Programmation Système

Alexis Wilhelm

alexis.wilhelm@gmail.com

hiver 2015–2016

version du
7 janvier 2016

Première partie

Processus

- Manipulation en bash
- Manipulation en C

Manipulation en bash

- bash
- proc
- ps
- nice
- Bilan

Manipulation simple : jobs, bg, fg, kill

```
~$ albert # bloquant
^Z # Ctrl Z
[1]+ Stopped albert
~$ jobs
[1]+ Stopped albert
~$ bg
[1]+ albert &
~$ jobs
[1]+ Running albert &
~$ fg # bloquant
albert
^Z # Ctrl Z
[1]+ Stopped albert
~$ rené &
[2] 12558
~$ marcel &
[3] 12559
~$ jobs
[1]+ Stopped albert
[2] Running rené &
[3]- Running marcel &
~$ bg
[1]+ albert &
~$ jobs
[1] Running albert &
[2]- Running rené &
[3]+ Running marcel &
~$ kill %1
[1] Terminated albert
~$ jobs
[2]- Running rené &
[3]+ Running marcel &
~$ kill %+
[3]+ Terminated marcel
~$ jobs
[2]+ Running rené &
```

☞ man sh

Observation : ls /proc

acpi/	iomem	pagetypeinfo	vmstat	20/
asound/	ioports	partitions	zoneinfo	21/
buddyinfo	irq/	sched_debug	1/	22/
bus/	kallsyms	self@	2/	23/
cgroups	kcore	slabinfo	3/	24/
cmdline	keys	softirqs	5/	26/
consoles	key−users	stat	7/	27/
cpuinfo	kmsg	swaps	8/	28/
crypto	kpagecount	sys/	9/	29/
devices	kpageflags	sysrq−trigger	10/	31/
diskstats	loadavg	sysvipc/	11/	32/
dma	locks	thread−self@	12/	38/
driver/	meminfo	timer_list	13/	39/
execdomains	misc	timer_stats	15/	41/
fb	modules	tty/	16/	77/
filesystems	mounts@	uptime	17/	78/
fs/	mtrr	version	18/	79/
interrupts	net@	vmallocinfo	19/	80/

Observation : ls /proc/1

attr/	environ	mem	personality	statm
autogroup	exe@	mountinfo	projid_map	status
auxv	fd/	mounts	root@	syscall
cgroup	fdinfo/	mountstats	sched	task/
clear_refs	gid_map	net/	schedstat	timers
cmdline	io	ns/	sessionid	uid_map
comm	limits	oom_adj	setgroups	wchan
coredump_filter	loginuid	oom_score	smaps	
cpuset	map_files/	oom_score_adj	stack	
cwd@	maps	pagemap	stat	

☞ man 5 proc

Observation : ps -ef

```
UID          PID  PPID  C STIME TTY          TIME CMD
alexis     11020 11019  0 13:33 ?        00:00:00 /bin/sh ./ps.sh
alexis     11022 11020  0 13:33 ?        00:00:00 ps -ef
alexis     11023 11020  0 13:33 ?        00:00:00 grep \b11020\b\|\bPID\b
```

☞ man 1 ps

Priorité : nice, renice

~$ nice ./gros-calcul &
[1] 15267

~$ renice -n 50 15267
15267 (process ID) old priority 10, new priority 19

~$ renice -n -50 15267
renice: failed to set priority for 15267 (process ID): Permission denied

~$ sudo !!
sudo renice -n -50 15267
15267 (process ID) old priority 19, new priority -20

