

Objectif: Ce cours a pour but de présenter les concepts de base et les applications de la programmation parallèle.
Ils sont illustrés à travers une démarche de développement et des applications

- Expression des activités parallèles: processus et threads
- Modèles de parallélisme: synchrone, asynchrone
- Modélisation par réseaux de Pétri
- Mécanismes de communication et de synchronisation
- variables partagées, envoi de messages
- Gestion de processus, gestion de la mémoire, virtualisation
- Problèmes et applications

Programmation Parallèle

Pourquoi ?

Système complexe

Exigence:
Décomposition en
activités « indépendantes »

Matériel

monoprocasseur

multiprocasseur

réseaux

Plateforme

Centralisé

Pseudo-parallélisme

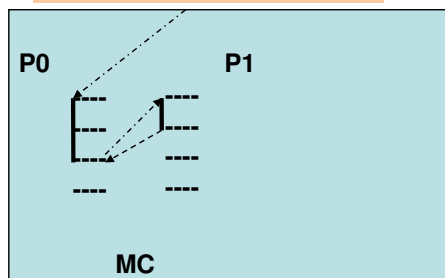
Distribué

Parallélisme réel

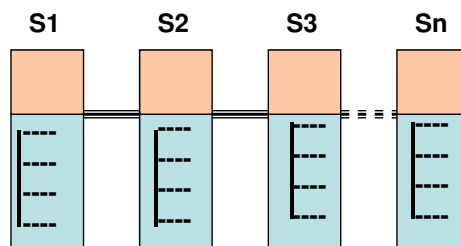
Monoprocasseur: UC

Multiprocasseur: UC1..UCn

réseaux



Passage de contrôle

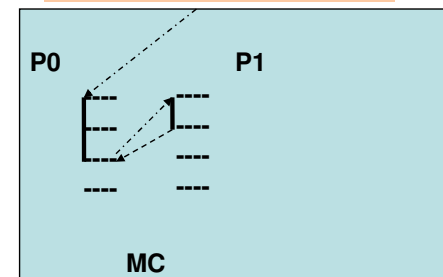


Centralisé: les programmes qui s'exécutent (processus) sont dans la même mémoire

Pseudo-parallélisme
Un ou plusieurs processeurs
partagés par les processus

Monoprocasseur: UC

Multiprocasseur: UC1..UCn



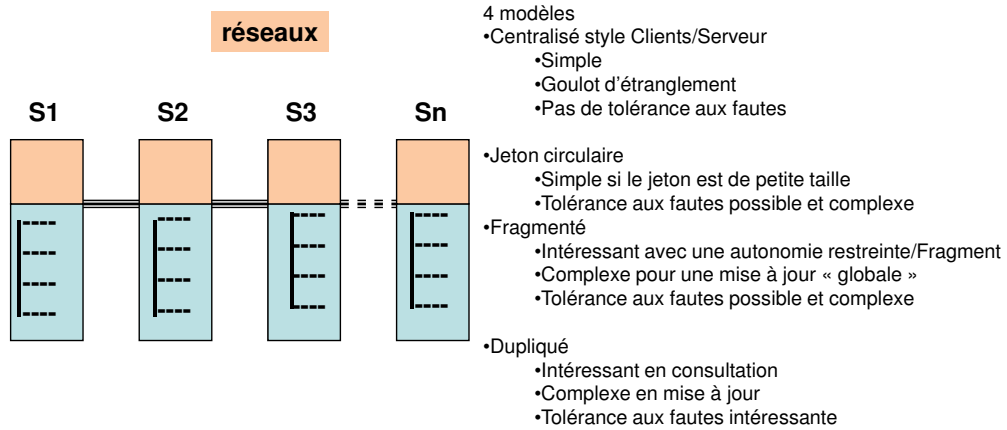
- Monoprogrammation
- Un seul processus

- Multiprogrammation

- Plusieurs processus qui partagent le processeur
- Pendant une opération d'entrée/sortie le processus
- Passe à un autre processus

- Temps partagé
 - Multiprogrammé
 - Un quantum de temps pour chaque processus à tour de rôle

Distribué Parallélisme réel



Expression du parallélisme

Processus

Définition utilisateur

Programme en cours d'exécution

Processus

Définition système

Descripteur
identité
état
pcontexte
:

