



Systeme d'exploitation avancé

Thread POSIX

Pierre LEROY – leroy.pierre1@gmail.com

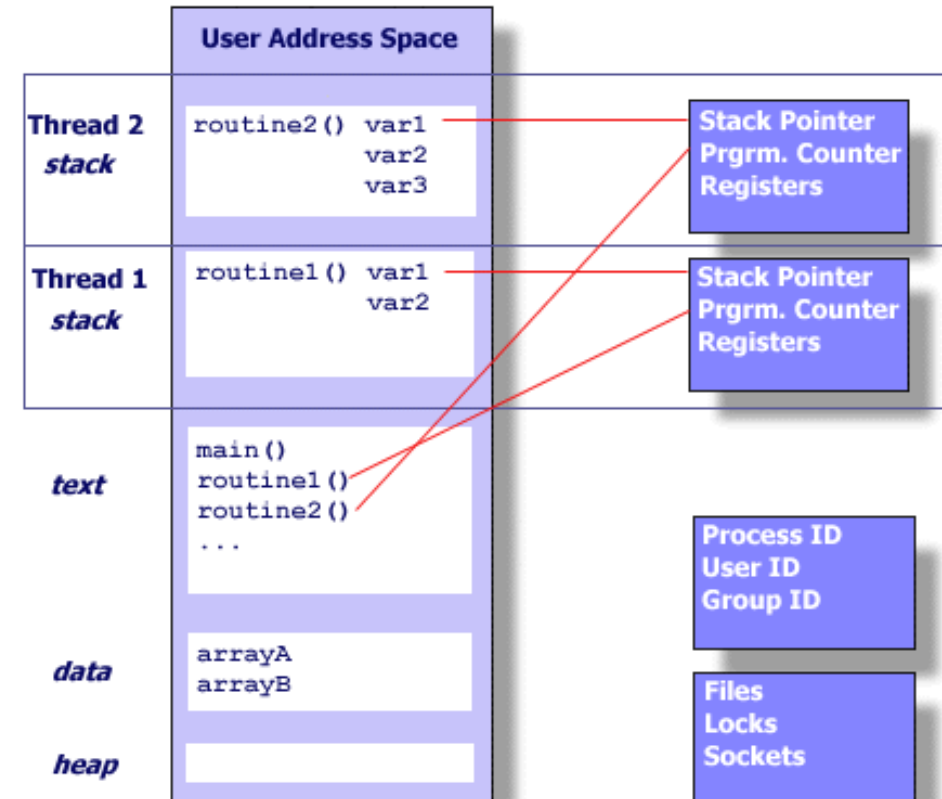
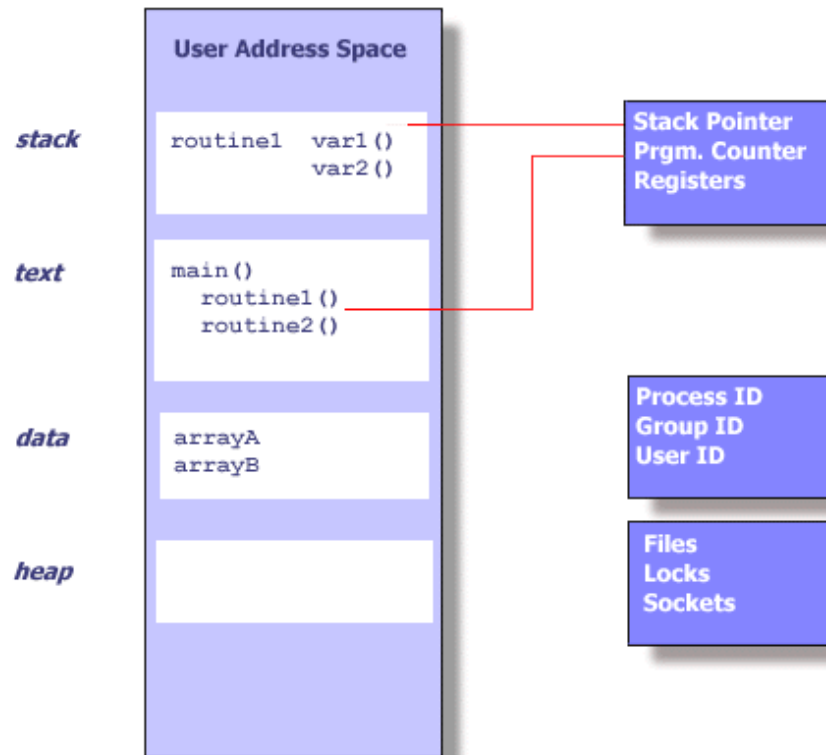
Sommaire

