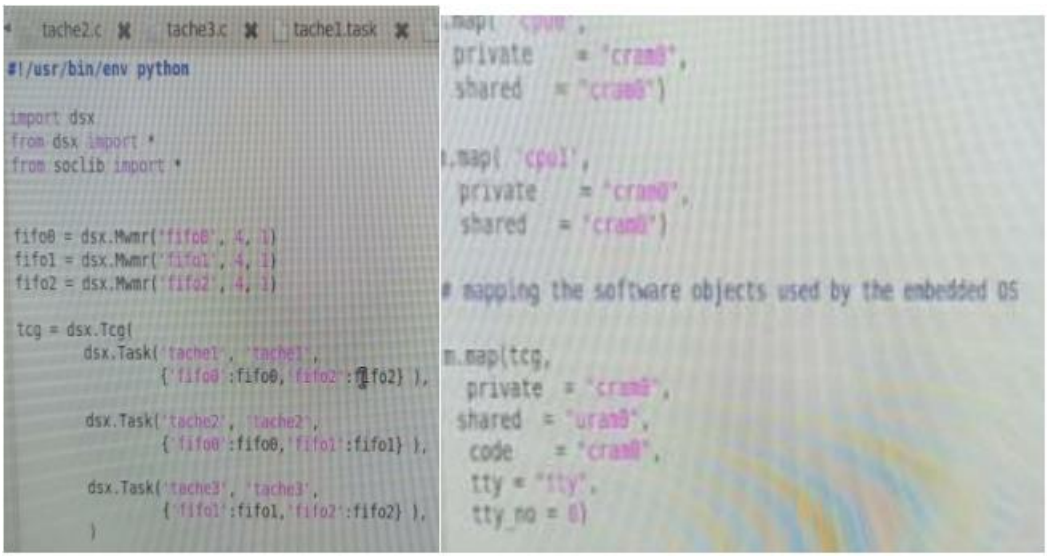
Code « Tache2.c » **:**



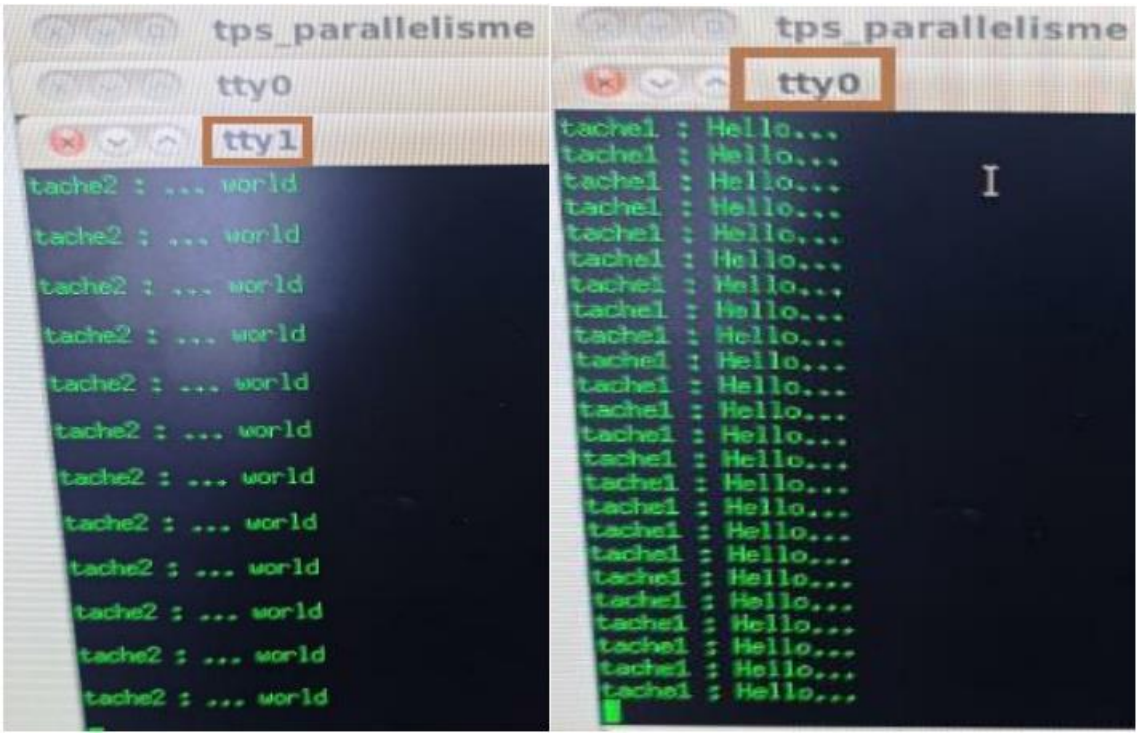
Code du ‘’Tache3.c’’ :