Pile du processus

—
—
:
—
Ad_SDonnées
Ad_Code

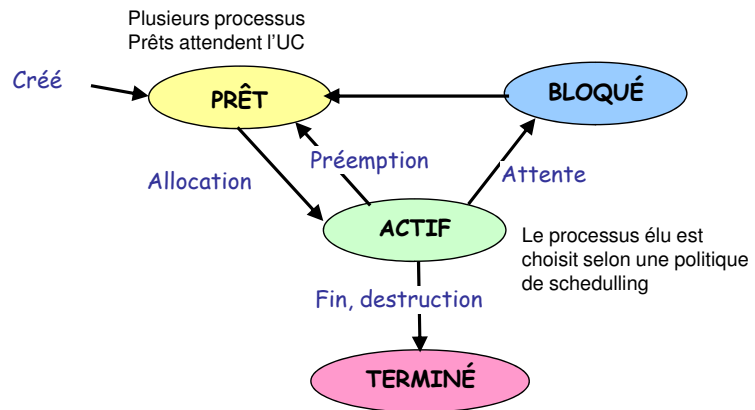
S_Données

—
—
—

Ad_Code

—
—
—

États d'un processus



Processus Unix

S_Code

```

Main()
{ int x, pid;
  x=0; pid = fork();
  if (!pid) x=1;
  else x=2;
}

```

S_Pile_Père

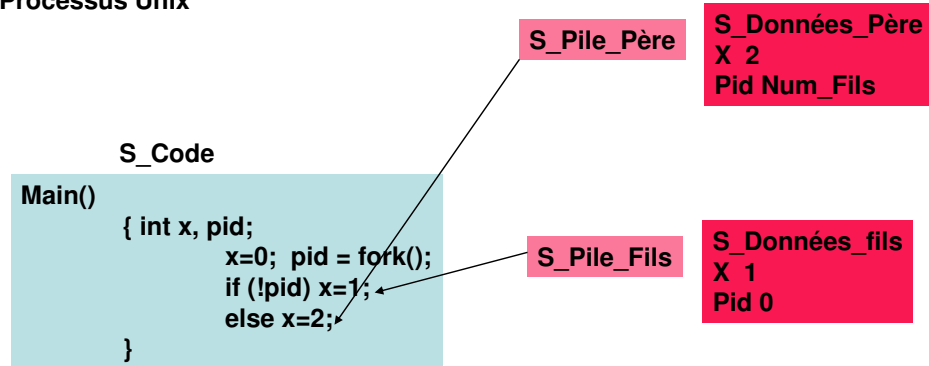
S_Données_Père
X 0
Pid Num_Fils

S_Pile_Fils

S_Données_fils
X 0
Pid 0

- Selon implantations, sous-états possibles

Processus Unix



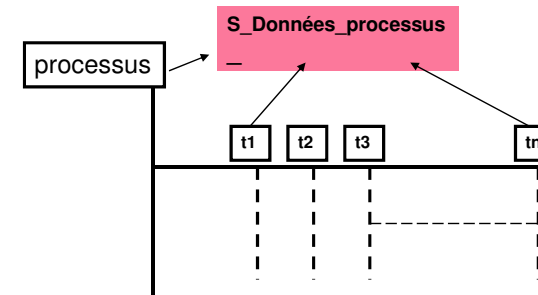
Expression du parallélisme

Thread « processus léger »

