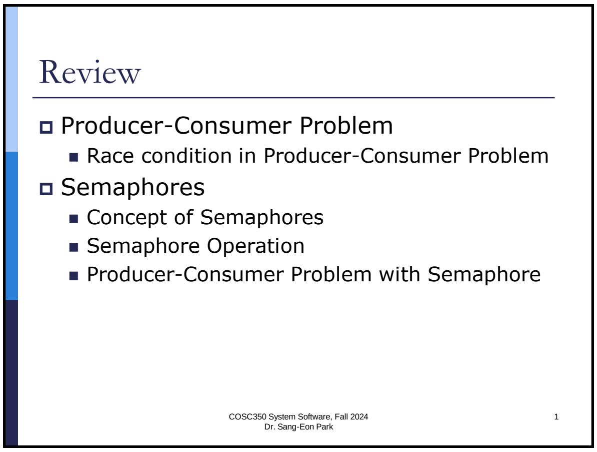
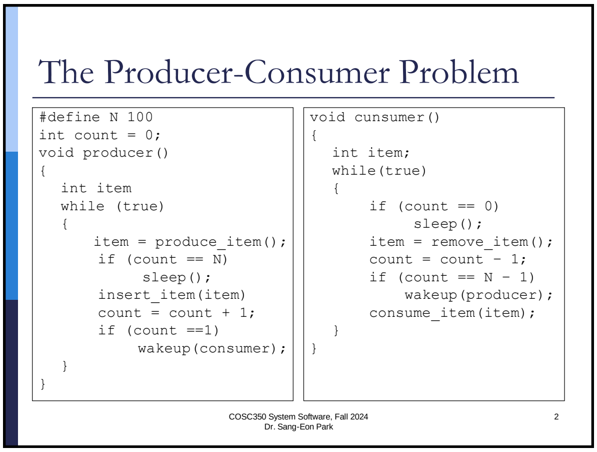


Review

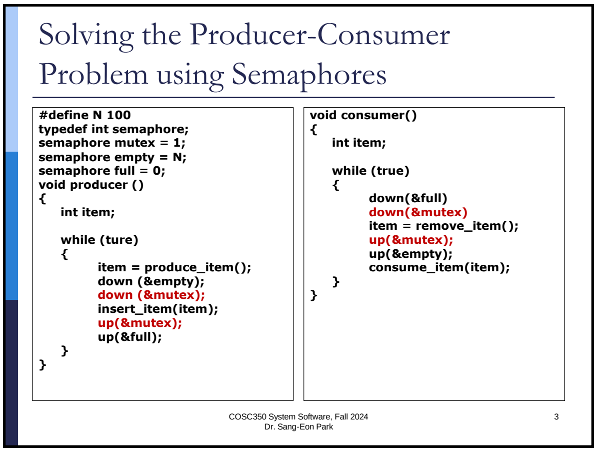
- Producer-Consumer Problem
  - Race condition in Producer-Consumer Problem
- Semaphores
  - Concept of Semaphores
  - Semaphore Operation
  - Producer-Consumer Problem with Semaphore



The Producer-Consumer Problem

```
#define N 100
int count = 0;
void producer()
{
   int item
   while (true)
   {
        item = produce_item();
        if (count == N)
             sleep();
        insert_item(item)
        count = count + 1;
        if (count ==1)
             wakeup(consumer);
   }
}
```

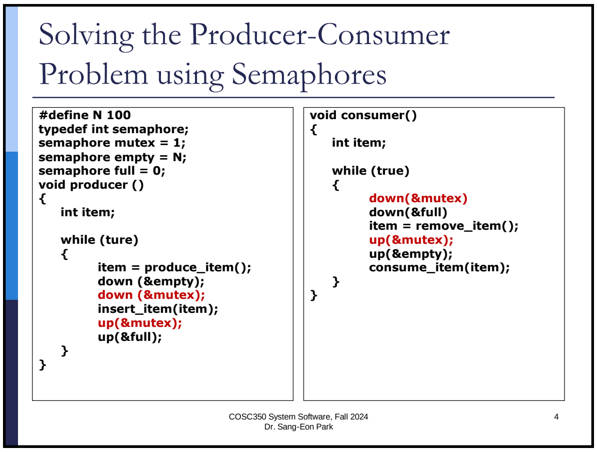
```
void cunsumer()
{
   int item;
   while(true)
   {
        if (count == 0)
              sleep();
        item = remove_item();
        count = count - 1;
        if (count == N - 1)
             wakeup(producer);
        consume_item(item);
   }
}
```



