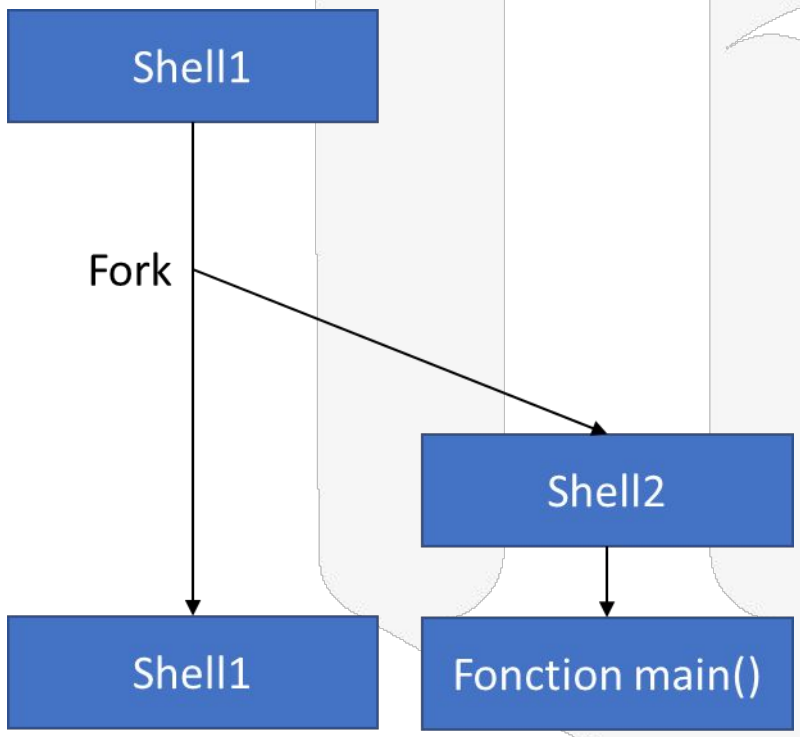
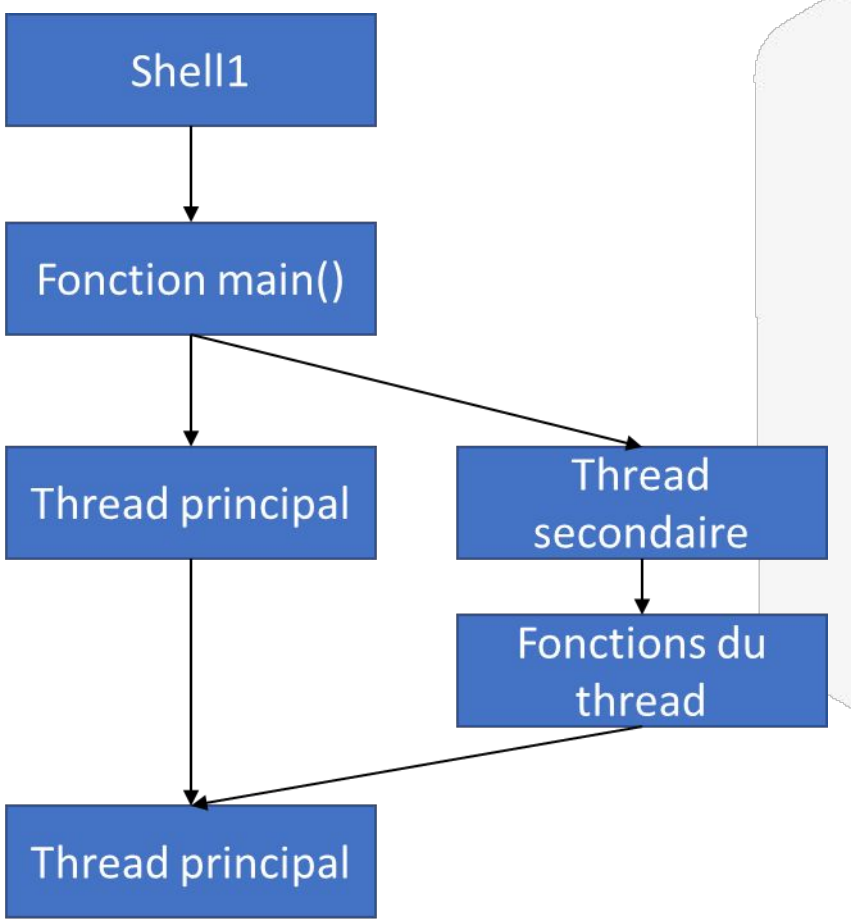
Les threads