☞ man 1 nice
☞ man 1 renice

Définition d'un processus

Système

- Identifiant numérique
- Parent
- Propriétaire
- Permissions

Mémoire

- Code
- Tas
- Pile
- Registres

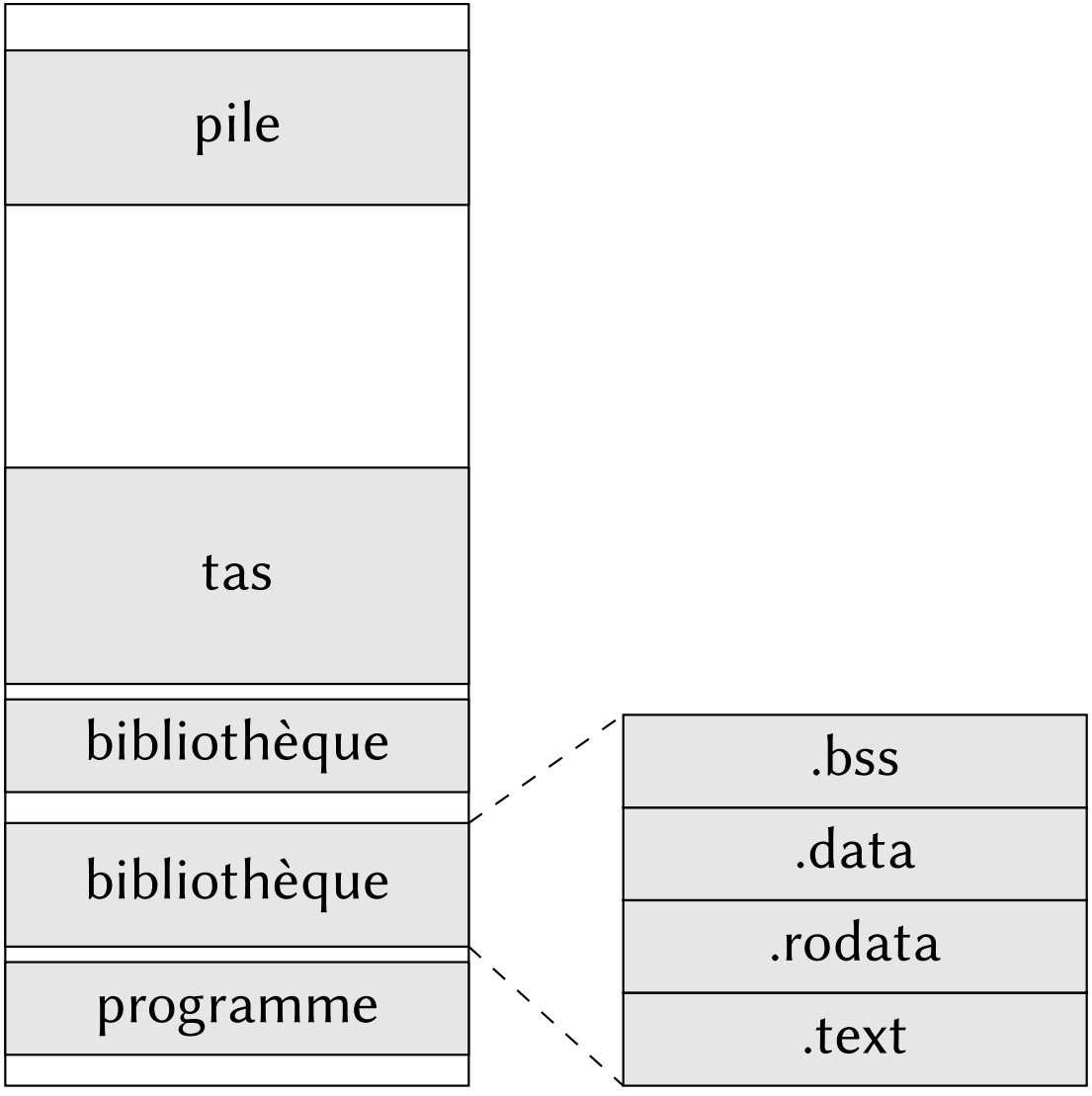
Ressources

- Répertoire
- Variables d'environnement
- Descripteurs de fichiers

Ordonnancement

- État (en cours, prêt, arrêté...)
- Temps CPU

Organisation de la mémoire d'un processus



Manipulation en C

- fork
- wait
- exec
- environ
- Documentation

Duplication : fork()

```c
#include <assert.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int child = fork(); // duplication
    assert(child != -1);
    if(child) printf("%d: Je suis le père de %d.\n", getpid(), child);
    else printf("%d: Je suis le fils de %d.\n", getpid(), getppid());
    sleep(1);
}
```

```
29696: Je suis le père de 29697.
29697: Je suis le fils de 29696.
```

Duplication : fork()

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork(); // duplication
    fork(); // duplication
    printf("Je suis %d.\n", getpid()); // qui est là ?
}
```

```
Je suis 20129.
Je suis 20131.
Je suis 20130.
Je suis 20132.
```

Attente : waitpid()

```
int main() {
    int child = fork();
    if(child) {
        printf("%d: Je suis le père de %d.\n", getpid(), child);
        int status;
        child = waitpid(child, &status, 0); // lecture
        printf("%d: Mon fils %d a fini avec un code %d.\n", getpid(), child,
            WEXITSTATUS(status));
    } else {
        printf("%d: Je suis le fils de %d.\n", getpid(), getppid());
        return 5; // écriture
    }
}
```

