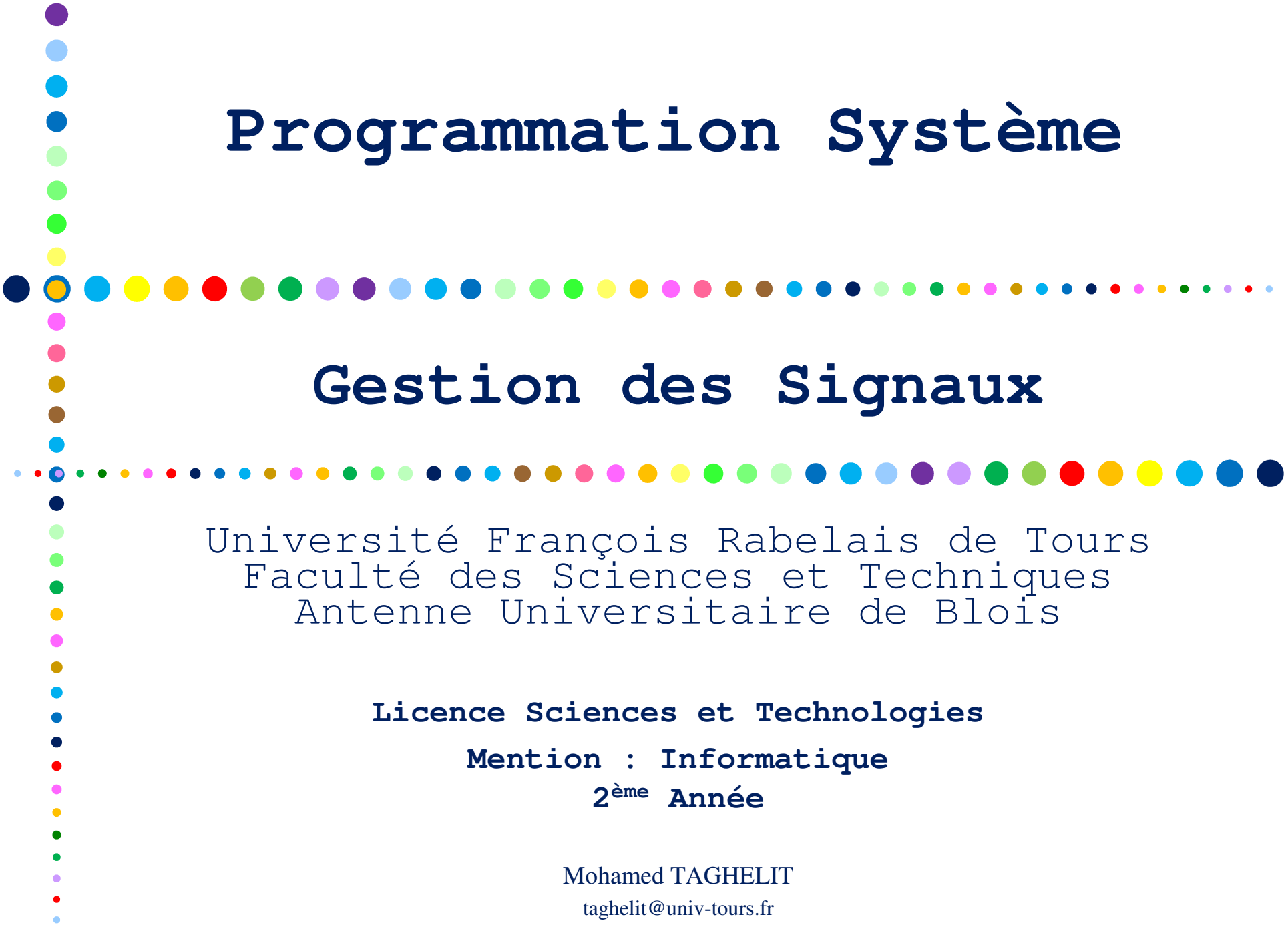
# Programmation Système



# Gestion des Signaux

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

**Licence Sciences et Technologies**

**Mention : Informatique**

**2ème Année**

Mohamed TAGHELIT
taghelit@univ-tours.fr

# La Gestion des Signaux

- Identification des signaux
- États et prise en compte d'un signal
- Émission des signaux
- Installation d'un gestionnaire de signaux
- Fonctions POSIX de manipulation de signaux
- Opérations sur les ensembles de signaux

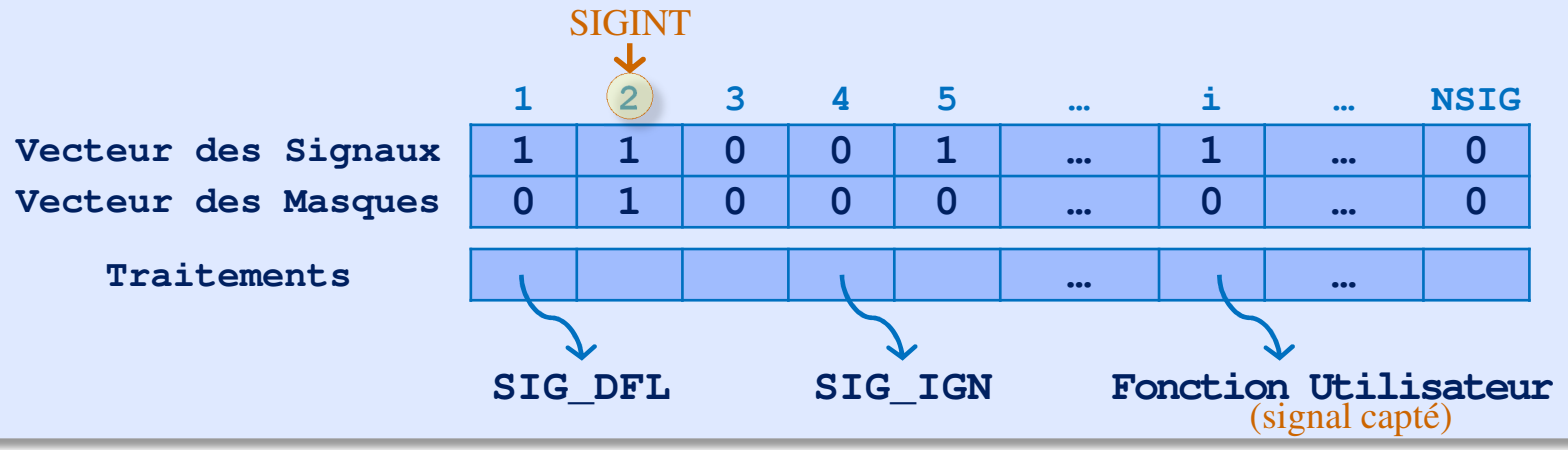
# Identification des Signaux

- Les signaux constituent un mécanisme de communication asynchrone inter-processus :
  - notions donc d'émission/réception de signal
  - la réception correspond au positionnement d'un drapeau dans le bloc de contrôle
- **`NSIG`** types de signaux différents (**`<bits/signum.h>`**) identifiés par des entiers (**`[1..NSIG]`**).
- Chaque signal possède un nom symbolique de préfixe **`SIG`**

