Topic 7: IPC

K24: Systems Programming

Interprocess Communication (System V)

- Until now we've learned
  - Sockets
  - Pipes
  - Named pipes (aka FIFO)
  - Signals, etc…
- Pipes are limiting
  - If unnamed
    - Processes must be related
    - Pipe must be created before one of the processes is born
    - File descriptors must be inherited
  - Data typically flows in only one direction
  - Data must be read in order written
    - Cannot prioritize data message types
  - Potential performance hit for large data transfers
- Named pipes
  - Processes don't need to be related
  - But exhibit remaining limitations of regular pipes

Interprocess Communication (System V)

- Three more types of IPC
  - Message Queues
  - Shared Memory
  - Semaphores
  - For IPC on **same machine**

(System V) IPC Characteristics

- Lifetime of IPC objects lasts until kernel reboots.
- Each IPC structure is referred to by a non-negative integer *identifier)*
  - When an IPC is created the program doing the creation provides a *key* of type **key_t**
  - The OS converts this key into an IPC identifier
- There are no inodes or pathnames so traditional file management syscalls (read, write, stat, unlink) unusable

ipcrm and ipcs – commands to remove an IPC object or to see the state of an IPC object

Keys

- Access to an IPC object can be done via an identifier
- You can also refer to an IPC object via a key which remains n in the system even across machine reboots
- Keys should be unique

Keys (cont'd)

- Keys can be created in three ways:
  - Server and client can agree on a pathname to an existing file in the file system AND a project ID (1..255) and then call *ftok*() to convert these two values into a unique key
  - The server creates a new structure by specifying a key of IPC_PRIVATE.
    - Key guaranteed to be unique
    - Client has to become aware of this private key
    - This is often done via a file that is generated by the server and then looked up by the client
  - Server and client agree on a key value (often defined and hard-coded in a header)

ftok()

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

key_t   ftok(const char *pathname, int proj_id);
```

```
if ( (thekey=ftok("/tmp/ad.tempfile",23))  == -1)
        perror("Failed to create key
                from /tmp/ad.tempfile");
```

- Converts a pathname and project identifier to a (Systems V) IPC-key
- File must exist and be accessible to the calling process
- proj_id must be integer between 1 and 255

Access Rights for IPC objects

- Keys identify and provide access to IPC objects which are data structures of type:
  - struct msqid_ds // for message queues
  - struct shmid_ds // for shared memory segments
  - struct semid_ds // for semaphores
- If IPC objects were files could use file permissions to manage access
  - They are NOT, so need separate access right management
  - Looks a lot like file access management
  - 9 bits for read, write, execute (execute bits unused)
- Wrongly accessing resources returns -1

Access Rights for IPC objects

- Access rights for IPC mechanisms: read/write stored in struct ipc_perm

```
Struct ipc_perm {
        uid_t uid;
        gid_t gid;
        uid_t  cuid;
        gid_t cgid;
        mode_t mode;
}
```

- Include:
  - #include <sys/ipc.h>
  - #include <sys/types.h>

Message Queues

- Message queues allow for the exchange of messages between processes.
- The dispatching process sends a specific type of message and the receiving process may request the specific type of message.
- Each message consists of its "*type*" and the "payload".
- Messages are pointers to stuctures:

```
struct message {
    long type;
    char messagetext[MESSAGESIZE;
};
```

- Syscalls for sending/receiving messages take as argument a pointer to a structure like the above
- Header needed: #include<sys/msg.h>

Creating/using a queue with msgget()

int msgget(key_t key, int msgflg)

- returns (creates) a message queue identifier associated with the value of the key argument
- A new message queue is created, if key has the value IPC_PRIVATE
- If key isn't IPC PRIVATE and no message queue with the given key exists, the msgflg must be specified to IPC_CREAT (to create the queue)
- If a queue with key key exists and both IPC_CREAT and IPC_ EXCL are set in msgflg, then msgget() fails with errno set to EEXIST
  - IPC_EXCL is used with IPC_CREAT to ensure failure if the segment already exists.

Use cases of msgflag

- Upon creation, the least significant bits of msgflg define the permissions of the message queue
- These permission bits have the same format and semantics as the permissions specified for the mode argument of open().
- The various use-cases of msgflg are:

	PERMS	PERMS | IPC_CREAT	PERMS | IPC_CREAT | IPC_EXCL
resource exists	use resource	use resource	error
resource does not exist	error	create and use new resource	create and use new resource

Placing a message in a queue with msgsnd()

```
int msgsnd(int msqid, const void *msgp,
                size_t msgsz, int msgflg);
```

- Places message pointed to by msgp in message queue with id msqid; Message must have the following structure:
- ```
  struct msgbuf {
        long mtype ; /* msg type - must be >0 */
        char mtext [MSGSZ]; /* msg data */
  };
  ```
- sender must have write-access permission on the message queue to send a message.
  ```

Fetching a message from a queue with msgrcv()

```
ssize_t msgrcv(int msqid, void *msgp,
                    size_t msgsz, long msgtyp,
                     int msgflg);
```

- Reads a message msgp from a message queue with id msqid
- msgtyp is an integer value
- if msgtyp is zero, the first message is retrieved regardless its type
  - This value can be used by the receiving process to designate message selection (next slide).
- mesgsz specifies the size of the field mtext
- msgflg is mostly set to 0.