Rappel : processus

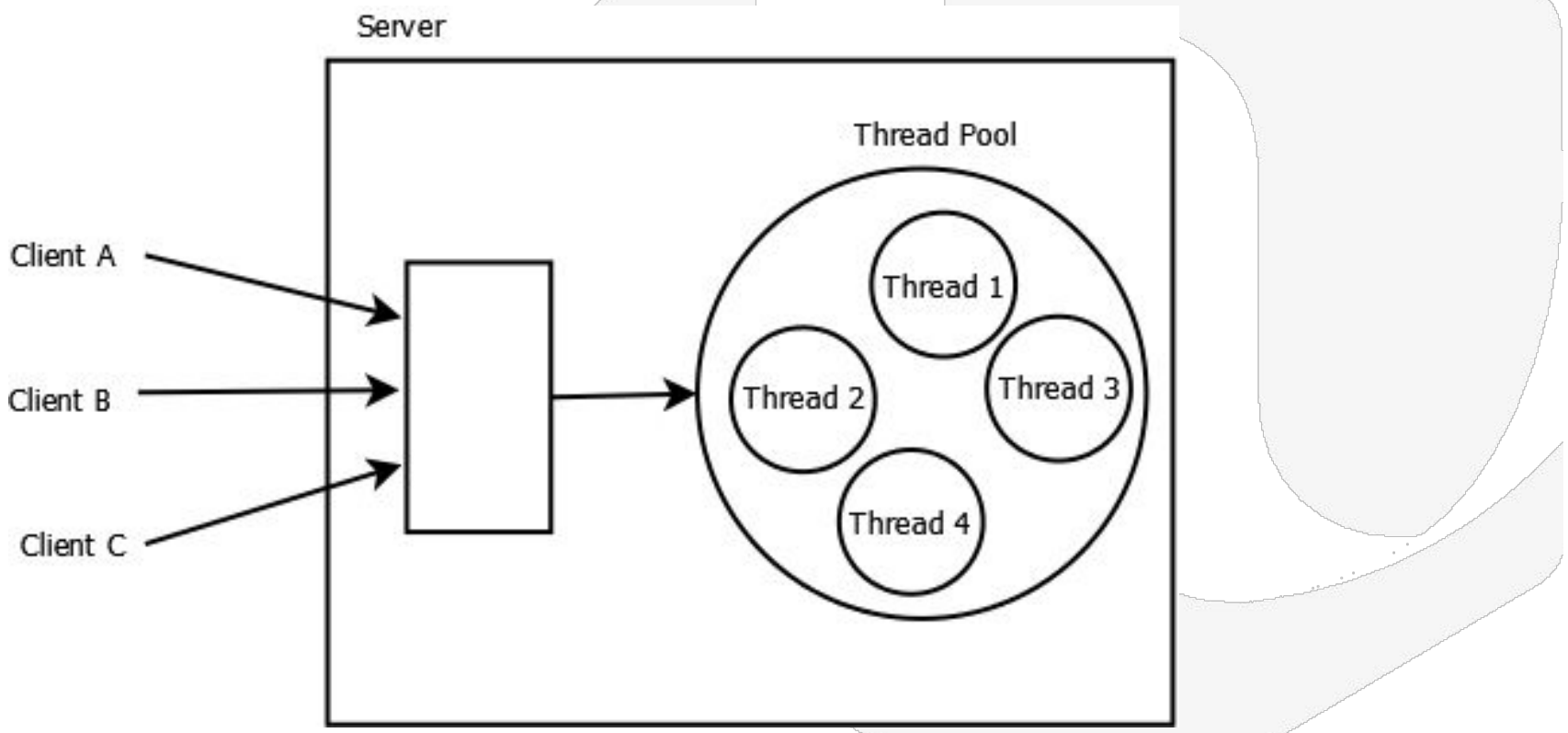


Thread



- Thread = processus « léger »
- Même espace mémoire
- Parallélisation des tâches
- Travail coopératif
- Plus léger… mais moins robuste
  - Isolation
  - Réaction aux signaux