  **`SIGHUP, SIGINT, SIGKILL, …`**
- Un événement particulier est associé à tout signal.
- Le type est la seule information véhiculée par un signal (l'événement associé peut ou pas s'être produit).
- L'événement associé à un signal peut être soit :
  - un événement externe au processus (frappe de caractère sur un terminal, terminaison d'un autre processus, … )
  - un événement interne au processus (erreur arithmétique, violation mémoire, … )

# États et Prise en Compte d'un Signal

- **Un signal peut être dans l'un des états suivants :**
  - **`pendant`** : signal émis mais pas encore pris en compte.
    Un seul exemplaire d'un même type de signal peut être pendant.
  - **`traité`** : signal pris en compte par le processus au cours de son exécution.
  - **`bloqué`** ou masqué : signal dont la délivrance est volontairement différée (BSD et POSIX).
- **Prise en compte d'un signal**
  La délivrance d'un signal entraîne l'exécution d'une fonction (**`handler`**) particulière :
  - fonction **`SIG_DFL`**, comportement par défaut,
  - fonction **`SIG_IGN`**, ignorer le signal,
  - fonction utilisateur (le signal est capté ).
- **Lors de l'exécution du gestionnaire d'un signal, ce dernier est bloqué**



# Liste des Signaux de Base

| Nom du signal | Événement associé | Traitement par défaut |
|---|---|---|
| **SIGHUP** | Terminaison du processus leader de session | (1) |
| **SIGINT** | Frappe du caractère **intr** sur le clavier du terminal de contrôle | (1) |
| **SIGQUIT** | Frappe du caractère **quit** sur le clavier du terminal de contrôle | (2) |
| **SIGILL** | Détection d'une instruction illégale | (2) |
| **SIGABRT** | Terminaison anormale provoquée par l'exécution de la fonction **abort** | (1) |
| **SIGFPE** | Erreur arithmétique (division par zéro, … ) | (1) |
| **SIGKILL** | Signal de terminaison | (1)* |
| **SIGSEGV** | Violation mémoire | (2) |
| **SIGPIPE** | Écriture dans un tube sans lecteur | (1) |
| **SIGALRM** | Fin de temporisation (fonction **alarm**) | (1) |
| **SIGTERM** | Signal de terminaison | (1) |
| **SIGUSR1** | Signal émis par un processus utilisateur | (1) |
| **SIGUSR2** | Signal émis par un processus utilisateur | (1) |
| **SIGCHLD** | Terminaison d'un fils | (3) |
| **SIGSTOP** | Signal de suspension | (4)* |
| **SIGTSTP** | Frappe du caractère **susp** sur le clavier du terminal de contrôle | (4) |
| **SIGCONT** | Signal de continuation d'un processus stoppé | (5)* |
| **SIGTTIN** | Lecture par un processus en arrière-plan | (4) |
| **SIGTTOU** | Écriture par un processus en arrière-plan | (4) |
| **SIGIO** | Avis d'arrivée de caractères à lire | (1) |
| **SIGURG** | Avis d'arrivée de caractères urgents | (1) |

(1) terminaison du processus, (2) terminaison du processus avec image mémoire (fichier **core**),
(3) signal ignoré (sans effet), (4) suspension du processus,
(5) continuation : reprise d'un processus stoppé et ignoré sinon.
* : signal ne pouvant être ni bloqué ni ignoré

# Les Primitives d'Émission d'un Signal

- Un processus peut émettre un signal vers un autre (groupe de) processus

```
#include <sys/types.h>
#include <signal.h>
        int kill(pid_t pid, int sig);
        int raise(int sig);
```

ID du processus cible

numéro du signal à émettre

| pid | Cible(s) du signal |
|---|---|
| > 0 | processus d'identifiant **pid** |
| = 0 | tous les processus du même groupe que l'appelant |
| = -1 | tous les processus accessibles par le processus appelant sauf **init** |
| < -1 | tous les processus du groupe d'identifiant **-pid** |

| sig | Signal concerné |
|---|---|
| < 0 **OU** > NSIG | valeur incorrecte |
| = 0 | pas d'émission (test d'existence du destinataire) |
| ∈ [1..NSIG] | signal de numéro **sig** |

- Valeur de retour :
  - 0 en cas de succès (au moins un signal a été envoyé)
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

# Installation d'un Gestionnaire de Signaux

- Un processus peut modifier le traitement associé à la réception de certains signaux

```
#include <signal.h>
      sighandler_t signal(int signum, sighandler_t handler);
```

Numéro du signal

Gestionnaire du signal

- Installe le gestionnaire spécifié par **handler** pour le signal **signum**.
- Valeur de retour :
  - en cas de succès, un pointeur sur la valeur antérieure du **handler**,
  - **SIG_ERR** en cas d'erreur.

| handler | Événement lors de l'arrivée du signal |
|---|---|
| **SIG_IGN** | le signal est ignoré |
| **SIG_DFL** | exécution de l'action par défaut associée au signal |
| fonction utilisateur | appel de la fonction utilisateur avec l'argument **signum** |

