

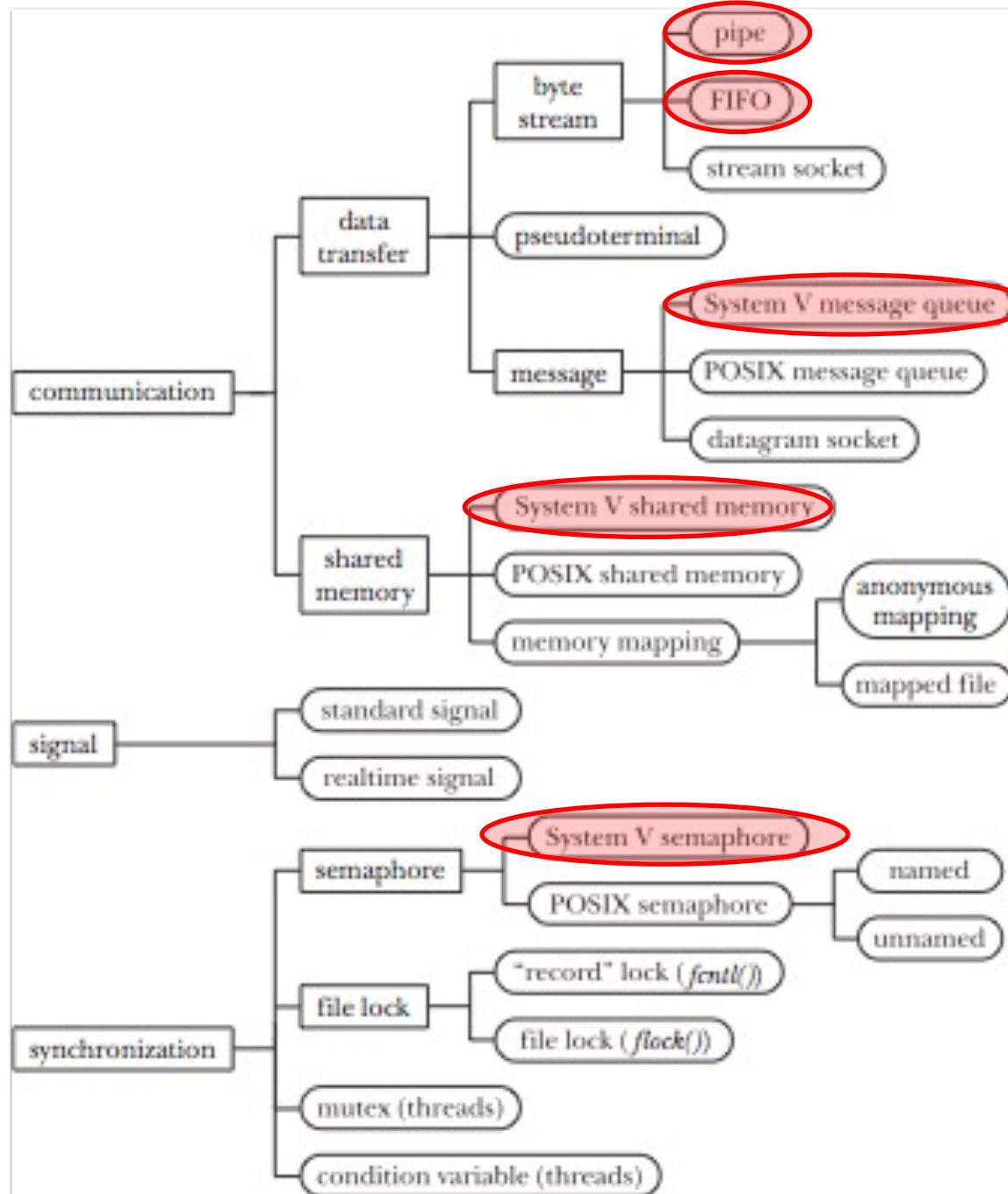
Laboratorio di sistemi operativi – T4

I semafori

Il programma

1. Introduzione a UNIX
2. Nozioni integrative del linguaggio C
3. controllo dei processi
4. pipe e fifo;
5. code di messaggi
6. memoria condivisa
- 7. semafori**
8. segnali
9. introduzione alla programmazione bash

Che cosa studieremo durante il corso?