Permettre à une activité parallèle

contrairement aux processus

de partager l'espace d'adressage avec d'autres



Un thread est créé, seulement, par un processus

Un processus peut créer plusieurs threads

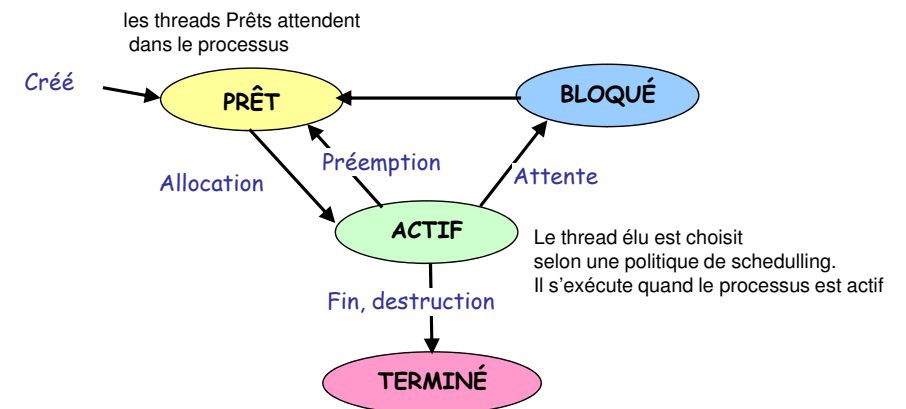
Le processus père joue le rôle du processeur / à ses threads

Les threads s'exécutent d'une façon général en multiprogrammé

- Ils partagent le temps du processus père
- Le contrôle passe à un autre thread si le thread courant:
 - se termine
 - se bloque
 - Lance une opération E/S

D'autres politiques de scheduling sont possibles

États d'un thread




- Selon implantations, sous-états possibles

S_Code

```
int x;
/*-----*/
void *traitementThread () {
/*-----*/

    int i;
    x++; printf(...) x++;
}

main()
{
    int etat;
    int numThreads;
    pthread_t idThread;
    pthread_attr_t attribut;
    /* Creation d'un thread */
    x = 1;
    etat = pthread_create(&idThread, &attribut, traitementThread,
        &numThreads);
    Printf (.....);
    X--;
}
```



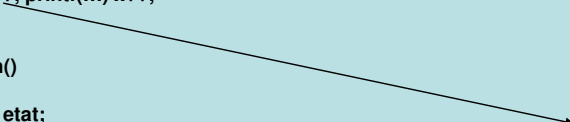
A black arrow originates from the line `x = 1;` in the `main()` function and points to a red rectangular box containing the text `X == 1`.

S_Code

```
int x;
/*-----*/
void *traitementThread () {
/*-----*/

    int i;
    x++; printf(...) x++;
}

main()
{
    int etat;
    int numThreads;
    pthread_t idThread;
    pthread_attr_t attribut;
    /* Creation d'un thread */
    x = 1;
    etat = pthread_create(&idThread, &attribut, traitementThread,
        &numThreads);
    Printf (.....);
    X--;
}
```




A black arrow originates from the line `x++; printf(...) x++;` in the `traitementThread()` function and points to a red rectangular box containing the text `X == 2`.

S_Code

```
int x;
/*-----*/
void *traitementThread () {
/*-----*/

    int i;
    x++; printf(...) x++;
}

main()
{
    int etat;
    int numThreads;
    pthread_t idThread;
    pthread_attr_t attribut;
    /* Creation d'un thread */
    x = 1;
    etat = pthread_create(&idThread, &attribut, traitementThread,
        &numThreads);
    Printf (.....);
    X--;
}
```



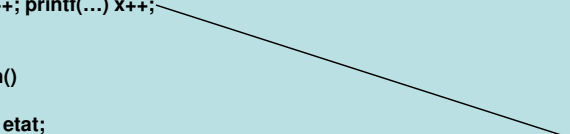
A black arrow originates from the line `x = 1;` in the `main()` function and points to a red rectangular box containing the text `X == 1`.

S_Code

```
int x;
/*-----*/
void *traitementThread () {
/*-----*/

    int i;
    x++; printf(...) x++;
}

main()
{
    int etat;
    int numThreads;
    pthread_t idThread;
    pthread_attr_t attribut;
    /* Creation d'un thread */
    x = 1;
    etat = pthread_create(&idThread, &attribut, traitementThread,
        &numThreads);
    Printf (.....);
    X--;
}
```



A black arrow originates from the line `x++; printf(...) x++;` in the `traitementThread()` function and points to a red rectangular box containing the text `X == 2`.