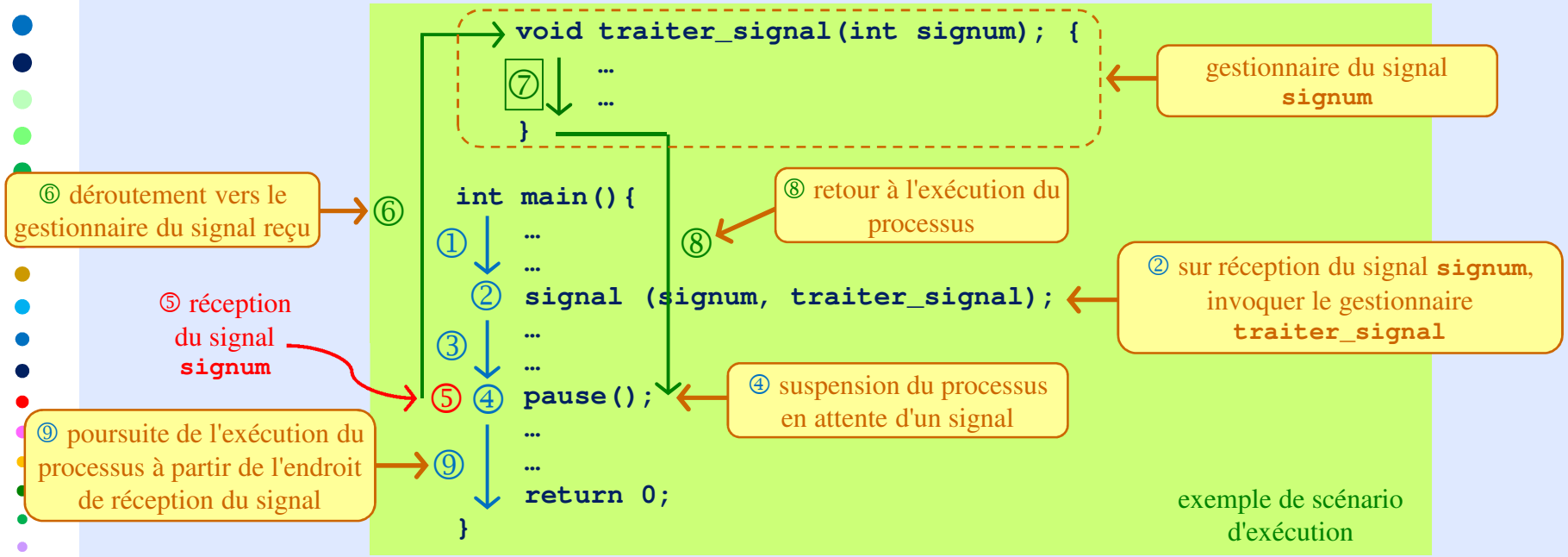
- Par défaut, durant l'exécution d'un gestionnaire, le signal qui l'a provoquée est bloqué. Le signal est débloqué au retour du gestionnaire.

# Attente d'un Signal

- Un processus peut à tout moment suspendre son exécution en attente de la réception d'un quelconque signal.

```
#include <unistd.h>
        int pause(void);
```

- Valeur de retour :
  - -1 et **errno=EINTR** (après réception du signal et retour de la fonction de gestion de ce même signal).



# Traitements associés à un Signal

## Programme `signal_dfl.c` (associer le traitement par défaut)

```
int main(){
    printf("Avant le \"pause\" !\n");
    pause();
    printf("Après le \"pause\" !\n");
    return 0;
}
```

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

## Exécution de `signal_dfl.c`

```
[student]$ ./signal_dfl
Avant le "pause" !
^C
[student]$
```

frappe au clavier de **Ctrl+c** (correspondant à l'émission du signal **SIGINT** au processus)

la réception du signal **SIGINT** interrompe le processus

## Programme `signal_ign.c` (ignorer un signal)

```
int main(){
    signal(SIGINT, SIG_IGN);
    printf("Avant le \"pause\" !\n");
    pause();
    printf("Après le \"pause\" !\n");
    return 0;
}
```

spécifie que le signal **SIGINT** doit être ignoré

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

## Exécution de `signal_ign.c`

```
[student]$ ./signal_ign
Avant le "pause" !
^C^C^C^C^C^C^C^C^C^C^C^C^\Quitter
[student]$
```

frappes successives au clavier de **Ctrl+c**

frappe au clavier de **Ctrl+** (correspondant à l'émission du signal **SIGQUIT** au processus)

la réception du signal **SIGINT** n'interrompe pas le processus, il est ignoré

# Exemple d'un Signal Capté

## Programme `signal_spec.c` (associer un traitement spécifique)

```
void traiter_signal(int signum){
printf("Signal numéro %d reçu !\n", signum);
}

int main(int argc, char **argv){
    signal(SIGINT, traiter_signal);
    while(1){
        printf("Avant le \"pause\" !\n");
        pause();
        printf("Après le \"pause\" !\n");
    }
    return 0;
}
```

fonction utilisateur (gestionnaire)

spécifie que sur réception du signal **SIGINT**, le gestionnaire **traiter_signal** doit être invoqué

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

## Exécution de `signal_spec.c`

```
[student]$ ./signal_spec
Avant le "pause" !
^CSignal numéro 2 reçu !
Après le "pause" !
Avant le "pause" !
^CSignal numéro 2 reçu !
Après le "pause" !
Avant le "pause" !
^\Quitter
[student]$
```

affiché par la fonction **traiter_signal**

la frappe au clavier de **Ctrl+c** entraine l'exécution du gestionnaire **traiter_signal** puis la reprise de l'exécution du processus à l'instruction qui suit celle interrompue par le signal

la frappe au clavier de **Ctrl+** entraine la terminaison du processus (signal ni ignoré, ni capté)

# Caractéristiques de la Fonction signal

- Réinitialisation du comportement par défaut après toute occurrence du signal (systèmes Unix d'origine), sinon

```
void traiter_signal(int signum){
   signal(SIGINT, traiter_signal);
    printf("Signal numéro %d reçu !\n", signum);
}
```

Réinitialiser le traitement voulu pour la future occurrence du signal

- Possibilités d'ignorer ou de capter uniquement un signal (pas de blocage).
- Impossibilité de connaître le traitement courant associé à un signal sans le modifier.
- Interruption des appels système.

