Advanced Inter-process Communications – Part2

Semaphores

- An integer variable that can be accessed only through two standard atomic operations:
  - *P* - wait
    - P(S): while S <= 0
      do skip;
      S = S-1;
  - *V* - signal

    V(S): S = S+1;

    P(S);
    CriticalSection();
    V(S);

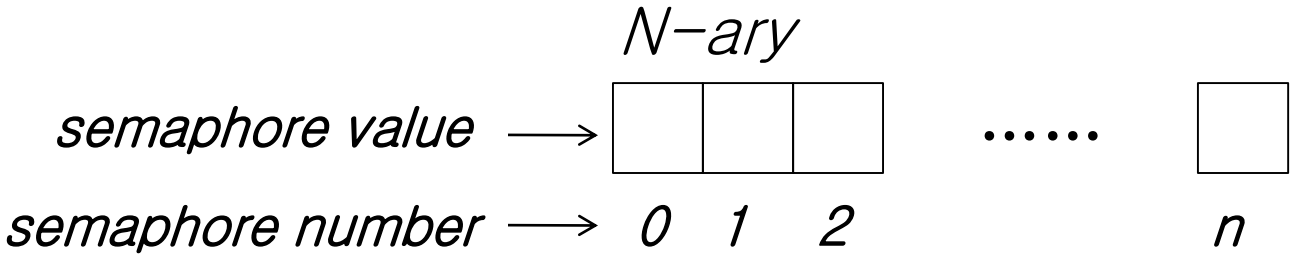
P: proberen
(= to try test)
V: verhogen
(= to raise)

How It Works

- If the semaphore value is positive, the process can use the resource. The process decrements the semaphore value by 1, indicating it has used one unit of the resource.
- If the value tested is 0, the process goes to sleep until the semaphore value is greater than 0.
- The controlling semaphore value is incremented by 1 when a process is done with the shared resource. If any other processes are asleep, waiting for the semaphore, they are awakened.
- Semaphores are typically implemented inside the kernel.
  - To ensure that the test of a semaphore's value and the de/incrementing of the value are implemented as atomic operations.

Two types of semaphores

- *Binary* (lock/unlock)
  - Having a value of 0 or 1.
  - Usually used for locking a resource.
- *N*-ary (*counting*)
  - *N* is the available number of an intended resource on a system.
  - Usually used for maintaining a pool of resources.
- Note that the type/nature of a semaphore (whether binary or *N*-ary) is imposed by the application (user-defined), not by the system.



UNIX Implementation

- A semaphore is a sharable unsigned short integer variable.
- Semaphores are grouped together to form sets.
  - No single semaphore in UNIX
- A semaphore set (array) contains:
  - 1 data structure, struct semid_ds, containing administrative information about the set.
- Each semaphore set is uniquely identified by a nonnegative integer.
- One or more semaphores in a set can be accessed at the same time. (AND synchronization can be possible)
- All semaphores involved in an operation are changed "atomically".
- Only one process can access a semaphore set at any given time.

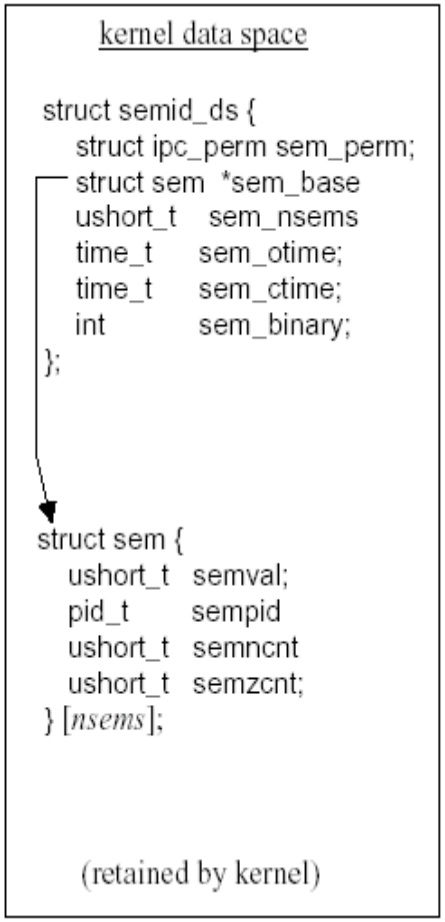
Semaphore Operations

- Basic semaphore operations:
  - *P* operation
    - Testing for zero
    - Decrementing
  - *V* operation
    - Incrementing
- A semaphore operation can be either:
  - Blocking: Processes wait for resources to be released.
  - Non-blocking: Processes return an error code -1, and the external errno variable is set accordingly. (IPC_NOWAIT is set.)
- UNIX puts the blocked process to sleep to eliminate the *busy waiting* phenomenon.

Semaphore System Calls

- semget( )
  - To create or gain access to a set of one or more semaphores.
- semctl( )
  - To get or set the value of a semaphore(s) in a set.
  - To get or modify the status of a semaphore set.
  - To identify process waiting for a specific semaphore value.
  - To identify process that did the last operation on a semaphore.
  - To remove a semaphore set.
- semop( )
  - To wait for the value of a semaphore(s) to become 0.
  - To increment/decrement semaphore value(s).
  - IPC_NOWAIT for non-blocking.
  - SEM_UNDO for undo operations.
  - Note: sleep( ) and wakeup( ) are used internally.

