

Programmation système – UNIX

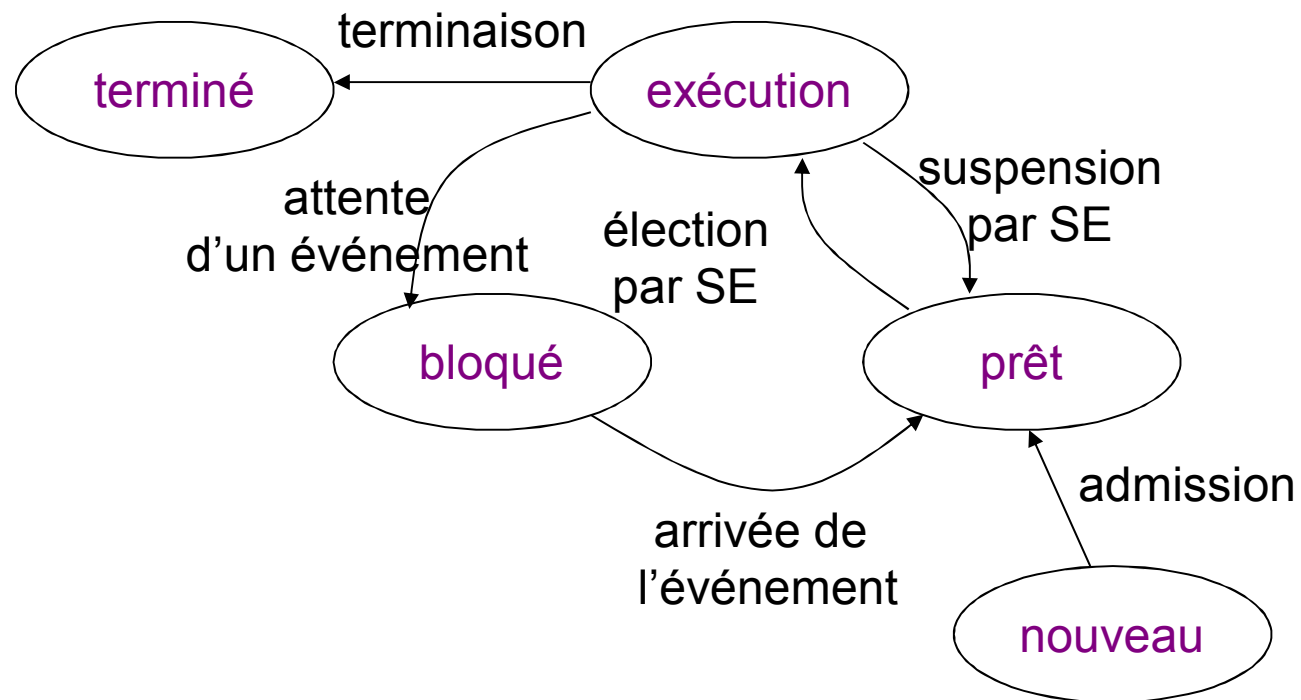
V. Felea / vafelea at femto-st dot fr /
411C

- communication inter-processus par mémoire partagée
- notions de synchronisation
- communication inter-thread dans la programmation multithreadée

Rappel sur les processus

- *définition*
 - entité dynamique représentant l'exécution d'un programme sur un processeur
 - possède un espace d'adressage
 - code exécutable
 - données statiques et dynamiques
 - pile d'exécution + tas
- processus versus programme
 - le **programme** : une *description statique*
 - le **processus** : une *activité dynamique*
 - il a un début, un déroulement et une fin
 - il a un état qui évolue au cours du temps

Etats des processus/Transitions



Rappel sur la communication entre processus locaux (inter-processus)

- échange de données
 - fichiers (lecture/écriture)
- synchronisation
 - signaux
- échange de données et synchronisation
 - file d'attente de messages
 - tubes anonymes (processus d'une même filiation), nommés (processus non forcément liés par filiation)

Communication locale inter-processus

- notion de mémoire partagée

Mémoire partagée

- IPC : Inter-Process Communication
- objectif : donner accès aux informations d'un processus à d'autres processus (communication explicite) s'exécutant sur une même machine (communication locale)
- moyen
 - segment de mémoire créé par le processus qui détient l'information
 - segment de mémoire attaché par les processus qui souhaitent accéder aux informations de l'espace partagé
- contrainte : aucun mécanisme de synchronisation

IPC

- files de messages
- mémoire partagée
- sémaphores

- System V / SysV (1983)