Exemple : le serveur réseau

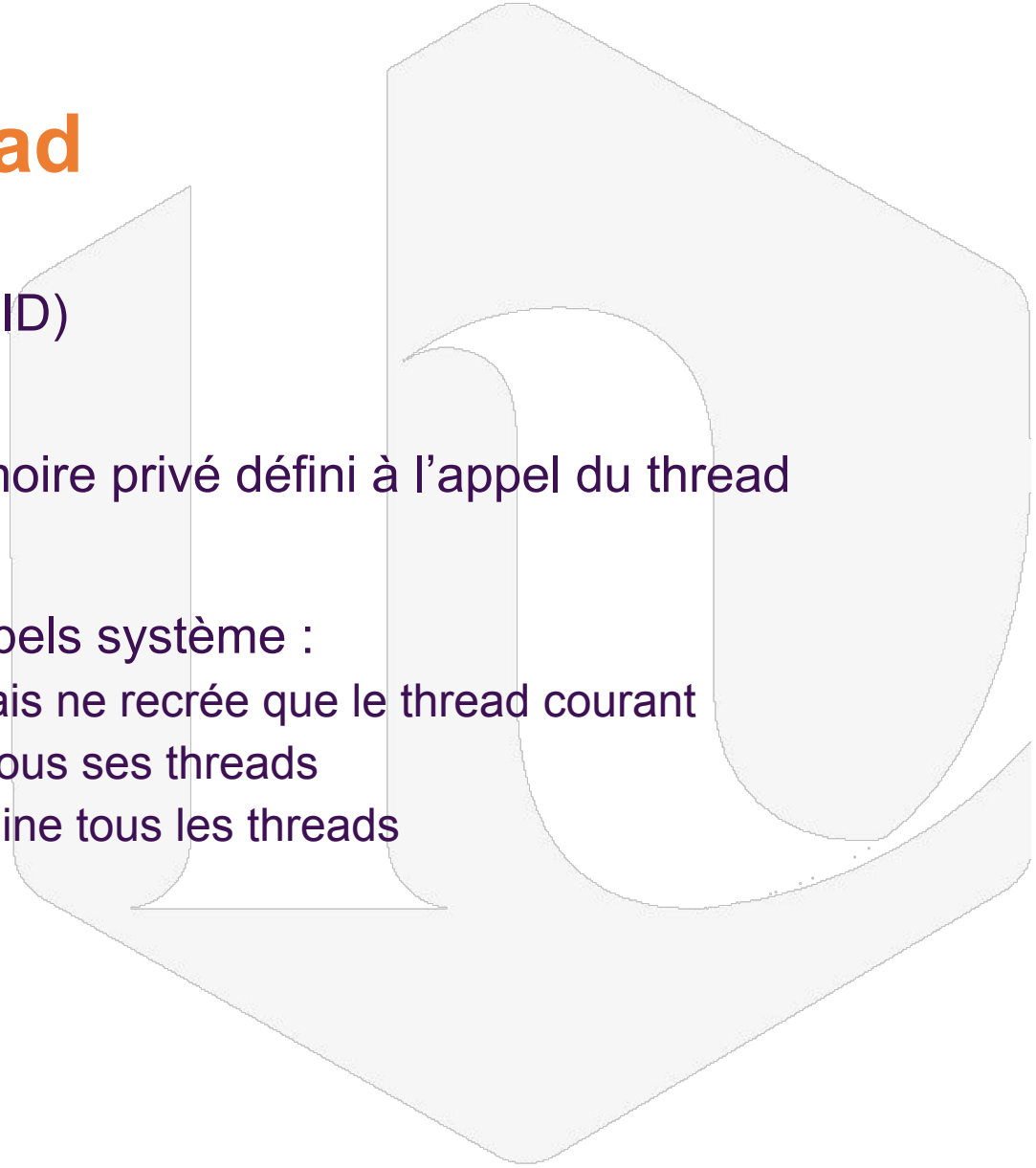


Partages entre thread

- intégration au sein d'un même processus
- les threads partagent :
  - En couche utilisateur :
    - code exécutable
    - variables de la section données et du tas
    - variables allouées dynamiquement
    - les bibliothèques et autres objets projetés en mémoire
  - En couche noyau :
    - le pid et le ppid
    - les privilèges
    - la table des descripteurs de fichiers ouverts
    - les verrous de fichiers
    - …
- Ils ne partagent pas la pile !!!!
  - variables locales à la fonction, etc...

