



**Compte Rendu : Ingénierie des systèmes**

***multicore et multiprocesseur***

**Réaliser Par : ROUIS Houssem**

**Année Universitaire : 2020-2021**

## **Introduction :**

La notion de parallélisme, qui consiste à utiliser plusieurs processeurs ou opérateurs matériels pour exécuter un ou plusieurs programmes, est ancienne.

Les multiprocesseurs datent des années 1960. De cette période jusqu'à la fin des années 1990, des architectures parallèles ont été utilisées pour les applications nécessitant des besoins de calcul que les monoprocesseurs étaient incapables de fournir. Étaient concernés les mainframes et serveurs d'une part, et les machines vectorielles puis parallèles utilisées pour le calcul scientifique hautes performances d'autre part. Les années 1980 ont vu l'apparition de différentes sociétés proposant des machines parallèles, sociétés qui ont assez rapidement disparu. La raison essentielle est liée aux progressions exponentielles des performances des microprocesseurs, utilisés dans les PC et les serveurs multiprocesseurs. L'utilisation massive du parallélisme se limitait aux très grandes applications de simulation numérique avec les architectures massivement parallèles. Le début des années 2000, avec les limitations des monoprocesseurs et le « mur de la chaleur », a complètement changé la situation.

Les processeurs multicœurs sont présents en 2016 dans les architectures matérielles pour tous les types de composants: appareils mobiles (smartphones, tablettes), systèmes embarqués, télévisions, PC portables et PC de bureau, et jusqu'aux machines parallèles et superordinateurs pour la très haute performance.

# TP1 : MULTI-TACHES

Tache 1: Affichage “hello”

Write FIFO 0

Read(Fifo2,2)

Tache 2: Read (Fifo 0,2)

Affichage “world”

Write (Fifo 1,3)

Tache3 : Read (Fifo 1,3)

Affichage “Poly”

Write Fifo 2

Les relations entre les different taches:

T1 => Fifo 0 => T2

T1 => Fifo 2 => T3

T2 => Fifo 1 => T3

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**" :

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

FUNC(world)
{
    int a[1];

    srl_mwmr_t fifo0 = GET_ARG(fifo0);
    srl_mwmr_t fifol = GET_ARG(fifol);

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

```
#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.Mwmm('fifo0', 4, 1)
fifo1 = dsx.Mwmm('fifo1', 4, 1)
fifo2 = dsx.Mwmm('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 exécutable 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 :

```
tâche1 : Hello...
tâche2 : ... world
tâche3 : ... polytech

tâche1 : Hello...
tâche2 : ... world
tâche3 : ... polytech

tâche1 : Hello...
tâche2 : ... world
tâche3 : ... polytech

tâche1 : Hello...
tâche2 : ... world
tâche3 : ... polytech^C
```

## TP2: Calcul de Somme

### Tache 1 :

Initialisation du T1

Calculer S1

Affichage « j'ai terminé s1 »

Write Fifo 0

Lire (Fifo 3,s2)

### Tache 2 :

Initialisation du T2

Calculer S2

Lire Fifo 0

Affichage « j'ai terminé s2 »

Write Fifo1

Write (Fifo 3,s2)

### Tache 3 :

Initialisation du T3

Calculer S3

Affichage « j'ai terminé s3 »

Write (Fifo 2,s3)

T	Taches	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[10];
    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_mwmmr_t fifo0 = GET_ARG(fifo0);
    srl_mwmmr_t fifo2 = GET_ARG(fifo2);
    srl_mwmmr_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_mwmmr_read(fifo0, &(s1[0]), 4);
    srl_log_printf(NONE, "s2 : j'ai termine s2...\n");
    srl_mwmmr_write(fifo1, &(a[0]), 4);
    srl_mwmmr_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_mwmmr_t fifo1 = GET_ARG(fifo1);
    srl_mwmmr_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_mwmmr_read(fifo1, &(s2[0]), 4);
    srl_log_printf(NONE, "s3 : j'ai termine s3...\n");
    srl_mwmmr_write(fifo3, &(s3[0]), 4);
}
```