- une des premières versions majeures d'Unix
- base pour l'élaboration du standard POSIX

- norme POSIX (1988) = Portable Operating System Interface (définie par IEEE)



POSIX

Procédure d'utilisation d'un segment de mémoire partagée – System V

- créer une clé (sert d'identifiant pour le segment de mémoire)
- créer segment de mémoire partagée (utilisant la clé) = « allouer » de la mémoire
- attacher le segment (utilisant l'adresse du segment partagé) à l'espace d'adressage du processus
- lire/écrire dans la mémoire partagée
- détacher le segment de mémoire partagée
- supprimer (libérer) le segment de mémoire partagée

Bibliothèques

- [sys/types.h](#)
- [sys/ipc.h](#)
- **sys/shm.h**
- `man cmd`

Création de clé - méthodes

- la créer explicitement

```
key_t someKey;
```

```
someKey = 1234;
```

- fonction **ftok**
- demander au système d'en fournir une à la création du segment

Création de clé

- **key_t ftok(char *pathname, int proj_id);**
(bibliothèques : sys/types.h, sys/ipc.h)
- *pathname* : l'identité du fichier (doit exister et être accessible)
- *proj_id* : entier (seuls les 8 bits de poids faible sont utilisés, qui ne doivent pas être nuls)
- (créer une clé IPC System V de type key_t)
- valeur retournée : la même pour tous les chemins d'accès identifiant le même fichier, en utilisant une valeur identique pour *proj_id*
- valeur différente lorsque des fichiers différents (existant simultanément), ou des identificateurs de projet différents sont employés

Création de segment (1)

- **int shmget(key_t key, size_t size, int flag)**
(bibliothèques : sys/shm.h, sys/ipc.h)
- *key* : entier, identifiant du segment mémoire / IPC_PRIVATE
- *size* : nombre d'octets du segment de mémoire
- *flag* : IPC_CREAT/IPC_EXCL **ou sur bits** avec
 - drapeau de permissions (similaires à celles utilisées pour la création de fichiers sous UNIX)
 - (les 9 bits de poids faible) indiquant les permissions pour le propriétaire, le groupe et les autres

Création de segment (2)

- sémantique : créer un nouveau segment de mémoire partagée ou renvoyer identifiant associé avec la clé *key* pour un segment de mémoire existant
- nouveau segment de mémoire partagée si
 - soit *key* = IPC_PRIVATE
 - soit *key* n'a pas d'identifiant de segment de mémoire partagée associé et IPC_CREAT est précisé dans dernier paramètre
- valeur retournée : descripteur du segment de mémoire partagée associée à la clé *key*

Exemple (1)

```
struct Data { int val; char opt; };
```

```
int shmID;
```

```
key_t key = ftok("./" , 'h');
```

```
shmID = shmget(
```

```
    key,                                /* la clé */
```

```
    sizeof(struct Data), /* taille sg mém part */
```

```
    IPC_CREAT | 0644); /* cr & rw (u) r(g,o) */
```



valeur en octal

Exemple (2)

```
struct Data { int val; char opt; };
```

```
int shmID;
```

```
shmID = shmget(  
    IPC_PRIVATE,          /* clé privée */  
    sizeof(struct Data), /* taille */  
    IPC_CREAT | 0644); /* cr & droits */
```

- IPC_CREAT ignoré car IPC_PRIVATE utilisé

Attachement d'un segment


```
void* shmat (int shm_id, const void* shm_addr,  
             int flag);
```

(bibliothèques : sys/shm.h, sys/types.h)

- *shm_id* : descripteur du segment de mémoire (obtenu par **shmget**)
- *shm_addr* : adresse à laquelle la mémoire partagée doit être attachée au processus courant (NULL – permet au SE de choisir l'adresse à laquelle la mémoire est attachée)
- *flag* : options (SHM_RDONLY, SHM_REMAP,...) – défaut (0) :
rw
- valeur retournée : un pointeur sur le premier octet de la mémoire partagée si succès, valeur négative sinon : (void*)-1

Exemple

```
key = ftok("./", 'h');  
shmID = shmget (key, sizeof(struct Data),  
                IPC_CREAT | 0644);  
p = (struct Data *) shmat(shmID, NULL, 0);  
if ((void*) p == -1) {  
    printf("erreur shmat()\n"); exit(1);  
}  
p->val = 1; p->opt = '.';
```



Détachement d'un segment

int shmdt(const void* *shm_addr*);
(bibliothèques : sys/shm.h, sys/types.h)