The role of msgtyp in msgrcv()

msgtyp specifies the type of message requested as follows:

- if msgtyp=0 then the first message in the queue is read
- if msgtyp > 0 then the first message in the queue of type msgtyp is read.
- if msgtyp < 0 then the first message in the queue with the lowest type value is read.
  - Example: assume a queue has 3 messages with mtype 40, 1, 554 and and msgtyp is set to -554; If msgrcv is called three times, the messages will be received in the following order: 1, 40, 554.

Controlling a queue with msgctl()

int msgctl(int msqid , int cmd, struct msqid_ds *buf

- Performs the control operation specified by cmd on the message queue with identifier msqid
- msqid_ds structure defined in <sys/msg.h> as:

```
struct msqid_ds {
    struct ipc_perm msg_perm; /*Owner & perms */
    time_t msg_stime; /*Time of last msgsnd(2) */
    time_t msg_rtime; /* Time of last msgrcv(2) */
    time_t msg_ctime; /* Time of last change */
    unsigned long __msg_cbytes; /* Current number of
                                   bytes in queue (non -standard)*/
    msgqnum_t msg_qnum; /* Current number of
                                   messages in queue */
    msglen_t msg_qbytes; /* Maximum number of bytes
                                   allowed in queue */
    pid_t msg_lspid; / * PID of last msgsnd(2) */
    pid_t msg_lrpid; /* PID of last msgrcv(2) */
};
```

Operations on message queues

Some values for cmd:

- IPC_STAT: Copy information from the kernel data structure associated with msqid into the msqid_ds structure pointed to by buf.
- IPC_SET: Write the values of some members of the msqid_ds structure pointed to by buf to the kernel data structure associated with this message queue, updating also its msg_ctime element.
- IPC_RMID: Immediately remove the message queue, awakening all waiting reader and writer processes (with an error return and errno set to EIDRM).

Server

```
#include <header file lines omitted here>
…
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L
struct message{
        long mtype;
        char mtext[MSGSIZE];
        };

main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;

        the_key = ftok("/home/ad/K24/MoIC/MesgQueues",
                                226);
        if ( (qid = msgget(the_key,
                                PERMS | IPC_CREAT)) < 0 ){
                perror("megget"); exit(1);
                }
        printf("Creating message queue with
                  identifier %d \n",qid);
```

Server

```
sbuf.mtype = SERVER_MTYPE;
strcpy(sbuf.mtext,"A message from server");
if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0)
        perror("msgsnd"); exit(1);
        }
printf("Sent message: %s\n",sbuf.mtext);

if ( msgrcv(qid, &rbuf, MSGSIZE,
                CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
printf("Received message: %s\n",rbuf.mtext);

if (msgrcv(qid, &rbuf, MSGSIZE, CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
printf("Received message: %s\n",rbuf.mtext);

if (msgctl(qid, IPC_RMID, (struct msqid_ds *)0) < 0){
                perror("msgctl"); exit(1);}
printf("Removed message queue with
          identifier %d\n",qid);
}
```

client1

```
.........
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
        long mtype;
        char mtext[MSGSIZE];
        };
main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;

        the_key = ftok("/home/ad/K24/MoIC/MesgQueues",
                                226);
        if ( (qid = msgget(the_key, PERMS)) < 0 ){
                perror("megget"); exit(1);
                }
```

client1

```
printf("Accessing message queue with
        identifier %d \n",qid);

      if ( msgrcv(qid, &rbuf, MSGSIZE,
                  SERVER_MTYPE, 0) < 0){
           perror("msgrcv"); exit(1);}
      printf("Received message: %s\n",rbuf.mtext);

      sbuf.mtype = CLIENT_MTYPE;
      strcpy(sbuf.mtext,"A message from client 1");

      if (msgsnd(qid, &sbuf,
                  strlen(sbuf.mtext)+1, 0) < 0){
           perror("msgsnd"); exit(1);
           }
      printf("Sent message: %s\n",sbuf.mtext);
}
```

client2

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
        long mtype;
        char mtext[MSGSIZE];
        };

main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;
```

client2

```
the_key = ftok("/home/ad/K24/MoIC/MesgQueues", 226)

if ( (qid = msgget(the_key, PERMS)) < 0 ){
        perror("megget"); exit(1);
        }
printf("Accessing message queue with
           identifier %d \n",qid);

sbuf.mtype = CLIENT_MTYPE;
strcpy(sbuf.mtext,"A message from client 2");

if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0)
        perror("msgsnd"); exit(1);
        }
printf("Sent message: %s\n",sbuf.mtext);
}
```

Output