Semaphore Data Structures



Semaphore system calls

```
semid = semget()

        semctl()

union semun {
  int val
  struct semid_ds  *buf;
  unshort          *array;
};

        semop()

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

- In union semun
  - int val is used for SETVAL
  - struct semid_ds *buf is used for IPC_STAT and IPC_SET
  - ushort *array is used for GETALL and SETALL
- In struct sembuf
  - short sem_op has the value: 0: zero test, +n: release n, or -n: request n
  - short sem_flg has the value: IPC_NOWAIT or SEM_UNDO
  - struct sembuf is the data structure of a semaphore. It comes in an *array*.
- In struct sem
  - semval : semaphore value
  - sempid : process id that last acted on the semaphore
  - semcnt : # of processes waiting for the semaphore to become positive
  - semzcnt : # of processed that are waiting for the semaphore to become 0

semget

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/sem.h>

  int semget(key_t *key*, int *nsems*, int *flag*);
  - semget creates an array of semaphores, corresponding to *key*.
  - *nsems* gives the number of semaphores required in the semaphore set.
  - *key* and *flag* take on the same meaning as they do in msgget.
  - Return
    - Semaphore set ID

semctl (1)

- #include <sys/types.h>
  #include <sys/ipc.h>
  #include <sys/sem.h>
  int semctl(int *id*, int *num*, int *cmd*, union semun *arg*);

  union semun {
      int val;
      struct semid_ds *buf;
      ushort *array;
  } arg;

semctl (2)

- semctl does the specified *cmd* on the semaphore queue indicated by *id*.
  - GETVAL return the semaphore whose index is *num*.
  - SETVAL set the value of the semaphore whose index is *num* to *arg.val*.
  - GETPID return value of last PID that did a semop on the semaphore whose index is *num*.
  - GETNCNT return the number of process waiting for semaphore value to become positive.
  - GETZCNT return the number of process waiting for semaphore value to become 0.
  - GETALL return values of all semaphores into array *arg.array*.
  - SETALL set values of all semaphores according to array *arg.array*.

semctl (3)

- IPC_SET set sem_perm.uid, sem_per.gid, and sem_perm.mode (low-order 9 bit) according to *arg.buf*.
- IPC_RMID remove the semaphores associated with *id*.
- IPC_STAT place status information into *arg.buf*.

– *num* gives the semaphore number in the set to be processed.

– The structure semid_ds is defined by:

```
struct semid_ds {
    struct ipc_perm sem_perm; /*permission struct*/
    ushort  sem_nsems; /*number of semaphores in set*/
    time_t  sem_otime; /*last semop operation time*/
    time_t  sem_ctime; /*last change time*/
};
```

– The structure ipc_perm is the same as defined in msgctl.

semop (1)

- int semop(int *id*, struct sembuf ***ops*, int *num*);
  - semop applies the semaphore operations in the array of structures *ops*, to the set of semaphores identified by *id*.
  - num is the number of entries in *ops*.
  - The structure of sembuf is:
    - struct sembuf {

      short sem_num; /*semphore number*/

      short sem_op; /*semphore operation*/

      short sem_flg; /*flag*/

      };
  - sem_num:
    - Specifies the index in the semaphore array for the particular operation, and sem_flg specifies flags for the operation.

semop (2)

- sem_op :
  - *Negative*:
    - Generalized form of **P(s)** operation
    - If the semaphore value is large enough, it is decremented.
    - If not, the process waits until it becomes large enough.
  - *Positive*:
    - Traditional **V(s)** operation
    - The value of sem_op is simply added to the semaphore value.
  - *Zero*:
    - Wait until the semaphore value becomes zero, but semval is not altered.
- sem_flag :
  - If IPC_NOWAIT is set for a particular operation, semop returns immediately for those occasions it would have slept.
  - If SEM_UNDO flag is set, it tells the system to "undo" operation when the process exits
- semop returns the value of the last semaphore operation in *ops* at the time of the call.

Example #4: Semaphore (1)

```
/* pv.h -- semaphore example header file */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#include <errno.h>

extern int errno;

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

/* function prototypes */
int initsem(key_t semkey);
int p(int semid);
int v(int semid);
```

Example #4: Semaphore (2)

```
#include "pv.h"

int initsem(key_t semkey)
{
    int status = 0, semid;

    if ((semid = semget(semkey, 1, SEMPERM|IPC_CREAT|IPC_EXCL)) == -1) {
        if (erron == EEXIST)
            semid = semget(semkey, 1, 0);
    }
   else /* if created ..*/
    {
        semun arg;
        arg.val = 1;
        status = semctl(semid, 0, SETVAL, arg);
    }

    if (semid == -1 || status == -1) {
        perror("initsem failed");
        return (-1);
    }
    else
        return semid; /*all okay*/
}
```

Example #4: Semaphore (3)

```
/* p.c -- semaphore p operation */

#include "pv.h"

int p(int semid)
{
    struct sembuf p_buf;

    p_buf.sem_num = 0;
    p_buf.sem_op = -1;
    p_buf.sem_flg = SEM_UNDO;

    if (semop(semid, &p_buf, 1) == -1) {
        perror("p(semid) failed");
        exit(1);
    }
    else
        return(0);
}
```

Example #4: Semaphore (4)

```
/* v.c -- semaphore v operation */

#include "pv.h"

int v(int semid)
{
    struct sembuf v_buf;

    v_buf.sem_num = 0;
    v_buf.sem_op = 1;
    v_buf.sem_flg = SEM_UNDO;

    if (semop(semid, &v_buf, 1) == -1) {
        perror("v(semid) failed");
        exit(1);
    }
    else
        return(0);
}
```

Example #4: Semaphore (5)