On fait l'exécution avec "**./exe.posix**"

## TP3 : Partage de données entre tâches

### Tache 1 :

Déclaration de la chaine1

Write(FiFo 0, chaine2)

Read (FiFo 0, chaine 3)

Affichage chaine3

Write(FiFo1,chaine1)

### Tache 2 :

Déclaration de la chaine2

Read(FiFo1,chaine1)

Affichage chaine1

Write(FiFo3,chaine2)

### Tache 3 :

Déclaration de la chaine3

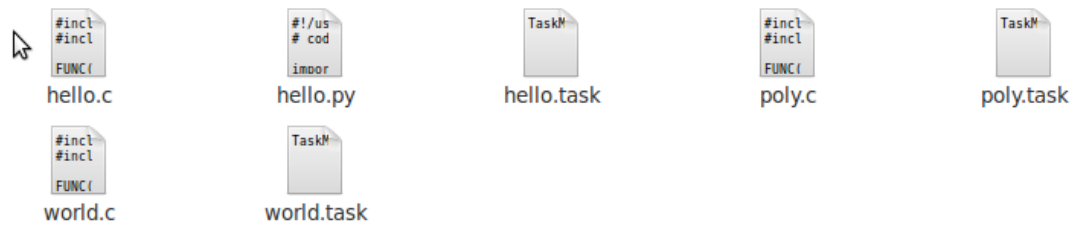
Write(FiFo 0, chaine3)

Read(FiFo2,chaine2)

Affichage chaine2

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	bloquée

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** » :

```
#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** » :

```

#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** » :

```

# 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() )

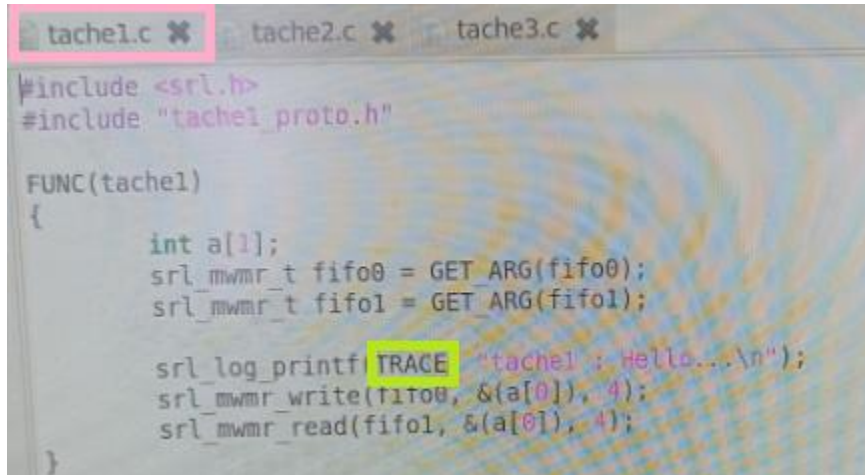
```

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 » :