I semafori: introduzione

A cosa servono i semafori

- diversamente dagli altri strumenti per IPC visti finora, i semafori non servono a trasferire dati fra processi.
 - sono utilizzati per permettere ai processi di sincronizzare le proprie azioni: per esempio permettono di sincronizzare l'accesso a un blocco di memoria condivisa, per impedire a un processo di accedere alla memoria condivisa mentre un altro processo la sta aggiornando

Esempio

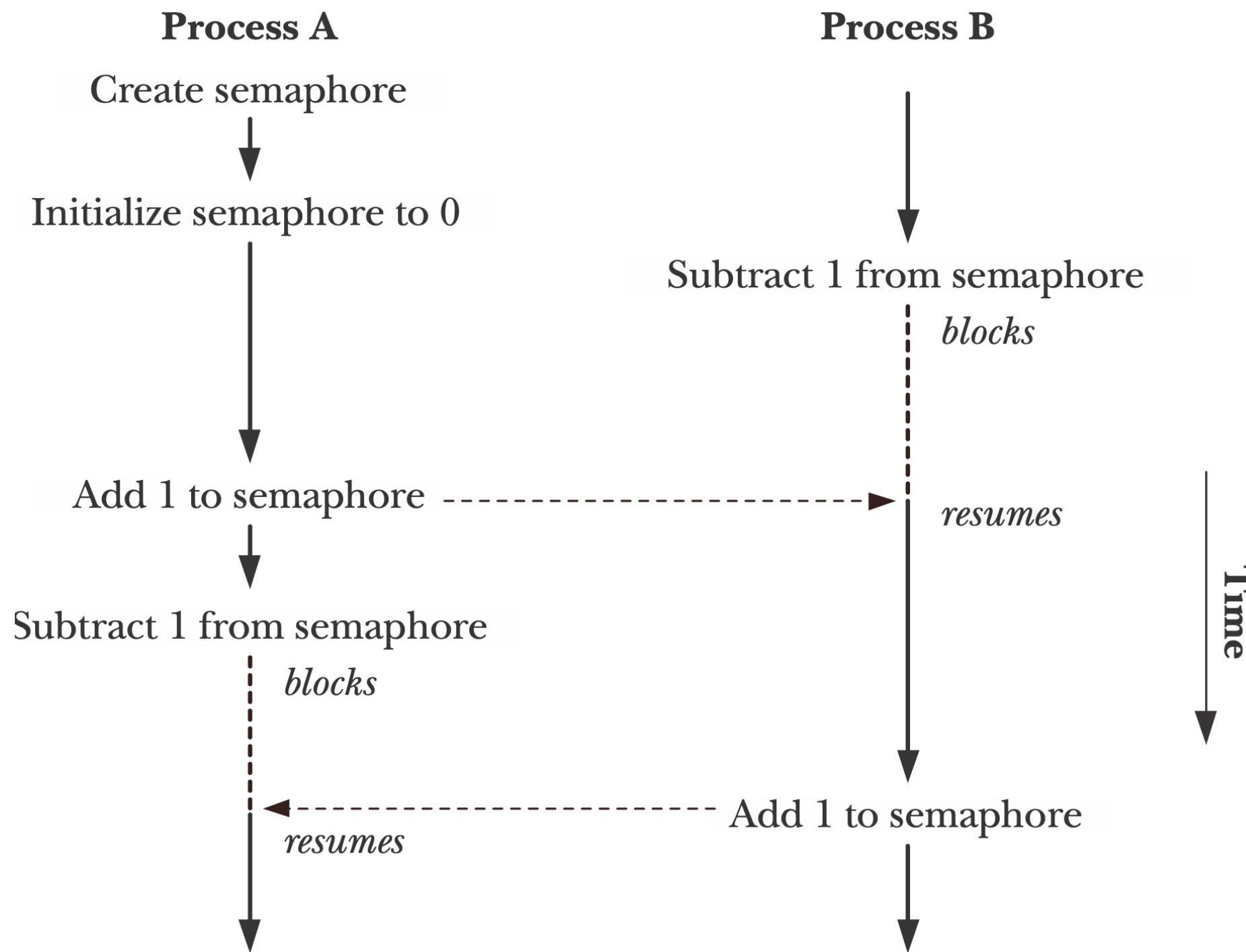
- immaginate 10 processi che vogliono incrementare di 1 il valore di un contatore (in memoria condivisa)
 - in teoria, al termine dell'esecuzione di tutti e 10 i processi, ci si aspetta che la variabile venga incrementata di 10
 - essendo però che è incerta la lettura (reading) di una variabile e la sua riscrittura (writing) da parte dei processi, questo può portare a diversi problemi sull'integrità del dato

Che cosa è un semaforo

- Un semaforo è un intero mantenuto dal kernel il cui valore è sempre ≥ 0 .

