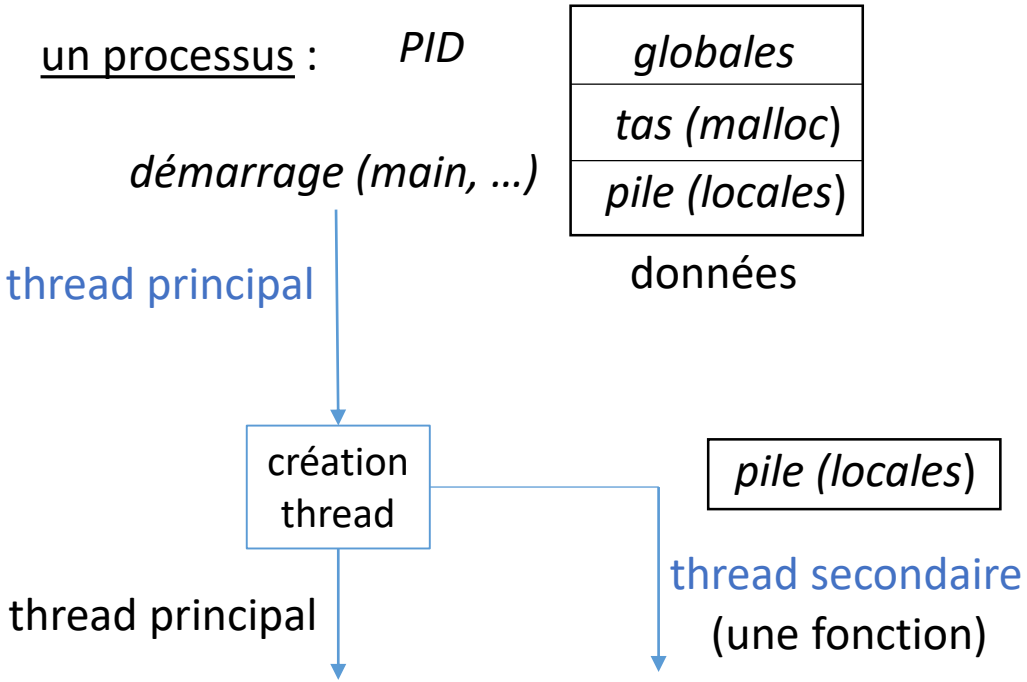
Séance 5
Threads

Thread : définition

Un **thread** : un chemin d'exécution à l'intérieur d'un processus

- matérialisé par l'exécution d'une fonction
- s'exécutant en parallèle du reste du processus



- un processus possède au moins un thread (le thread principal)
- on peut créer d'autres threads, qui s'exécuteront en parallèle
- tous les threads partagent la zone de données du processus : variables globales, tas (malloc)
- chaque thread a sa pile (variables locales)

Création d'un thread : pthread_create

pthread_create (*adr. de la variable où récupérer l'identifiant du thread,*
attributs – toujours NULL,
la fonction à exécuter par le thread,
l'argument à passer à la fonction du thread)

retourne 0 (OK) ou != 0 (échec)

identifiant d'un thread : type pthread_t

fonction à exécuter : passer son adresse (son nom)

```
pthread_t idThread;       pour y récupérer l'identifiant

pthread_create(&idThread, NULL, mafonction, …);
```

fonction du thread (prototype imposé)

```
void *mafonction(void *arg)  {
   …
}
```

voir diapositive suivante

Argument de la fonction d'un thread

Possibilité de passer une donnée à la fonction d'un thread :

- par le dernier paramètre de pthread_create
- on passe l'adresse de la donnée
- la fonction du thread récupère cette adresse dans son paramètre arg

Donnée passée en argument de la fonction d'un thread : doit être **variable globale** ou **malloc.**

```
int x = 5;        // variable globale
```

```
pthread_t idThread;

pthread_create(&idThread, NULL, mafonction,  &x);
```

```
void *mafonction(void *arg)  {
      int *p;    int val;
      p = (int *)arg;
      val = *p;
      …
}
```

arg contient &x

Si pas de donnée à passer au thread :
pthread_create(&idThread, NULL, mafonction, NULL);

Terminaison d'un thread

Un thread se termine quand sa fonction se termine par return ou pthread_exit.

Si le thread n'a pas de donnée à retourner :
return NULL; ou **pthread_exit(NULL);**

Un thread peut retourner d'une donnée sous forme de l'adresse de cette donnée (globale ou malloc).

```
float y = 3.2;       // variable globale
```

```
void *mafonction(void *arg)  {
   …
   return &y;    ou    pthread_exit(&y);
}
```

Lorsque le processus (le thread principal) se termine : tous les threads se terminent.

Attente de la fin d'un thread : pthread_join

pthread_join (*identifiant du thread,*
adresse du pointeur où récupérer la valeur d'exit du thread ou NULL)

retourne 0 (OK) ou != 0 (échec)

- fonction bloquante (sauf si le thread est déjà fini) en attente de la fin du thread
- n'importe quel thread peut attendre la fin d'un autre thread

Si le thread ne retourne pas de donnée ou si on n'est pas intéressé par cette donnée :

```
…
pthread_join(idThread, NULL);
```

Pour récupérer la donnée (son adresse) retournée par le thread :

```
float *p;    float val;
…
pthread_join(idThread, (void **) &p);
val = *p;
```