```
mema@bowser> ./msg-server
Creating message queue with identifier 0
Sent message: A message from server
Received message: A message from client 1
Received message: A message from client 2
Removed message queue with identifier 0
mema@bowser>
```

Output

```
mema@bowser> ./msg-client1
Accessing message queue with identifier 0
Received message: A message from server
Sent message: A message from client 1
mema@bowser>
```

```
mema@bowser> ./msg-client2
Accessing message queue with identifier 0
Sent message: A message from client 2
mema@bowser>
```

Differences with pipes

- Message queues have bidirectional data flow vs pipes (unidirectional)
- Can choose message types to read from the queue (pipes can only read bytes in order)

Developing a Priority Queue

- Implement a Queue in which jobs have Priorities
- A server gets the items from the queue and in some way “processes” these items

“q.h”

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>

#define QKEY          (key_t) 111
#define QPERM          0660
#define MAXOBN          50
#define MAXPRIOR        10

struct q_entry{
        long mtype;
        char mtext[MAXOBN+1];
        };
```

“init_queue.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int warn(char *s){
        fprintf(stderr,"Warning: %s\n",s);
}

int init_queue(void){
        int queue_id;

        if ( (queue_id = msgget(QKEY,
                                IPC_CREAT | QPERM)) == -1 )
                perror("msgget failed");
        return(queue_id);
}
```

“enter.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

/* Prepares a job to be submitted to message queue */
int enter(char *objname, int priority){
        int len, s_qid;
        struct q_entry s_entry;

        if ( (len=strlen(objname)) > MAXOBN){
                warn("name too long\n"); exit(1);
                }
        if ( priority > MAXPRIOR || priority < 0 ){
                warn("invalid priority level"); return(-1);
                }
        if ( (s_qid = init_queue()) == -1 ) return(-1);

        s_entry.mtype= (long)priority;
        strncpy(s_entry.mtext, objname, MAXOBN);

        if (msgsnd(s_qid, &s_entry, len, 0) == -1 ){
                perror("msgsnd failed"); return(-1);}
        else    return(0);
}
```

"etest.c"

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

main(int argc, char *argv[]){
        int priority;

        if ( argc!= 3){
                fprintf(stderr,"usage: %s objname
                                        priority\n",argv[0]);
                }
        if ((priority = atoi(argv[2])) <=0 ||
              priority > MAXPRIOR){
                warn("invalid priority"); exit(2);
                }

        if ( enter(argv[1], priority) < 0 ){
                warn("enter failure"); exit(3);
                }
        exit(0);
}
```

```
gcc enter.c init_queue.c etest.c -o etest
```

“serve.c”

```
#include "q.h"

int serve(void){
        int mlen, r_qid;
        struct q_entry r_entry;

        if ( (r_qid=init_queue()) == -1) return(-1);

      for(;;){
         if ( (mlen=msgrcv(r_qid, &r_entry, MAXOBN,
                 (-1 * MAXPRIOR),
                 MSG_NOERROR) ) == -1 ){
             perror("mesgrcv failed"); return(-1);
         }
         else {
             r_entry.mtext[mlen]='\0';
             proc_obj(&r_entry);
         }
      }
}
```

“stest.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"
extern void serve();
main(){
        pid_t pid;

        switch (pid=fork()){
        case 0: // child
                serve();
                break;
        case -1:
                warn("fork to start the server failed");
                break;
        default:
                printf("server process pid is %d \n", pid);
        }
exit(pid != 1 ? 0 : 1);
}

int proc_obj(struct q_entry *msg){
        printf("\npriority: %ld name: %s\n",
                msg->mtype, msg->mtext);
        }
```