Che cosa è un semaforo

- Il valore di un semaforo s regola l'accesso alla risorsa protetta da s
- Ogni processo può compiere le seguenti azioni su un semaforo s
 - Inizializzare il valore di s ad un valore intero a (numero di accessi concorrenti alla risorsa): $v(s) = a$
 - Uso di una risorsa condivisa protetta da s
 - Se $v(s) == 0$, allora blocca il processo sino a che $v(s) > 0$
 - Decrementa $v(s)$ e usa la risorsa
 - Rilasciare una risorsa dopo il suo uso
 - Incrementare $v(s)$ (operazione non bloccante)
 - Attendere fino a che $v(s)$ non diventa
 - Punto di sincronizzazione



Tornando all'esempio di prima

- immaginate 10 processi che vogliono incrementare di 1 il valore di un contatore (in memoria condivisa)
 - se ogni processo usa un semaforo per accedere al contatore (decrementa semaforo, legge e aggiorna valore variabile e incrementa semaforo rendendo la risorsa disponibile) si eliminano possibili inconistenze
 - è il kernel ad assicurare tutto ciò!

Panoramica

header file	<sys/sem.h>
get semaphore set	semget
semaphore action	semop
data structure	semid_ds
control	semctl

Utilizzo dei semafori

- I passi per utilizzare i semafori sono:
 - Creazione o apertura di un set di semafori utilizzando la `semget()`
 - Inizializzazione dei semafori presenti nel set, usando l'operazione `SETVAL` o `SETALL` della `semctl()`
 - Esecuzione delle operazioni sui valori del semaforo, utilizzando `semop()`
 - I processi che utilizzano il semaforo tipicamente usano tali operazioni per indicare l'acquisizione e il rilascio di una risorsa condivisa.
 - Quando tutti i processi hanno terminato di usare il set di semafori, rimozione del set per mezzo dell'operazione `IPC_RMID` della `semctl()`

Set di semafori

- I semafori di System V sono resi complessi dal fatto di essere allocati in gruppi detti set di semafori.
 - Il numero di semafori in un set è specificato al momento della creazione del set, per mezzo della system call `semget()`.
 - Molto spesso si utilizza un solo semaforo alla volta, ma la system call `semop()` consente di eseguire atomicamente un gruppo di operazioni su vari semafori in uno stesso set.

Creare o agganciare un set di semafori

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
//Returns semaphore set identifier on success,
//or -1 on error
```

- L'argomento `key` è una chiave generata usando il valore `IPC_PRIVATE` o una chiave restituita da `ftok()`
- Se stiamo usando `semget()` per creare un nuovo set di semafori, allora `nsems` specifica il numero di semafori in quell'insieme, e deve essere maggiore di 0. Se invece stiamo usando la `semget()` per ottenere l'identificatore di un set esistente, `nsems` deve essere minore o uguale alla dimensione del set (o si incorrerà nell'errore `EINVAL`).
 - NB: non è possibile modificare il numero di semafori presenti in un dato set.

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
//Returns semaphore set identifier on success,
//or -1 on error
```

- L'argomento `semflg` è una maschera di bit che specifica i permessi da assegnare a un nuovo set di semafori o da verificare su un set esistente.