- *shm_addr* : adresse (dans l'espace d'adressage du processus appelant) du segment de mémoire partagée à détacher (retournée par **shmat**)
- attention : pas de suppression du segment de mémoire partagée
- valeur retournée : 0 si succès, -1 en cas d'erreur

Contrôle de la mémoire partagée

```
int shmctl(int shm_id, int command, struct shmid_ds* buf);
```

```
struct shmid_ds {  
    struct ipc_perm shm_perm ;  
    size_t shm_segsz ; ...  
    pid_t shm_cpid ;  
    shmatt_t shm_nattach; ...  
}  
  
struct ipc_perm {  
    key_t __key ;  
    uid_t uid ;  
    uid_t gid ; ...  
}
```

(bibliothèques : sys/shm.h, sys/ipc.h)

- *shm_id* : descripteur du segment de mémoire (fourni par **shmget**)
- *command* : action à réaliser (IPC_RMID : marquer le segment pour suppression, IPC_SET : appliquer des permissions sur le segment, IPC_STAT : obtenir infos sur le segment - copiées en **buf*)
- *buf* : infos segment (processus créateur du segment, permissions)
- valeur retournée : 0 si succès, -1 en cas d'erreur

Exemple de communication inter-processus - sans filiation (1)

en connaissant
id segment

processus à l'origine
de la création du sg mém part

processus
accédant au sg mém part

- (créer **clé**)
 - créer segment de mémoire partagée, utilisant (clé + IPC_CREAT) / (IPC_PRIVATE)
 - attacher le segment (utilisant l'adresse du segment partagé)
 - écrire dans/lire depuis la mémoire partagée
 - détacher le segment de mémoire partagée
 - supprimer le segment de mémoire partagée
- attacher le segment (utilisant l'**identifiant** du segment partagé)
 - écrire dans/lire depuis la mémoire partagée
 - détacher le segment de mémoire partagée

Exemple de communication inter-processus - sans filiation (2)

sans connaître
id segment

processus à l'origine
de la création du sg mém part

- (créer **clé**)
- créer segment de mémoire partagée en utilisant **clé + IPC_CREAT**
- attacher le segment (utilisant l'adresse du segment partagé)
- écrire dans/lire depuis la mémoire partagée
- détacher le segment de mémoire partagée
- supprimer le segment de mémoire partagée

processus
accédant au sg mém part

- (créer **clé**)
- créer segment de mémoire partagée, utilisant **clé**
- attacher le segment (utilisant l'identifiant du segment partagé)
- écrire dans/lire depuis la mémoire partagée
- détacher le segment de mémoire partagée

Exemple de communication interprocessus – relation père/fils

processus père

- créer segment de mémoire partagée, en utilisant une (clé + IPC_CREAT) / (IPC_PRIVATE)
- **attacher le segment**
- (lire depuis /) écrire dans la mémoire

créer fils
(fork)

processus fils

- **attacher le segment**
- lire depuis (/ écrire) dans la mémoire
- détacher le segment de mémoire partagée

attendre
termination
fils (wait)

- **détacher le segment de mémoire partagée**
- supprimer le segment de mémoire partagée

Mémoire partagée POSIX/System V - similarités

- mémoire partagée attachée au processus –
comme si elle fait partie de l'espace
d'adressage du processus
 - /proc/PID/maps
- outils associés de synchronisation

Mémoire partagée POSIX/System V - différences

- moyen d'identifier et référencer l'espace mémoire partagé
 - System V : un mécanisme de clés et descripteurs (non compatible avec le modèle standard I/O d'Unix) et appels système spécifiques et commandes
 - POSIX : noms et descripteurs de fichiers (les espaces mémoire partagés peuvent être analysés et manipulés en utilisant les appels système existants)
- taille des espaces mémoire partagée : fixe (System V), ajustable (POSIX)
- portabilité
 - System V : tous les noyaux Linux
 - POSIX IPC : depuis la version 2.6 du noyau

Problème accès mémoire partagée

- cohérence des données
- solution : synchronisation
- cas relation père-fils : wait – solution partielle
- cas général :
 - attente active : tantque (non dispo) ;
 - attente passive : outils spécifiques de synchronisation d'un espace partagé (e.g. sémaphores)