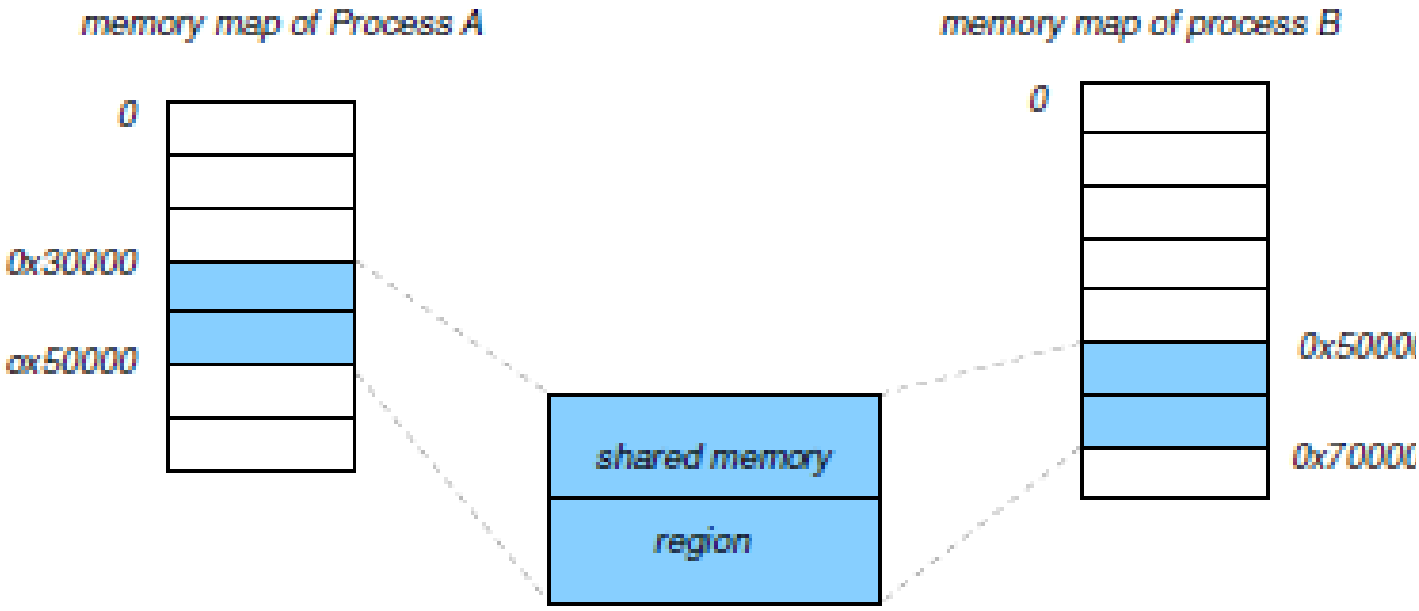
```
gcc stest.c serve.c init_queue.c -o stest
```

```
mema@bowser> ./etest object1 3
mema@bowser> ./etest object2 1
mema@bowser> ./etest object3 6
mema@bowser> ./etest object4 8

mema@bowser> ./stest
server process pid is 2213
priority : 1 name: object2
priority : 3 name: object1
priority : 6 name: object3
priority : 8 name: object4
mema@bowser>
```

Shared Memory

- A shared memory region is a portion of physical memory that is shared by multiple processes
- In this region, structures can be set up by processes and others may read/write on them
- Synchronization among processes using the segment (if required) is achieved with the help of semaphores



Creating shared memory segment with shmget()

include <sys/ipc.h>

include <sys/shm.h>

int shmget (key_t key, size_t size, int shmflg)

- returns the identifier of the shared memory segment associated with the value of the argument key
- the returned size of the segment is equal to *size* rounded up to a multiple of PAGE_SIZE
- shmflg helps designate the access rights for the segment (IPC_CREAT and IPC_EXCL are used in a way similar to that of message queues)
- If shmflg specifies *both* IPC_CREAT and IPC_ EXCL and a shared memory segment already exists for key, then shmget() fails (returns -1 with errno set to EEXIST)

Attaching and Detaching a segment with shmat(), shmdt()

void *shmat(int shmid const void *addr, int flag)

- attaches the shared memory segment identified by shmid to the address space of the calling process
- If shmaddr is NULL, the OS chooses a suitable (unused) address at which to attach the segment (frequent choice)
- Otherwise, addr must be a page-aligned address at which the attach occurs.

int shmdt (const void * addr)

- detaches the shared memory segment located at the address specified by addr from the address space of the calling process.

shmctl()

```
int shmctl (int shmid, int cmd,
                struct shmid_ds *buf)
```

- performs the control operation specified by cmd on the shared memory segment whose identifier is given in shmid
- buf is a pointer to a shmid _ds structure:

```
struct shmid_ds {
    struct ipc_perm shm_perm; /* Owner & perms */
    size_t shm_segsz;   /* Size of segment (bytes)*/l
    tme_t shm_atime;     /* Last attach time */
    time_t shm_dtime; /* Last detach time */
    time_t shm_ctime; /* Last change time */
    pid_t shm_cpid; /* PID of creator */
    pid_t shm_lpid; /* PID of last shmat (2) / shmdt (*/
    shmatt_t shm_nattch ; /* #  current attaches */
    ...
};
```

shmctl()

Usual values for cmd are:

- IPC STAT: copy information from the kernel data structure associated with shmid into the shmid_ds structure pointed to by buf
- IPC_SET: write the value of some member of the shmid_ds structure pointed to by buf to the kernel data structure associated with this shared memory segment, updating also its shm ctime member
- IPC_RMID: mark the segment to be destroyed. The segment will be destroyed after the last process detaches it (i.e., shm_nattch is zero)