Propriété d'un thread

- Possède un *Thread Identifier* (TID)
- Peut disposer d'un espace mémoire privé défini à l'appel du thread
- Comportement vis-à-vis des appels système :
  - fork : duplique le processus mais ne recrée que le thread courant
  - exec : écrase le processus et tous ses threads
  - exit (ou sortie du main()) : termine tous les threads



Etats des threads

- joignabilité
  - il est possible de récupérer son code de retour
  - non joignable (détaché) espace mémoire libéré à la fin
- Annulabilité
  - il accepte les requêtes de terminaison que les autres threads lui envoient
- mode d'annulation
  - Immédiat : terminaison dès réception de la requête
  - Différé : examen des requêtes d'annulation à certains moments seulement

Le défi : la synchronisation

- Partage d'un même espace mémoire = accès aux mêmes variables !
  - Utilisation de verrous
  - Un thread doit libérer ses accès sous peine de bloquer les autres…

- L'ordonnancement doit être adapté :
  - Modèle 1:1 □ les threads sont traités comme des processus individuels
  - Modèle M:1 □ le temps processus est partagé entre ses threads
  - Modèle M:N □ hybridation des 2 solutions

Le défi : la synchronisation

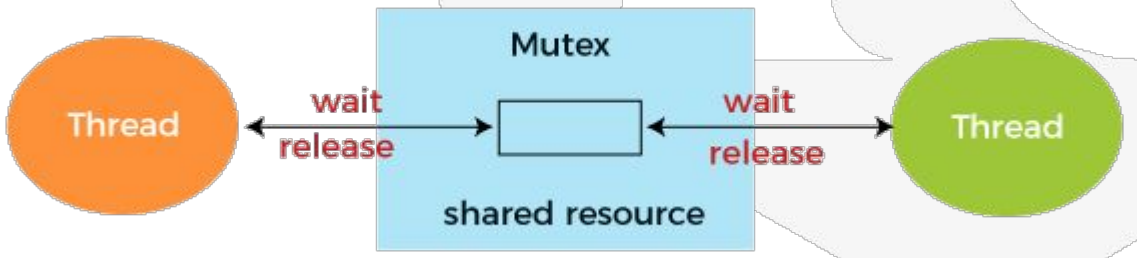
- éviter les incohérences liées aux accès concurrents à ces ressources
  - *Race condition*

- deux fonctions accèdent à une même variable globale autrement qu'en lecture seule…
  - Définition d'une **section critique**
  - Mise en place d'un verrou par le thread 1, puis relâche pour le thread 2…

- Problèmes des verrous :
  - interblocage (deadlock)
    - Chaque thread attend un événement de l'autre…
  - Autoblocage
    - Un thread attend un événement que seul lui peut produire
  - Famine
    - L'ordonnanceur ne donne jamais la main à un thread il bloque une ressource !