Préférer l'utilisation de **`sigaction()`**

# Fonctions POSIX de Manipulation de Signaux

- Installation d'un gestionnaire de signal.

```
#include <signal.h>
      int sigaction(int signum, const struct sigaction *act,
                    struct sigaction *oldact);
```

numéro du signal concerné

décrit le gestionnaire à installer

sauvegarde le précédent gestionnaire

- Permet de déterminer ou de modifier l'action associée à un signal particulier.
  - Si **act** ≠ **NULL**, il s'agit d'une modification de l'action associée au signal **signum**.
  - Si **oldact** ≠ **NULL**, l'ancienne action pour le signal **signum** est sauvegardée dans **oldact**.
  - Si **act** = **NULL** et **oldact** = **NULL**, test de la validité du signal

```
struct sigaction {
    void (*sa_handler)(int);   /* adresse du handler, ou SIG_IGN, ou SIG_DFL */
    sigset_t sa_mask;          /* signaux additionnels à bloquer */
    int sa_flags;              /* options :
    …                          SA_RESETHAND : restaurer l'action par défaut après exécution du gestionnaire
}                              SA_NODEFER : ne pas empêcher la réception du signal concerné
                               SA_RESTART : Sémantique BSD, reprise appels systèmes lents interrompus
                               … */
```

- Valeur de retour :
  - 0 en cas de succès
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

# Exemple d'un Signal Capté

- Programme **sigaction_spec.c** (associer un traitement spécifique)

```
void traiter_signal(int signum){
    printf("\t\tDébut gestionnaire !\n");
    sleep(5);
    printf("\t\tFin gestionnaire !\n");
}
```

fonction utilisateur (gestionnaire)

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

```
int main(int argc, char **argv){
struct sigaction fonc;
char buf[128];
fonc.sa_handler = traiter_signal;
```

initialise la structure **sigaction** en spécifiant que le gestionnaire est la fonction **traiter_signal**

```
    // fonc.sa_flags = SA_RESETHAND;
    // fonc.sa_flags = SA_NODEFER;
    // fonc.sa_flags = SA_RESTART;
```

option 1 / option 2 / option 3

```
    if ( sigaction(SIGINT, &fonc, NULL) == -1 )
        perror("Erreur sigaction !\n");
    else
        printf("Installation du gestionnaire pour SIGINT\n");
```

installe le gestionnaire **traiter_signal** (spécifié dans **fonc**) pour le signal **SIGINT**

```
    while(1){
        printf("\tAvant le \"pause\" !\n");
        pause();
        printf("\tAprès le \"pause\" !\n");
    }
    return -1;
}
```

**code 1**

```
printf("\tAvant la lecture !\n");
read(STDIN_FILENO, buf, sizeof(buf));
printf("\tAprès la lecture !\n");
```

**code 2**

à utiliser avec l'option **3** en remplacement du code **1**

# Exemple d'un Signal Capté

## Exécution de **sigaction_spec.c** (**code 1** sans aucune option)

```
[student]$ ./sigaction_spec
Installation du gestionnaire pour SIGINT
    Avant le "pause" !
^C      Début gestionnaire !
^C^C
        Fin gestionnaire !
        Début gestionnaire !
        Fin gestionnaire !
    Après le "pause" !
    Avant le "pause" !
^\Quitter
[student]$
```

première exécution de la fonction **traiter_signal**

deuxième exécution de la fonction **traiter_signal**

le signal est bloqué lors de l'exécution de son gestionnaire et une seule occurrence du signal est sauvegardée

émission du signal **SIGQUIT** (non capté) ⇒ terminaison du processus

## Exécution de **sigaction_spec.c** (**code 1** avec l'option 1 **SA_RESETHAND**)

```
[student]$ ./sigaction_spec
Installation du gestionnaire pour SIGINT
    Avant le "pause" !
^C      Début gestionnaire !
        Fin gestionnaire !
    Après le "pause" !
    Avant le "pause" !
^C
[student]$
```

exécution de la fonction **traiter_signal**

le traitement par défaut est rétabli après la réception du premier signal **SIGINT**
la réception du second signal **SIGINT** termine le processus.

# Exemple d'un Signal Capté

## Exécution de `sigaction_spec.c` (`code 1` avec l'option 2 `SA_NODEFER`)

```
[student]$ ./sigaction_spec
Installation du gestionnaire pour SIGINT
    Avant le "pause" !
^C      Début gestionnaire !
^C      Début gestionnaire !
        Fin gestionnaire !
        Fin gestionnaire !
    Après le "pause" !
    Avant le "pause" !
^\Quitter
[student]$
```

première exécution de la fonction **`traiter_signal`**

deuxième exécution de la fonction **`traiter_signal`**

le signal n'est pas bloqué lors de l'exécution de son gestionnaire

## Exécution de `sigaction_spec.c` (`code 2` sans option)

```
[student]$ ./sigaction_spec
Installation du gestionnaire pour SIGINT
    Avant la lecture !
^C      Début gestionnaire !
        Fin gestionnaire !
    Après la lecture !
    Avant la lecture !
^\Quitter
[student]$
```

l'appel système **`read`** est interrompu

affichage de "**`Après la lecture !`**" suite à l'interruption de l'appel système **`read`**

## Exécution de `sigaction_spec.c` (`code 2` avec l'option 3 `SA_RESTART`)

```
[student]$ ./sigaction_spec
Installation du gestionnaire pour SIGINT
    Avant la lecture !
^C      Début gestionnaire !
        Fin gestionnaire !
^\Quitter
[student]$
```

l'appel système **`read`** n'est pas interrompu

pas d'affichage !

# Test de l'Existence d'un Signal

## Programme `sigaction_test.c`

teste si le 1er argument passé en ligne de commande est un signal valide

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

int main(int argc, char **argv){

    if ( sigaction( atoi(argv[1]), NULL, NULL ) == -1 )
        printf("le signal numéro ' %d ' n'existe pas !\n", atoi(argv[1]));
    else
        printf("le signal numéro ' %d ' existe !\n", atoi(argv[1]));