Use cases of shared memory calls

- Only one process creates the segment:

```
int id;
id = shmget ( IPC_PRIVATE , 10, 0666) ;
if ( id == -1 ) perror (" Creating ");
```

- Every (interested) process attaches the segment:

```
int *mem;
mem = ( int *) shmat (id , (void *)0, 0);
if ( ( int ) mem == -1 ) perror (" Attachment ");
```

- Every process detaches the segment:

```
int err;
err = shmdt ((void *) mem );
if ( err == -1 ) perror (" Detachment ");
```

- Only one process has to remove the segment:

```
int err;
err = shmctl (id , IPC_RMID , 0);
if ( err == -1 ) perror (" Removal ");
```

Creating and accessing shared memory (sharedMem1.c)

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

int main(int argc, char **argv){
        int id=0, err=0;
        int *mem;

        /* Make shared memory segment */
        id = shmget(IPC_PRIVATE,10,0666);
        if (id == -1)   perror ("Creation");
        else     printf("Allocated. %d\n",(int)id);

        /* Attach the segment */
        mem = (int *) shmat(id, (void*)0, 0);
        if ((int) mem == -1)  perror("Attachment.");
        else
         printf("Attached. Mem contents %d\n",*mem)
```

sharedMemory1.c

```
/* Give it initial value */
        *mem=1;
        printf("Start other process. >"); getchar();
        /* Print out new value */
        printf("mem is now %d\n", *mem);

        /* Remove segment */
        err = shmctl(id, IPC_RMID, 0);
        if (err == -1)   perror ("Removal.");
        else   printf("Removed. %d\n", (int)(err));
        return 0;
}
```

Creating and accessing shared memory (sharedMem2.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char **argv) {
        int id, err;
        int *mem;
        if (argc <= 1) {
                printf("Need shmem id. \n"); exit(1); }

        /* Get id from command line. */
        sscanf(argv[1], "%d", &id);
        printf("Id is %d\n", id);

        /* Attach the segment */
        mem = (int *) shmat(id, (void*) 0,0);
        if ((int) mem == -1) perror("Attachment.");
        else
          printf("Attached. Mem contents %d\n",*mem)
```

sharedMem2.c

```
/* Give it a different value */
        *mem=2;
        printf("Changed mem is now %d\n", *mem);

        /* Detach segment */
        err = shmdt((void *) mem);
        if (err == -1) perror ("Detachment.");
        else printf("Detachment %d\n", err);
        return 0;
}
```

Running the two programs

```
mema@bowser> ./sharedMem1
Allocated. 262145
Attached. Mem contents 0
Start other process. >
```

```
mema@bowser> ./sharedMem2 262145
Id is 262145
Attached. Mem contents 1
Changed mem is now 2
Detachment 0
mema@bowser>
```

```
Start other process. >s
mem is now 2
Removed. 0
mema@bowser>
```

Shared Memory vs Message Queues

- May prefer shared memory when
  - Messages (data) exchanged VERY large (e.g., 100,000 bytes)
  - For performance, to avoid copying messages from user to kernel to user space
- Downside of shared memory
  - Synchronization necessary!

Semaphores

- Fundamental mechanism that facilitates synchronization and coordinated accessing of resources placed in shared memory.
- A semaphore is an integer whose value is never allowed to fall below zero.
- Two operations can be atomically performed on a semaphore:

  1) increment a semaphore value by one (UP or V() ala Dijkstra).

  2) decrement a semaphore value by one (DOWN or P() ala Dijkstra).
- If the value of semaphore is currently zero, then the invoking process will block until the value becomes greater than zero.
- Semaphores are advisory in nature. If a process does not check the semaphore before accessing a resource, chaos may result.

System-V Semaphores

- In general, (System-V) system calls create sets of semaphores:
- The kernel warrants atomic operations on these sets.
- Should we have more than one resources to protect, we can “lock” all of them simultaneously.