Solving the Producer-Consumer Problem using Semaphores

```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer ()
{
   int item;

   while (ture)
   {
        item = produce_item();
        down (&empty);
        down (&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
   }
}
```

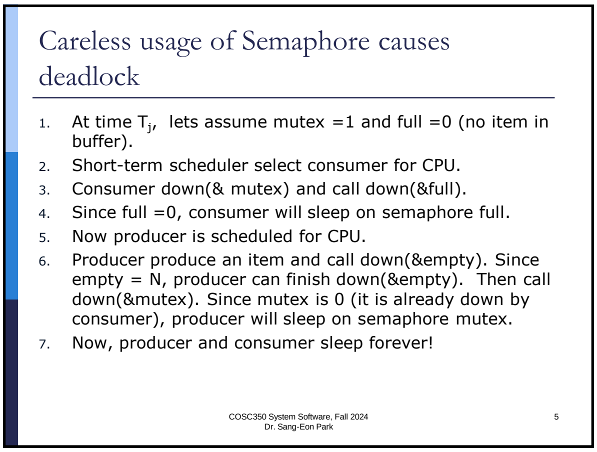
```
void consumer()
{
   int item;

   while (true)
   {
        down(&full)
        down(&mutex)
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
   }
}
```



Solving the Producer-Consumer Problem using Semaphores

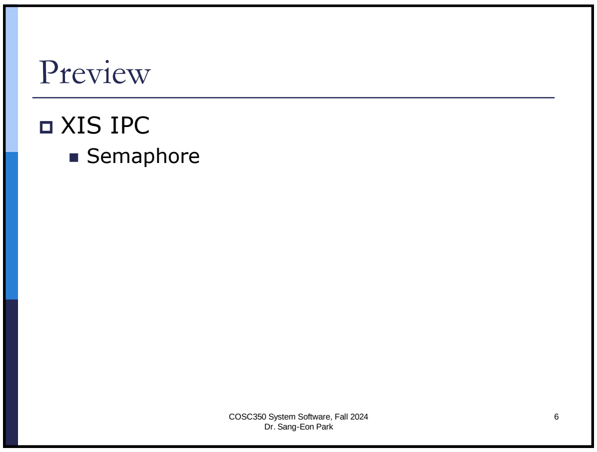
```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer ()
{
   int item;

   while (ture)
   {
        item = produce_item();
        down (&empty);
        down (&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
   }
}
```

```
void consumer()
{
   int item;

   while (true)
   {
        down(&mutex)
        down(&full)
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
   }
}
```



Careless usage of Semaphore causes deadlock

1. At time T_j, lets assume mutex =1 and full =0 (no item in buffer).
2. Short-term scheduler select consumer for CPU.
3. Consumer down(& mutex) and call down(&full).
4. Since full =0, consumer will sleep on semaphore full.
5. Now producer is scheduled for CPU.
6. Producer produce an item and call down(&empty). Since empty = N, producer can finish down(&empty). Then call down(&mutex). Since mutex is 0 (it is already down by consumer), producer will sleep on semaphore mutex.
7. Now, producer and consumer sleep forever!



Preview

- XIS IPC
  - Semaphore

XSI IPC (Semaphore)

- A semaphore is not a form of IPC similar to the others (pipes, FIFOs or message queue).
- A semaphore is a counter used to protect to a shared data object for multiple processes.
- To access (read or write) a shared data object, a process must check semaphore.
- Modification to the a semaphore are executed **indivisibly**

XSI IPC (Semaphore)

- To access a shared resources, a process needs to do the followings:
  - Test the semaphore that controls the resources.
  - If the value of the semaphore is >0, the process reduce the value by 1 and access resources.
  - If the value of the semaphore is 0, the process need go to sleep on the semaphore until the value becomes greater than 0.

XSI IPC (Semaphore)

- To implement semaphores correctly, the test and modification must be an indivisible operation – usually implemented inside kernel.
- XSI **semaphore** are not simply an integer value.
- Instead, we have to define a semaphore as a set of one or more semaphore values.
- The kernel maintains a **semid_ds** structure for each semaphore set.