```
/* testsem -- test semaphore routines */

#include <stdio.h>
#include "pv.h"

int handlesem(key_t skey)
{
    int semid, pid = getpid();
    if ((semid = initsem(skey)) < 0)
        exit(1);

    printf("\nprocess %d before critical section\n", pid);
    p(semid);
    printf("process %d in critical section\n", pid);
    sleep(10); /*in real life do something interesting*/
    printf("process %d leaving critical section\n", pid);
    v(semid);
    printf("process %d exiting\n", pid);
    exit(0);
}
```

Example #4: Semaphore (6)

```
void main(void)
{
    key_t semkey = 0x200;

    if (fork() == 0)
        handlesem(semkey);
    if (fork() == 0)
        handlesem(semkey);
    if (fork() == 0)
        handlesem(semkey);
}
```

Example #4: Sample Run

```
dhlee@kde:~/Course/SP/example>a.out

process 5406 before critical section
process 5406 in critical section

process 5407 before critical section

process 5408 before critical section
dhlee@kde:~/Course/SP/example>process 5406 leaving critical section
process 5406 exiting
process 5407 in critical section
process 5407 leaving critical section
process 5407 exiting
process 5408 in critical section
process 5408 leaving critical section
process 5408 exiting

dhlee@kde:~/Course/SP/example>
```

Shared Memory (1)

- Shared memory allows multiple processes to share memory.
- The implementation of shared memory is machine dependent.
  - It depends on the memory management of the host machine.
  - System configuration values determine the minimum and maximum sizes of a segment.
  - The number of segments that may be attached to a process is also restricted.
- Typically one process creates a shared memory segment, maps the segment into its data space and initializes the segment.
  - The term for mapping a shared memory segment into a user's data space is called *attachment*.
  - The creator process specifies R/W permissions associated for the shared memory segment.

Shared Memory (2)

- Other processes may then attach the shared memory segment and access its contents.
  - Unrelated processes can thus communicate via shared memory.
- A process using a shared memory segment may *detach* it explicitly when finished.
  - If not done explicitly, an exit( ) call will detach the segment *automatically*.
  - Detachment does *not* remove the shared memory.
- Shared memory must be removed explicitly.
  - Usually by the same process that does the creation.
- Synchronized access to shared memory must be done explicitly.
  - Usually used with semaphores.

Shared Memory System Calls

- shmget( )
  - To obtain a shared memory identifier.
- shmat( )
  - To attach a shared memory segment to a process data segment.
- shmdt( )
  - To detach a shared memory segment from a process data segment.
- shmctl( )
  - To determine the status of a shared memory segment.
  - To change permissions or ownership of a shared memory segment.
  - To remove a shared memory.

shmget

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/shm.h>

  int shmget(key_t *key*, int *size*, int *flag*);
  - shmget accesses or creates a shared memory region of *size* bytes.
  - *size* gives the required minimum size (in byte) of the memory segment.
  - The parameters *key* and *flag* have the same meaning as the do for msgget.

shmctl (1)

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/shm.h>

  int shmctl(int *id*, int *cmd*, struct shmid_ds **buf*);
  - shmctl does various control operation on the shared memory region identified by *id*.

shmctl (2)

- The structure shmid_ds is defined by:
  - struct shmid_ds {

```
    struct ipc_perm  shm_perm;   /*permission struct*/
    int              shm_segsz;  /*size of segment*/
    int*             pad1;       /*used by system*/
    ushort           shm_lpid;   /*pid of last operation */
    ushort           shm_cpid;   /*pid of creator*/
    ushort           shm_nattch; /*number currently attached*/
    short            pad2;       /*used by system*/
    time_t           shm_atime;  /*last attach time*/
    tiem_t           shm_dtime;  /*last detach time*/
    time_t           shm_ctime;  /*last change time*/
};
```

shmctl (3)

- The operations (*cmd*) are:
  - IPC_STAT read values of shared memory header for *id* into *buf*.
  - IPC_SET set shm_perm.uid, shm_perm.gid, and shm_perm.mode (9 low-order bits) in shared memory header according to values in *buf*.
  - IPC_RMID remove shared memory region for *id*.

shmop (1)

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/shm.h>

  void *shmat(int *id*, char **addr*, int *flag*);

  int shmdt(char **addr*);
  - shmat attaches the shared memory region identified by *id* to the address space of a process.

shmop (2)

- If *addr* is 0, the kernel chooses an appropriate address to attach the region. Otherwise, it attempts to attach the region at the specified address.
- *flag*
  - If the SHM_RND bit is on in *flag*, the kernel rounds off the address, if necessary.
  - SHM_RDONLY requests that the segment is attached for reading only.
- shmat returns the address where the region is attached.
- shmdt detaches the shared memory region previously attached at *addr*.

Example #5: Shared Mem (1)

```
/* share_ex.h -- head file for shared memory example */

#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define SHMKEY1 (key_t)0x10 /*shared mem key*/
#define SHMKEY2 (key_t)0x15 /*shared mem key*/
#define SEMKEY  (key_t)0x20 /*semaphore key*/

/*buffer size for reads and writes*/
#define SIZ  5*BUFSIZ
```

Example #5: Shared Mem (2)

```
/*will hold data and read count*/
struct databuf {
    int d_nread;
    char d_buf[SIZ];
};