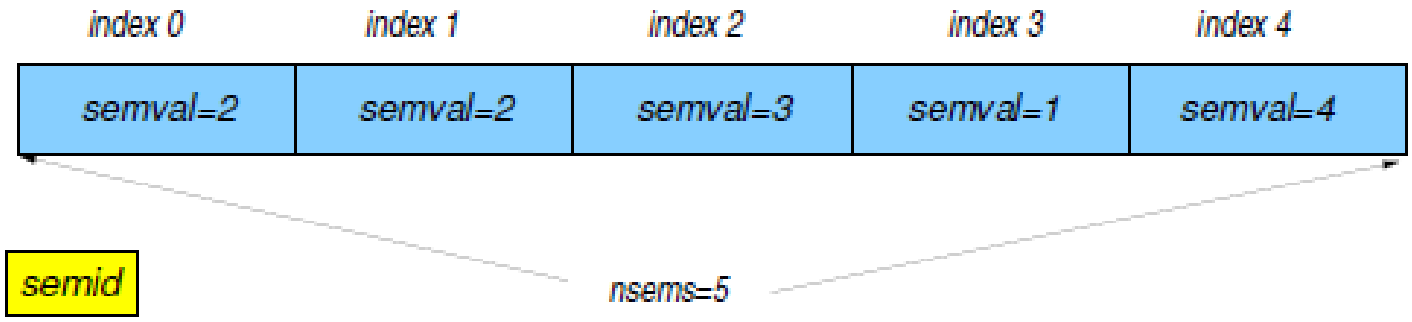
semget

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

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

- returns the semaphore set identifier associated with the argument key
- A new set of nsems semaphores is created if key has the value IPC_PRIVATE **or** if no existing semaphore set is associated with key and IPC_CREAT is specified in semflg
- semflg helps set the access rights for the semaphore set
- If semflg specifies both IPC_CREAT and IPC_EXCL and a semaphore set already exists for key, then semget() fails with errno set to EEXIST

Structure of a Semaphore set



Associated with *EACH* semaphore in the set are the following (among others) values:

- *semval*: the semaphore value, always a positive number
- *sempid*: the pid of the process that last "acted" on the semaphore
- *semcnt:* the number of processes waiting for the semaphore to reach value greater than its current one
- *semzcnt:* the number of processes waiting for the semaphore to reach value zero

Operating on a set of semaphores

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

- performs operations on selected semaphores in the set indicated by semid.
- each of the nsops elements in the array pointed to by sops specifies an operation to be performed on a single semaphore in the set.

Operating on a set of semaphores

- The elements of the struct sembuf are as follows:

```
struct sembuf {
  unsigned short sem_num; /* semaphore # */
  short sem_op; /* semaphore operation */
  short sem_flg; /* operation flags */
};
```

- sem_num identifies the ID of the specific semaphore on the set on which sem_ op operates
- - The value of sem_op is set to:
  - < 0 for locking
  - > 0 for unlocking

- sem_flg often set to 0.

Παράδειγμα Πρόσβασης σε 2 Ταινίες από 3 διεργασίες

```
#include <sys/sem.h>
void setsembuf(strcut sembuf *s, int num, int op,
     int flg) {
   s->sem_num = (short) num;
   s-> sem_op = (short) op;
   s->sem_flg = (short) flg;
   return;
}

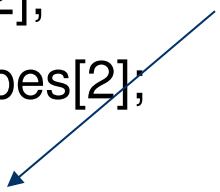
struct sembuf get_tapes[2];
struct sembuf release_tapes[2];

setsembuf(&(get_tapes[0]), 0, -1, 0);
setsembuf(&(get_tapes[1]), 1, -1, 0);
setsembuf(&(release_tapes[0]),0,1,0);
setsembuf(&(release_tapes[1]),1,1,0);
```

- Πρώτο Σημαφόρο (0)

  Ενέργεια = -1

- Σημαία (flag) = 0



Παράδειγμα (συνέχεια)

Προσπάθησε να αφαιρέσεις 1 (P) από τον πρώτο σηματοφόρο

```
Process 1: semop(S, get_tapes, 1);
        <use tape A>
        semop(S, release_tapes, 1);
```

Προσπάθησε να αφαιρέσεις (P) από τους 2 πρώτους σηματοφόρους

```
Process 2: semop(S, get_tapes, 2);
            <use tapes A and B>
            semop(S, release_tapes, 2);
```

Προσπάθησε να αφαιρέσεις 1 (P) από το δεύτερο σηματοφόρο

```
Process 3: semop(S, get_tapes + 1, 1);
            <use tape B>
            semop(S, release_tapes + 1, 1);
```

Προσπάθησε να προσθέσεις 1 (V) στο δεύτερο σηματοφόρο

semget() not enough for initialization

- After a semget() call, semaphores cannot be used immediately
- First they must be “prepared” with a call to semctl()

semctl()

int semctl (int semid , int semnum , int cmd,

[union semun arg])

- performs the control operation specified by cmd on the semnum-th semaphore of the set identified by semid.
- The 4th parameter above –if it exists– has the following layout:

```
union semun {
    int val; /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for
                             IPC_STAT , IPC_SET */
    unsigned short *array; /* Array for GETALL
                              SETALL */
    struct seminfo *__buf; /* Buffer for IPC_INFO
                              ( Linux - specific ) */
};
```

The semid_ds structure

- The semaphore data structure semid ds, is as follows:

```
struct semid_ds {
    struct ipc_perm sem_perm; /* Owner & perms */
    time_t sem_otime; /* Last semop time */
    time_t sem_ctime; /* Last change time */
    unsigned short sem_nsems; /*  # sems in set */
};
```

semctl()

Values for the cmd parameter:

- **IPC_STAT**: copy information from the kernel data structure associated with semid into the semid ds structure pointed to by arg.buf.
- **IPC_SET**: write the value of some member of the semid_ds structure pointed to by arg.buf to the kernel data structure associated with his semaphore set; its sem_ctime member gets updated as well.
- **IPC_SETALL**: Set semval for all semaphores of the set using arg.array, updating also the sem_ctime member of the semid ds structure associated with the set.
- **IPC_GETALL**: Return to semval the current values of all semaphores of the set arg.array.
- **IPC_RMID:** remove the semaphore set while awakening all processes blocked by the respective semop().