XSI IPC (Semaphore)

```
/* One semid data structure for each set of semaphores in the system. */
struct semid_ds {
     struct ipc_perm sem_perm;     /* permissions  */
     time_t          sem_otime;    /* last semop time */
     time_t          sem_ctime;    /* last change time */
     unsighed short    sem_nsems;  /* no. of semaphores in array */
     struct sem      *sem_base;    /* ptr to first semaphore in array */
     struct wait_queue *eventn; /* processes awaiting semval > curval */
     struct wait_queue *eventz; /* processes awaiting semval == 0*/
     struct sem_undo  *undo;       /* undo requests on this array */
};
```

```
struct ipc_perm {
   uid_t uid; /* owner's effective user ID*/
   gid_t gid; /* owner's effective group ID */
   uid_t cuid; /* creator effective user ID */
   gid_t cgid; /* creator effective group ID */
   mode_t mode /* access mode */
};
```

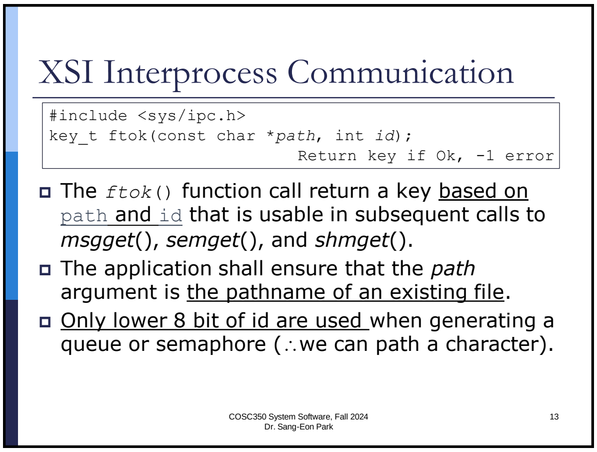
XSI IPC (Semaphore)

- Each semaphore is represented by an anonymous structure containing at least following members.

```
struct sem{
       unsigned short semval;  /* semaphore values, always >=0 */
       pid_t sempid;           /* pid for last operation */
       unsigned short semncnt; /* # of processes awaiting semval > curval */
       unsigned short semzcnt; /* # of processes awaiting semval == 0 */

};
```

XSI Interprocess Communication

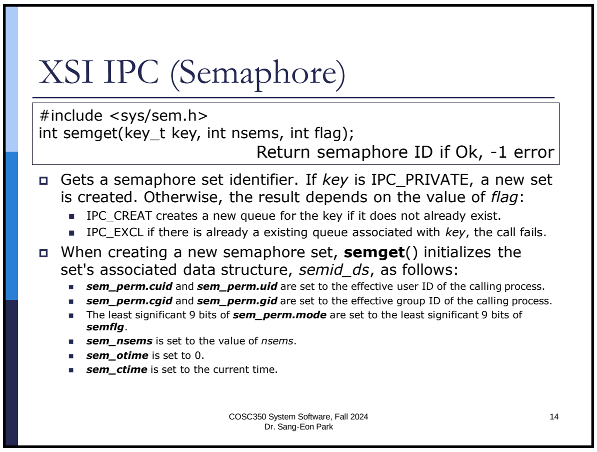
- When an XSI IPC structure is created (by calling `msgget()`, `semget()` or `shmget()`), a **key** must be specified.
- The data type **key_t** for a key is specified in the header file <sys/types.h>.



XSI Interprocess Communication

```
#include <sys/ipc.h>
key_t ftok(const char *path, int id);
                                    Return key if Ok, -1 error
```

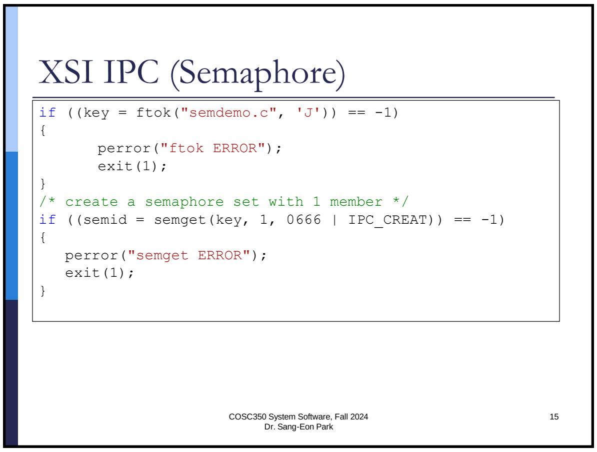
- The *ftok*() function call return a key based on path and id that is usable in subsequent calls to *msgget*(), *semget*(), and *shmget*().
- The application shall ensure that the *path* argument is the pathname of an existing file.
- Only lower 8 bit of id are used when generating a queue or semaphore (∴we can path a character).



