# Threads - Création

Programmation Système — R3.05

C. Raïevsky

IUT A Valence
Université Grenoble Alpes

Département Informatique
**BUT Informatiques 2ème année**



## Thread - Définition

Fil d'exécution – processus léger

### Fil d'exécution

Un thread est

- L'unité d'exécution d'un OS
- Le concept qui encapsule l'exécution

Un *processus*

- a toujours au moins un thread : Le thread principal - *Main Thread*
- peut créer d'autres threads

Lorsque l'OS doit attribuer une tâche au *core* d'un CPU

↪ il choisit un thread





## Création

Lors de la création d'un thread

- Un nouveau contexte d'exécution est créé :
  - Pile dédiée
  - Ensemble de registres

Lancement de l'exécution

Dans une fonction





## Création d'un thread POSIX – Exemple minimal

```
1  void* printDots(void*) {
2     for (int i = 0; i < 100000; ++i){ printf("."); }
3     printf("Thread␣done␣printing\n");
4  }
5
6  int main(int argc, char *argv[]) {
7
8     pthread_t th;
9     pthread_create(&th, NULL, printDots, NULL);
10
11    for (int i = 0; i < 100000; ++i){ printf("*"); }
12    printf("Main␣done␣printing\n");
13
14    pthread_join(th, NULL); /* On attend la fin du thread */
15
16    printf("Main␣exiting\n");
17    return EXIT_SUCCESS;
18 }
```

## Espace mémoire `man pthreads`

Les threads d'un même processus **partagent** :

- L'espace mémoire (programme, data, bss, tas, mmap)
- Les PID et PPID
- Le propriétaire et les droits
- Les descripteurs de fichier ouverts
- Les programmes d'interruption
- etc.

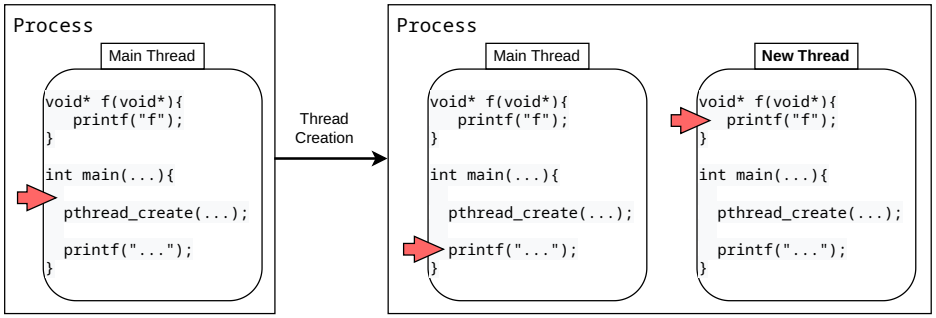
Les threads d'un même processus **ne partagent pas** :

- **La stack ! !**
- Les filtres d'interruption
- etc.