```
29680: Je suis le fils de 29679.
29679: Je suis le père de 29680.
29679: Mon fils 29680 a fini avec un code 5.
```

Attente : wait

```
for i in 1 2 3
do sleep $i && echo $i &
done
echo début
wait
echo fin
```

```
début
1
2
3
fin
```

Recouvrement : exec()

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    execlp("date", "", "+%A %-d %B", NULL); // changement de programme
    puts("Cette ligne ne sera pas exécutée.");
}
```

mercredi 14 octobre

Recouvrement : exec()

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    char *args[] = {"", "-r", "execve.c", 0}; // nouvelle ligne de commande
    char *env[] = {"LC_ALL=C", 0}; // nouvelles variables d'environnement
    execve("/bin/date", args, env); // nouveau programme
    puts("Cette ligne ne sera pas exécutée.");
}
```

```
Mon Oct 12 16:57:20 CEST 2015
```

Environnement : getenv(), setenv(), unsetenv()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    (void) argc;
    char const *var = getenv("var"); // lecture
    printf("var=%s\n", var);
    if(!var)
    {
        setenv("var", "OK", 1); // écriture
        fflush(0);
        execv(argv[0], argv);
    }
}
```

```
var=(null)
var=OK
```

Environnement : export

```
./setenv.exe

var=1 ./setenv.exe # local à cette commande

export var=2 # pour toutes les prochaines commandes
./setenv.exe
```

```
var=(null)
var=OK
var=1
var=2
```

Documentation

POSIX

- ☞ The Open Group Base Specifications Issue 7
  http://pubs.opengroup.org/onlinepubs/9699919799/

C

- ☞ C reference
  http://en.cppreference.com/w/c
- ☞ C gibberish ↔ English
  http://www.cdecl.org/