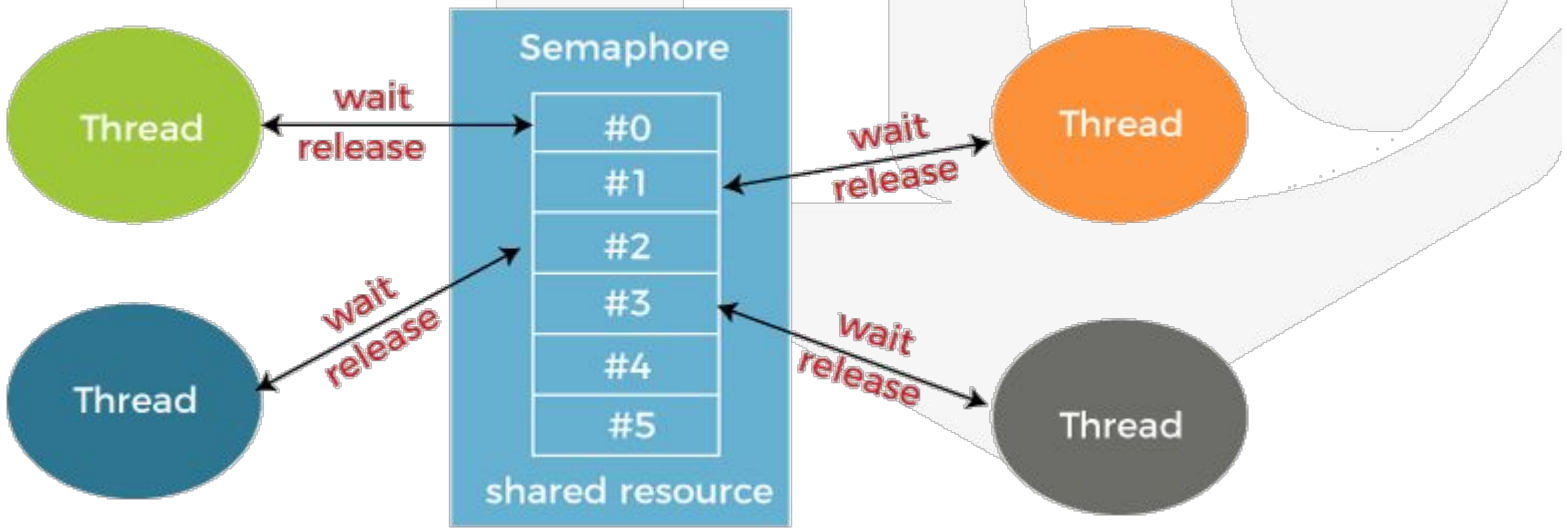
La synchronisation : mécanismes

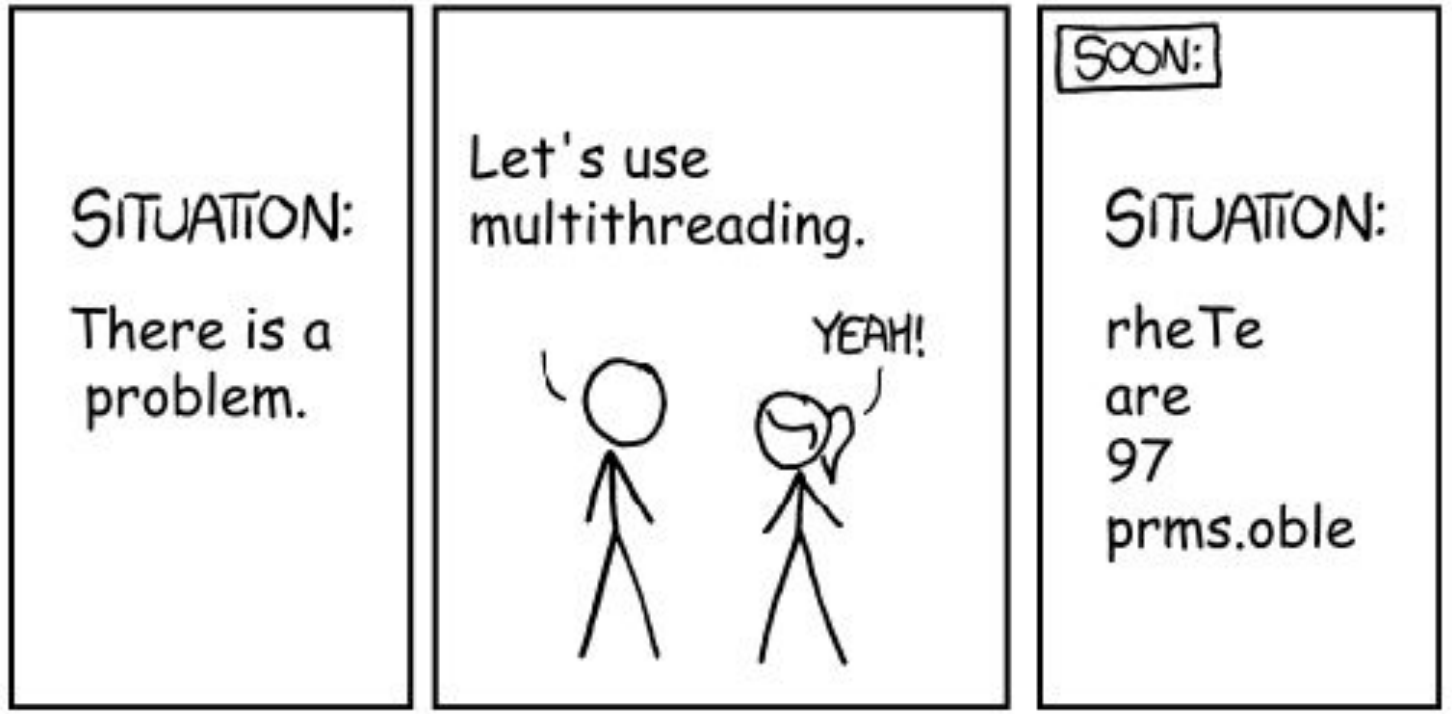
- MUTEX
  - *MUTually Exclusive*
  - au plus un thread peut posséder le verrou à un instant donné (verrou exclusif)
  - seul le possesseur d'un mutex peut le relâcher
  - parallélisme impossible dans les sections protégées
    - restriction forte voire excessive…
    - À réserver à des sections critiques très courtes !



La synchronisation : mécanismes

- Sémaphore
  - Verrou indiquant la disponibilité d'un certain nombre de ressources
  - 3 opérations :
    - I : initialisation du sémaphore
    - V : ajout d'une ressource
    - P : prise d'une ressource




SITUATION:
There is a problem.
Let's use multithreading.
YEAH!
SOON:
SITUATION:
rheTe
are
97
prms.oble