Code « Taches.py » :

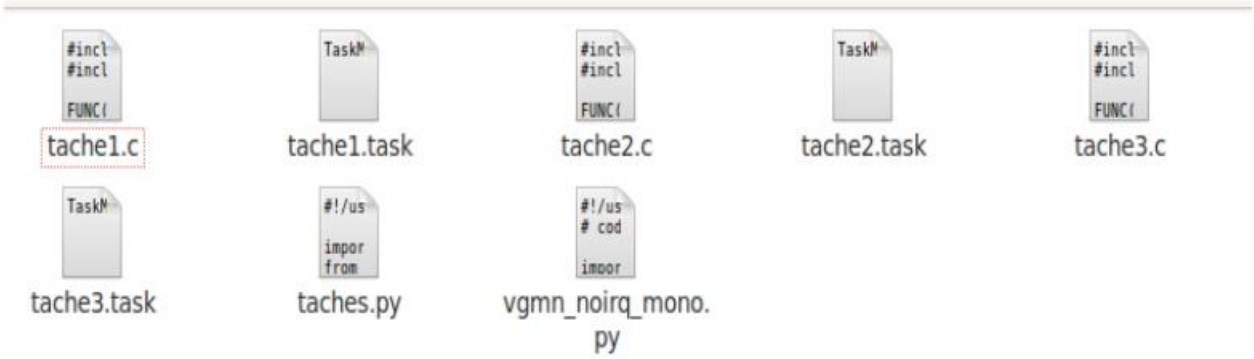


On fait l’exécution avec ‘’ ./exe.muteks_hard ‘’



Calcul de somme avec microprocessur

Notre TP contient les fichiers suivants :



Code **« tache1.c »**

```
FUNC(tache1)
{
        int a[1];
        char T1[10]={1, 2, 3, 4, 50, 6, 7, 8, 9, 10};
int i=0, s1=0, s2=0, s3=0, s=0;
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);
        srl_mwmr_t fifo3 = GET_ARG(fifo3);

        for(i=0; i<10; i++){
                s1+=T1[i];
        }

        srl_log_printf(TRACE, "tache1 : J'ai termine S1: %d\n", s1);
        srl_mwmr_write(fifo0, &(a[0]), 4);

        srl_mwmr_read(fifo3, &s2, 4);
        srl_mwmr_read(fifo2, &s3, 4);

        s=s1+s2+s3;
        srl_log_printf(TRACE, "tache1 : s=%d\n", s);
}
```

Code du **« tache2.c » :**

```
#include <srl.h>
#include "tache2_proto.h"

FUNC(tache2)
{
        int a[1];
        char T2[10]={1, 2, 3, 4, 5, 6, 7, 80, 9, 10};
        int i=0, s2=0;
        srl_mwmr_t fifo0 = GET_ARG(fifo0);
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo3 = GET_ARG(fifo3);

        for(i=0; i<10; i++){
                s2+=T2[i];
        }

        srl_mwmr_read(fifo0, &a[0], 4);
        srl_log_printf(TRACE, "tache2 : J'ai termine S2: %d\n", s2);

        srl_mwmr_write(fifo1, &(a[0]), 4);
        srl_mwmr_write(fifo3, &s2, 4);
```

Code **« tache3.c » :**

```c
#include <srl.h>
#include "tache3_proto.h"

FUNC(tache3)
{
        int a[1];
        char T3[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int i=0, s3=0;
        srl_mwmr_t fifo1 = GET_ARG(fifo1);
        srl_mwmr_t fifo2 = GET_ARG(fifo2);

        for(i=0; i<10; i++){
                s3+=T3[i];
        }

        srl_mwmr_read(fifo1, &a[0], 4);
        srl_log_printf(TRACE, "tache3 : J'ai termine S3: %d\n", s3);

        srl_mwmr_write(fifo2, &s3, 4);
}
```