Commandes shell connexes

- si pas de suppression des segments mémoire partagée (**shmctl**) → fuite mémoire
- **ipcs** : liste l'ensemble des entités IPC disponibles
 - option **-m** : information sur les segments de mémoire partagée
 - option **-i id** : informations sur la ressource de descripteur **id**
- **ipcrm** : supprimer une entité IPC connue par son descripteur ou sa clé
 - option **-m id** : segment de mémoire partagée donné par son descripteur
 - option **-M clé** : segment de mémoire partagée donné par sa clé
- **lsuf** : lister les fichiers ouverts ainsi que leurs programmes ou utilisateurs liés
 - un segment de mémoire partagée est un fichier
 - champ **NODE** : identifiant segment (pour un segment de mémoire partagé)

Synchronisation

- gestion des accès à des informations communes (mémoire partagée)
- mécanisme de base : les sémaphores
 - conceptuel : sémaphore Dijkstra
 - implémenté : sémaphores System V (bibliothèque **sys/sem.h**) / sémaphores POSIX (bibliothèque **semaphore.h**)
 - sémaphores POSIX plus légers que les sémaphores System V

Sémaphores Dijkstra (1965)

- objet (S) contenant un entier (S.value) et une file d'attente (S.queue)
- opérations :
 - Init(S, val) – valeur initiale de l'entier ($val \geq 0$) et file vide
 - P(S) : décrémente la valeur du sémaphore et laisse le processus continuer son exécution si valeur ≥ 0 ; sinon, le processus bloque
 - V(S) : incrémente la valeur du sémaphore et s'il y a des processus en attente, en débloquent un

Sémaphores POSIX

- objet permettant la synchronisation entre processus (locaux) ou entre threads d'un même processus
- bibliothèque : *semaphore.h*
- fonctions : **sem_init**, **sem_destroy**, **sem_wait** (**sem_trywait**), **sem_post**
- Compilation : option -pthread (bibliothèque pthreads)
 - gcc -pthread ...

Sémaphores POSIX - init

int sem_init(sem_t **sem*, int *pshared*, unsigned int *value*);

- initialise le sémaphore à l'adresse *sem* à *value* (équivalent de Init(S,value) de Dijkstra) – opération unique
- *sem* : adresse du sémaphore
- *pshared*
 - = 0 si sémaphore partagé par les threads du processus et doit être placé à une adresse visible par tous les threads
 - != 0 si sémaphore partagé par les processus et doit être placé dans un espace mémoire partagé par les processus

Sémaphores POSIX - wait

- **int sem_wait(sem_t* sem);**
- *sem* : adresse du sémaphore
- essaie de décrémenter la valeur du sémaphore (équivalent à **P** de Dijkstra)
- si valeur > 0 alors décrémentation possible et la fonction rend la main
- sinon (valeur courante nulle) l'appel bloque jusqu'à ce qu'il soit possible de réaliser la décrémentation (valeur > 0)

Sémaphores POSIX - trywait

- **int sem_trywait(sem_t* *sem*);**
- *sem* : adresse du sémaphore
- identique à **sem_wait**, sauf pour l'aspect bloquant : si la décrémentation ne peut pas être effectuée, la fonction rend la main avec une erreur (-1)

Sémaphores POSIX - post

- **int sem_post(sem_t* *sem*);**
- *sem* : adresse du sémaphore
- incrémente la valeur du sémaphore (équivalent à **V** de Dijkstra)
- si valeur > 0 alors au moins un processus/thread est bloqué – en débloquent un

Sémaphores POSIX - destroy

- **int sem_destroy(sem_t* *sem*);**
- *sem* : adresse du sémaphore
- supprime le sémaphore

Mémoire partagée – accès en exclusion mutuelle

- assurer qu'un seul processus au maximum accède à la mémoire partagée : propriété d'exclusion mutuelle sur la mémoire partagée
- sémaphore de valeur initiale 1 (***mutex*** = ***mutual exclusion***)
- Section Critique (SC)

