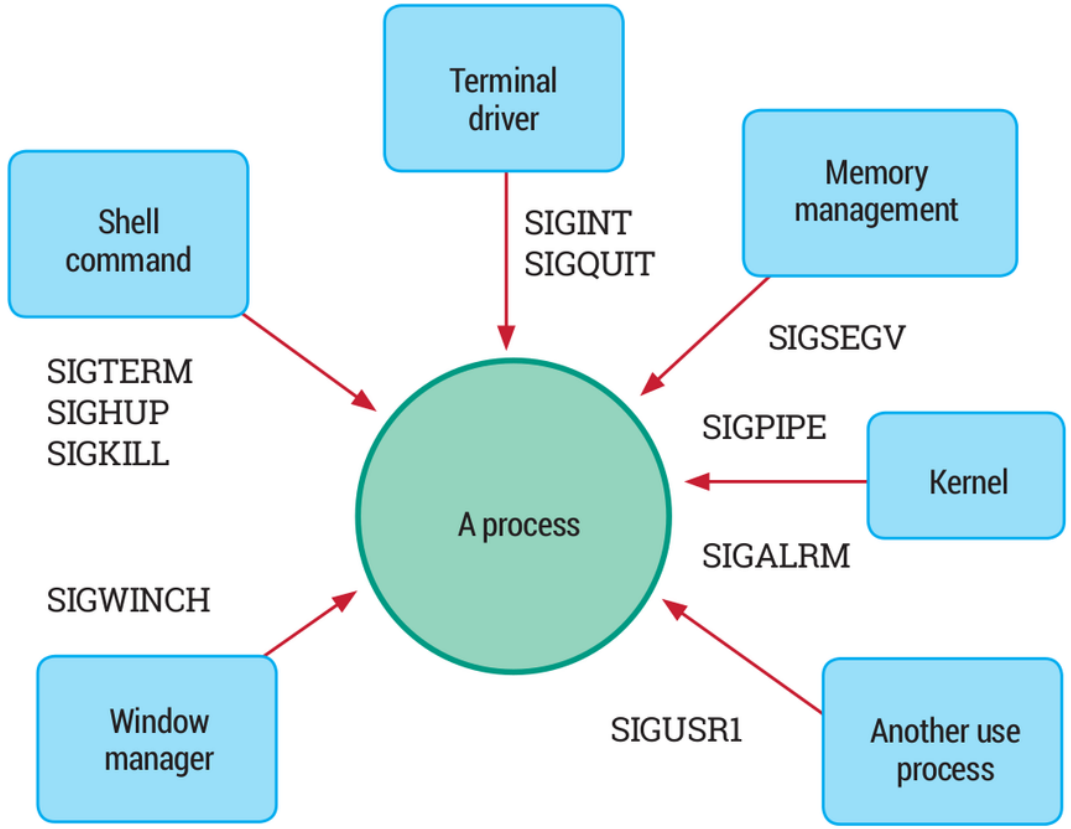
LINUX – LE NOYAU
LES SIGNAUX

Les signaux

- Un **signal** est un **message** qui peut être envoyé à un **processus en cours d'exécution**

- Les signaux peuvent être initiés par :
  - Un programme
  - Un utilisateur
  - Un administrateur