Server (for shared memory + semaphore)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

union semnum{
        int val;
        struct semid_ds *buff;
        unsigned short *array;
        };

main(){
        int shmid, semid;
        char line[128], *shmem;
        struct sembuf oper[1]={0,   1,   0};
```

Increase by 1 the value of semaphore #0

Server

```
union semnum arg;

 if ((shmid = shmget (SHMKEY, SHMSIZE,
        PERMS | IPC_CREAT)) < 0) {
        perror("shmget");
        exit(1);
        }

 printf("Creating shared memory with
           ID: %d\n",shmid);

 /* access a SINGLE (1) semaphore from the system */
 if ((semid = semget(SEMKEY, 1,
                  PERMS| IPC_CREAT)) <0) {
        perror("semget");
        exit(1);
        }
 printf("Creating a semaphore with
           ID: %d \n",semid);
```

Server

```
arg.val=0;
/* initialize the semaphore #0's value to value
   arg*/
  if (semctl(semid, 0, SETVAL, arg) <0) {
        perror("semctl");
        exit(1);
        }
  printf("Initializing semaphore to lock\n");

   if ( (shmem = shmat(shmid, (char *)0, 0))
          == (char *) -1) {
        perror("shmem");
        exit(1);
        }
  printf("Attaching shared memory
         segment \nEnter a string: ");
  fgets(line, sizeof(line), stdin);
  line[strlen(line)-1]='\0';

  /* Write message in shared memory */
  strcpy(shmem, line);
```

Server

```
 printf("Writing to shared memory region:
         %s\n", line);

  /* Make shared memory available for reading */
  if ( semop(semid, &oper[0], 1) < 0 ) {
       perror("semop");
       exit(1);
  }
 shmdt(shmem);
 printf("Releasing shared memory region\n");
}
```

Take a single action (3rd argument)

Client

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

main(){
        int shmid, semid;
        char *shmem;
        struct sembuf oper[1]={ 0,  -1,   0};     // down
```

Down semaphore

Client

```
//  get access to shared memory segment
if ((shmid = shmget (SHMKEY, SHMSIZE,
                              PERMS )) < 0) {
              perror("shmget");
              exit(1);
              }
printf("Accessing shared memory
                with ID: %d\n",shmid);

  // accessing the SINGLE  (one) semaphore
     (from the system)
 if ((semid = semget(SEMKEY, 1,
                              PERMS )) <0) {
              perror("semget");
              exit(1);
              }
printf("Accessing semaphore with
          ID: %d \n",semid);
```

Client (continued)

```
if ( (shmem = shmat(shmid, (char *) 0, 0))
                    == (char *) -1 ) {
          perror("shmat");
          exit(1);
          }
   printf("Attaching shared memory segment\n");

   printf("Asking for access to shared memory
             region \n");
   if (semop(semid, &oper[0], 1) <0)  {
          perror("semop");
          exit(1);
          }
   printf("Reading from shared memory
             region: %s\n", shmem);

   /* detach shared memory */
   shmdt(shmem);
```

Take one (1) action

Client (continued)

```
/* destroy shared memory */
if (shmctl(shmid, IPC_RMID,
            (struct shmid_ds *)0 ) <0) {
        perror("semctl");
        exit(1);
        }
printf("Releasing shared segment with
          identifier %d\n", shmid);

/* destroy semaphore set */
if (semctl(semid, 0, IPC_RMID, 0) <0 ) {
        perror("semctl");
        exit(1);
        }
printf("Releasing semaphore with
          identifier %d\n", semid);
}
```

Output

```
mema@bowser> ./sem-server
Creating shared memory with ID: 360449
Creating a semaphore with ID: 819203
Initializing semaphore to lock
Attaching shared memory segment
Enter a string: have a good day!
Writing to shared memory region: have a good day
Releasing shared memory region
```

```
mema@bowser> ./sem-client
Accessing shared memory with ID: 360449
Accessing semaphore with ID: 819203
Attaching shared memory segment
Asking for access to shared memory region
Reading from shared memory region: have a good
day!
Releasing shared segment with identifier 360449
Releasing semaphore with identifier 819203
mema@bowser>
```