Accès en exclusion mutuelle – sémaphore Dijkstra

- initialisation : `Init(mutex,1)`
- code symétrique pour chaque processus

`P(mutex)`

`// accès mémoire partagée`

`V(mutex)`

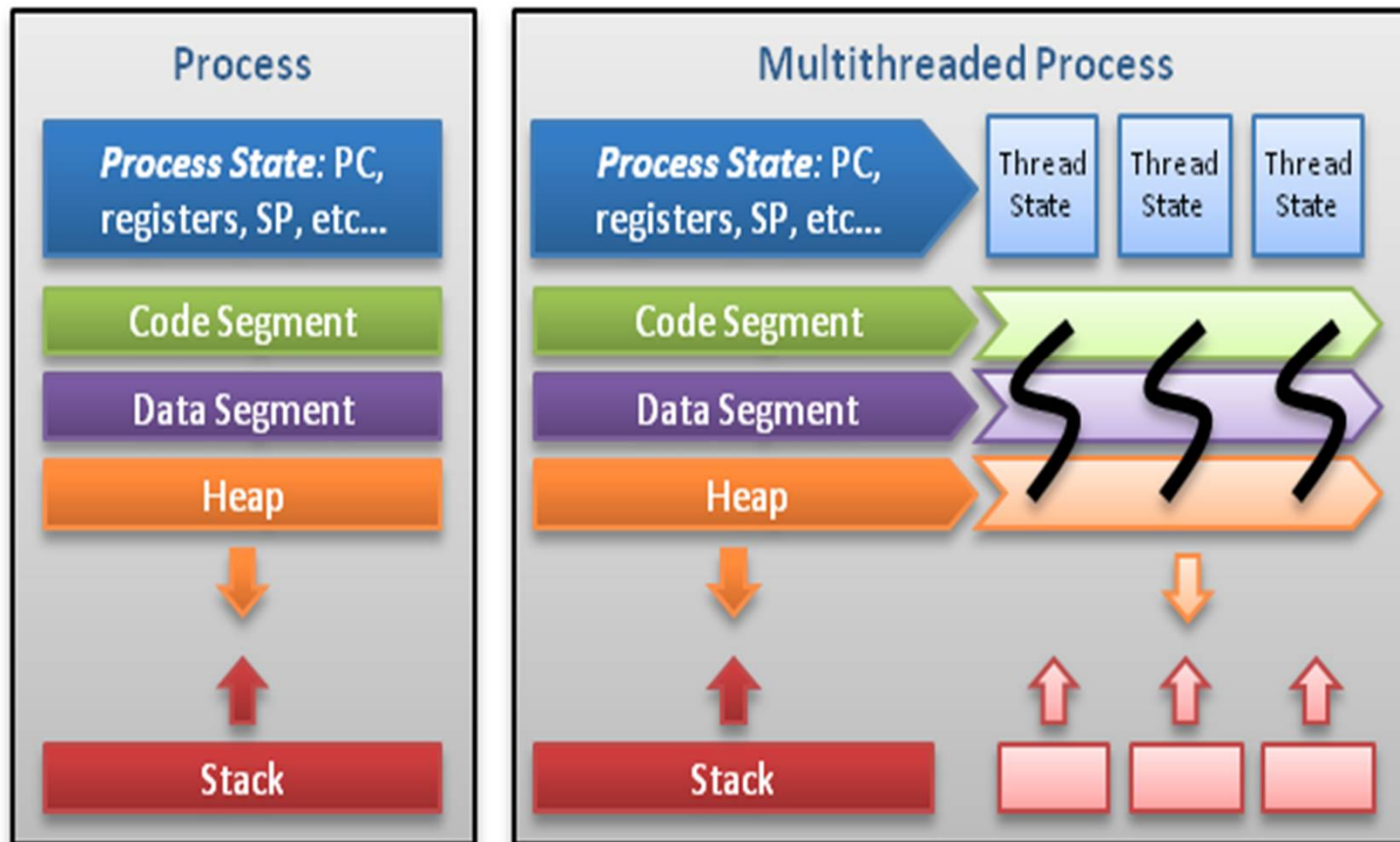
Threads

- concept
- threads POSIX
- synchronisation

Threads

- processus léger
- flux de contrôle au sein d'un processus
- les threads partagent les données d'un processus (sans mécanisme explicite, mais par passage de paramètres/variables globales)
- leur propre identifiant
- leurs propres pile et registres
- support SE (Unix, Windows) et langages (Java, Ada, C-11/C++, Pascal, Python, C#, ...)

Threads



Threads contain only necessary information, such as a stack (for local variables, function arguments, return values), a copy of the registers, program counter and any thread-specific data to allow them to be scheduled individually. Other data is shared within the process between all threads.