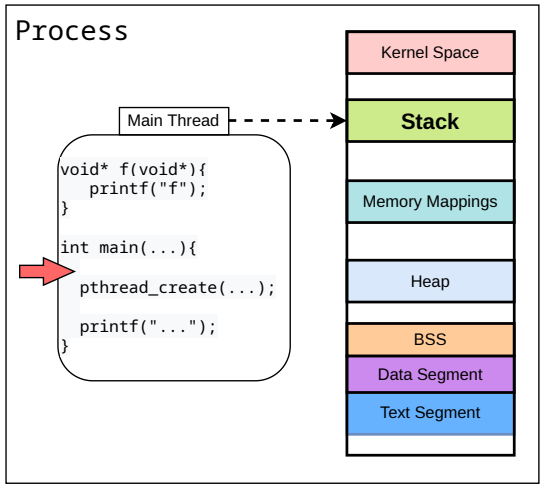
## Création d'un thread







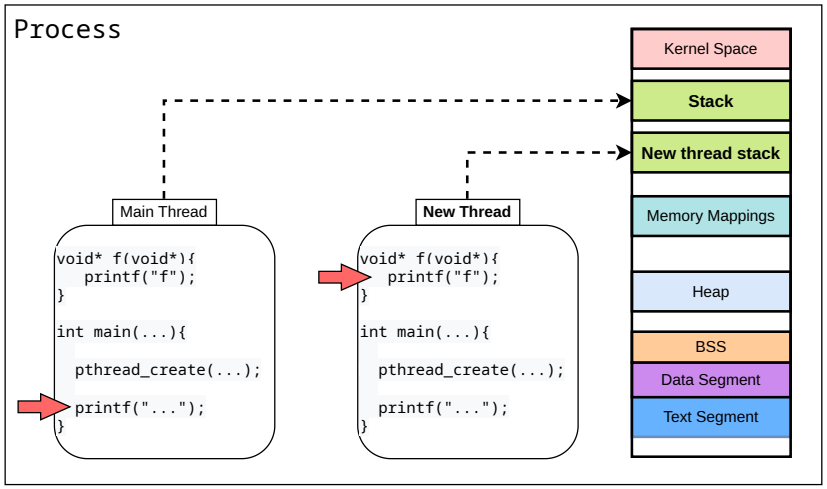

## Mémoire du processus avant la création d'un thread



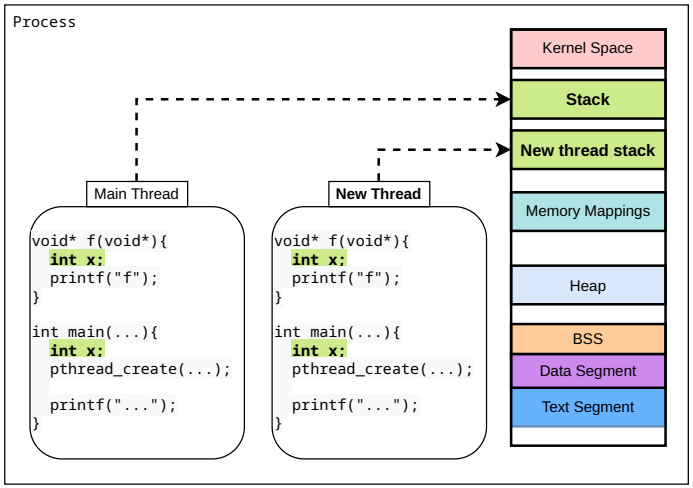





## Mémoire du processus après la création d'un thread



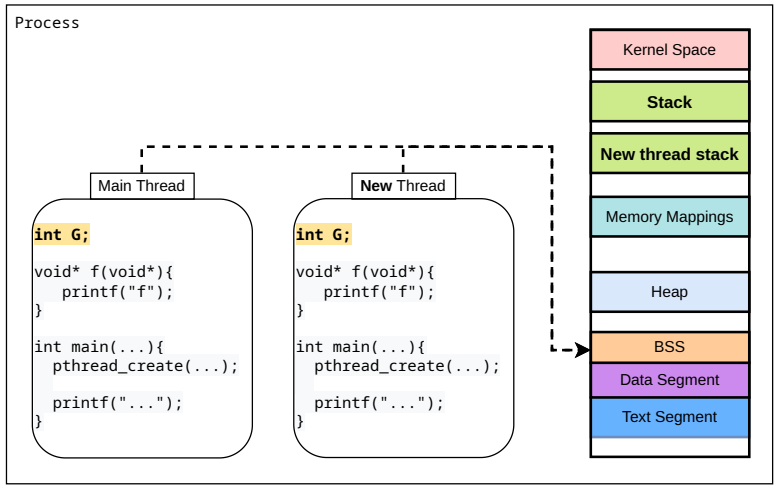
## Variables locales








## Variable globale








## Passer des paramètres

L'interface `pthread` impose la signature de la fonction de démarrage

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

Troisième paramètre :

```
void *(*start_routine)(void*)
```

Pointeur vers une fonction qui :

- Prend un void* en paramètre
- Renvoie un void*





## Passage de paramètres à un thread - Exemple

```
1  typedef struct {
2     int nbRep;
3     char toPrint;
4  } param_t;
5
6  void* printChar(void* p) {
7     param_t params = *((param_t*)p);
8
9     for (int i = 0; i < params.nbRep; ++i){
10       printf("%c",params.toPrint);
11    }
12
13    printf("Thread done printing\n");
14 }
```

```
15 int main(int argc, char *argv[]) {
16    param_t p = {10000, 'a'};
17    pthread_t th;
18    pthread_create(&th, NULL, printChar, &p);
19
20    for (int i = 0; i < 10000; ++i){
21       printf("*");
22    }
23    printf("Main done printing\n");
24
25    pthread_join(th, NULL);
26
27    printf("Main exiting\n");
28    return EXIT_SUCCESS;
29 }
```

```
param_t params = *((param_t*)p);
```

## Récupérer un résultat

La valeur de retour de `pthread_create` n'a rien à voir avec le résultat du thread

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

Résultat :

- 0 si tout va bien
- Un code d'erreur sinon (cf `man pthread_create`)





