

# Thread POSIX

Florent Gluck, *V. Pilloux*

Version 1.1

# Librairie POSIX Threads

- Librairie C normalisée POSIX pour la gestion et synchronisation des threads.
- Existe pour la plupart des plateformes : Linux, OSX, Windows, Solaris, etc.
- Fichier d'en-tête : `pthread.h`
- Compilation et édition des liens :
  - `gcc -Wall -Wextra -std=gnu11 prg.c -o prg -lpthread`
  - **ATTENTION** : `-lpthread` **doit** être le dernier argument !
- Déclaration d'un thread : `pthread_t thread;`

# Création d'un thread

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine)(void *),  
                  void *arg);
```

- `thread` est **l'identifiant** du thread créé (argument de sortie).
- `attr` permet de définir les attributs du thread (NULL = attributs par défaut)
- `start_routine` est la fonction exécutée par le thread créé. Celle-ci **doit** obligatoirement avoir le prototype suivant :  
`void *fonction(void *data)`
- `arg` est l'argument passé à la fonction (NULL = pas d'argument)
- Renvoie 0 en cas de succès → **toujours** tester le succès de l'appel !

# Création thread : exemple trivial

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

void *func(void __attribute__((unused)) *arg) {
    printf("Hello from our first thread!\n");
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t thread;
    if (pthread_create(&thread, NULL, func, NULL) != 0) {
        perror("thread creation error");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

- Compilation et édition des liens avec :

```
gcc -Wall -std=gnu11 hello.c -o hello -lpthread
```


# Création : exemple de passage d'argument

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

void *func(void *arg) {
    char *msg = (char *) arg;
    printf("Message = %s\n", msg);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t;
    char *msg = "Threads are awesome!";
    if (pthread_create(&t, NULL, func, msg) != 0) {
        perror("thread creation error");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Passage d'un argument



# Creation : quiz 1

Quel sera l'affichage produit ?

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

```
void *func(void *arg) {
    char *msg = (char *) arg;
    printf("Message = %s\n", msg);
    return NULL;
}
```

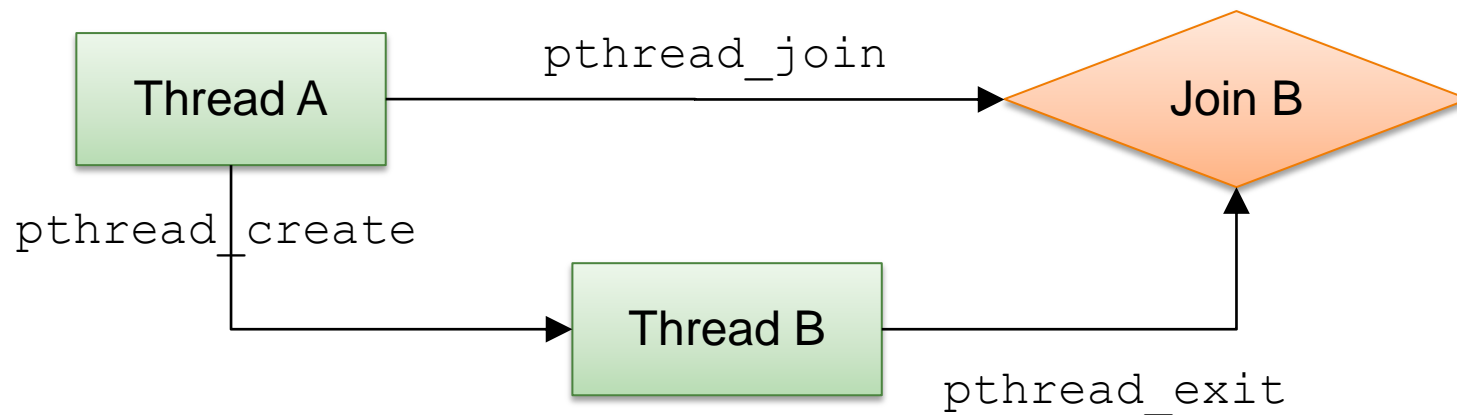
Passage d'un argument

```
int main(int argc, char *argv[]) {
    pthread_t t;
    char *msg = "Threads are awesome!";
    if (pthread_create(&t, NULL, func, msg) != 0) {
        perror("thread creation error");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

➤ Aucun!! Le main() se termine immédiatement et tue le thread!