Deuxième partie

Threads

- Introduction
- Manipulation avec pthread
- Autres API

Introduction

- Processus léger

Processus léger

Système

- Identifiant numérique
- Parent
- Propriétaire
- Permissions

Mémoire

- Code
- Tas
- Pile
- Registres

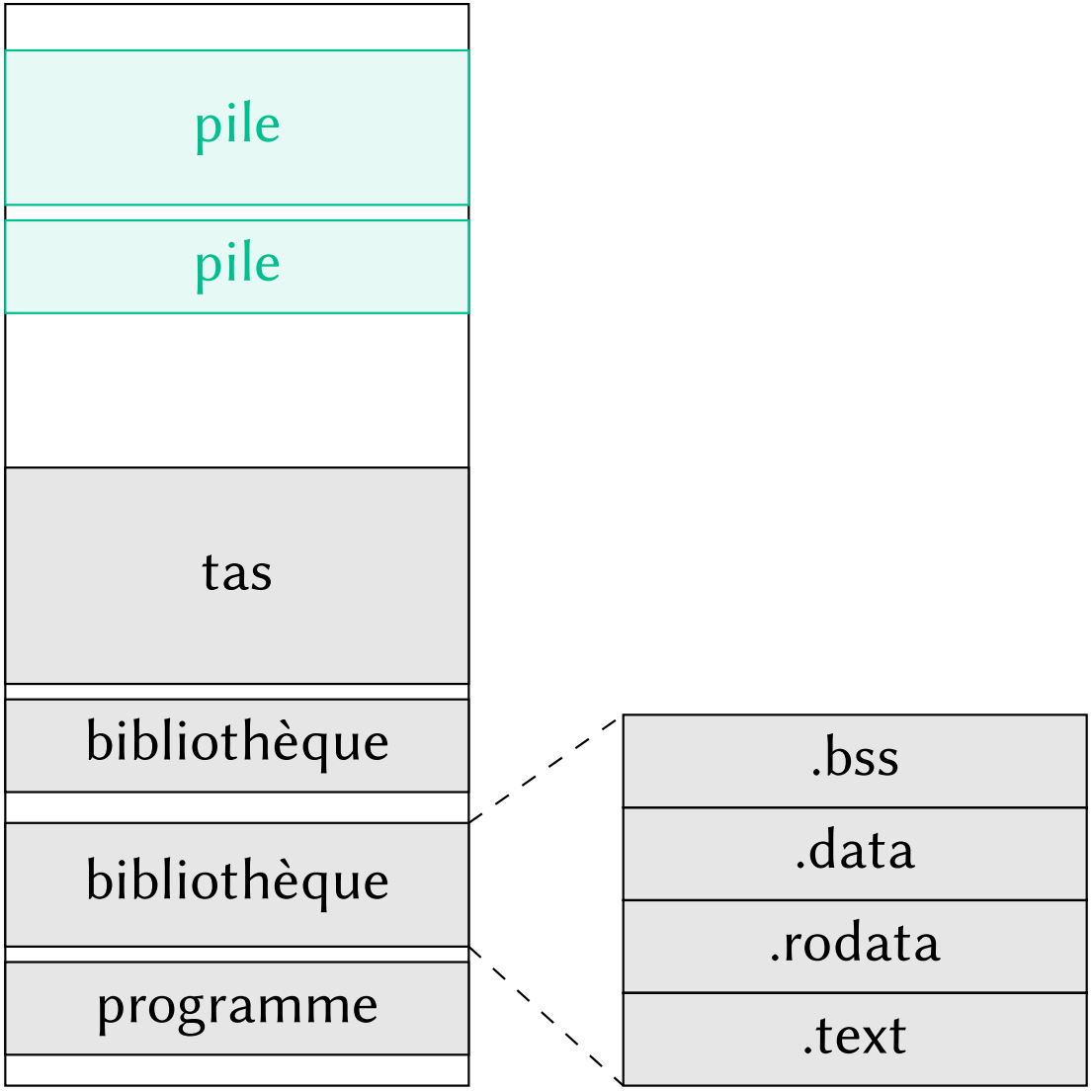
Ressources

- Répertoire
- Variables d'environnement
- Descripteurs de fichiers

Ordonnancement

- État (en cours, prêt, arrêté...)
- Temps CPU

Processus léger



Processus léger

```
[Thread debugging using libthread_db enabled]
Using host libthread_db library
      "/lib/i386-linux-gnu/i686/cmov/libthread_db.so.1".
[New Thread 0xb7ddfb40 (LWP 19372)]
[Thread 0xb7ddfb40 (LWP 19372) exited]
résultat : 5
[Inferior 1 (process 19368) exited normally]
```

Observation : ps -L

```
  PID   LWP TTY          TIME CMD
28751 28751 tty7     00:03:05 iceweasel
28751 28794 tty7     00:00:00 Gecko_IOThread
28751 28795 tty7     00:00:00 Link Monitor
28751 28796 tty7     00:00:06 Socket Thread
28751 28797 tty7     00:00:00 iceweasel
28751 28798 tty7     00:00:01 JS Helper
28751 28799 tty7     00:00:01 JS Helper
28751 28800 tty7     00:00:00 JS Helper
28751 28801 tty7     00:00:01 JS Helper
28751 28802 tty7     00:00:01 JS Helper
28751 28803 tty7     00:00:01 JS Helper
28751 28804 tty7     00:00:00 JS Watchdog
28751 28805 tty7     00:00:00 Hang Monitor
28751 28806 tty7     00:00:00 BgHangManager
```

Manipulation avec pthread

- Démarrer une tâche
- Attendre qu'une tâche soit terminée
- Communiquer avec une autre tâche
- Attendre un événement
- Annuler une tâche

Démarrage : pthread_create()

```c
void *function(void *args)
{
    printf("numéro %d\n", *(int *)args);
    return 0;
}

int main()
{
    int numbers[] = {1, 2, 3, 4};
    for(int i = 0; i < 4; ++i)
    {
        pthread_t thread;
        pthread_create(&thread, 0, function, numbers + i); // nouvelle tâche
    }
    sleep(1); // attente
}
```

```
numéro 1
numéro 4
numéro 2
numéro 3
```

Attente : pthread_join()

```c
void *function(void *args)
{
    (void)args;
    int *result = malloc(sizeof(int));
    *result = 5;
    return result;
}

int main()
{
    pthread_t thread;
    pthread_create(&thread, 0, function, 0); // nouvelle tâche
    void *result = 0;
    pthread_join(thread, &result); // attente
    printf("résultat : %d\n", *(int *)result);
    free(result);
}
```

résultat : 5

Attente : pthread_join()

```c
void *function(void *result)
{
    *(int *)result = 5;
    return 0;
}

int main()
{
    int result = 0;
    pthread_t thread;
    pthread_create(&thread, 0, function, &result); // nouvelle tâche
    pthread_join(thread, 0); // attente
    printf("résultat : %d\n", result);
}
```

résultat : 5

Variable globale

```c
int result = 0;

void *function(void *args)
{
    (void)args;
    result = 5;
    return 0;
}

int main()
{
    pthread_t thread;
    pthread_create(&thread, 0, function, 0); // nouvelle tâche
    sleep(1); // attente
    printf("résultat : %d\n", result);
}
```

résultat : 5

Variable globale et accès concurrents

```
int global = 0;

void *function(void *args)
{
    (void)args;
    sleep(1);
    printf("numéro %d\n", global);
    ++global;
    return 0;
}
```

```
numéro 0
numéro 0
numéro 0
numéro 3
numéro 3
```