/* function prototypes */
int getseg(struct databuf **p1, struct databuf **p2);
int getsem(void);
int remove(void);
void reader(int semid, struct databuf *buf1, struct databuf *buf2);
void writer(int semid, struct databuf *buf1, struct databuf *buf2);
```

Example #5: Shared Mem (3)

```
/*init -- initialization routines*/

#include "share_ex.h"
#define IFLAGS (IPC_CREAT|IPC_EXCL)
#define ERR ((struct databuf *) -1)

static int shmid1, shmid2, semid;

/* getseg -- create + attach shared memory sements */
int getseg(struct databuf **p1, struct databuf **p2)
{
    /*create shared memory segments*/
    if ((shmid1 = shmget(SHMKEY1, sizeof(struct databuf), 0600|IFLAGS)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shmid2 = shmget(SHMKEY2, sizeof(struct databuf), 0600|IFLAGS)) < 0) {
        perror("shmget");
        exit(1);
    }
```

Example #5: Shared Mem (4)

```
    /*attach shared memory segments*/
    if ((*p1 = (struct databuf *) shmat(shmid1, 0, 0)) == ERR) {
        perror("shmget");
        exit(1);
    }
    if ((*p2 = (struct databuf *) shmat(shmid2, 0, 0)) == ERR) {
        perror("shmget");
        exit(1);
    }
}
```

Example #5: Shared Mem (5)

```
/* getsem -- get semaphore set */

int getsem(void)
{
    /*create two semaphore set*/
    if ((semid = semget(SEMKEY, 2, 0600|IFLAGS)) < 0) {
        perror("semget");
        exit(1);
    }
    /*set initial values*/
    if (semctl(0, SETVAL, 0) < 0) {
        perror("semctl");
        exit(1);
    }
    if (semctl(1, SETVAL, 0) < 0) {
        perror("semctl");
        exit(1);
    }
    return(semid);
}
```

Example #5: Shared Mem (6)

```
/* remove -- remove shared memory identifiers + sem set id */

int remove(void)
{
    if (shmctl(shmid1, IPC_RMID, (struct shmid_ds *)0) < 0) {
        perror("shmctl");
        exit(1);
    }
    if (shmctl(shmid2, IPC_RMID, (struct shmid_ds *)0) < 0) {
        perror("shmctl");
        exit(1);
    }
    if (semctl(semid, IPC_RMID, (struct semid_ds *)0) < 0) {
        perror("semctl");
        exit(1);
    }
}
```

Example #5: Shared Mem (7)

```
/* shmcopy -- main function */

#include "share_ex.h"

void main(void)
{
    int semid, pid;
    struct databuf *buf1, *buf2;

    /*initialize semaphore set*/
    semid = getsem();

    /*create and attach shared memory segments*/
    getseg(&buf1, &buf2);
```

Example #5: Shared Mem (8)

```
    switch (pid = fork()) {
      case -1:
          perror("fork");
          exit(1);
      case 0: /*child*/
          writer(semid, buf1, buf2);
          remove();
          break;
      default: /*parent*/
          reader(semid, buf1, buf2);
          break;
    }
    exit(0);
}
```

Example #5: Shared Mem (9)

```
/* reader -- handle reading of file */

#include “share_ex.h”

/*these define p() and v() for semaphores*/
struct sembuf p1 = {0, -1, 0}, p2 = {1, -1, 0};
struct sembuf v1 = {0, 1, 0}, v2 = {1, 1, 0};

void reader(int semid, struct databuf *buf1, struct databuf *buf2)
{
    for (;;) {

        /*read into buffer buf1*/
        buf1->d_nread = read(0, buf1->d_buf, SIZ);

        /*synchronization point*/
        semop(semid, &v1, 1);
        semop(semid, &p2, 1);
```

Example #5: Shared Mem (10)

```
        /*test here to avoid writer sleeping*/
        if(buf1->d_nread <= 0)
            return;

        buf2->d_nread = read(0, buf2->d_buf, SIZ);

        semop(semid, &v1, 1);
        semop(semid, &p2, 1);

        if (buf2->d_nread <= 0)
            return;
    }
}
```

Example #5: Shared Mem (11)

```
/* wrinter -- handle writing */

#include “shared_ex.h”

extern struct sembuf p1, p2; /*defined in reader.c*/
extern struct sembuf v1, v2; /*defined in reader.c*/

void writer(int semid, struct databuf *buf1, struct databuf *buf2)
{
    for (;;) {

        semop(semid, &p1, 1);
        semop(semid, &v2, 1);


        if(buf1 ->d_nread <= 0)
            return;
        write(1, buf1->d_buf, buf1->d_nread);
```

Example #5: Shared Mem (12)

```
        semop(semid, &p1, 1);
        semop(semid, &v2, 1);

        if(buf2->d_nread <= 0)
            return;

        write(1, buf2->d_buf, buf2->d_nread);
    }
}
```

```
$> shmcp < big > /tmp/big
```

Example #5: Shared Mem (12)

```
void reader(int semid, struct databuf *buf1, struct
      databuf *buf2)
{
   for (;;) {
        /*read into buffer buf1*/
R1:     buf1->d_nread = read(0, buf1->d_buf,  SIZ);

        /*synchronization point*/
        semop(semid, &v1, 1);
        semop(semid, &p2, 1);

        /*test here to avoid writer sleeping*/
        if(buf1->d_nread <= 0) return;