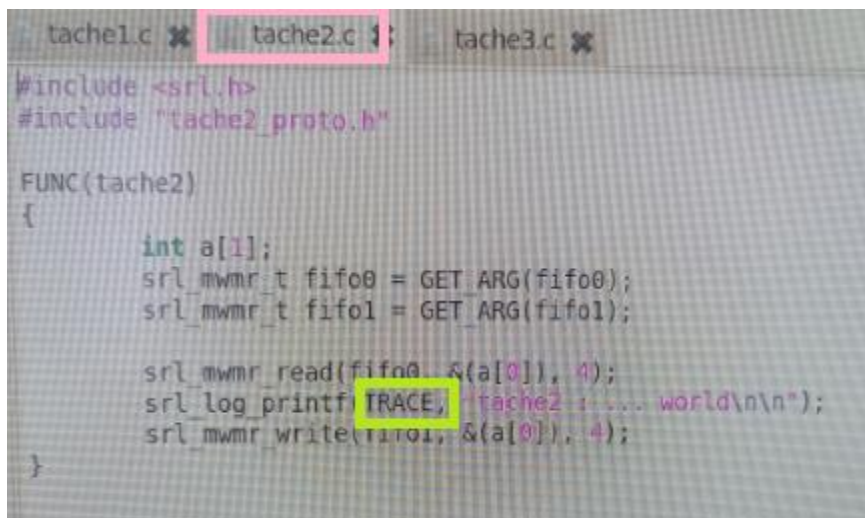


```
tache1.c ✕ tache2.c ✕ tache3.c ✕
#include <srl.h>
#include "tache1_proto.h"

FUNC(tache1)
{
    int a[1];
    srl_mwmr_t fifo0 = GET_ARG(fifo0);
    srl_mwmr_t fifo1 = GET_ARG(fifo1);

    srl_log_printf(TRACE, "tache1 : Hello...\n");
    srl_mwmr_write(fifo0, &a[0], 4);
    srl_mwmr_read(fifo1, &a[0], 4);
}
```

Code « Tache2.c » :



```
tache1.c ✕ tache2.c ✕ tache3.c ✕
#include <srl.h>
#include "tache2_proto.h"

FUNC(tache2)
{
    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(TRACE, "tache2 : ... world\n\n");
    srl_mwmr_write(fifo1, &a[0], 4);
}
```

Code du "Tache3.c" :

```
tache1.c x tache2.c x tache3.c x
#include <srl.h>
#include "tache3_proto.h"

FUNC(tache3)
{
    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 TRACE "tache3 : ... polytech\n\n";
    srl_mwmr_write(fifo2, &a[0], 4);
}
```

Code « Taches.py » :

```
tache2.c x tache3.c x tache1.task x
#!/usr/bin/env python

import dsx
from dsx import *
from soclib import *

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

tcg = dsx.Tcg(
    dsx.Task('tache1', 'tache1',
             {'fifo0':fifo0, 'fifo2':fifo2}),
    dsx.Task('tache2', 'tache2',
             {'fifo0':fifo0, 'fifo1':fifo1}),
    dsx.Task('tache3', 'tache3',
             {'fifo1':fifo1, 'fifo2':fifo2}),
)

m.map('cpu0',
      private = 'crand',
      shared = 'crand')

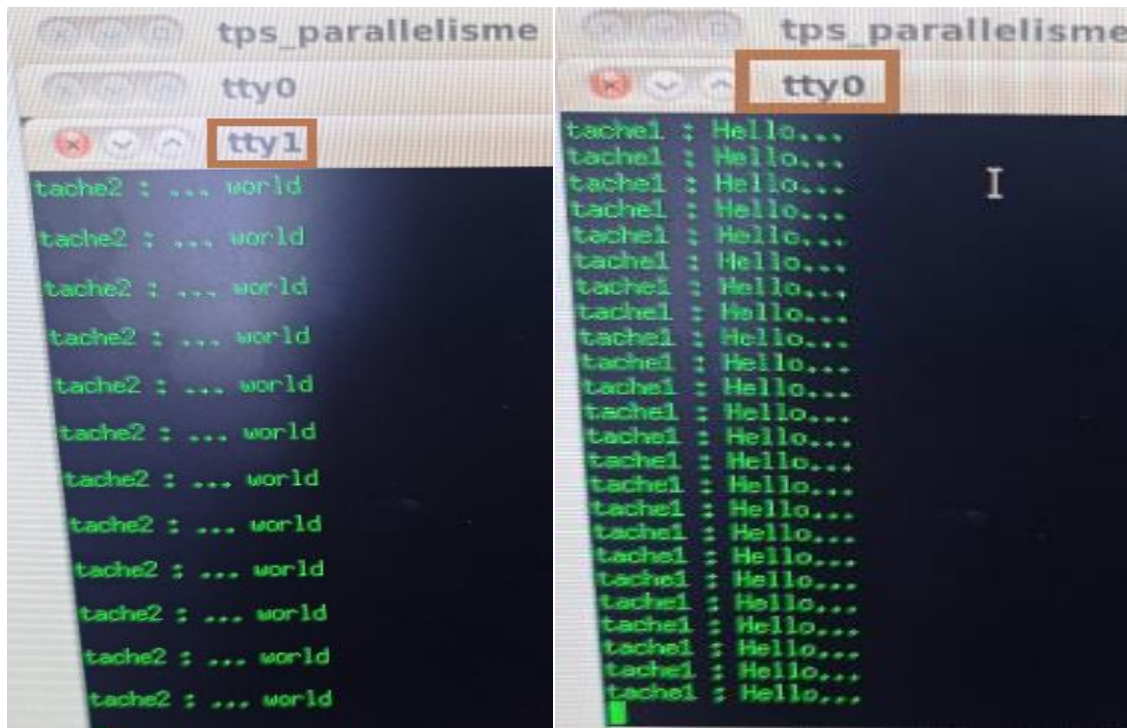
m.map('cpu1',
      private = 'crand',
      shared = 'crand')

# mapping the software objects used by the embedded OS

m.map(tcg,
      private = 'crand',
      shared = 'urand',
      code = 'crand',
      tty = 'tty',
      tty_no = 0)
```