XSI IPC (Semaphore)

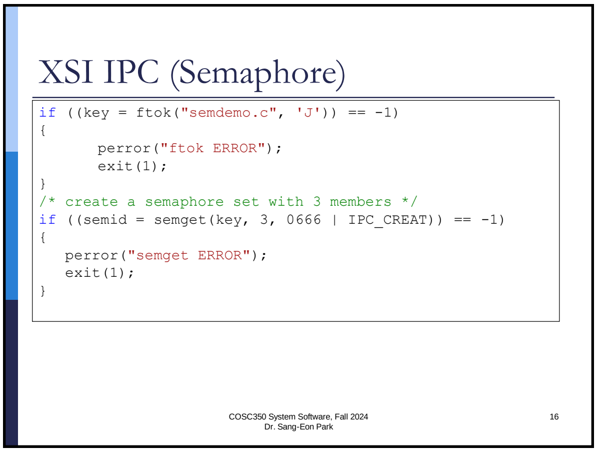
```
#include <sys/sem.h>
int semget(key_t key, int nsems, int flag);
                              Return semaphore ID if Ok, -1 error
```

- Gets a semaphore set identifier. If *key* is IPC_PRIVATE, a new set is created. Otherwise, the result depends on the value of *flag*:
  - IPC_CREAT creates a new queue for the key if it does not already exist.
  - IPC_EXCL if there is already a existing queue associated with *key*, the call fails.
- When creating a new semaphore set, **semget**() initializes the set's associated data structure, *semid_ds*, as follows:
  - ***sem_perm.cuid*** and ***sem_perm.uid*** are set to the effective user ID of the calling process.
  - ***sem_perm.cgid*** and ***sem_perm.gid*** are set to the effective group ID of the calling process.
  - The least significant 9 bits of ***sem_perm.mode*** are set to the least significant 9 bits of ***semflg***.
  - ***sem_nsems*** is set to the value of *nsems*.
  - ***sem_otime*** is set to 0.
  - ***sem_ctime*** is set to the current time.



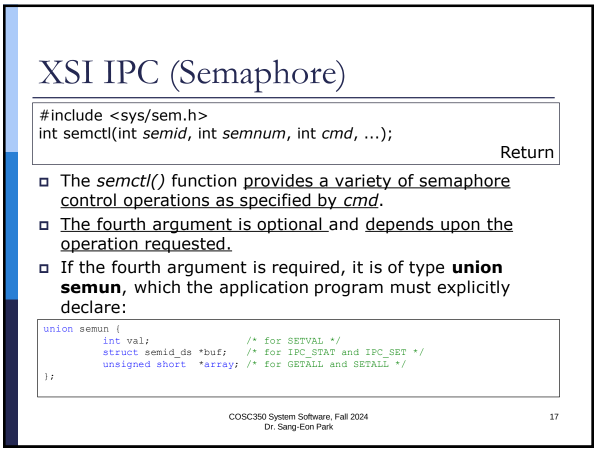
XSI IPC (Semaphore)

```
if ((key = ftok("semdemo.c", 'J')) == -1)
{
       perror("ftok ERROR");
       exit(1);
}
/* create a semaphore set with 1 member */
if ((semid = semget(key, 1, 0666 | IPC_CREAT)) == -1)
{
   perror("semget ERROR");
   exit(1);
}
```



XSI IPC (Semaphore)

```
if ((key = ftok("semdemo.c", 'J')) == -1)
{
       perror("ftok ERROR");
       exit(1);
}
/* create a semaphore set with 3 members */
if ((semid = semget(key, 3, 0666 | IPC_CREAT)) == -1)
{
   perror("semget ERROR");
   exit(1);
}
```