R2:     buf2->d_nread = read(0, buf2->d_buf, SIZ);

        semop(semid, &v1, 1);
        semop(semid, &p2, 1);

        if (buf2->d_nread <= 0) return;
    }
}
```

```
void writer(int semid, struct databuf *buf1, struct
      databuf *buf2)
{
    for (;;) {

        semop(semid, &p1, 1);
        semop(semid, &v2, 1);

        if(buf1->d_nread <= 0) return;

W1:     write(1, buf1->d_buf, buf1->d_nread);

        semop(semid, &p1, 1);
        semop(semid, &v2, 1);

        if(buf2->d_nread <= 0) return;

W2:     write(1, buf2->d_buf, buf2->d_nread);

    }
}
```

More Examples #3: Sem (1)

```
/* semget.c
   This program shows how to use the semget() system call.
   Before a semaphore set can be used, an IPC key needs to be assembled.
   A way to do it is to call ftok(). This function needs two arguments.
   The first argument is a filename. It is a dummy file but must exist
   in order to assemble a key. The second argument is an integer.
   This program has 3 different names, based on their functions:
       csem:   to create a semaphore set
       semid:  to get the semid associated with a specific key
       rsem:   to remove a semaphore set  (note: semctl() is used)
 */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>

#define  ERR      -1
#define  UNUSED    0          /* for padding unused arguments */
#define  SEMMSL   25          /* Max no of semaphores in a set */
#define  SEMKEY   "semkey"    /* full pathname is preferred */
#define  SEMID    'a'         /* any arbitrary char */
#define  semkey(x, y)  (key_t)ftok(x, y)
```

More Examples #3: Sem (2)

```
int main(int argc, char **argv)
{
    int semid, nsems;
    key_t key;

    if ((key = semkey(SEMKEY, SEMID)) == ERR) { /* needs every time */
        printf("\nCan't assemble a key: ");     /* key is used to generate semid*/
        printf("Error no = %d", errno);
        exit(ERR);
    }  /* key better be unique */

    if (!strcmp(argv[0], "csem")) {             /* to create a semaphore set */
        printf("\nPlease enter number of semaphores desired > ");
        scanf("%d", &nsems);
        if ((semid =semget(key, nsems, IPC_CREAT | IPC_EXCL | 0666)) == ERR){
            printf("\nCan't create a semaphore set: ");
            printf("Error no = %d\n", errno);
            exit(ERR);
        }
        printf("\nA semaphore set with ID = %d is created OK\n", semid);
    }
```

More Examples #3: Sem (3)

```
    else if (!strcmp(argv[0], "semid")) {       /* get semaphore set ID */
        if ((semid = semget(key, UNUSED, UNUSED)) == ERR) {
            printf("\nSemaphore set with the specified key does not exit: ");
            printf("Error no = %d\n", errno);
            exit(ERR);
        }
        printf("\nSemaphore set ID = %d\n", semid);
    }
    else if (!strcmp(argv[0], "rsem")) {        /* to remove a semaphore set */
        semid = semget(key, UNUSED, UNUSED);    /* get semid for given key */
        if ((semctl(semid, UNUSED, IPC_RMID, UNUSED)) == ERR) {
            printf("\nSemaphore set %d not removed: ", semid);
            printf("Error no = %d\n", errno);
            exit(ERR);
        }
        printf("\nSemaphore set %d is removed\n", semid);
    }
    exit(0);
}  /* main */
```

More Examples #4: Sem (1)

```
/* semctl.c
   This program shows some examples of using semctl().
   The are 4 names linked to this file. These filenames have been chosen
   based on some of the symbolic names that are passed as the 3rd argument
   to semctl(). They are:
     setval:  set value for one semaphore in a set
     getval:  get value for one semaphore in a set
     setall:  set values for all semaphores in a set
     getall:  get values for all semaphores in a set
   The data structure "union semun" is also introduced in this program.
   For some reason, if the declaration for union semun is left out, the
   compiler would complain.
   Also, as one can see, two elements in this union are just pointers.
   Memory allocation is thus needed before use.
*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
```

More Examples #4: Sem (2)

```
#define  ERR        -1
#define  UNUSED      0          /* for padding unused arguments */
#define  SEMMSL     25          /* default max # of sems in a set */

union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
}

int main(int argc, char **argv)
{
    int i, cflag, semid, semval, semnum, nsems;
    struct semid_ds semun_buf;
    ushort semun_array[SEMMSL];
    void semerr(char *);
    union semun sems;

    if (!strcmp(argv[0], "setval"))
        cflag = 1;
    else if (!strcmp(argv[0], "getval"))
        cflag = 2;
```

More Examples #4: Sem (3)

```
else if (!strcmp(argv[0], "setall"))
    cflag = 3;
else if (!strcmp(argv[0], "getall"))
    cflag = 4;

printf("\nPlease enter semid > ");
scanf("%d", &semid);

switch(cflag) {
    case 1:                         /* set one semaphore value */
        printf("\nPlease enter semaphore number > ", i);
        scanf("%d", &semnum);
        printf("\nPlease enter desired value for this semaphore > ", i);
        scanf("%d", &semval);
        sems.val = semval;

        if ((semctl(semid, semnum, SETVAL, sems)) == ERR)
            semerr("\nCan't initialize a semaphore value");
        break;
```

More Examples #4: Sem (4)