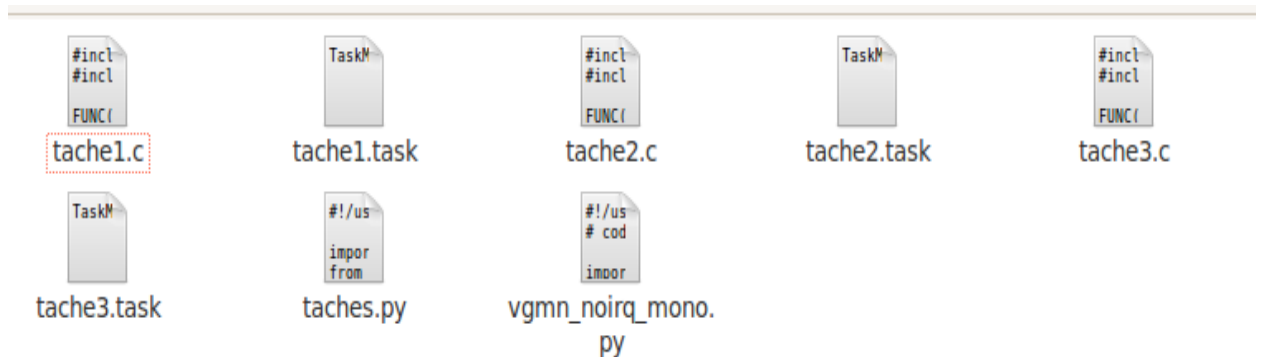
On fait l'exécution avec " ***./exe.muteks\_hard*** "





### Calcul de somme avec microprocesseur

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** » :

```
#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 « **vgmnoirq\_mono.py** » :

```

archi = VgmnNoirqMono(ntty = 3)

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

m = Mapper(archi, tcg)

# mapping the MMR 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')

    vgm = pf.create('caba:vci_vgm', 'vgm0',
                    min_latency=10,

```

```

    for i in range(3):###nous avons 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)

        vgm.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 // vgm.to_target.new()

```



**Conclusion :**

Le parallélisme est dorénavant utilisé dans la majorité des architectures, des systèmes embarqués aux superordinateurs.

Les monoprocesseurs sont remplacés par des processeurs multicœurs. Cet article décrit la notion de parallélisme et ses différents types.

Il présente les grandes classes d'architectures parallèles avec leurs ressources et organisations mémoire, en distinguant les architectures homogènes et hétérogènes.

Les principes des techniques de programmation sont introduits avec les extensions parallèles des langages de programmation couramment utilisés et les modèles de programmation visant à rapprocher la programmation parallèle de la programmation séquentielle, en incluant les spécificités des architectures.

Enfin, les modèles et métriques d'évaluation des performances sont examinés.