## Récupérer un résultat

Prototype de `pthread_join`

```
int pthread_joint(pthread_t thread, void **retval);
```

Deuxième paramètre :

- Pointeur vers le résultat renvoyer par le thread
- La valeur retournée par la fonction de départ ou
- La valeur passée à `pthread_exit`





## Récupération du résultat d'un thread - Exemple

```
1  typedef struct {
2     int* array;
3     int first;
4     int last;
5  } param_t;
6
7  void* sumIntArray(void* v) {
8     param_t p = *((param_t*)v);
9
10    long int* sum = malloc(sizeof(long int));
11
12    for (int i = p.first; i < p.last; ++i){
13       *sum += p.array[i];
14    }
15    return (void*)sum;
16 }
```

```
17 int main(int argc, char *argv[]) {
18    int myArray[1000];
19
20    param_t p = {myArray, 0, 1000};
21    pthread_t th;
22    pthread_create(&th, NULL, sumIntArray, &p);
23
24    printf("Main waiting for result.\n");
25
26    long int* result;
27    pthread_join(th, (void**)&result);
28
29    printf("Result : %ld\n", *result);
30
31    free(result);
32    return EXIT_SUCCESS;
33 }
```





## Récupération du résultat d'un thread - Exemple - Autre méthode

```
1  typedef struct {
2     int* array;
3     int first;
4     int last;
5     int* result;
6  } param_t;
7
8  void* sumIntArray(void* v) {
9     param_t p = *((param_t*)v);
10
11    *p.result = 0;
12
13    for (int i = p.first; i < p.last; ++i){
14       *p.result += p.array[i];
15    }
16
17    return NULL;
18 }
```

```
19 int main(int argc, char *argv[]) {
20    int myArray[1000];
21    int result = 0;
22
23    param_t p = {myArray, 0, 1000, &result};
24    pthread_t th;
25    pthread_create(&th, NULL, sumIntArray, &p);
26
27    printf("Main waiting for result.\n");
28
29    pthread_join(th, NULL);
30
31    printf("Result : %ld\n", result);
32    return EXIT_SUCCESS;
33 }
```

## Paramètres et résultat - Variables globales

La tentation des variables globales est forte

- Au vu de la complexité des solutions proposées
- Bien fait ça peut marcher...

Mais le péril est grand...





## Problèmes posés par l'exécution non séquentielle

Que ce soit multi-threads ou multi-processus

Exécution

- Non séquentielle
- Difficilement prédictible
- **C'est l'OS qui décide de l'ordre dans lequel les threads sont exécutés.**

Accès concurrent aux ressources partagées

- Mémoire - Notamment les variables globales
- Fichier disque
- Réseau





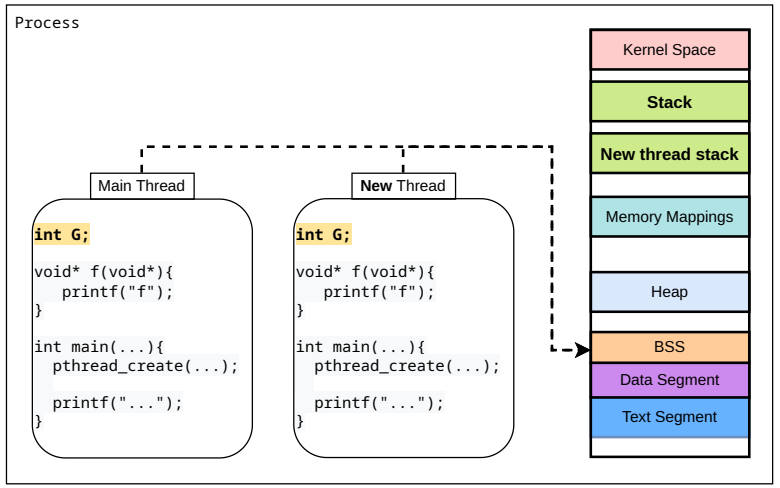
## Accès concurrent – Exemple minimal

```c
int G = 0;
void* incG() {
   for (int i = 0; i < 500000; ++i) { G++; }
}

int main(int argc, char* argv[]){
   pthread_t th1, th2;

   pthread_create(&th1, NULL, incG, NULL);
   pthread_create(&th2, NULL, incG, NULL);

   pthread_join(th1, NULL);
   pthread_join(th2, NULL);

   printf("Final value: %d\n", G);
   return EXIT_SUCCESS;
}
```





## Variable globale



## Solutions

Tous ces problèmes et leurs (éventuelles) solutions sont le domaine de

# la programmation concurrente