# Jointure (1)

- La jointure est la forme de communication entre threads la plus simple.
- Joindre un thread signifie bloquer jusqu'à la terminaison de celui-ci.
- Le thread terminé peut retourner une valeur au thread ayant effectué la jointure.



# Jointure (2)

```
int pthread_join(pthread_t thread,  
                 void **value_ptr);
```

- Attend que le thread passé en paramètre se termine.
- Le thread appelant est bloqué jusqu'à la terminaison du thread spécifié.
- `value_ptr` contient la valeur de retour du thread terminé.
- Renvoie 0 en cas de succès.



# Jointure : exemple trivial

```
void *func(void *arg) {
    char *msg = (char *) arg;
    printf("Message = %s\n", msg);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t;
    char *msg = "Threads are awesome!";
    if (pthread_create(&t, NULL, func, msg) != 0) {
        perror("pthread_create");
        return EXIT_FAILURE;
    }
    if (pthread_join(t, NULL) != 0) {
        perror("pthread_join");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

# Jointure : quiz 1

```
void *func(void *arg) {  
    char *msg = (char *) arg;  
    printf("Message = %s\n", msg);  
    return NULL;  
}
```

```
int main(int argc, char *argv[]) {  
    pthread_t t;  
    char *msg = "Threads are awesome!";  
    if (pthread_create(&t, NULL, func, msg) != 0) {  
        perror("pthread_create");  
        return EXIT_FAILURE;  
    }  
    if (pthread_join(t, NULL) != 0) {  
        perror("pthread_join");  
        return EXIT_FAILURE;  
    }  
    return EXIT_SUCCESS;  
}
```

Quel sera l'affichage produit ?

➤ Message = Threads are awesome!

# Jointure : quiz 2

```
void *func(void *arg) {
    char *msg = (char *) arg;
    printf("Message = %s\n", msg);
    int exit_code = 33;
    return &exit_code;
}

int main(int argc, char *argv[]) {
    pthread_t t;
    char *msg = "Threads are awesome!";
    if (pthread_create(&t, NULL, func, msg) != 0) {
        perror("pthread_create");
        return EXIT_FAILURE;
    }
    int *exit_code;
    if (pthread_join(t, (void **)&exit_code) != 0) {
        perror("pthread_join");
        return EXIT_FAILURE;
    }
    printf("Thread exit code = %d\n", *exit_code);
    return EXIT_SUCCESS;
}
```

Quel sera l'affichage produit ?

➤ Un nombre quelconque! La variable est locale et jetée!

# Jointure : quiz 3

```
void *func1(void *arg) {
    static int exit_code = 0;
    printf("Thread 1\n");
    return &exit_code;
}

void *func2(void *arg) {
    static int exit_code = 4;
    printf("Thread 2\n");
    pthread_exit((void *)&exit_code);
}

int main(int argc, char *argv[]) {
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, func1, NULL);
    pthread_create(&thread2, NULL, func2, NULL);
    int *status1, *status2;
    pthread_join(thread1, (void **)&status1);
    pthread_join(thread2, (void **)&status2);
    printf("Status1: %d\n", *status1);
    printf("Status2: %d\n", *status2);
    return EXIT_SUCCESS;
}
```

Quel sera l'affichage produit ?

Un exemple possible:

- Thread 1
- Thread 2
- Status1: 0
- Status2: 4

\* Tests d'erreurs omis pour des raisons de places

# Jointure : quiz 4

```
void *func(void *arg) {  
    char *msg = (char *) arg;  
    printf("Message = %s\n", msg);  
    int *code = malloc(sizeof(int));  
    *code = 47;  
    return code;  
}
```