Threads POSIX

- Linux ne fait pas la différenciation entre threads et processus
 - le processus est constitué d'un seul thread (celui exécutant le main)
- introduction dans la norme POSIX en 1995 (POSIX 1.c)
- bibliothèque : *pthread.h*

Threads POSIX - création

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

- création et lancement d'un thread dont les instructions sont celles de la fonction *start_routine* et les paramètres, *arg* (éventuellement NULL)
- *attr* : les attributs du thread (adresse pile, taille pile, type ordonnancement) - éventuellement NULL
- l'identificateur du nouveau thread est sauvegardé à l'adresse *thr_id*

Exemple – fonction sans paramètre

```
void* fct(void* arg) {  
    printf("pid du thread fils = % d\n ", getpid());  
    while (1);  
    return NULL;  
}  
  
int main( ) {  
    pthread_t thr;  
    printf("pid de main = % d\n", getpid());  
    pthread_create(&thr, NULL, fct, NULL);  
    while (1) ;  
    return 0;  
}
```

Exemple – fonction avec paramètre

```
void* fonction (void* arg) {  
    /* argument : le pid du processus auquel le thread appartient */  
    printf("pid du thread fils = %d  
           s'exécutant dans le processus %d\n",  
           getpid(), *(int*) arg);  
    while (1);  
    return NULL;  
}  
int main (...) {  
    pthread_t thread; int pid;  
    printf("pid de main = % d\n", getpid());  
    pid = getpid();  
    pthread_create(&thread, NULL, fonction, &pid);  
    while (1) ;  
    return 0;  
}
```

Compilation

- option `-pthread`
`gcc -pthread ...`

Threads POSIX - terminaison

- **void pthread_exit(void* *valeur_de_retour*);**
- permet à un thread de terminer son exécution en retournant, à l'appelant, un état de terminaison *valeur_de_retour*
-

Threads POSIX - join

```
int pthread_join(pthread_t* thr_id,  
                 void** valeur_de_retour);
```

- attendre la terminaison du thread d'identifiant *thr_id*
- *valeur_de_retour* : si pas NULL, l'état de terminaison du thread est copié à l'adresse pointée par **valeur_de_retour*
- ...un début de synchronisation

Commandes système – threads (1)

- `ps -aux`
- codes d'état du processus (entête STAT/S)
 - D Uninterruptible sleep (usually IO)
 - R Running or runnable (on run queue)
 - S Interruptible sleep (waiting for an event to complete)
 - T Stopped, either by a job control signal or because it is being traced.
 - W paging (not valid since the 2.6.xx kernel)
 - X dead (should never be seen) Z Defunct ("zombie") process, terminated but not reaped by its parent.
- caractère supplémentaire
 - / le processus est multi-threadé

Commandes système – threads (2)

- `ps -T -p <pid>`
- `top -H`
- `/usr/bin/pstree $PID`
- `ps -eLf`

Threads - mémoire

- situation de compétition (race condition)
 - contexte : plusieurs threads partagent les mêmes données
 - désigne un cas où le comportement d'un programme dépend de l'ordre d'évènements particuliers, alors que cet ordre ne peut pas être garanti
- exemple : compteur partagé *cpt* sur lequel
 - thr1 : *cpt++*
 - thr2 : *cpt--*

Threads POSIX – synchronisation (exclusion mutuelle)

- initialiser un sémaphore (de type mutex)

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

- *mutex* est la variable à initialiser
- *attr* : attribut permettant de paramétrer le mutex (NULL → valeurs par défaut)
- détruire le sémaphore (de type mutex)

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

Threads POSIX – synchronisation (exclusion mutuelle)

- primitive P (bloquante)

**pthread_mutex_lock(pthread_mutex_t*
mutex);**

- primitive V

**pthread_mutex_unlock(pthread_mutex_t*
mutex);**

Exemple

```
pthread_mutex_t mutex ;  
pthread_mutex_init(&mutex , NULL) ;  
  
...  
pthread_mutex_lock(&mutex);  
/*****  
section critique  
*****/  
pthread_mutex_unlock(&mutex) ;  
  
...  
/* fin du programme */  
pthread_mutex_destroy(&mutex) ;
```

Threads POSIX – synchronisation avec sémaphores

- la bibliothèque des threads POSIX fournit également une implémentation des sémaphores POSIX 1003.1b
- *semaphore.h*
- rappel
 - `int sem_init(sem_t *sem, int pshared, unsigned int value);`
 - *pshared = 0*

Threads POSIX – bilan synchronisation

- join
- mutex, semaphores
- ***variables conditions***
 - attendre qu'une condition soit vérifiée dans un autre thread
 - blocage du traitement en attendant

Bilan cours

- communication locale inter-processus
 - mémoire partagée (C - System V)
- cohérence mémoire partagée
 - synchronisation (exclusion mutuelle pour la SC)
 - sémaphore (concept Dijkstra)
 - sémaphore (C - POSIX)
 - synchronisation : exclusion mutuelle
- threads (C - POSIX)
 - principe et commandes de gestion
 - synchronisation : exclusion mutuelle