

# Semaphore (6A)

---

- Semaphore

Copyright (c) 2012 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice and Octave.

# Semaphore

```
int semget ( key_t key, int nsems, int semflg );  
int semop ( int semid, struct sembuf *sops, unsigned nsops);  
int semctl ( int semid, int semnum, int cmd, union semun arg );
```

cmd (arg)

```
struct sembuf {  
    ushort sem_num;  
    short  sem_op;  
    short  sem_flg;  
};
```

```
/* semaphore index in array */  
/* semaphore operation */  
/* operation flags */
```

/\* arg for semctl system calls. \*/

```
union semun {
```

```
    int          val;          /* value for SETVAL */  
    struct semid_ds * buf;     /* buffer for IPC_STAT & IPC_SET */  
    ushort       * array;     /* array for GETALL & SETALL */  
    struct seminfo * __buf;    /* buffer for IPC_INFO */  
    void          * __pad;
```

```
}
```

# Semaphore Example (1)

```
int semop ( int semid, struct sembuf *sops, unsigned nsops);
```

```
semid = semget(key, 2, IPC_CREATE);
```

```
struct sembuf lock[ ] = { {0, -1, SEM_UNDO}, {1, -1, SEM_UNDO} };
```

```
struct sembuf unlock[ ] = { {0, +1, SEM_UNDO}, {1, +1, SEM_UNDO} };
```

<pre>semop(semid, &amp;lock[0], 1); // dec 1st semaphore</pre>	<pre>← sops</pre>	<pre>sem_num = 0 sem_op = -1 sem_flg = SEM_UNDO</pre>	}	<pre>semop(semid, lock, 2);</pre>
<pre>semop(semid, &amp;lock[1], 1); // dec 2nd semaphore</pre>	<pre>← sops</pre>	<pre>sem_num = 1 sem_op = -1 sem_flg = SEM_UNDO</pre>		
<pre>semop(semid, &amp;unlock[0], 1); // inc 1st semaphore</pre>	<pre>← sops</pre>	<pre>sem_num = 0 sem_op = +1 sem_flg = SEM_UNDO</pre>	}	<pre>semop(semid, unlock, 2);</pre>
<pre>semop(semid, &amp;unlock[1], 1); // inc 2nd semaphore</pre>	<pre>← sops</pre>	<pre>sem_num = 1 sem_op = +1 sem_flg = SEM_UNDO</pre>		

# semget()

```
int semget ( key_t key, int nsems, int semflg );
```

returns semaphore set identifier (*sid*) on success  
*semaphore set – array of semaphores*

*key* – the return value of `ftok()`

*nsems* - the **number** of semaphores in a semaphore set (*array*)

*semflg*

IPC\_CREAT

Create the semaphore set  
if it doesn't already exist

IPC\_CREAT | IPC\_EXCL

Fails  
if semaphore set already exists.

```
sid = semget( mykey, 2, IPC_CREAT | 0660 )
```

# semop() - (1)

```
int semop ( int semid, struct sembuf *sops, unsigned nsops);
```

**semid** - the return value of **semget()**  
**sops** - a pointer to an **array of semaphore operations** to be performed on the semaphore set  
**nsops** - the **number of semaphore operations** in that array.

```
struct sembuf {  
    ushort sem_num; // semaphore index in array (sem set)  
                    // The index of the semaphore you wish to deal with  
    short  sem_op;  // semaphore operation (eg inc, dec)  
                    // The operation to perform (positive, negative, or zero)  
    short  sem_flg; // operation flags  
};
```

**sops** example

```
sem_num = 0  
sem_op  = -1  
sem_flg = SEM_UNDO
```

# semop() - (2)

```
int semop ( int semid, struct sembuf *sops, unsigned nsops);
```

**negative sem\_op** – lock  
is added to the semaphore.  
the calling process **sleeps** until the requested amount  
of resources are available (val > 0) in the semaphore

**positive sem\_op** – unlock  
is added to the semaphore.  
returning resources back to the semaphore set

**zero sem\_op**  
the calling process will **sleep()** until the semaphore's  
value is 0.  
waiting for a semaphore to reach 100% utilization

```
struct sembuf {  
    ushort sem_num;  
    short  sem_op;  
    short  sem_flg;  
};
```

# semop() - (3)

```
int semop ( int semid, struct sembuf *sops, unsigned nsops);
```

**nsops** - the number of operations in that array.

SEM\_UNDO : automatically undone when the process terminates

IPC\_NOWAIT : If IPC\_NOWAIT is **not** specified,  
then the calling process sleeps  
until the requested amount of resources  
are available in the semaphore  
(another process has released some).