Signaux de base

SIGHUP	1	Hangup (^?)
SIGINT	2	Interrupt (^c)
SIGQUIT	3	Quit (^\)
SIGILL	4	Illegal instruction
SIGTRAP	5	Trace trap
SIGIOT	6	IOT instruction
SIGEMT	7	EMT instruction
SIGFPE	8	Floating point exception
SIGKILL	9	Kill (cannot be caught or ignored)
SIGBUS	10	Bus error
SIGSEGV	11	Segmentation violation
SIGSYS	12	Bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	Alarm clock
SIGTERM	15	Software termination signal
SIGURG	16	Urgent condition present on socket
SIGSTOP	17	Stop (cannot be caught or ignored)
SIGTSTP	18	Stop signal generated from keyboard (^z)
SIGCONT	19	Continue after stop
SIGCHLD	20	Child status has changed
SIGTTIN	21	Background read attempted from ctrl term
SIGTTOU	22	Background write attempted to control term.
SIGIO	23	I/O is possible on a descriptor (see fcntl(2))
SIGXCPU	24	Cpu time limit exceeded (see setrlimit(2))
SIGXFSZ	25	File size limit exceeded (see setrlimit(2))
SIGVTALRM	26	Virtual time alarm (see setitimer(2))
SIGPROF	27	Profiling timer alarm (see setitimer(2))
SIGWINCH	28	Window size change
SIGLOST	29	lock not reclaimed after server recovery
SIGUSR1	30	User defined signal
SIGUSR2	31	User defined signal
SIGCLD		System V name for SIGCHLD
SIGABRT		

Envoi d'un signal à un processus : fonction kill()

```
#include <sys/types.h>
#include <signal.h>

kill(pid, sig)
      pid_t pid;
      int sig;
```

La fonction **kill()** envoie l'événement **sig** au processus désigné par pid

```
# kill –sig pid
```

```
# kill –9 5432
```

Un processus peut envoyer un événement à lui-même

Exemple avec la fonction kill()

```
#include <sys/types.h>
#include <signal.h>

pere( pid_t pid ) {
        system("ps");
        kill( pid, SIGKILL );
        sleep(5);
        system("ps");
    }

fils( ) {
        while(1) { };
}
```

```
main( ) {
        int pid;
        pid = fork();
        if (pid == 0)
          fils();
        else
          pere( pid );
}
```

Exemple avec la fonction kill()

Interprétation d'un signal : la fonction signal()

```
#include <signal.h>

void ( *signal(sig, func) )
            void (*func)();
```

signal() modifie le comportement d'un processus devant une interruption

- Le premier argument est le numéro d'une interruption

- Le second argument est soit l'adresse d'une fonction, soit un code demandant d'ignorer ou de rétablir le traitement par défaut de l'interruption (SIG_IGN ou SIG_DFL)