Constant	Octal value	Permission bit
<code>S_ISUID</code>	04000	Set-user-ID
<code>S_ISGID</code>	02000	Set-group-ID
<code>S_ISVTX</code>	01000	Sticky
<code>S_IRUSR</code>	0400	User-read
<code>S_IWUSR</code>	0200	User-write
<code>S_IXUSR</code>	0100	User-execute
<code>S_IRGRP</code>	040	Group-read
<code>S_IWGRP</code>	020	Group-write
<code>S_IXGRP</code>	010	Group-execute
<code>S_IROTH</code>	04	Other-read
<code>S_IWOTH</code>	02	Other-write
<code>S_IXOTH</code>	01	Other-execute

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
//Returns semaphore set identifier on success,
//or -1 on error
```

- Zero o più dei seguenti flag possono essere messi in OR (|) nel semflg per controllare l'operazione `semget()`:
 - `IPC_CREAT`. Se non esiste un set di semafori con la chiave specificata, crea un nuovo set.
 - `IPC_EXCL`. Se è stato specificato anche `IPC_CREAT`, e un set di semafori con la chiave specificata esiste già, restituisci un fallimento, con errore `EEXIST`

Operazioni di controllo sui semafori

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ... /* union semun arg */);
//Returns nonnegative integer on success; returns -1 on error
```

- L'argomento *semid* è l'identificatore del set di semafori sul quale l'operazione viene eseguita.
- Per le operazioni su un singolo semaforo l'argomento *semnum* identifica un semaforo all'interno del set. Per le altre operazioni questo argomento è ignorato, e possiamo lasciarlo a 0.
- L'argomento *cmd* specifica l'operazione.

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ... /* union semun arg */);
//Returns nonnegative integer on success; returns -1 on error
```

- Alcune operazioni richiedono un quarto argomento di nome *arg*.
- si tratta di una union che deve essere esplicitamente definita nei nostri programmi (a meno che non sia già presente in <sys/types.h>, come in Mac OS X).

```
union semun {
    // value for SETVAL
    int val;
    // buffer for IPC_STAT, IPC_SET
    struct semid_ds* buf;
    // array for GETALL, SETALL
    unsigned short* array;
    // Linux specific part
#if defined(__linux__)
    // buffer for IPC_INFO
    struct seminfo* _buf;
#endif
};
```

Operazioni di controllo sui semafori

```
int semctl(int semid, int semnum,  
            int cmd, ... /* union semun arg */);
```

- Le seguenti operazioni (IPC_RMID, IPC_STAT, IPC_SET) sono le stesse applicabili agli altri tipi di oggetti IPC di System V.
 - IPC_RMID. Rimuove immediatamente il set di semafori e l'associata struttura *semid_ds*. Qualsiasi processo bloccato in chiamate *semop()* in attesa su semafori è immediatamente svegliato, e *semop()* riporta l'errore EIDRM.
 - L'argomento *arg* non è richiesto.

Operazioni di controllo sui semafori

```
int semctl(int semid, int semnum,  
            int cmd, ... /* union semun arg */);
```

- IPC_STAT. Copia la struttura *semid_ds* associata con il set di semafori nel buffer puntato da *arg.buf*.
- IPC_SET. Aggiorna i membri della struttura *semid_ds* associata al set di semafori utilizzando i valori nel buffer puntato da *arg.buf*.

Leggere o inizializzare un semaforo

```
int semctl(int semid, int semnum,  
           int cmd, ... /* union semun arg */);
```

- Le seguenti operazioni prelevano o inizializzano il valore (o i valori) di un singolo semaforo o di tutti i semafori del set.
 - La copia del valore di un semaforo richiede permessi in lettura sul semaforo, mentre l'inizializzazione del valore richiede permessi in scrittura.
- GETVAL. Come risultato della chiamata, semctl() restituisce il valore del semaforo (numero) semnum nel set di semafori specificato da semid. L'argomento arg non è richiesto.
- SETVAL. Il valore del semaforo semnum nel set riferito da semid è inizializzato al valore arg.val.

Leggere o inizializzare un semaforo

```
int semctl(int semid, int semnum,  
            int cmd, ... /* union semun arg */);
```

- Le seguenti operazioni prelevano o inizializzano il valore (i valori) di un singolo semaforo o di tutti i semafori nel set.
- **GETALL**. Preleva i valori di tutti i semafori nel set riferito da *semid*, copiandoli nell'array puntato da *arg.array*. Il programmatore deve garantire che l'array sia abbastanza capiente. *semnum* è ignorato.
- **SETALL**. Inizializza tutti i semafori del set riferito da *semid*, usando i valori forniti nell'array puntato da *arg.array*. *semnum* è ignorato.