    return 0;
}
```

## Exécution de `sigaction_test.c`

```
[student]$ gcc sigaction_test.c -o sigaction_test
[student]$ ./sigaction_test -1
le signal numéro ' -1 ' n'existe pas !
[student]$ ./sigaction_test 0
le signal numéro ' 0 ' n'existe pas !
[student]$ ./sigaction_test 1
le signal numéro ' 1 ' existe !
[student]$ ./sigaction_test 13
le signal numéro ' 13 ' existe !
[student]$ ./sigaction_test 133
le signal numéro ' 133 ' n'existe pas !
[student]$
```

# Consultation/Modification du Masque des Signaux

- Un processus peut récupérer ou modifier son masque de signaux.
  - Un masque de signaux est l'ensemble des signaux dont la distribution est actuellement bloquée.

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

spécifie comment est modifié le masque courant

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

ensemble de signaux

sauvegarde le masque courant

  - Si **oldset** ≠ **NULL**, le masque courant du processus est sauvegardé dans **oldset**,
  - Si **set** ≠ **NULL**, alors **how** indique comment le masque courant du processus est modifié :

| how | nouveau masque de signaux |
|---|---|
| **SIG_BLOCK** | { masque courant } ∪ { **set** } |
| **SIG_UNBLOCK** | { masque courant } \ { **set** } |
| **SIG_SETMASK** | { **set** } |

- Impossible de bloquer **SIGKILL** et **SIGSTOP** (tentatives ignorées silencieusement)
- S'il existe de quelconques signaux pendants non-bloqués après l'appel à **sigprocmask**, alors au moins un de ces signaux est délivré au processus avant le retour de la primitive.
- Valeur de retour :
  - 0 en cas de succès, et -1 en cas d'erreur (et **errno** est modifiée en conséquence).

# Autres Primitives POSIX

- Un processus peut connaitre l'ensemble des signaux en attente (ou pendant)
  - Un signal en attente est un signal qui s'est déclenché (émis) en étant bloqué.

```
#include <signal.h>
        int sigpending(sigset_t *set);
```

stocke l'ensemble des signaux en attente

  - Valeur de retour, 0 en cas de succès, et -1 en cas d'erreur

- Un processus peut remplacer temporairement son masque de signaux

```
#include <signal.h>
        int sigsuspend(const sigset_t *mask);
```

masque des signaux de remplacement au masque courant

  - remplace temporairement le masque de signaux bloqués par celui fourni dans **mask** puis endort le processus jusqu'à l'arrivée d'un signal qui déclenche un gestionnaire de signal ou termine le processus (opération atomique).
  - Valeur de retour :
    -1 et **errno = EINTR** si signal capté,
    pas de retour sinon

```
…
sigprocmask(SIG_SETMASK, set, oldset);
…
    section de code critique
…
sigsuspend(oldset);
…
```

tout signal de **set** est bloqué durant l'exécution du code

# Opérations sur les Ensembles de Signaux

- Un ensemble de fonctions permettent la manipulation des ensembles de signaux **POSIX**

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

**`int sigemptyset(sigset_t *set);`**

- Initialise l'ensemble de signaux **`set`** à l'ensemble vide

**`int sigfillset(sigset_t *set);`**

- Initialise l'ensemble de signaux **`set`** à l'ensemble incluant tous les signaux

**`int sigaddset(sigset_t *set, int signum);`**

- Inclut le signal **`signum`** dans l'ensemble des signaux **`set`**

**`int sigdelset(sigset_t *set, int signum);`**

- Supprime le signal **`signum`** de l'ensemble des signaux **`set`**

**`int sigismember(const sigset_t *set, int signum);`**

- Teste si le signal **`signum`** appartient à l'ensemble des signaux **`set`**

- Valeur de retour
  - **`sigemptyset`**, **`sigfillset`**, **`sigaddset`** et **`sigdelset`** retournent 0 en cas de succès et -1 sinon
  - **`sigismember`** retourne 1 si **`signum`** est membre de **`set`**, 0 sinon et -1 en cas d'erreur

# Exemple de Blocage de Signaux

- Programme **block_sig.c**

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

void traiter_signal(int signum){
printf("\t\tSignal numéro %d reçu !\n", signum);
}

void affiche_signaux(sigset_t *set){
int i;
    for ( i = 1; i < NSIG; i++ )
        if ( sigismember(set, i) == 1 )
            printf("\t%d", i);
    printf("\n");
}

int main(int argc, char **argv){
struct sigaction fonc;
sigset_t set_bloques, set_pendants;

    fonc.sa_handler = traiter_signal;
    sigaction(SIGINT, &fonc, NULL);

    if ( sigemptyset(&set_bloques) != 0) perror("Erreur sigemptyset bloqués");
    if ( sigaddset(&set_bloques, SIGQUIT) != 0) perror("Erreur sigaddset QUIT bloqués");
    if ( sigaddset(&set_bloques, SIGKILL) != 0 ) perror("Erreur sigaddset KILL bloqués");

    if ( sigprocmask(SIG_SETMASK, &set_bloques, NULL) == -1 ){
        perror("Erreur sigprocmask !"); return -1;
    }
    else printf("SIGQUIT et SIGKILL bloqués !\n");

    printf("Signaux bloqués : "); fflush(stdout); affiche_signaux(&set_bloques);
    pause();
    sigemptyset(&set_pendants);
    sigpending(&set_pendants);

    printf("Signaux pendants : "); fflush(stdout); affiche_signaux(&set_pendants);
    exit(0);
}
```

fonction qui affiche le numéro de chacun des signaux présents dans **set**

installe le gestionnaire **traiter_signal** pour le signal **SIGINT**

bloque les signaux présents dans **set_bloques** (**SIGQUIT** et **SIGKILL**)

vide le buffer **stdout**

initialise à l'ensemble vide **set_pendants**

sauvegarde dans **set_pendants** les signaux pendants (reçus mais dans l'état bloqués)

# Exemple de Blocage de Signaux

## Exécution de `block_sig.c`

```
[student]$ gcc block_sig.c -o block_sig
[student]$ ./block_sig
SIGQUIT et SIGKILL bloqués !
Signaux bloqués :   3         9
^\
^C      Signal numéro 2 reçu !
Signaux pendants :  3
[student]$ ./block_sig
SIGQUIT et SIGKILL bloqués !
Signaux bloqués :   3         9
^Z
[2]+  Stopped                 ./block_sig
[student]$ ps
  PID TTY          TIME CMD
 1732 pts/0    00:00:00 bash
 7121 pts/0    00:00:00 block_sig
 7122 pts/0    00:00:00 ps
[student]$ kill -SIGKILL 7121
[student]$ ps
  PID TTY          TIME CMD
 1732 pts/0    00:00:00 bash
 7123 pts/0    00:00:00 ps
[2]+  Processus arrêté      ./block_sig
[student]$
```

← ???