Variable globale et pthread_mutex_t

```
int global = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *function(void *args)
{
    (void)args;
    sleep(1);
    pthread_mutex_lock(&mutex); // verrouillage
    printf("numéro %d\n", global);
    ++global;
    pthread_mutex_unlock(&mutex); // déverrouillage
    return 0;
}
```

```
numéro 0
numéro 1
numéro 2
numéro 3
numéro 4
```

Variable globale et pthread_mutex_t

```
int global = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *function(void *args)
{
    (void)args;
    sleep(1);
    pthread_mutex_lock(&mutex); // verrouillage
    pthread_mutex_lock(&mutex); // verrouillage
    printf("numéro %d\n", global);
    ++global;
    pthread_mutex_unlock(&mutex); // déverrouillage
    pthread_mutex_unlock(&mutex); // déverrouillage
    return 0;
}
```

Types de verrous

pthread_mutex_t

- pthread_mutex_lock()
- pthread_mutex_trylock()
- pthread_mutex_timedlock()
- pthread_mutex_unlock()

pthread_mutexattr_settype() permet de définir le comportement du mutex quand un même thread le verrouille plusieurs fois :

PTHREAD_MUTEX_NORMAL dead lock.

PTHREAD_MUTEX_ERRORCHECK code d'erreur EDEADLK.

PTHREAD_MUTEX_RECURSIVE verrouille une deuxième fois.

Types de verrous

pthread_rwlock_t

- pthread_rwlock_rdlock()
- pthread_rwlock_wrlock()
- pthread_rwlock_tryrdlock()
- pthread_rwlock_trywrlock()
- pthread_rwlock_timedrdlock()
- pthread_rwlock_timedwrlock()
- pthread_rwlock_unlock()

Types de verrous

pthread_spinlock_t

- pthread_spin_lock()
- pthread_spin_trylock()
- pthread_spin_unlock()

Attente : pthread_cond_t


```
int result = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition = PTHREAD_COND_INITIALIZER;
void *function(void *);

int main()
{
    pthread_t thread;
    pthread_create(&thread, 0, function, 0); // nouvelle tâche
    pthread_mutex_lock(&mutex); // verrouillage
    while(!result) pthread_cond_wait(&condition, &mutex); // attente
    printf("résultat : %d\n", result);
    pthread_mutex_unlock(&mutex); // déverrouillage
}
```

Attente : pthread_cond_t


```
void *function(void *args)
{
    (void)args;
    pthread_mutex_lock(&mutex); // verrouillage
    result = 5;
    pthread_cond_signal(&condition); // travail terminé !
    pthread_mutex_unlock(&mutex); // déverrouillage
    return 0;
}
```

résultat : 5

Attente : pthread_cond_t

- pthread_cond_signal()
- pthread_cond_broadcast()

- pthread_cond_wait()
- pthread_cond_timedwait()

Attente : pthread_barrier_t


```
pthread_barrier_t barrier;
void *function(void *);

int main()
{
    pthread_barrier_init(&barrier, 0, 3); // initialisation
    pthread_t threads[3];

    for(int i = 0; i < 3; ++i)
        pthread_create(&threads[i], 0, function, 0); // nouvelle tâche
    for(int i = 0; i < 3; ++i)
        pthread_join(threads[i], 0); // attente
}
```

Attente : pthread_barrier_t


```c
void *function(void *args)
{
    (void)args;
    puts("avant");
    pthread_barrier_wait(&barrier); // synchronisation
    puts("après");
    return 0;
}
```

```
avant
avant
avant
après
après
après
```

Annulation : pthread_cancel()


```c
void cleanup(void *args)
{
    (void)args;
    puts("Arrêté.");
}

void *function(void *args)
{
    pthread_cleanup_push(cleanup, args);
    for(;;) pthread_testcancel(); // peut être interrompu ici
    puts("Cette ligne ne sera pas exécutée.");
    pthread_cleanup_pop(1); // exécute le destructeur
    return 0;
}
```

Annulation : pthread_cancel()


```
int main()
{
    pthread_t thread;
    pthread_create(&thread, 0, function, 0); // nouvelle tâche
    sleep(1); // attente
    pthread_cancel(thread); // arrêt
    pthread_join(thread, 0); // attente
}
```

Arrêté.