```
union semun {  
    int val;  
    struct semid_ds* buf;  
    unsigned short* array;  
#if defined(__linux__)  
    struct seminfo* _buf;  
#endif  
};
```

Ottenerne informazioni sul semaforo

- Le seguenti operazioni restituiscono (attraverso il valore restituito dalla funzione) informazioni sul semaforo numero `semnum` del set riferito da `semid`.
 - Per tutte queste operazioni, è richiesto il permesso in lettura sul set di semafori, e l'argomento `arg` non è richiesto.
- **GETPID**. Restituisce il PID dell'ultimo processo che ha eseguito una `semop()` su questo semaforo; questo è riferito come il valore `sempid`. Se nessun processo ha ancora eseguito una `semop()` su questo semaforo, restituisce 0.
- **GETNCNT**. Restituisce il numero di processi attualmente in attesa di un incremento del valore del semaforo; questo è riferito come il valore `semncnt`.
- **GETZCNT**. Restituisce il numero di processi attualmente in attesa che il valore del semaforo divenga 0; questo è riferito come valore `semzcnt`.

```
struct semid_ds {  
    /* Ownership and permissions */  
    struct ipc_perm sem_perm;  
    /* Last semop time */  
    time_t             sem_otime;  
    /* Last change time */  
    time_t             sem_ctime;  
    /* No. of semaphores in set */  
    unsigned short    sem_nsems;  
};
```

```
union semun {  
    int val;  
    struct semid_ds* buf;  
    unsigned short* array;  
#if defined(__linux__)  
    struct seminfo* __buf;  
#endif  
};
```

- Ogni set di semafori ha associata una struttura `semid_ds` data;
- I membri della struttura `semid_ds` sono implicitamente aggiornati da varie system call sul semaforo, e alcuni membri della struttura `sem_perm` possono essere aggiornati esplicitamente con l'operazione `semctl()` `IPC_SET`.

```
struct semid_ds {  
    struct ipc_perm sem_perm; /* Ownership and permissions */  
    time_t          sem_otime; /* Last semop time */  
    time_t          sem_ctime; /* Last change time */  
    unsigned short  sem_nsems; /* No. of semaphores in set */  
};
```

- **sem_perm**. Quando il set di semafori è creato, i membri di questa struttura sono inizializzati. I membri uid, gid, e mode possono essere aggiornati con l'operazione IPC_SET
- **sem_otime**. Questo membro è settato a 0 alla creazione del set di semafori, ed aggiornato all'ora corrente ad ogni semop() che va a buon fine, o quando il valore del semaforo è modificato in seguito a un'operazione SEM_UNDO
- **sem_ctime**. Questo membro è impostato all'ora corrente al momento della creazione del semaforo, e in seguito a ogni operazione IPC_SET, SETALL, o SETVAL
- **sem_nsems**. Membro inizializzato al momento della creazione del set di semafori: contiene il numero di semafori nel set

Operazioni sui semafori

Operazioni sui semafori

```
#include <sys/types.h> /* For portability */
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, unsigned int nsops);
//Returns 0 on success, or -1 on error
```

- La system call `semop()` esegue una o più operazioni sui semafori nel set identificato da *semid*.
 - l'argomento *sops* è un puntatore a un array che contiene le operazioni da eseguirsi, e
 - *nsops* è la dimensione dell'array (che deve contenere almeno un elemento).

Specifica dell'operazione con sembuf

```
struct sembuf {  
    unsigned short sem_num; // numero semaforo  
    short sem_op; // operazione da eseguire  
    short sem_flg; // flags operazione  
        // (IPC_NOWAIT and SEM_UNDO)  
};
```

- Le operazioni sono eseguite atomicamente e nell'ordine in cui compaiono nell'array. Gli elementi dell'array sops sono strutture con queste caratteristiche:
 - Il membro `sem_num` identifica, all'interno del set, il semaforo sul quale si intende effettuare l'operazione.

Specifiche dell'operazione con sembuf