# semop() - (4)

```
int semop ( int semid, struct sembuf *sops, unsigned nsops);
```

```
struct sembuf sem_lock = { 0, -1, IPC_NOWAIT };  
a value of ``-1" will be added to semaphore number 0  
in the semaphore set.
```

```
semop(sid, &sem_lock, 1);
```

```
struct sembuf sem_unlock = { 0, 1, IPC_NOWAIT };  
a value of ``1" will be added to semaphore number 0  
in the semaphore set.
```

```
semop(sid, &sem_unlock, 1);
```

```
struct sembuf {  
    ushort sem_num;  
    short  sem_op;  
    short  sem_flg;  
};
```

# semctl() - (1)

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

```
/* arg for semctl system calls. */
```

```
union semun {
```

```
    int                val;        /* SETVAL (val)                */
    struct semid_ds *   buf;        /* IPC_STAT (buf) & IPC_SET (buf) */
    ushort             *   array;   /* GETALL (array) & SETALL (array) */
    struct seminfo     *   __buf;   /* IPC_INFO (__buf)            */
    Void               *   __pad;
```

```
}
```

cmd (arg)

IPC_STAT	GETPID	<u>GETALL</u>
IPC_SET	GETNCNT	<u>GETVAL</u>
IPC_RMID	GETZCNT	
		<u>SETALL</u>
		<u>SETVAL</u>

# semctl() - (2)

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

**IPC\_STAT** Retrieves the `semid_ds` structure for a set, and stores it in the address of the `buf` argument in the `semun` union.

**IPC\_SET** Sets the value of the `ipc_perm` member of the `semid_ds` structure for a set. Takes the values from the `buf` argument of the `semun` union.

**IPC\_RMID** Removes the set from the kernel.

**GETALL** Used to obtain the **values of all semaphores** in a set. The integer values are stored in an *array* of unsigned short integers pointed to by the `array` member of the union.

**GETNCNT** Returns the number of processes currently **waiting for resources**.

**GETPID** Returns the PID of the process which performed the **last semop call**.

**GETVAL** Returns the **value of a single** semaphore within the set.

**GETZCNT** Returns the number of processes currently **waiting for 100% resource utilization**.

**SETALL** Sets **all semaphore values** with a set to the matching values contained in the *array member* of the union.

**SETVAL** Sets the **value of an individual semaphore** within the set to the `val` member of the union.

# semctl() - semid\_ds

```
/* One semid data structure for each set of semaphores in the system. */
struct semid_ds {
    struct ipc_perm    sem_perm;        /* permissions .. see ipc.h */
    time_t             sem_otime;       /* last semop time */
    time_t             sem_ctime;       /* last change time */
    struct sem         *sem_base;       /* ptr to first semaphore in array */
    struct wait_queue *eventn;
    struct wait_queue *eventz;
    struct sem_undo    *undo;           /* undo requests on this array */
    ushort             sem_nsems;       /* no. of semaphores in array */
};
```

**sem\_perm** This is an instance of the ipc\_perm structure, which holds the [permission information](#) for the semaphore set, including the access permissions, and information about the creator of the set (uid, etc).

**sem\_otime** Time of the [last semop\(\) operation](#) (more on this in a moment)

**sem\_ctime** Time of the [last change](#) to this structure (mode change, etc)

**sem\_base** Pointer to the [first semaphore](#) in the array (see next structure)

**sem\_undo** Number of [undo requests](#) in this array

**sem\_nsems** Number of [semaphores](#) in the semaphore set (the array)

# semctl() - IPC\_STAT, IPC\_SET

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

**IPC\_STAT** Retrieves the `semid_ds` structure for a set, and stores it in the address of the `buf` argument in the `semun` union.

**IPC\_SET** Sets the value of the `ipc_perm` member of the `semid_ds` structure for a set. Takes the values from the `buf` argument of the `semun` union.