Annulation : drapeau

```c
int cancel = 0;

void *function(void *args)
{
    (void)args;
    while(!cancel); // peut être interrompu ici
    puts("Arrêté.");
    return 0;
}

int main()
{
    pthread_t thread;
    pthread_create(&thread, 0, function, 0); // nouvelle tâche
    sleep(1); // attente
    cancel = 1; // arrêt
    pthread_join(thread, 0); // attente
}
```

Arrêté.

Autres API

- C++
- Java

C++ : std::thread

```cpp
#include <iostream>
#include <thread>

void function(int a, int b, int *c)
{
    *c = a + b;
}

int main()
{
    int result = 0;
    std::thread thread {function, 1, 2, &result}; // nouvelle tâche
    thread.join(); // attente
    std::cout << "résultat : " << result << std::endl;
}
```

résultat : 3

C++ : std::unique_lock et std::condition_variable

```cpp
int result = 0;
std::mutex mutex;
std::condition_variable condition;

void function()
{
    std::unique_lock<std::mutex> lock {mutex}; // verrouillage
    result = 5;
    condition.notify_one(); // travail terminé !
}

int main()
{
    std::thread thread {function}; // nouvelle tâche
    std::unique_lock<std::mutex> lock {mutex}; // verrouillage
    while(!result) condition.wait(lock); // attente
    std::cout << "résultat : " << result << std::endl;
    thread.join(); // attente
}
```

résultat : 5

C++ : std::unique_lock et std::condition_variable

std::unique_lock

- lock()
- try_lock()
- try_lock_for()
- try_lock_until()
- unlock()

std::condition_variable

- notify_one()
- notify_all()

- wait()
- wait_for()
- wait_until()

Documentation

☞ C++ reference : Thread support library
http://en.cppreference.com/w/cpp/thread

```
public class Thread1
{
    public static void main(String[] args) throws InterruptedException
    {
        MyFunction function = new MyFunction(1, 2);
        Thread thread = new Thread(function); // nouvelle tâche
        thread.start(); // démarrage
        thread.join(); // attente
        System.out.format("résultat : %d%n", function.result);
    }
}
```

```
class MyFunction implements Runnable
{
    int a, b, result;

    MyFunction(int a, int b)
    {
        this.a = a;
        this.b = b;
    }

    public void run()
    {
        result = a + b;
    }
}
```

résultat : 3

Java : synchronized

```
public class Thread2
{
    public static void main(String[] args)
    {
        Object mutex = new Object();

        synchronized(mutex)
        {
            synchronized(mutex)
            {
                System.out.println("OK");
            }
        }
    }
}
```

OK

Java : wait() et notify()


```
public class Thread3
{
    public static void main(String[] args) throws InterruptedException
    {
        MyFunction function = new MyFunction();
        Thread thread = new Thread(function); // nouvelle tâche
        thread.start(); // démarrage
        synchronized(function) // verrouillage
        {
            while(function.result == 0) function.wait(); // attente
        }
        System.out.format("résultat : %d%n", function.result);
    }
}
```

Java : wait() et notify()


```
class MyFunction implements Runnable
{
    int result = 0;

    public void run()
    {
        synchronized(this) // verrouillage
        {
            result = 5;
            notifyAll(); // travail terminé !
        }
    }
}
```

résultat : 5

Java : interrupt()


```
public class Thread4
{
    public static void main(String[] args) throws InterruptedException
    {
        Thread thread = new Thread(new MyFunction()); // nouvelle tâche
        thread.start(); // démarrage
        Thread.sleep(100); // attente
        thread.interrupt(); // arrêt
        thread.join(); // attente
    }
}
```

Java : interrupt()


```
class MyFunction implements Runnable
{
    public void run()
    {
        try
        {
            Thread.sleep(10); // peut être interrompu ici
            while(!Thread.interrupted()); // peut être interrompu ici
        }
        catch(InterruptedException e) {}
        System.out.println("Arrêté.");
    }
}
```

Arrêté.

Documentation

- ☞ Lesson : Concurrency
  http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html