```
struct sembuf {  
    unsigned short sem_num;  
    // operazione da eseguire  
    short sem_op;  
    short sem_flg;  
};
```

- Il membro `sem_op` specifica l'operazione da eseguire:
 - Se `sem_op` è maggiore di 0, il valore di `sem_op` è aggiunto al valore del semaforo.
 - Quindi altri processi in attesa di diminuire il valore del semaforo possono essere svegliati per eseguire le proprie operazioni. Il processo chiamante deve avere diritti di scrittura sul semaforo.
 - Se `sem_op` è uguale a 0, il valore del semaforo è testato per vedere se attualmente è uguale a 0. Se sì, l'operazione è completata immediatamente; diversamente, la `semop()` si blocca finché il valore del semaforo diviene 0.
 - Il chiamante deve avere permessi in scrittura sul semaforo

Specifiche dell'operazione con sembuf

```
struct sembuf {  
    unsigned short sem_num;  
    // operazione da eseguire  
    short sem_op;  
    short sem_flg;  
};
```

- Se `sem_op` è minore di 0, decrementa il valore del semaforo del valore specificato da `sem_op`.
 - Se il valore corrente del semaforo è maggiore o uguale al valore assoluto specificato da `sem_op`, l'operazione è completata immediatamente.
 - Diversamente, `semop()` si blocca finché il valore del semaforo è stato aumentato tanto da permettere che l'operazione venga eseguita (senza produrre un valore negativo).
 - Il chiamante deve avere diritti di scrittura sul semaforo.

L'interpretazione delle operazioni su un semaforo

- l'aumento del valore di un semaforo corrisponde a rendere disponibile una risorsa così che altri processi possano utilizzarla;
- la diminuzione del valore di semaforo corrisponde a riservare la risorsa per un uso esclusivo da parte di questo processo.
 - Il tentativo di ridurre il valore di un semaforo può restare bloccato se il valore del semaforo è troppo basso, cioè se qualche altro processo ha già ottenuto l'uso esclusivo per la risorsa in questione.

Cosa accade ad un processo bloccato?

- Quando una system call semop() si blocca, il processo resta bloccato finché:
 - Un altro processo modifica il valore del semaforo così che la richiesta possa procedere;
 - Un segnale interrompe la system call semop() call. In questo caso la semop() fallisce con l'errore EINTR.
 - Un altro processo cancella il semaforo identificato da semid. In questo caso, la semop() fallisce con l'errore EIDRM.

Non-blocking semop()

```
struct sembuf {  
    unsigned short sem_num;  
    // operazione da eseguire  
    short sem_op;  
    short sem_flg;  
};
```

- È possibile prevenire il blocco della semop() nell'esecuzione di un'operazione su un dato semaforo specificando il flag IPC_NOWAIT nel membro sem_flg della struttura sembuf.
 - In questo caso, invece di restare bloccata, la semop() fallisce ritornando -1 e errno contiene l'errore EAGAIN.

Operazioni su molteplici semafori

- è possibile eseguire una semop() per compiere operazioni su molteplici semafori in uno stesso set.
- questo gruppo di operazioni è eseguito atomicamente; cioè, o la semop() esegue tutte le operazioni, o si blocca fino a quando non diventa possibile eseguirle simultaneamente tutte.

Operazioni su molteplici semafori

- consideriamo un esempio:
 - l'uso di `semop()` per eseguire operazioni su tre semafori in un set.
 - Le operazioni sui semafori 0 e 2 possono non essere in grado di procedere immediatamente, a seconda dei valori correnti dei semafori.
 - Se l'operazione sul semaforo 0 non può essere eseguita immediatamente, allora nessuna delle operazioni richieste viene eseguita, e la `semop()` si blocca.
 - Se l'operazione sul semaforo 0 può essere eseguita immediatamente, ma non l'operazione sul semaforo 2, allora se è stato specificato il flag `IPC_NOWAIT` sul semaforo 2, nessuna delle operazioni è eseguita, e la `semop()` restituisce immediatamente con l'errore `EAGAIN`.

```
struct sembuf sops[3];

sops[0].sem_num = 0;
sops[0].sem_op = -1; // DECREMENTO di 1 il semaforo 0
sops[0].sem_flg = 0;

sops[1].sem_num = 1;
sops[1].sem_op = 2; // INCREMENTO di 2 il semaforo 1 */
sops[1].sem_flg = 0;

sops[2].sem_num = 2;
sops[2].sem_op = 0; // ATTESA che il semaforo 2 valga 0 */
sops[2].sem_flg = IPC_NOWAIT; /* operazione NON SOSPENSIVA: se l'operazione
non può essere effettuata, NON attendere */