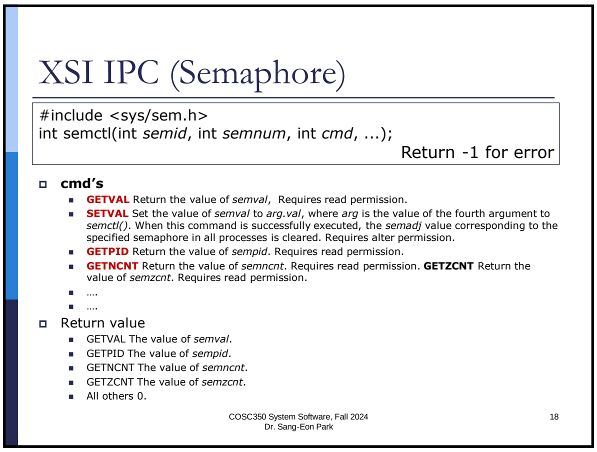


XSI IPC (Semaphore)

```
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
                                                      Return
```

- The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.
- The fourth argument is optional and depends upon the operation requested.
- If the fourth argument is required, it is of type **union semun**, which the application program must explicitly declare:

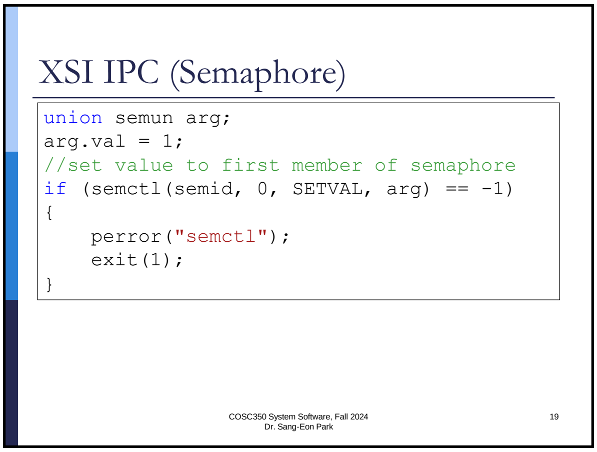
```
union semun {
          int val;                 /* for SETVAL */
          struct semid_ds *buf;    /* for IPC_STAT and IPC_SET */
          unsigned short  *array; /* for GETALL and SETALL */
};
```



XSI IPC (Semaphore)

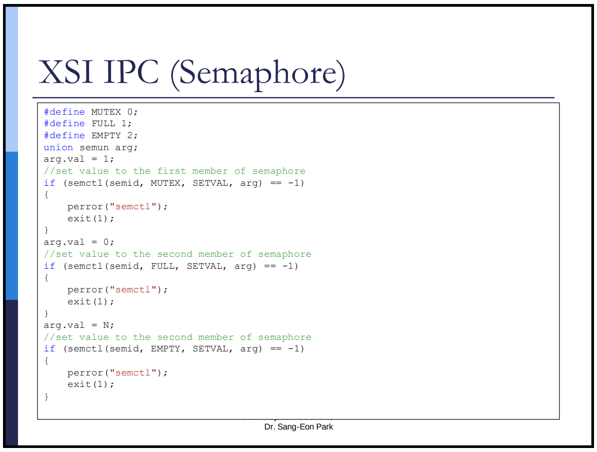
```
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
                                           Return -1 for error
```

- **cmd’s**
  - **GETVAL** Return the value of *semval*, Requires read permission.
  - **SETVAL** Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument to *semctl()*. When this command is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. Requires alter permission.
  - **GETPID** Return the value of *sempid*. Requires read permission.
  - **GETNCNT** Return the value of *semncnt*. Requires read permission. **GETZCNT** Return the value of *semzcnt*. Requires read permission.
  - ….
  - ….
- Return value
  - GETVAL The value of *semval*.
  - GETPID The value of *sempid*.
  - GETNCNT The value of *semncnt*.
  - GETZCNT The value of *semzcnt*.
  - All others 0.



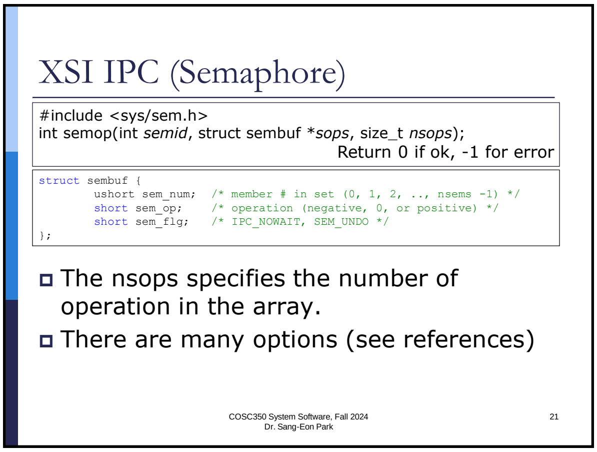
XSI IPC (Semaphore)

```
union semun arg;
arg.val = 1;
//set value to first member of semaphore
if (semctl(semid, 0, SETVAL, arg) == -1)
{
    perror("semctl");
    exit(1);
}
```