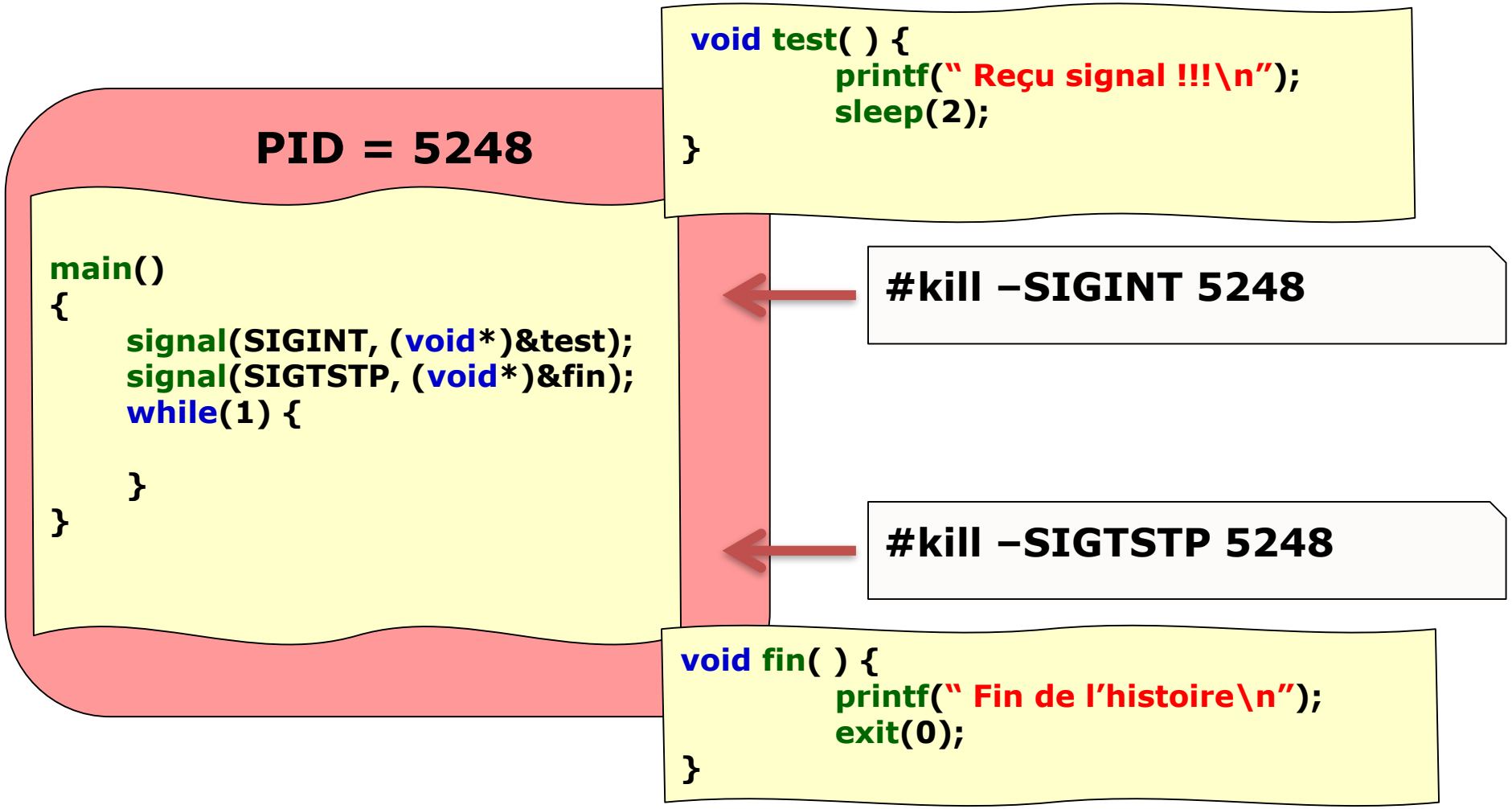
Exemple avec la fonction signal()

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

// Traitement pour SIGINT (CTRL+C)
void test( ) {
        printf(" Reçu signal !!!\n");
        sleep(2);
}

// Traitement pour SIGTSTP (CTRL+Z)
void fin( ) {
        printf(" Fin de l'histoire\n");
        exit(0);
}
```

```
main()
{
    signal(SIGINT, (void*)&test);
    signal(SIGTSTP, (void*)&fin);
    while(1) {

    }
}
```

Exemple avec la fonction signal()

Exemple avec la fonction signal()



Les fonctions alarm() et pause()

```
#include <unistd.h>

unsigned alarm(seconds)
    unsigned seconds;
```

alarm() envoie l'événement **SIGALRM** au processus appelant, après que le nombre de secondes précisé en argument soit écoulé

Cet événement provoque la fin du processus à moins qu'il ne soit masqué ou intercepté

```
#include <unistd.h>

int pause(void)
```

pause() est utilisée pour arrêter l'exécution et pour attendre un événement en provenance des fonctions **kill()** ou **alarm()**

Exemple
avec les fonctions alarm() et pause()

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

void reveil( )
{
     printf("Driiiiing \n");
}
```

```
main()
{
  signal(SIGALRM, &reveil);
  printf("Start alarm\n");
  alarm(5);
  pause();
  printf("Au boulot !\n");
}
```

Exemple
avec les fonctions alarm() et pause()

Attendre (passivement) un signal

```
#include <signal.h>

int sigsuspend (const sigset_t *mask)
```

sigsuspend() remplace le masque de signaux du processus appelant avec le masque fourni dans **mask** et suspend le processus jusqu'à la livraison d'un signal

Pendant que le processus est suspendu son état est "bloqué" : il n'utilise pas de ressources processeurs

Attendre (passivement) un signal