- I. Principes
 - II. Mise en œuvre
 - III. Essentiel
 - IV. Conclusion
-

Principes

- « Processus léger » :

NOTIONS

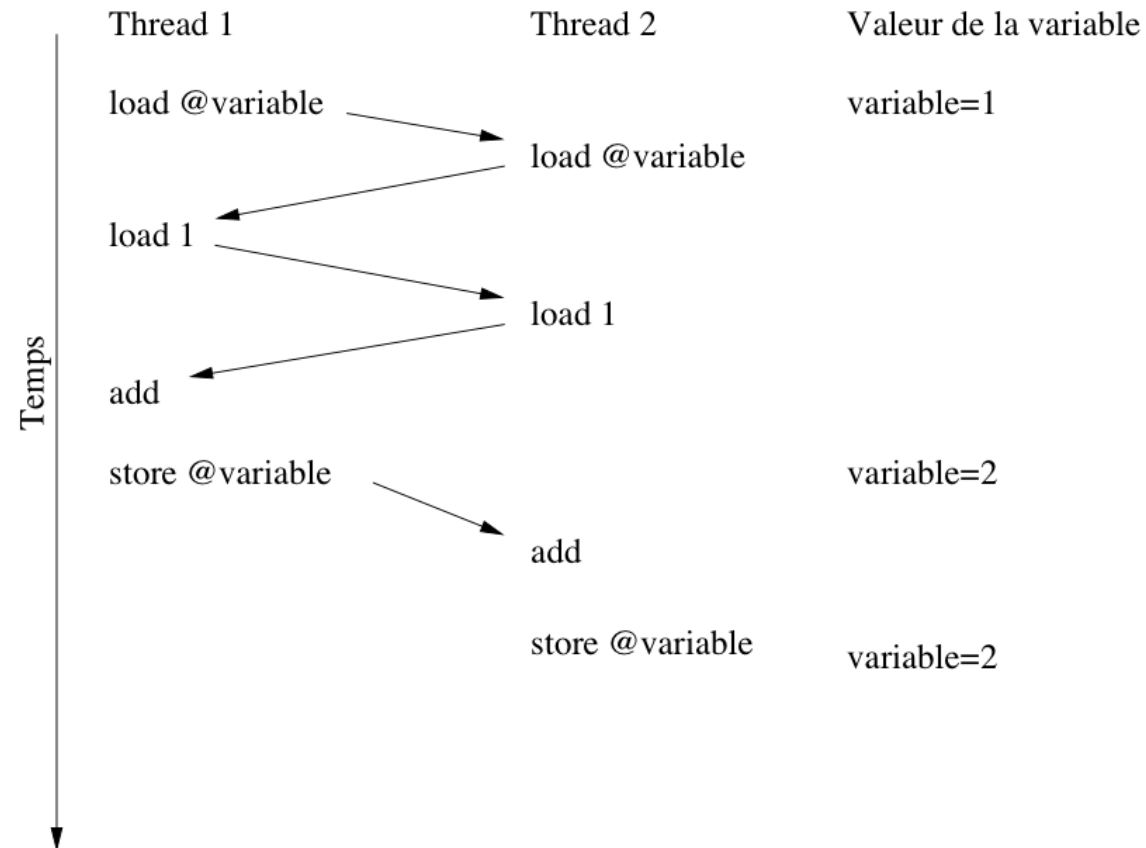


2 THREADS

Principes

- Concurrence et section critique :