```
int main(int argc, char *argv[]) {  
    pthread_t t;  
    char *msg = "Threads are awesome!";  
    if (pthread_create(&t, NULL, func, msg) != 0) {  
        perror("pthread_create");  
        return EXIT_FAILURE;  
    }  
    int *exit_code;  
    if (pthread_join(t, (void **)&exit_code) != 0) {  
        perror("pthread_join");  
        return EXIT_FAILURE;  
    }  
    printf("Thread exit code = %d\n", *exit_code);  
    return EXIT_SUCCESS;  
}
```

Quel sera l'affichage produit ?

- Message = Threads are awesome!
- Thread exit code = 47

# Terminaison d'un thread

- Un thread peut se terminer de deux manières possibles :
  - 1) Le thread se termine **lui-même** (auto-terminaison)
    - Instruction `return`
    - Fonction `pthread_exit`
  - 2) Un autre thread le termine (mécanisme d'annulation) :
    - Fonction `pthread_cancel`

# Auto terminaison

```
void pthread_exit(void *value);
```

- Termine le thread et retourne `value` au thread effectuant la jointure.
- Le mot clé **return** appelé dans le corps du thread permet aussi de terminer le thread et retourner une valeur au thread effectuant la jointure.
- La fonction **exit** permet de terminer le processus, ce qui a pour effet **de terminer instantanément tous les threads du processus**; ceci, quel que soit le thread ayant appelé **exit**.
- Note: si `pthread_exit` est appelé dans la fonction `main`, le programme bloque et attend que tous les threads se terminent avant de terminer le processus !

# Attributs de thread (1)

```
int pthread_attr_init(pthread_attr_t *attr);  
int pthread_attr_destroy(pthread_attr_t *attr);
```

- La fonction `pthread_attr_init` initialise un « attribut de thread » en vue d'être utilisé lors de la création d'un thread.
- La fonction `pthread_attr_destroy` détruit un « attribut de thread »
- Une fois le thread créé, l'attribut n'est plus nécessaire et peut être détruit.
- Renvoie 0 en cas de succès.
- Cf. man `pthread_attr_init` pour voir la liste des attributs configurables.



# Attributs de thread (2)

- Exemple : attributs spécifiant 4MB de pile et une plage de garde de 512 bytes:

```
void *func(void *arg) {  
    puts("hello");  
}  
  
int main(int argc, char *argv[]) {  
    pthread_attr_t attr;  
    pthread_attr_init(&attr);  
    pthread_attr_setguardsize(&attr, 512);  
    pthread_attr_setstacksize(&attr, 1024*4096);  
  
    pthread_t t;  
    pthread_create(&t, NULL, func, NULL);  
    pthread_attr_destroy(&attr);  
  
    pthread_join(t, NULL);  
    return EXIT_SUCCESS;  
}
```

\* Tests d'erreurs  
omis pour des  
raisons de lisibilité

# Rendre le processeur

- Il est possible d'indiquer à l'ordonnanceur que le thread **souhaite** rendre le processeur (« rendre la main »):

```
#include <sched.h>

int sched_yield();
```

- Le but est que le thread suivant en attente du processeur soit activé.
- Cependant, il n'y a **aucune garantie** que le thread courant sera mis en attente et que le prochain thread sera ordonnancé !
- Renvoie zéro en cas de succès.

# Auto identification

- Un thread peut obtenir son identifiant avec la fonction:

```
pthread_t pthread_self();
```

- Il s'agit donc de la même valeur d'identifiant que celle retournée en premier argument de la fonction `pthread_create`
- Chaque thread dispose d'un numéro d'identifiant **unique**
- Sur linux x64, l'identifiant est un entier long non signé (`unsigned long int`)
- **Attention:** il n'est pas garanti que l'identifiant d'un thread soit un entier ! Le type est propre à l'implémentation (voir slide suivante pour comparer des identifiants de threads)

# Comparaison d'identifiants

- Comparer deux identifiants de threads **doit** se faire avec la fonction:

```
int pthread_equal(pthread_t t1, pthread_t t2);
```

- Si les threads `t1` et `t2` sont égaux, alors la fonction renvoie une valeur différente de zéro
- Il est nécessaire d'utiliser cette fonction pour comparer les identifiants, car rien **ne garanti** que les types `pthread_t` soient des valeurs comparables avec l'opérateur d'égalité `=`

# Ressources

## POSIX Threads Programming

- <https://computing.llnl.gov/tutorials/pthreads/>

## Initiation à la programmation multitâche en C avec Pthreads

- <http://franckh.developpez.com/tutoriels/posix/pthreads/>

## Le manuel Pthreads

- « man pthreads » dans un terminal UNIX