← l'émission du signal **SIGQUIT** n'a aucun effet sur le processus

← la réception du signal **SIGINT** déclenche l'exécution de **traiter_signal**

← le signal **SIGQUIT** est bien considéré comme reçu mais dans l'état bloqué

← émission du signal **SIGSTOP** au processus

← la réception du signal **SIGSTOP** suspend le processus

← émission du signal **SIGKILL** au processus

← la réception du signal **SIGKILL** termine le processus

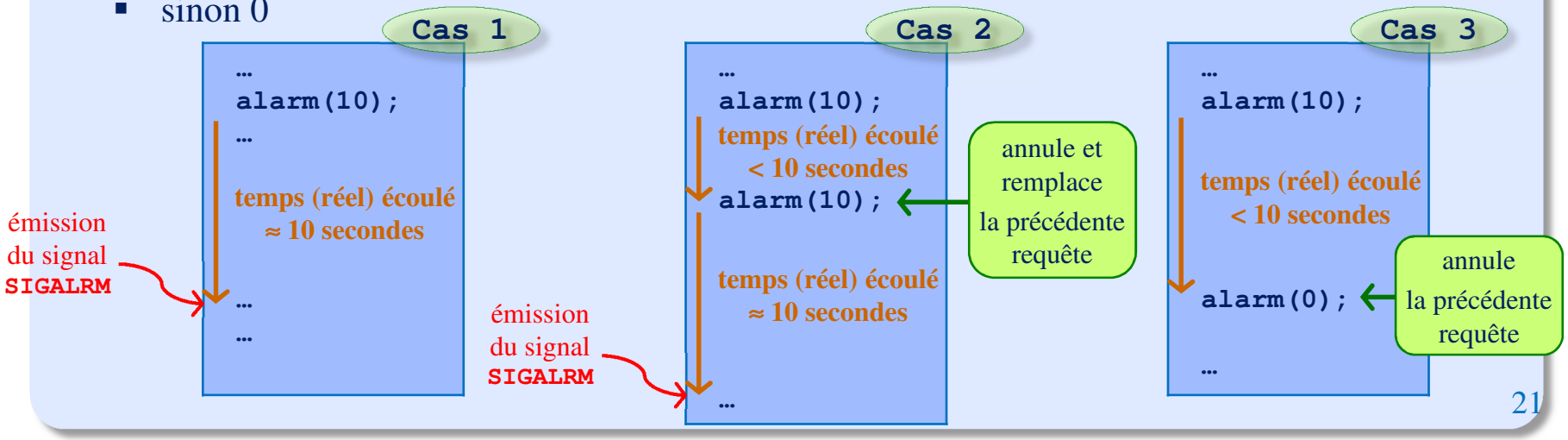
# Mise en Place d'une Temporisation

- Un processus peut demander qu'un signal **SIGALRM** lui soit envoyé au bout d'un certain temps

```
#include <unistd.h>
      unsigned int alarm(unsigned int nb_sec);
```

durée au bout de laquelle le signal **SIGALRM** est émis au processus

- Transmet au système une requête d'émission du signal **SIGALRM** au processus courant (appelant) dans **nb_sec** secondes
- Usage : implémentation de temporisateurs
- Si **nb_sec** = 0, alors la requête pendante précédente, s'il en existe une, est annulée
- Valeur de retour
  - s'il existe une précédente requête dont la durée n'a pas expirée, alors retour du temps restant
  - sinon 0



# Programmation d'un Temporisateur

- Programme **tempo.c**

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

int delai = 10;
```

variable globale

gestionnaire du signal **SIGALRM**

```c
void expiration_delai(){
    printf("\n\tLe délai de saisi a expiré ! Vous perdez %d seconde.\n", (delai > 5) ? 1 : 0);
    (delai > 5) ? delai-- : delai;
}

int main(){
char buf[128]; int nl;  int dr;
struct sigaction fonc;

fonc.sa_handler = expiration_delai;
sigaction(SIGALRM, &fonc, NULL);
```

installe le gestionnaire **expiration_delai** pour le signal **SIGALRM**

```c
while(1){
    printf("Introduire 3 noms de fruits (délai = %d secondes) : \n", delai);

    alarm(delai);
```

requête d'émission du signal **SIGALRM** au bout de **delai** secondes

```c
    nl = read(STDIN_FILENO, buf, sizeof(buf));
    if ( (dr = alarm(0)) > 0){
        printf("Vous avez utilisé %d seconde(s). Vous gagnez 1 seconde de plus !\n", delai-dr);
        delai++;
    }
}
return 0;
}
```

annule la précédente requête si existante

# Programmation d'un Temporisateur

## Exécution de `tempo.c`

```
[student]$ gcc tempo.c -o tempo
[student]$ ./tempo
Introduire 3 noms de fruits (délai = 10 secondes) :
pomme banane poire
Vous avez utilisé 9 seconde(s). Vous gagnez 1 seconde de plus !
Introduire 3 noms de fruits (délai = 11 secondes) :
kiwi orange c
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 10 secondes) :
kiwi cerise
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 9 secondes) :
kiwi cerise orange
Vous avez utilisé 8 seconde(s). Vous gagnez 1 seconde de plus !
Introduire 3 noms de fruits (délai = 10 secondes) :
mangue ananas
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 9 secondes) :
prune ana
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 8 secondes) :
ananas prune
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 7 secondes) :
prune a
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 6 secondes) :
anan
    Le délai de saisie a expiré ! Vous perdez 1 seconde.
Introduire 3 noms de fruits (délai = 5 secondes) :

    Le délai de saisie a expiré ! Vous perdez 0 seconde.
Introduire 3 noms de fruits (délai = 5 secondes) :
^C
[student]$
```

introduit par l'utilisateur

expiration du délai avant introduction

# Contrôle du Point de Reprise

- Un processus peut gérer les erreurs qu'il peut rencontrer lors de son exécution en sauvegardant le contexte de pile (avant erreur avec **`setjmp`**) pour le restaurer ultérieurement (après erreur avec **`longjmp`**)