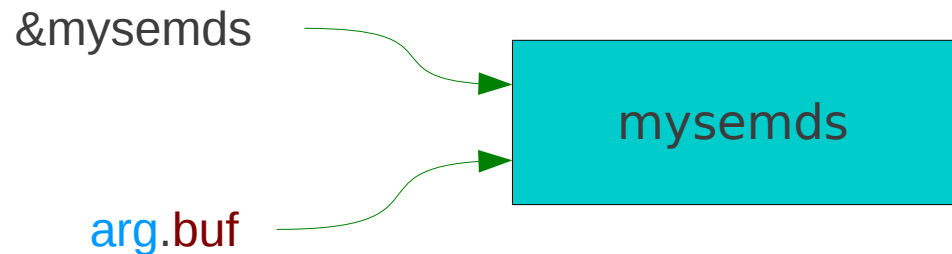
```
union semun {  
    int          val;  
    struct semid_ds * buf;  
    ushort      * array;  
    struct seminfo * __buf;  
    void        * __pad;  
}
```

```
struct semid_ds mysemids;           // allocate ds in memory
```

```
union semun arg;  
arg.buf = &mysemids;               // buf must point to an allocated ds
```

```
semctl(semid, 0, IPC_STAT, arg);
```

```
semctl(semid, 0, IPC_SET, arg);
```



# semctl() - SETVAL, SETALL

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

**SETVAL** Sets the value of an individual semaphore within the set to the val member of the union.

**SETALL** Sets all semaphore values with a set to the matching values contained in the array member of the union.

```
union semun {  
    int val;  
    struct semid_ds * buf;  
    ushort * array;  
    struct seminfo * __buf;  
    void * __pad;  
}
```

```
union semun arg;
```

```
arg.val = 5;  
semctl(semid, 1, SETVAL, arg);
```

```
unsigned short val[] = {3, 5, 6};  
arg.array = val;  
semctl(semidc, 0, SETALL, arg);
```

↑  
unused

# semctl() - GETVAL, GETALL

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

**GETVAL** Returns the value of a single semaphore within the set.

**GETALL** Used to obtain the values of all semaphores in a set. The integer values are stored in an *array* of unsigned short integers pointed to by the array member of the union.

```
union semun {  
    int val;  
    struct semid_ds * buf;  
    → ushort * array;  
    struct seminfo * __buf;  
    void * __pad;  
}
```

```
unsigned short val;
```

```
val = semctl(semid, 1, GETVAL, 0);
```

```
union semun arg;  
unsigned short semarr[3];  
arg.array = semarr;
```

```
semctl(semid, 0, GETALL, arg);
```

↑  
unused

# Union Examples

```
union myunion {  
    char    ch;  
    int     in;  
    short   sh;  
} arg;
```

```
int     *    pint;  
short   *    pshort;  
char    *    pchar;  
double  *    pdouble;
```

```
printf(" add(arg)=%p \n", &arg);  
printf(" size(arg)=%ld \n", sizeof(arg));  
printf(" size(arg.ch)=%ld \n", sizeof(arg.ch));  
printf(" size(arg.in)=%ld \n", sizeof(arg.in));  
printf(" size(arg.sh)=%ld \n", sizeof(arg.sh));
```

```
arg.ch = 'A';  
printf("arg.ch = %#010x %d %c \n", arg.ch, arg.ch, arg.ch);
```

```
arg.sh = 0x0102;  
printf("arg.sh = %#010x %d %c \n", arg.sh, arg.sh, arg.sh);
```

```
arg.in = 0x01020304;  
printf("arg.in = %#010x %d %c \n", arg.in, arg.in, arg.in);
```

```
printf("arg.ch = %#010x %d %c \n", arg.ch, arg.ch, arg.ch);  
printf("arg.sh = %#010x %d %c \n", arg.sh, arg.sh, arg.sh);  
printf("arg.in = %#010x %d %c \n", arg.in, arg.in, arg.in);
```

```
printf("size(pint) = %ld size(int) = %ld \n", sizeof(pint), sizeof(int));  
printf("size(pshort) = %ld size(short) = %ld \n", sizeof(pshort), sizeof(short));  
printf("size(pchar) = %ld size(char) = %ld \n", sizeof(pchar), sizeof(char));  
printf("size(pdoube) = %ld size(double) = %ld \n", sizeof(pdoube), sizeof(double));
```



# Reference

---

## References

- [1] <http://en.wikipedia.org/>
- [2] <http://www.tldp.org/LDP/lpg/node46.html>