XSI IPC (Semaphore)

```
#define MUTEX 0;
#define FULL 1;
#define EMPTY 2;
union semun arg;
arg.val = 1;
//set value to the first member of semaphore
if (semctl(semid, MUTEX, SETVAL, arg) == -1)
{
    perror("semctl");
    exit(1);
}
arg.val = 0;
//set value to the second member of semaphore
if (semctl(semid, FULL, SETVAL, arg) == -1)
{
    perror("semctl");
    exit(1);
}
arg.val = N;
//set value to the second member of semaphore
if (semctl(semid, EMPTY, SETVAL, arg) == -1)
{
    perror("semctl");
    exit(1);
}
```

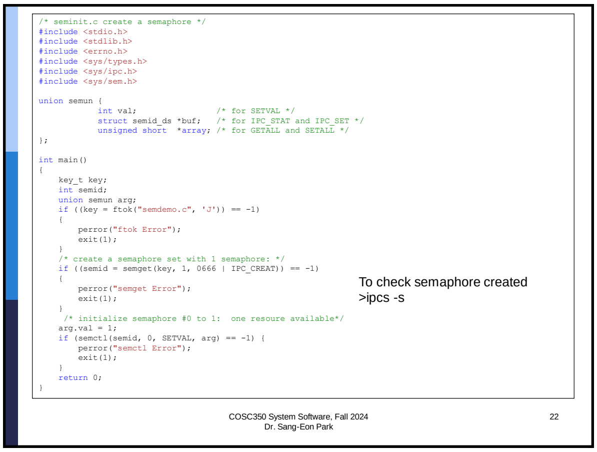


XSI IPC (Semaphore)

#include <sys/sem.h>
int semop(int *semid*, struct sembuf **sops*, size_t *nsops*);
Return 0 if ok, -1 for error

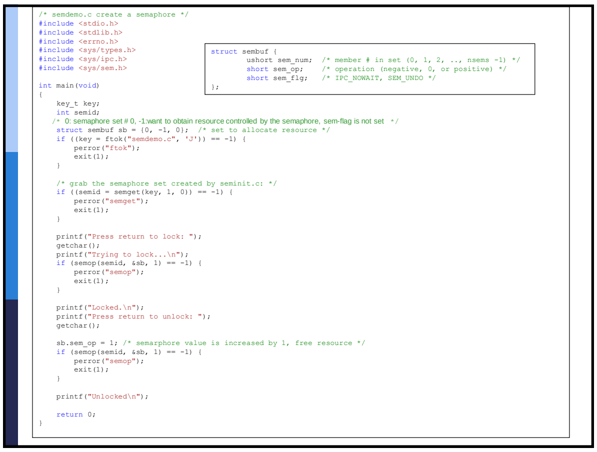
```
struct sembuf {
        ushort sem_num;  /* member # in set (0, 1, 2, .., nsems -1) */
        short sem_op;    /* operation (negative, 0, or positive) */
        short sem_flg;   /* IPC_NOWAIT, SEM_UNDO */
};
```

- The nsops specifies the number of operation in the array.
- There are many options (see references)



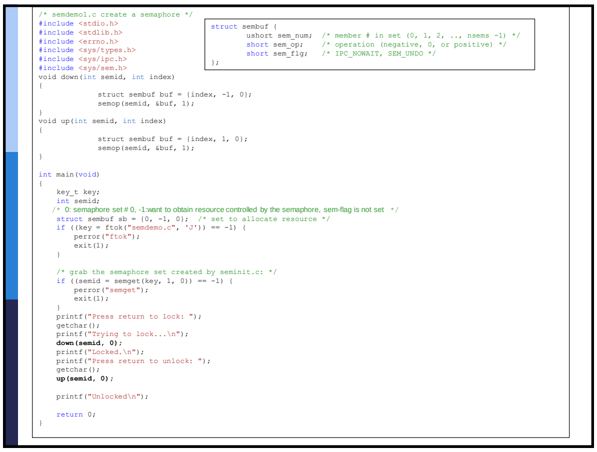
```
/* seminit.c create a semaphore */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun {
            int val;               /* for SETVAL */
            struct semid_ds *buf;  /* for IPC_STAT and IPC_SET */
            unsigned short  *array; /* for GETALL and SETALL */
};

int main()
{
    key_t key;
    int semid;
    union semun arg;
    if ((key = ftok("semdemo.c", 'J')) == -1)
    {
        perror("ftok Error");
        exit(1);
    }
    /* create a semaphore set with 1 semaphore: */
    if ((semid = semget(key, 1, 0666 | IPC_CREAT)) == -1)
    {
        perror("semget Error");
        exit(1);
    }
     /* initialize semaphore #0 to 1:  one resoure available*/
    arg.val = 1;
    if (semctl(semid, 0, SETVAL, arg) == -1) {
        perror("semctl Error");
        exit(1);
    }
    return 0;
}
```