```
case 2:                         /* get one semaphore value */
    printf("\nPlease enter semaphore number > ");
    scanf("%d", &semnum);
    if ((semval = semctl(semid, semnum, GETVAL, UNUSED)) == ERR)
        semerr("\nCan't get a semaphore value");

    printf("\nValue of semaphore %d = %d", semnum, semval);
    break;

case 3:                     /* set values for all semaphores  */
    sems.buf = &semun_buf;        /* initialize the buf pointer */
    if ((semctl(semid, UNUSED, IPC_STAT, sems)) == ERR)
        semerr("\nCan't access semaphore set");
    nsems = sems.buf->sem_nsems;  /* get num of semaphores */

    sems.array = semun_array;     /* initialize the array pointer */
    for (i = 0; i < nsems; i++) {
        printf("\nPlease enter value for semaphore %d > ", i);
        scanf("%d", &semval);
        sems.array[i] = semval;
    }  /* for */
    if ((semctl(semid, UNUSED, SETALL, sems)) == ERR)
        semerr("\nCan't initialize a semaphore set");
```

More Examples #4: Sem (5)

```
        case 4:                         /* get all semaphores values */
          sems.buf = &semun_buf;        /* initialize the buf pointer */
          if ((semctl(semid, UNUSED, IPC_STAT, sems)) == ERR)
              semerr("\nCan't access semaphore set");
          nsems = sems.buf->sem_nsems;  /* get num of semaphores */

          sems.array = semun_array;     /* initialize the array pointer */
          semctl(semid, UNUSED, GETALL, sems);
          for (i = 0; i < nsems; i++)
              printf("Value of semaphore %d = %d\n", i, sems.array[i]);
          break;
    }  /* switch */
    exit(0);
}  /* main */

void semerr(char *mesg)
{
    fprintf(stderr, mesg);
    fprintf(stderr, ": errno = %d\n", errno);
    exit(ERR);
}  /* semerr */
```

More Examples #5: Sem (1)

```
/* semop.c
   This program manipulates semop().
   Again, 3 filenames are associated with this program:
     semreg:  request resources
     semrel:  release resources
     semzro:  test for zero
   The struct sembuf data structure introduced in this program is used for
   semaphore operations. The meaning of each element is as follows:
     sem_num   -->  semaphore number in set to be used (0, 1, ..., nsems-1)
     sem_op    -->  operation desired (request, zerotest, release)
     sem_flg   -->  IPC_NOWAIT, SUM_UNDO
   Note: More than one semaphore in a set can be acted upon atomically.
*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define  ERR      -1
#define  UNUSED    0        /* for padding unused arguments */
#define  SEMOPM    10        /* default max # of sem ops at a time */

extern int errno;
```

More Examples #5: Sem (2)

```
int main(int argc, char **argv)
{
    int i, cflag, semid, semnum, semopr;
    unsigned nsops = 1;  /* # of simu sem ops; let's do one at a time */
    struct sembuf sembuf[SEMOPM];
    struct sembuf *semoprs = &sembuf[0];   /* to deal with one at a time */
    void semerr(char *);

    if (!strcmp(argv[0], "semreq"))        /* request resources */
        cflag = 1;
    else if (!strcmp(argv[0], "semrel"))   /* release resources */
        cflag = 2;
    else if (!strcmp(argv[0], "semzro"))   /* test for zero */
        cflag = 3;
    printf("\nPlease enter semid > ");
    scanf("%d", &semid);

    switch(cflag) {
        case 1:                          /*  process requests resources */
            printf("\nPlease enter semaphore number > ");
            scanf("%d", &semnum);
            printf("\nPlease enter number of instances requested > ");
            scanf("%d", &semopr);
```

More Examples #5: Sem (3)

```
        semopr = (- semopr);          /* minus means requested */
        semoprs->sem_num = semnum;  /* set up sembuf elements */
        semoprs->sem_op = semopr;
        semoprs->sem_flg = IPC_NOWAIT; /* otherwise process may block */
        if (semop(semid, semoprs, nsops) == ERR)
            semerr("Can't allocate requested resources");
        else
            printf("Requested resources granted\n");
        break;
    case 2:                         /* process releases resources */
        printf("\nPlease enter semaphore number > ");
        scanf("%d", &semnum);
        printf("\nPlease enter number of resources to be released > ");
        scanf("%d", &semopr);
        semoprs->sem_num = semnum;  /* set up sembuf elements */
        semoprs->sem_op = semopr;
        semoprs->sem_flg = IPC_NOWAIT; /* otherwise process may block */
        if ((semop(semid, semoprs, nsops)) == ERR)
            semerr("Resources not released");
        else
            printf("Resources released\n");
        break;
```

More Examples #5: Sem (4)

```
        case 3:                         /* test for zero */
            printf("\nPlease enter semaphore number being waited > ");
            scanf("%d", &semnum);
            semoprs->sem_num = semnum;  /* set up sembuf elements */
            semoprs->sem_op = 0;   /* test for zero */
            semoprs->sem_flg = IPC_NOWAIT; /* otherwise process may block */
            if (semop(semid, semoprs, nsops))       /* if it is non zero */
                printf("Sem %d is nonzero\n", semnum);
            else
                printf("Sem %d is zero\n", semnum);
            break;
    }  /* switch */
    exit(0);
}  /* main */

void semerr(char *mesg)
{
    fprintf(stderr, mesg);
    fprintf(stderr, ": errno = %d\n", errno);
    exit(ERR);
}  /* semerr */
```

More Examples #6: Sem (1)