NOTIONS



Sommaire

I. Principes

II. Mise en œuvre

III. Essentiel

IV. Conclusion

Implémentation

- Manipulations usuelles concernant les threads :

MISE EN OEUVRE

- La mise œuvre des thread ***posix*** s'implémente via les procédures **pthread_xxx** :
 - ✓ Les actions usuelles sont : **création** / **attente** / **terminaison**
 - ✓ La gestion des accès concurrents : via **mutex**

Implémentation

- Création d'un thread :

MISE EN OEUVRE

- Appel système ***pthread_create*** :

```
int pthread_create (  
    pthread_t *tid, pthread_attr_t *attr,  
    void* (*func)(void*), void* arg );
```

- ✓ **tid** : identifiant du thread créé.
- ✓ **attributs** :
 - voir man ***pthread_attr_init*** ; ie : joignable/détache ; si **NULL** attributs par défaut
- ✓ **func** : fonction à appeler ; code exécuté par le thread.
- ✓ **arg** : argument de la fonction appelée;
 - ie: pointeur, données via structure

Implémentation

- Gestion de la vie du thread :

MISE EN OEUVRE

- Appel système ***pthread_exit***:

- ✓ Fin d'exécution d'un thread

```
pthread_exit(void *retval);
```

↑
Pointeur sur code retour du thread

- Appel système ***pthread_cancel***:

- ✓ Demande à un thread de terminer son exécution (similaire à ***kill***)

```
pthread_cancel(pthread_t thread);
```

↑
Identifiant du thread

Implémentation

- Gestion de la vie du thread :

MISE EN OEUVRE

- Appel système ***pthread_join***:
 - ✓ Attente de la fin d'exécution d'un thread (similaire à ***wait***)

```
pthread_join(pthread_t thread, void **retval);
```

↑
Identifiant du thread

↑
sur code retour du thread

Implémentation

- Terminaison d'un thread :

MISE EN OEUVRE

- Chaque thread définit sa réaction aux demandes d'arrêt selon deux paramètres :
 - ✓ **cancel_state** : *accepte-t-il les demandes d'arrêt ?*
 - ✓ **cancel_type** : *quand exactement s'arrêtera-t-il ?*
- Défini par les fonctions suivantes :
 - ✓ **pthread_setcancelstate**
 - ✓ **pthread_setcanceltype**

Implémentation

- Acceptation des demandes d'arrêt :

MISE EN OEUVRE

- Appel système ***pthread_setcancelstate*** :
 - ✓ Attente de la fin d'exécution d'un thread (similaire à ***wait***)

```
int pthread_setcancelstate (int new, int *old);
```

Nouvelle valeur souhaitée

Précédente valeur si != NULL

- ✓ **PTHREAD_CANCEL_ENABLE** : (par défaut) le thread accepte les demandes d'arrêt
- ✓ **PTHREAD_CANCEL_DISABLE** : le thread refuse les demandes d'arrêt (elles sont alors simplement ignorées)

Implémentation

- Temporalité des demandes d'arrêt :

MISE EN OEUVRE

- Appel système ***pthread_setcanceltype*** :
 - ✓ Attente de la fin d'exécution d'un thread (similaire à ***wait***)

```
int pthread_setcanceltype (int new, int *old);
```

Nouvelle valeur souhaitée

Précédente valeur si != NULL

- ✓ **PTHREAD_CANCEL_ASYNCHRONOUS** : Action immédiate, lorsqu'il est sollicité
- ✓ **PTHREAD_CANCEL_DEFERRED** : (par défaut) Au prochain point d'arrêt :
fonction bloquante ou appel système

Implémentation

- Gestion des accès concurrents via ***mutex*** :

MISE EN OEUVRE

- Appel système ***pthread_mutex_init***:

✓ Création d'un mutex

```
int pthread_mutex_init (pthread_mutex_t* mutex,  
                        pthread_mutexattr_t *attr);
```

mutex à initialiser

Attributs du mutex

- Appel système ***pthread_mutex_destroy***:

✓ Destruction d'un mutex

```
int pthread_mutex_destroy (pthread_mutex_t* mutex)
```

Sémaphore à détruire

Implémentation

- Gestion des accès concurrents via ***mutex*** :

MISE EN OEUVRE

- Appel système ***pthread_mutex_lock / pthread_mutex_unlock***:
 - ✓ Positionnement des verrous

```
pthread_mutex mutex; // variable partagée
pthread_mutex_init (&mutex, NULL)
...
pthread_mutex_lock (&mutex);
// Section critique ici
pthread_mutex_unlock (&mutex);
...
```

Implémentation

- Gestion des accès concurrents via ***mutex*** :

MISE EN OEUVRE

- Appel système ***pthread_mutex_trylock***:
 - ✓ Vérification de la disponibilité sans rester bloqué dans un contexte bloquant

```
if (pthread_mutex_trylock(&mutex) == 0)
{
    // Section critique ici
}
else
{
    // Autre chose, non critique ici
}
```

Implémentation

- Gestion des accès concurrents via ***mutex*** && ***variable conditionnelle*** :

MISE EN OEUVRE

- Mutex :
 - ✓ Protection d'une section critique
 - ✓ *Synchronisation de thread*
- Séparation des rôles avec des variables conditionnelles :
 - ✓ Mutex : Section critique
 - ✓ *Variable conditionnelle : synchronisation*
- Type : **pthread_cond_t cond ;**
- Fonctions :
 - ✓ **pthread_cond_init**
 - ✓ **pthread_cond_signal**
 - ✓ **pthread_cond_wait**

Thread A

```
pthread_mutex_lock(&mutex);
// traitement
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);
```

Thread B

```
pthread_mutex_lock(&mutex);
pthread_cond_wait(&cond, &mutex);
// traitement
pthread_mutex_unlock(&mutex);
```


Implémentation

- Gestion des accès concurrents via *sémaphore* :

MISE EN OEUVRE

- Sémaphore (généralisation du *mutex*) :
 - ✓ Protection d'une section critique
 - ✓ *Synchronisation de thread*
- Appels systèmes **sem_XXX** (possédant des usages spécifiques) :
 - ✓ Initialisation : **sem_init**
 - ✓ Consultation : **sem_getvalue**
 - ✓ Destruction : **sem_destroy**
 - ✓ Usage : P / V : **sem_wait** / **sem_post**

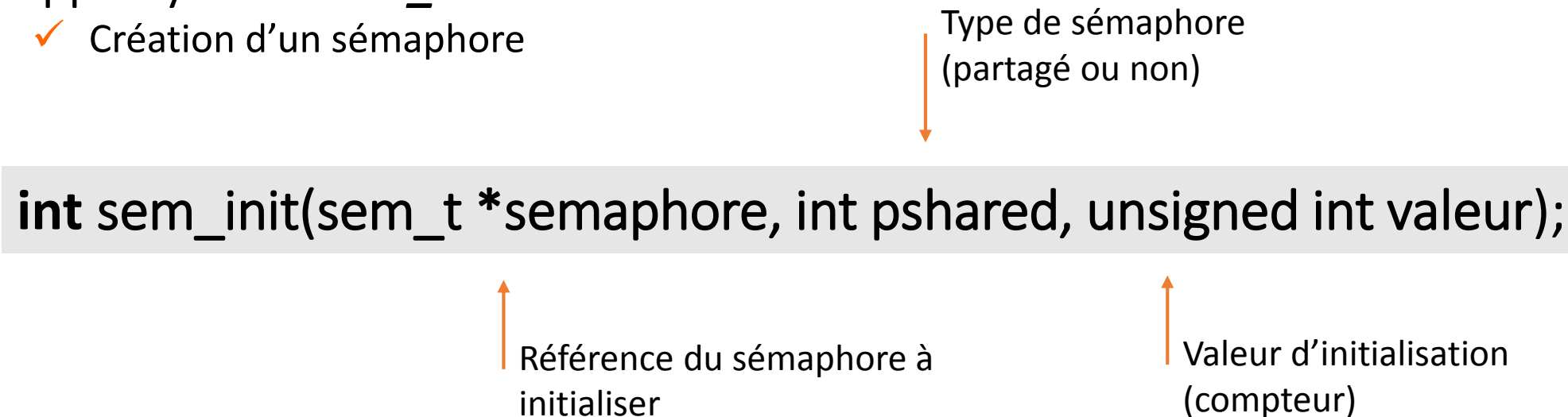
Implémentation

- Gestion des accès concurrents via ***sémaphore*** :