```
#include <setjmp.h>
        int setjmp(jmp_buf env);
```

stocke le contexte courant

- Sauvegarde le contexte de pile courant dans **`env`**
- Valeur de retour
  - 0 pour un appel direct et une valeur non nulle si retour d'un appel à **`longjmp`**

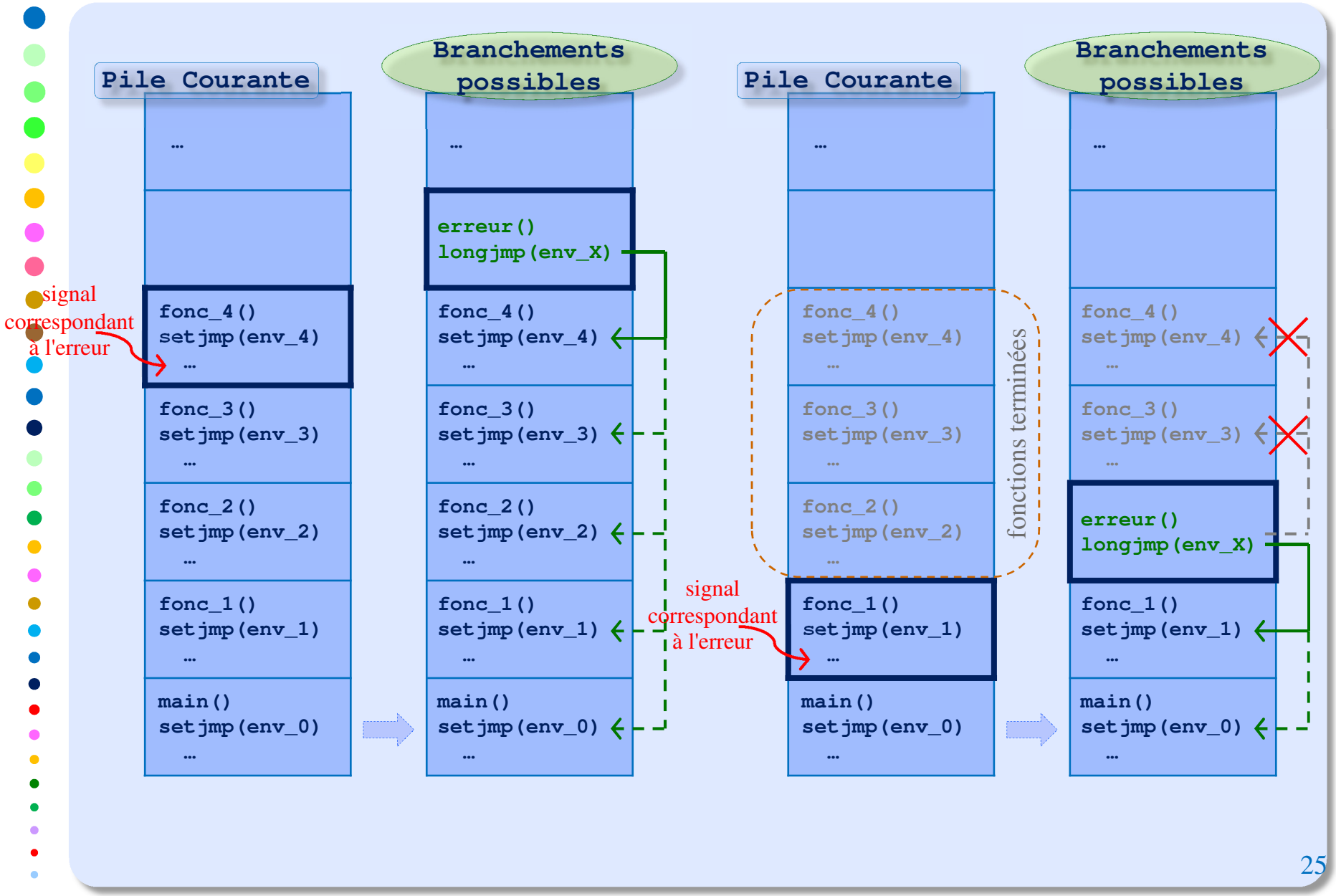
```
#include <setjmp.h>
        int longjmp(jmp_buf env, int val);
```

définit le contexte à restaurer

valeur de retour de **`setjmp`**

- Fonction qui ne revient jamais
- Simule un retour de l'appel à la fonction **`setjmp`** avec retour de la valeur **`val`** (si **`val`** = 0, alors retour de la valeur 1)
- Ces deux fonctions permettent un branchement non local
- Ces deux fonctions ne prennent pas en compte le masque de blocage des signaux

# Contrôle du Point de Reprise



# Sauvegarde/Restauration du Masque des Signaux

- **POSIX** ne spécifie pas si la fonction **setjmp()** doit sauvegarder le masque des signaux ni si la fonction **longjmp()** doit le restaurer.

- La fonction **longjmp()** est souvent appelée depuis un gestionnaire de signal pour retourner à la boucle principale d'un programme, au lieu d'y retourner depuis le gestionnaire (la norme **ANSI C** stipule qu'un gestionnaire de signal peut soit "retourner", soit appeler **abort**, **exit** ou **longjmp()**).

- Que devient le masque des signaux du processus si **longjmp()** est appelée ?
  - Sur plate-forme **4.3+BSD**, les fonctions **setjmp()** et **longjmp()** sauvegardent et restaurent le masque des signaux.

    **4.3+BSD** propose également les fonctions **_setjmp()** et **_longjmp()** qui elles ne sauvegardent pas et donc ne restaurent pas non plus le masque.
  - Sur plate-forme **SVR4**, le masque des signaux n'est pas sauvegardé pour être ensuite restauré.

# Exemple de non restauration du masque

- Programme **set_longjmp.c**

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

```
jmp_buf env;
sigset_t set;
```

variables globales

fonction qui affiche le numéro de chacun des signaux présents dans **set**

```
void affiche_signaux(sigset_t *set){
    int i;
    printf("{ ");
    for ( i = 1; i < NSIG; i++ )
        if ( sigismember(set, i) == 1 )
            printf("%d ", i);
    printf("}\n");
}
```

gestionnaire du signal **SIGINT**

```
void sig_int() {
    printf("\tDébut gestionnaire SIGINT\n");
    sigemptyset(&set); sigprocmask(0, NULL, &set);
    printf("\tSignaux actuellement bloqués : "); fflush(stdout);
    affiche_signaux(&set);

    alarm(1);
    pause();

    sigemptyset(&set); sigprocmask(0, NULL, &set);
    printf("\tSignaux actuellement bloqués : "); fflush(stdout);
    affiche_signaux(&set);
    printf("\tFin gestionnaire SIGINT par longjmp\n");
    longjmp(env, 1);
}
```

sauvegarde dans **set** le masque courant

requière l'émission du signal **SIGALRM** au bout d'une seconde et se met en attente de réception d'un signal

restaure le contexte **env** sauvegardé (branchement non local)

# Exemple de non restauration du masque

- Programme **set_longjmp.c** (suite)

gestionnaire du signal **SIGALRM**