```
/* sema.h */
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/wait.h>

enum { ERR    = -1,
       REQ    = -1,          /* need one instance */
       REL    =  1,          /* release one instance */
       UNUSED =  0,          /* for padding unused arguments */
       SEMOPM = 10,          /* default max # of sem ops at a time */
       R1     =  0,          /* reource type 1 */
       R2     =  1,          /* reource type 2 */
       NUMR1  =  1,          /* # of resource type  R1 */
       NUMR2  =  3,          /* # of resource type  R2 */
       SEMID  = 'a',         /* any arbitrary char */
       SEMMSL =  25          /* default max # of sems in a set */
};
char SEMKEY[] = "semkey";    /* pathname used as part of semaphore key */
```

More Examples #6: Sem (2)

```
union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
};
unsigned nsops = 1;
struct sembuf sembuf[SEMOPM], *semoprs = sembuf;

#define semkey(x, y)    (key_t)ftok(x, y)
#define  P(device)      { int pid = getpid();\
                        printf("Process %d requests resources: wait ...\n", \
                        pid); \
                        semoprs->sem_num = device; \
                        semoprs->sem_op = REQ; \
                        semoprs->sem_flg = UNUSED; \
                        semop(semid, semoprs, nsops);\
                        printf("Resources allocated (pid=%d)\n", pid); }
#define  V(device)      { int pid = getpid();\
                        printf("Proc %d  will release resources: ... good news\
                        \n", pid); \
                        semoprs->sem_num = device; \
```

More Examples #6: Sem (3)

```
                semoprs->sem_op = REL; \
                semoprs->sem_flg = UNUSED; \
                semop(semid, semoprs, nsops);\
                printf("Resources released (pid= %d)\n", pid); }

#define  Pn(device, no) { int pid = getpid();\
                printf("Process %d requests resources: wait ...\n", \
                pid); \
                semoprs->sem_num = device; \
                semoprs->sem_op = REQ * no; \
                semoprs->sem_flg = UNUSED; \
                semop(semid, semoprs, nsops);\
                printf("Resources allocated (pid=%d)\n", pid); }
#define  Vn(device, no) { int pid = getpid();\
                printf("Proc %d  will release resources: ... good news\
                \n", pid); \
                semoprs->sem_num = device; \
                semoprs->sem_op = REL * no; \
                semoprs->sem_flg = UNUSED; \
                semop(semid, semoprs, nsops);\
                printf("Resources released (pid=%d)\n", pid); }
/* end of sema.h */
```

More Examples #6: Sem (4)

```
/* sema.c
  Scenairo:
  1. There are NUMRS different types of resources.
     Resource type 1 (R1):  NUMR1 instances.
     Resource type 2 (R2):  NUMR2 instances.
     ...
     Resources are fairly competed for by processes in the system
  2. A process issues P(R-type) to request for R-type resource
          Then enjoys that resource upon allocation.
     It issues V(R-type) to release R-type resource after it is done with it.
  3. Pn(R-type, instances) is used for requesting multiple instances
          of R-type resource.
     Likewise, Vn(R-type, instances) is for releasing multiple
          instances of R-type resource.
  4. If the intended resource is not available at the request time,
          then the requesting process will block.
     It will be awakened by OS when enough instances are ready.
     However, available resources are allocated to all waiting process
          on an early-bird-get-the-worm basis.
*/
```

More Examples #6: Sem (5)

```
#include "sema.h"

int main(void)
{
    int pid;
    struct sembuf sembuf[SEMOPM], *semoprs = sembuf;
    union semun sems;
    key_t key = semkey(SEMKEY, SEMID);        /* no error checking */
    int semid = semget(key, UNUSED, UNUSED);  /* no error checking */

    sems.val=NUMR1;
    semctl(semid, R1, SETVAL, sems);  /* no error checking */
    sems.val=NUMR2;
    semctl(semid, R2, SETVAL, sems);  /* no error checking */

    if (!fork())  {  /* 1st child */
        pid = getpid();
        P(R1);          /* needs one instance of R1 */
            printf("\n\t(%d): GOT it *** \n\t\t*** Using it \n\n", pid);
            sleep(5);
        V(R1);          /* done with it */
            printf("\n\t(%d):DOING something else\n\t\t\t...\n", pid);
```

More Examples #6: Sem (6)

```
    Pn(R2, 2);      /* needs two instances of R2 */
        printf("\n\t(%d): GOT'em *** \n\t\t*** Using'em \n\n", pid);
        sleep(10);
    Vn(R2, 2);      /* done with'em  */
    exit(0);
}

if (!fork())  {  /* 2nd child */
    pid = getpid();
    Pn(R2, 2);      /* needs two instances of R2 */
        printf("\n\t(%d): Got'em ***\n\t\t*** Using'em \n\n", pid);
        sleep(3);
    Vn(R2, 2);      /* done with'em  */
        printf("\n\t(%d):Doing something else\n\t\t\t...\n", pid);
    P(R1);          /* needs one instance of R1 */
        printf("\n\t(%d): Got it ***\n\t\t*** Using it \n\n", pid);
        sleep(6);
    V(R1);          /* done with it */
    exit(0);
}
wait((int *)0);
exit(0);
}  /* main */
```

More Examples #7: Shm (1)

```
/* shmem.c
   This program shows two unrelated processes communicate via shared memory.