MISE EN OEUVRE

- Appel système ***sem_init***:
 - ✓ Création d'un sémaphore

Type de sémaphore
(partagé ou non)



```
int sem_init(sem_t *semaphore, int pshared, unsigned int valeur);
```

Référence du sémaphore à
initialiser

Valeur d'initialisation
(compteur)

Implémentation

- Gestion des accès concurrents via ***sémaphore*** :

MISE EN OEUVRE

- Appel système ***sem_getvalue***:

- ✓ Consultation de la valeur d'un sémaphore

Valeur courante du compteur



```
int sem_getvalue(sem_t *semaphore, int * sval);
```



Référence du sémaphore à utiliser

Implémentation

- Gestion des accès concurrents via *sémaphore* :

MISE EN OEUVRE

- Appel système *sem_destroy*:
 - ✓ Destruction d'un sémaphore

```
int sem_destroy(sem_t *semaphore);
```



Référence du sémaphore à utiliser

Implémentation

- Gestion des accès concurrents via ***sémaphore*** :

MISE EN OEUVRE

- Appel système ***sem_wait***:
 - ✓ Opération « P » tentative de récupération d'un jeton

```
int sem_wait(sem_t *semaphore);
```



Référence du sémaphore à utiliser

Implémentation

- Gestion des accès concurrents via ***sémaphore*** :

MISE EN OEUVRE

- Appel système ***sem_pwait***:
 - ✓ Opération « P » tentative de récupération d'un jeton version non bloquante

```
int sem_trywait(sem_t *semaphore);
```

↑
Référence du sémaphore à utiliser

Implémentation

- Gestion des accès concurrents via ***sémaphore*** :

MISE EN OEUVRE

- Appel système ***sem_post***:
 - ✓ Opération « V » remise à disposition d'un jeton

```
int sem_post(sem_t *semaphore);
```

↑
Référence du sémaphore à utiliser

Cas d'usage

- Gestion des accès concurrents via *sémaphore* :

MISE EN OEUVRE

- Problématique Producteur / Consommateur :
 - ✓ Assurer que les N consommateurs consomment tant qu'il y a des produits disponibles.
 - ✓ Assurer que le producteur ne tente pas de remplir une file pleine.

Producteur

- ✓ créer un document D
- ✓ verrouiller F
- ✓ ajouter D à la fin de la file F
- ✓ déverrouiller F
- ✓ envoyer un signal au consommateur

Consommateur

- ✓ répéter
 - ✓ attendre un signal de F
 - ✓ tant que F n'est pas vide
 - ✓ pour chaque élément E de F
 - ✓ verrouiller F
 - ✓ imprimer E
 - ✓ supprimer E de F
 - ✓ déverrouiller F
 - ✓ fin pour
- ✓ fin tant que
- ✓ attendre un signal d'un producteur
- ✓ fin répéter

Sommaire

I. Principes

II. Mise en œuvre

III. Essentiel

IV. Conclusion

Essentiel

- Toutes les notions abordées dans ce chapitre sont fondamentales



Conclusion



Annexes

Annexes

- Liens annexes :

- ✓ IBM – « POSIX threads explained » : <http://www.ibm.com/developerworks/library/l-posix1/l-posix1-pdf.pdf>
- ✓ Producteur/Consommateur : https://fr.wikipedia.org/wiki/Probl%C3%A8me_des_producteurs_et_des_consommateurs
- ✓ Lecteur/Ecrivain : https://fr.wikipedia.org/wiki/Probl%C3%A8me_des_lecteurs_et_des_r%C3%A9dacteurs