if (semop(semid, sops, 3) == -1) {
    if (errno == EAGAIN) /* il semaforo 2 si sarebbe bloccato; invece grazie
a IPC_NOWAIT ha restituito EAGAIN */
        printf("Operation would have blocked\n");
else
    errExit("semop"); // si è verificato qualche altro errore
```

Semafori e segnali

- Quando un processo è bloccato in una `semop()` e riceve un segnale non mascherato
 - Il corrispondente handler è eseguito
 - La `semop()` si sblocca, ritornando `-1` e `errno` è settato a `EINTR`
- Anche se il flag `SA_RESTART` è stato precedentemente settato nell'handler del segnale dalla `sigaction()`, una `semop()` che è stata interrotta, terminerà sempre con un fallimento e `errno` settato a `EINTR`

Da non fare con i semafori!

```
sop.sem_flg = IPC_NOWAIT;  
do {  
    semop(semId, &sop, 1);  
}  
while (errno == EAGAIN);
```

```
while (semctl(semId, nsem, GETVAL) > 0)  
{  
    .....  
}
```

Semafori con soglia temporale di attesa

- E' possibile anche sbloccare un processo in attesa su un semaforo dopo un certo intervallo di tempo anche se la risorsa non è disponibile
- Allo sblocco, la `semtimedop` restituisce -1 con `errno == EAGAIN` se il timeout è scaduto

```
int semtimedop(int semid, struct sembuf *sops, size_t nsops, const
struct timespec *timeout);
```

```
struct timespec timeout;
timeout.tv_sec=5; // seconds
timeout.tv_nsec=0; // nanoseconds
```

Semafori binari

Una implementazione

Semafori binari: una implementazione

- La API per i semafori di System V semaphores è complessa,
 - perché il valore dei semafori può essere modificato di quantità arbitrarie,
 - perché i semafori sono allocati in set di semafori, e le operazioni sono eseguite su set di semafori.
- Entrambe queste caratteristiche forniscono funzionalità più estese di quelle tipicamente necessarie, quindi è utile implementare protocolli più semplici.

Semafori binari: una implementazione

- Un protocollo normalmente utilizzato è quello dei semafori binari. Un semaforo binario ha due valori: available (libero) e reserved (in uso). Sui semafori binari sono definite due operazioni:
 - Reserve (wait, o P): Tenta di riservare questo semaforo per uso esclusivo.
 - Se il semaforo è già stato riservato da un altro processo, l'operazione si blocca fino a quando il semaforo è rilasciato.
 - Release (signal, o V): Libera un semaforo correntemente riservato, così che possa essere riservato da un altro processo.

Semafori binari: una implementazione

- un modo largamente usato per rappresentare questi stati è utilizzare il valore 1 per indicare free e il valore 0 per riservato, con le operazioni reserve e release: l'una decrementa di uno il valore del semaforo, l'altra lo incrementa di uno.

Semafori binari: una implementazione

- tutte le funzioni in questa implementazione hanno due argomenti, che identificano un set di semafori e il numero di un semaforo all'interno di quel set

```
int initSemAvailable(int semId, int semNum);  
int initSemInUse(int semId, int semNum);  
int reserveSem(int semId, int semNum);  
int releaseSem(int semId, int semNum);
```

Semafori binari: una implementazione

```
// Initialize semaphore to 1 (i.e., "available")
int initSemAvailable(int semId, int semNum) {
    union semun arg;

    arg.val = 1;
    return semctl(semId, semNum, SETVAL, arg);
}

// Initialize semaphore to 0 (i.e., "in use")
int initSemInUse(int semId, int semNum) {
    union semun arg;

    arg.val = 0;
    return semctl(semId, semNum, SETVAL, arg);
}
```

Semafori binari: una implementazione

```
// Reserve semaphore - decrement it by 1
int reserveSem(int semId, int semNum) {
    struct sembuf sops;

    sops.sem_num = semNum;
    sops.sem_op = -1;
    sops.sem_flg = 0;

    return semop(semId, &sops, 1);
}
```

```
// Release semaphore - increment it by 1
int releaseSem(int semId, int semNum) {
    struct sembuf sops;

    sops.sem_num = semNum;
    sops.sem_op = 1;
    sops.sem_flg = 0;

    return semop(semId, &sops, 1);
}
```