```
void sig_alarm() {
    printf("Début gestionnaire ALARM\n");
    sigemptyset(&set); sigprocmask(0, NULL, &set);
    printf("Signaux actuellement bloqués : "); fflush(stdout);
    affiche_signaux(&set);
    printf("Fin gestionnaire ALARM\n");
    return;
}
```

sauvegarde dans **set** le masque courant

```
main() {
    signal(SIGINT, sig_int);
    signal(SIGALRM, sig_alarm);

    if ( setjmp(env) == 0 ){
        printf("\t\tMain : Retour de setjmp\n");
        sigemptyset(&set); sigprocmask(0, NULL, &set);
        printf("\t\tSignaux actuellement bloqués : "); fflush(stdout);
        affiche_signaux(&set);
        pause();
    }
    else {
        printf("\t\tMain : Retour de longjmp\n");
        sigemptyset(&set); sigprocmask(0, NULL, &set);
        printf("\t\tSignaux actuellement bloqués : "); fflush(stdout);
        affiche_signaux(&set);
        exit(0);
    }
}
```

installent des gestionnaires pour les signaux **SIGINT** et **SIGALRM**

sauvegarde le contexte courant dans **env**

# Exemple de non restauration du masque

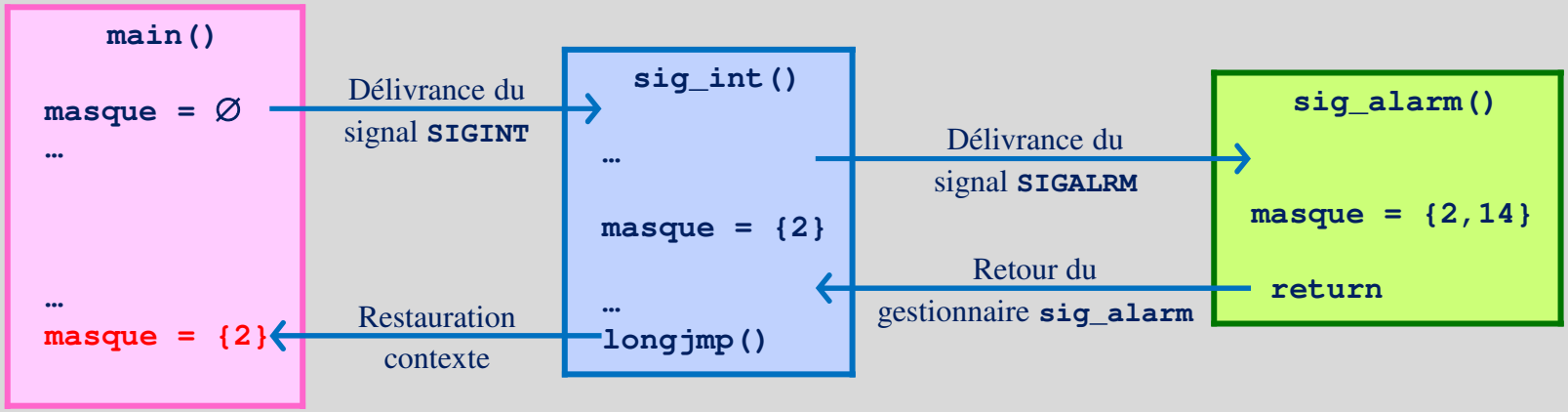
- Exécution de **set_longjmp.c**

```
[student]$ gcc set_longjmp.c -o set_longjmp
[student]$ ./set_longjmp
       Main : Retour de setjmp
       Signaux actuellement bloqués : { }
^C     Début gestionnaire SIGINT
       Signaux actuellement bloqués : { 2 }
Début gestionnaire ALARM
Signaux actuellement bloqués : { 2 14 }
Fin gestionnaire ALARM
       Signaux actuellement bloqués : { 2 }
       Fin gestionnaire SIGINT par longjmp
       Main : Retour de longjmp
       Signaux actuellement bloqués : { 2 }
[student]$
```

exécution de la fonction **main**

exécution du gestionnaires **sig_int**

exécution du gestionnaires **sig_alarm**



# Fonctions POSIX sigsetjmp et siglongjmp

- La fonction **`sigsetjmp()`** sauvegarde le contexte de pile ainsi que le masque courant des signaux pour les restaurer ultérieurement avec la fonction **`siglongjmp()`**

```
#include <setjmp.h>
        int sigsetjmp(sigjmp_buf env, int savesigs);
```

stocke le contexte et le masque courants

indique si le masque doit être aussi sauvegardé dans **`env`**

- Sauvegarde le contexte de pile courant dans **`env`** (ainsi que le masque courant si **`savesigs`** ≠ 0)
- Valeur de retour
  - 0 pour un appel direct et une valeur non nulle si retour d'un appel à **`siglongjmp`**

```
#include <setjmp.h>
        int siglongjmp(sigjmp_buf env, int val);
```

définit le contexte à restaurer

valeur de retour de **`setjmp`**

- Fonction qui ne revient jamais
- Simule un retour de l'appel à la fonction **`sigsetjmp()`** avec retour de la valeur **`val`** (si **`val`** = 0, alors retour de la valeur 1)
- Restaure le contexte (et éventuellement le masque des signaux) sauvegardé(s) par l'appel à **`sigsetjmp()`**

# Exemple de restauration du masque

- Programme **sigset_longjmp.c** (en rouge les modification par rapport à **set_longjmp.c**)

| | | | |
|---|---|---|---|
| Déclaration : | `jmp_buf env;` | ⇒ | `sigjmp_buf env;` |
| Fonction `sig_int()` : | `longjmp(env, 1);` | ⇒ | `siglongjmp(env, 1);` |
| Fonction `main()` : | `setjmp(env)` | ⇒ | `sigsetjmp(env, 1);` |

- Exécution de **sigset_longjmp.c**

```
[student]$ gcc sigset_longjmp.c -o sigset_longjmp
[student]$ ./sigset_longjmp
        Main : Retour de setjmp
        Signaux actuellement bloqués : { }
^C      Début gestionnaire SIGINT
        Signaux actuellement bloqués : { 2 }
Début gestionnaire ALARM
Signaux actuellement bloqués : { 2 14 }
Fin gestionnaire ALARM
        Signaux actuellement bloqués : { 2 }
        Fin gestionnaire SIGINT par longjmp
        Main : Retour de longjmp
        Signaux actuellement bloqués : { }
[stud
```

exécution de la fonction **main** !

exécution du gestionnaires **sig_int**

exécution du gestionnaires **sig_alarm**

exécution de la fonction **main** !