Exemple: Thread Unix

Créer un thread

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

- start_routine = fonction exécutée par le thread
- arg = argument de cette fonction
- attr = attributs optionnels de création
- thread = identificateur
- Toutes les ressources nécessaires au thread doivent avoir été initialisées.
- Erreurs possibles :
 - EINVAL : attributs invalide
 - EAGAIN : ressources insuffisantes

Autres opérations sur un thread

```
int pthread_detach(pthread_t thread);
```

- Détacher un thread
- Erreurs :
 - (EINVAL : thread non « joignable »
 - ESRCH : thread invalide)

```
pthread_t pthread_self(void);
```

- Retourne l'identificateur de l'appelant
- int pthread_yield(void);
- <sched.h>, TR
- Rend prêt l'appelant, élection nouveau thread
- Erreur :
 - Retour -1 + errno
 - ENOSYS : non supporté

Terminer un thread

```
void pthread_exit(void *value_ptr);
```

- Terminaison de l'appelant
 - value_ptr = valeur (non adr) de retour pour jointure

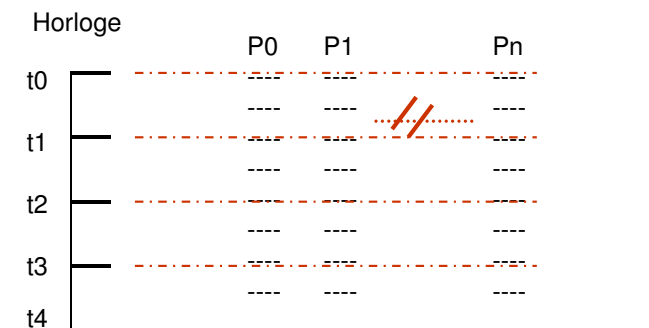
```
int pthread_join(pthread_t thread, void **value_ptr);
```

- Attente de la terminaison d'un thread non détaché
 - thread = identificateur du thread concerné
 - value_ptr = valeur retournée (si non NULL)
- Erreurs :
 - EINVAL : thread non « joignable »
 - ESRCH : thread invalide
 - (EDEADLK : join avec self)
- Exemple : create, exit/join

2 Modèles du parallélisme

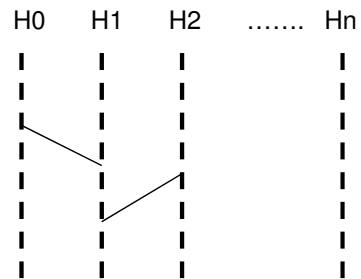
I- Modèle Synchrone

- Une seule horloge
- A chaque top tous les processus s'activent
- exécution // entre 2 top
- interaction 0



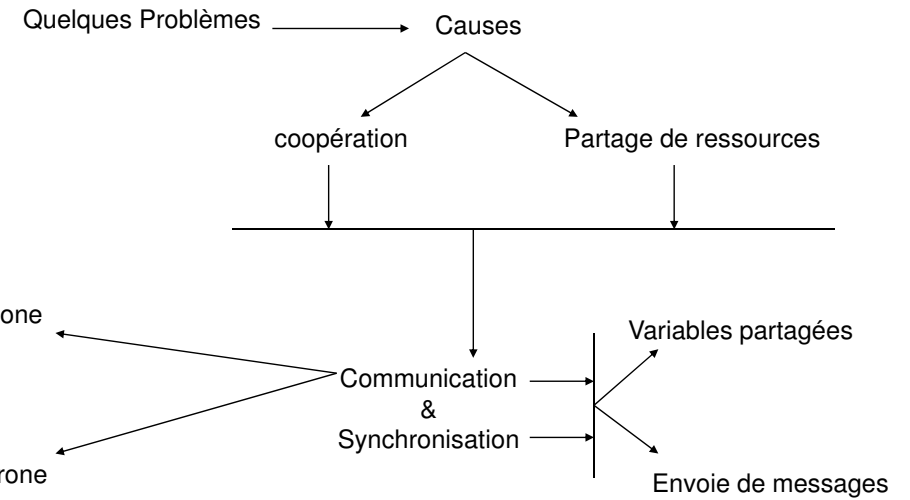
2 Modèles du parallélisme

II- Modèle Asynchrone



• Indépendance

• interaction # 0



Quelques Problèmes

Propriétés Comportementales

Section Critique

Communication Synchrones

Communication Asynchrone

Attendre & Signaler un événement

Diffuser un événement

Allocation de Ressources

Fichier Partagé

Pilote d'Entrée /Sortie

.....

Sûreté

Exclusion Mutuelle

Non Inter blocage

Vivacité

Non Famine

Équité