   Each time invoked, this program will create a new shared memory segment,
   attach it to the current process, and initialize it with some data.
   Then, fork a child to bring in an unrelated process to change the infor
   stored in the shared memory.
   Next, detach the shared memory from the process.
   Last, remove the shared memory segment created in the process.
   Since a shared memory can be attached to more than one process, a
   semaphore is created in this program to associate it with the shared
   memory segment to prevent two processes accessing the shared memory
   at the same time HOPEFULLY.
*/
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

struct shmdata {        /* a user-defined data structure for shmem */
    int  ndata;
    char sdata[24];
};
```

More Examples #7: Shm (2)

```
union semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
};

int main(void)
{
    int shmid, semid;
    char *shmptr, cshmid[10], csemid[10];   /* 10 is arbitrary */
    pid_t pid;
    union semun sems;
    struct shmdata shmbuf = {               /* some data */
        11,
        "UNIX System Programming"
    };

    /* to get a shared memory identifier */
    shmid = shmget(IPC_PRIVATE, sizeof(struct shmdata), IPC_CREAT|0666);
    if (shmid < 0) {
        fprintf(stderr, "shmget failed\n");
        exit(1);
    }
```

More Examples #7: Shm (3)

```
/* to attach a shared memory segment to the process data area */
/* up to the system to decide where to fit ((char *)0), for r/w (0) */
shmptr = shmat(shmid, (char *)0, 0);
if (shmptr == (char *) -1) {
    fprintf(stderr, "shmat failed\n");
    exit(2);
}

/* to copy data into attached memory segment */
memcpy(shmptr, (char *)&shmbuf, sizeof(struct shmdata));

/* to create a new semaphore set with one semaphore in it */
semid = semget(IPC_PRIVATE, 1, IPC_CREAT | 0666);
if (semid < 0) {
    fprintf(stderr, "semget failed\n");
    exit(3);
}

/* to initialize the 0th semaphore in the set to 1 */
sems.val = 1;
if ((semctl(semid, 0, SETVAL, sems)) < 0) {
    fprintf(stderr, "semctl SETVAL failed\n");
    exit(4);
```

More Examples #7: Shm (4)

```
    /* to fork a child to read/upate the shared memroy */
    switch (pid = fork()) {
        case -1:
            fprintf(stderr, "fork failed\n");
            exit(5);
        case 0:    /* child */
            sprintf(cshmid, "%d", shmid);      /* to convert int to str */
            sprintf(csemid, "%d", semid);
            execl("./shmemexec", "shmemexec", cshmid, csemid, (char *)0);
            exit(6);                          /* in case execl() fails */
        default:   /* parent */
            wait((int *)0);    /* wait for child to terminate */
            printf("In parent: %d   %s\n", ((struct shmdata *)shmptr)->ndata,
                                    ((struct shmdata *)shmptr)->sdata);
            shmdt(shmptr);      /* detach shared memory */
            shmctl(shmid, IPC_RMID, (struct shmid_ds *)0);   /* remove shm ID */
            semctl(semid, 0, IPC_RMID, 0);     /* remove sem ID */
    }
    return(0);
}  /* shmem.c */
```

More Examples #7: Shm (5)

```
/* shmemexec.c
   This program is called from shmem; however, it is a standalone program.

   It gets a shmem ID as its first commandline argument.
   Then, it attachs the shared memory segment to the process.
   First, it reads what is in the shared memory to a buffer; changes one
   piece of data in the buffer; then writes the updated infor back to the
   shared memory
   Next, it detachs the shared memory from the process.

   For mutual exclusion, a semphore is used. It is passed as the second
   commandline argument to the program.
   This program first lock the shared memory with the semaphore associated
   with it before use, and then unlock it after use.
*/
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
```

More Examples #7: Shm (6)

```
struct shmdata {
    int  ndata;
    char sdata[24];
};
struct sembuf lock = {0, -1, IPC_NOWAIT};    /* to make it readable */
struct sembuf unlock = {0, 1, IPC_NOWAIT};

int main(int argc, char *argv[])
{
    int shmid, semid;
    char *shmptr;
    struct shmdata shmbuf;

    sscanf(argv[1], "%d", &shmid);   /* to get shmem ID from argv[1] */
    shmptr = shmat(shmid, (char *)0, 0);
    if (shmptr == (char *) -1) {

        fprintf(stderr, "shmat failed\n");
        exit(1);
    }
```

More Examples #7: Shm (7)

```
    /* to lock the attched memory with a semaphore */
    semid = atoi(argv[2]);              /* another way to convert */
    if (semop(semid, &lock, 1) < 0) {
        fprintf(stderr, "semop lock failed\n");
        exit(2);
    }

    /* to read data from attached memory to buffer */
    memcpy((char *)&shmbuf, shmptr, sizeof(struct shmdata));
    printf("In child: %d   %s\n", ((struct shmdata *)shmptr)->ndata,
                            ((struct shmdata *)shmptr)->sdata);
    shmbuf.ndata = 10;                  /* to change value */

    /* to update shared memory segment */
    memcpy(shmptr, (char *)&shmbuf, sizeof(struct shmdata));
    shmdt(shmptr);                      /* detach memory */
    if (semop(semid, &unlock, 1) < 0) {
        fprintf(stderr, "semop unlock failed\n");
        exit(3);
    }
    return(0);
}  /* shmemexec.c */
```

Home Work #2

- Synchronize *the dining-philosopher problem* using the IPC primitives (Semaphore).
  - There are 5 philosophers running concurrently who spend their lives thinking and eating.
  - Assume that each thinking and eating only takes 1 second.
  - Each philosopher should print short message saying what he does: start thinking, finish thinking, start eating or finish eating.
  - No two neighboring philosophers can eat simultaneously.
  - Is your solution deadlock-free? If it is, is there any possibility of starvation?