To check semaphore created
>ipcs -s



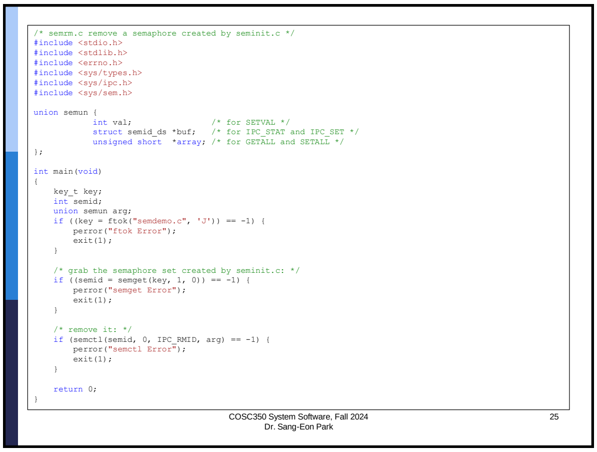
```
/* semdemo.c create a semaphore */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main(void)
{
    key_t key;
    int semid;
   /*  0: semaphore set # 0, -1:want to obtain resource controlled by the semaphore, sem-flag is not set  */
    struct sembuf sb = {0, -1, 0};  /* set to allocate resource */
    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    printf("Press return to lock: ");
    getchar();
    printf("Trying to lock...\n");
    if (semop(semid, &sb, 1) == -1) {
        perror("semop");
        exit(1);
    }

    printf("Locked.\n");
    printf("Press return to unlock: ");
    getchar();

    sb.sem_op = 1; /* semarphore value is increased by 1, free resource */
    if (semop(semid, &sb, 1) == -1) {
        perror("semop");
        exit(1);
    }

    printf("Unlocked\n");

    return 0;
}
```

```
struct sembuf {
        ushort sem_num;  /* member # in set (0, 1, 2, .., nsems -1) */
        short sem_op;    /* operation (negative, 0, or positive) */
        short sem_flg;   /* IPC_NOWAIT, SEM_UNDO */
};
```



```
/* semdemo1.c create a semaphore */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
void down(int semid, int index)
{
             struct sembuf buf = {index, -1, 0};
             semop(semid, &buf, 1);
}
void up(int semid, int index)
{
             struct sembuf buf = {index, 1, 0};
             semop(semid, &buf, 1);
}

int main(void)
{
    key_t key;
    int semid;
   /*  0: semaphore set # 0, -1:want to obtain resource controlled by the semaphore, sem-flag is not set  */
    struct sembuf sb = {0, -1, 0};  /* set to allocate resource */
    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }
    printf("Press return to lock: ");
    getchar();
    printf("Trying to lock...\n");
    down(semid, 0);
    printf("Locked.\n");
    printf("Press return to unlock: ");
    getchar();
    up(semid, 0);

    printf("Unlocked\n");

    return 0;
}
```

```
struct sembuf {
        ushort sem_num;  /* member # in set (0, 1, 2, .., nsems -1) */
        short sem_op;    /* operation (negative, 0, or positive) */
        short sem_flg;   /* IPC_NOWAIT, SEM_UNDO */
};
```



```c
/* semrm.c remove a semaphore created by seminit.c */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun {
            int val;                /* for SETVAL */
            struct semid_ds *buf;   /* for IPC_STAT and IPC_SET */
            unsigned short  *array; /* for GETALL and SETALL */
};

int main(void)
{
    key_t key;
    int semid;
    union semun arg;
    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok Error");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget Error");
        exit(1);
    }

    /* remove it: */
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {
        perror("semctl Error");
        exit(1);
    }

    return 0;
}
```