Code **« taches.python » :**

```
fifo0 = dsx.Mwmr('fifo0', 4,1)
fifo1 = dsx.Mwmr('fifo1', 4, 1)
fifo2 = dsx.Mwmr('fifo2', 4, 1)
fifo3 = dsx.Mwmr('fifo3', 4, 1)

tcg = dsx.Tcg(
        dsx.Task('tache1', 'tache1',
                 {'fifo0':fifo0,'fifo2':fifo2,'fifo3':fifo3} ),

        dsx.Task('tache2', 'tache2',
                 {'fifo0':fifo0,'fifo1':fifo1,'fifo3':fifo3} ),

        dsx.Task('tache3', 'tache3',
                 {'fifo1':fifo1,'fifo2':fifo2} ),
        )
```

Code **« vgmn_noirq_mono.py» :**

```
archi = VgmnNoirqMono(ntty = 3)

#########################################################
# Section C : Mapping
#
#########################################################

m = Mapper(archi, tcg)

# mapping the MWMR channel

m.map( "fifo0",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
m.map( "fifo1",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
m.map( "fifo2",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
m.map( "fifo3",  buffer  = "cram0",  status  = "cram0",  desc    = "cram0")
```

```
# mapping the software objects associated to a processor

m.map( 'cpu0',
  private    = "cram0",
  shared   = "cram0")

m.map( 'cpu1',
  private    = "cram0",
  shared   = "cram0")

m.map( 'cpu2',
  private    = "cram0",
  shared   = "cram0")

# mapping the software objects used by the embedded OS

m.map(tcg,
  private  = "cram0",
 shared  = "uram0",
  code    = "cram0",
  tty = "tty",
  tty_no = 0)
```

```
#########################################################
# Section D : Code generation
#########################################################

m.generate( dsx.MutekS() )
tcg.generate( dsx.Posix() )
```

```
def VgmnNoirqMono(ntty = 1):
        pf = soclib.Architecture(cell_size = 4,
                                                                plen_size = 8,
                                                                addr_size = 32,
                                                                rerror_size = 1,
                                                                clen_size = 1,
                                                                rflag_size = 1,
                                                                srcid_size = 8,
                                                                pktid_size = 1,
                                                                trdid_size = 1,
                                                                wrplen_size = 1
                                                                )
        pf.create('common:mapping_table',
                                'mapping_table',
                                addr_bits = [8],
                                srcid_bits = [8],
                                cacheability_mask = 0xc00000)
        pf.create('common:loader', 'loader')

        vgmn = pf.create('caba:vci_vgmn', 'vgmn0',
                                                min_latency=10,
```

```
        for i in range (3):###nous avions besoin de 2 processeur donc 2 sinon si n
processeurs, on mettra n
            cpu = pf.create('caba:vci_xcache_wrapper', 'cpu%d' % i,
                                iss_t = "common:mips32el",
                                ident = i,
                                icache_ways = 2,
                                                icache_sets = 128,
                                                icache_words = 32,
                                                dcache_ways = 2,
                                                dcache_sets = 128,
                                                dcache_words = 32)

            vgmn.to_initiator.new() // cpu.vci



        for i in range(1):
                ram = pf.create('caba:vci_ram', 'ram%d'%i)
                base = 0x10000000*i
                ram.addSegment('cram%d'%i, base, 0x200000, True)
                ram.addSegment('uram%d'%i, base + 0x400000, 0x200000, False)
                ram.vci // vgmn.to_target.new()
```

```
                ram.addSegment('cram%d'%i, base, 0x200000, True)
                ram.addSegment('uram%d'%i, base + 0x400000, 0x200000, False)
                ram.vci // vgmn.to_target.new()
        ram.addSegment('boot', 0xbfc00000, 0x1000, True) # Mips boot address, 0x1000 octets,
cacheable
        ram.addSegment('excep', 0x80000000, 0x1000, True) # Mips exception address, 0x1000
octets, cacheable



        tty = pf.create('caba:vci_multi_tty', 'tty', names = map(lambda x:'tty%d'%x, range
(ntty)))
        tty.addSegment('tty', 0x95400000, 0x20*ntty, False)
        tty.vci // vgmn.to_target.new()

        return pf

# This is a python quirk to generate the platform
# if this file is directly called, but only export
# methods if imported from somewhere else

if __name__ == '__main__':
        VgmnNoirqMono().generate(soclib.PfDriver())
```

On fait l'exécution avec ‘’ ***./exe.muteks_hard*** ‘’

Le résultat nous donne la somme du trois taches :