Example: Access to Critical Section

```
/* Example code using "POSIX-like" calls for
semaphores and shared memory */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>

/* Union semun */
union semun {
   int val;                          /* value for SETVAL */
   struct semid_ds *buf; /*buffer for IPC_STAT, IPC_SET */
   unsigned short *array;  /* array for GETALL, SETALL */
};
void free_resources(int shm_id, int sem_id) {
   /* Destroy the shared memory segment */
   shmctl(shm_id,IPC_RMID,NULL);
   /* Destroy the semaphore */
   semctl(sem_id,0,IPC_RMID,0);
}
```

Example: Access to Critical Section

```
/* Semaphore P - down operation, using semop
    [simulates sem_wait call in POSIX for reserving
      a resource.] */
int sem_P(int sem_id) {
   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = -1;
   sem_d.sem_flg = 0;
   if (semop(sem_id, &sem_d, 1) == -1) {
       perror("# Semaphore down (P) operation ");
       return -1;
   }
   return 0;
}
```

Example: Access to Critical Section

```
/* Semaphore V - up operation, using semop
   [simulates sem_post call in POSIX for releasing
    a resource.] */
int sem_V(int sem_id) {

   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = 1;
   sem_d.sem_flg = 0;
   if (semop(sem_id, &sem_d, 1) == -1) {
       perror("# Semaphore up (V) operation ");
       return -1;
   }
   return 0;

}
```

Example: Access to Critical Section

```
/* Semaphore Init - set a semaphore's value
    to val */
int sem_Init(int sem_id, int val) {

   union semun arg;

   arg.val = val;
   if (semctl(sem_id, 0, SETVAL, arg) == -1) {
      perror("# Semaphore setting value ");
      return -1;
   }
   return 0;

}
```

Example: Access to Critical Section

```
int main () {
   int shm_id;
   int sem_id;
   int t = 0;
   int *sh;
   int pid;

   /* Create a new shared memory segment */
   shm_id = shmget(IPC_PRIVATE,sizeof(int),
                              IPC_CREAT | 0660);
   if (shm_id == -1) {
       perror("Shared memory creation");
       exit(EXIT_FAILURE);
   }
   /* Create a new semaphore id */
   sem_id = semget(IPC_PRIVATE, 1,
                              IPC_CREAT | 0660);
   if (sem_id == -1) {
       perror("Semaphore creation ");
```

Example: Access to Critical Section

```
    shmctl(shm_id,IPC_RMID,
        (struct shmid_ds *)NULL);
    exit(EXIT_FAILURE);
}

/* Set the value of the semaphore to 1 */
if (sem_Init(sem_id, 1) == -1) {
    free_resources(shm_id,sem_id);
    exit(EXIT_FAILURE);
}

/* Attach the shared memory segment */
sh = (int *)shmat(shm_id,NULL,0);
if (sh == NULL) {
    perror("Shared memory attach ");
    free_resources(shm_id,sem_id);
    exit(EXIT_FAILURE);
}
/* Setting shared memory to 0 */
*sh = 0;
```

```
/* New process */
 if ((pid = fork()) == -1) {
     perror("fork");
     free_resources(shm_id,sem_id);
     exit(EXIT_FAILURE);
 }

 if (pid == 0) {
    /* Child process */
    printf("# I am the child process with
              process id: %d\n", getpid());
 } else {
    /* Parent process */
    printf("# I am the parent process with
              process id: %d\n", getpid());
    sleep(2);
 }

 printf("(%d): trying to access the critical
            section\n", getpid());
 sem_P(sem_id);
 printf("(%d): accessed the critical
           section\n", getpid());
```

```
    (*sh)++;
    printf("(%d): value of shared memory is
            now: %d\n", getpid(), *sh);

    printf("(%d): getting out of the critical
            section\n", getpid());
    sem_V(sem_id);

    printf("(%d): got out of the critical
             section\n", getpid());

    /* Child process */
    if (!pid)
      exit(EXIT_SUCCESS);

    /* Wait for child process */
    wait(NULL);

    /* Clear recourses */
    free_resources(shm_id,sem_id);
    return 0;
}
```

Output

```
mema@bowser> ./posix-sample
# I am the child process with process id: 9109
(9109): trying to access the critical section
(9109): accessed the critical section
(9109): value of shared memory is now: 1
(9109): getting out of the critical section
(9109): got out of the critical section
# I am the parent process with process id: 9108
(9108): trying to access the critical section
(9108): accessed the critical section
(9108): value of shared memory is now: 2
(9108): getting out of the critical section
(9108): got out of the critical section
mema@bowser>
```

Semaphore challenges

Like threads, synchronizing processes via semaphores entails three challenges:

1. Ensuring mutual exclusion when using a resource
2. Avoiding deadlocks
3. Avoiding starvation (where a process is never able to access the resource)

Locking a File

- Another synchronization mechanism between processes
- File = resource
  - Could use semaphores to synchronize access by processes to files
  - But Unix supplies special syscall in case of files called *fcntl*

Locking a file

read – write locks (on sections of) files.

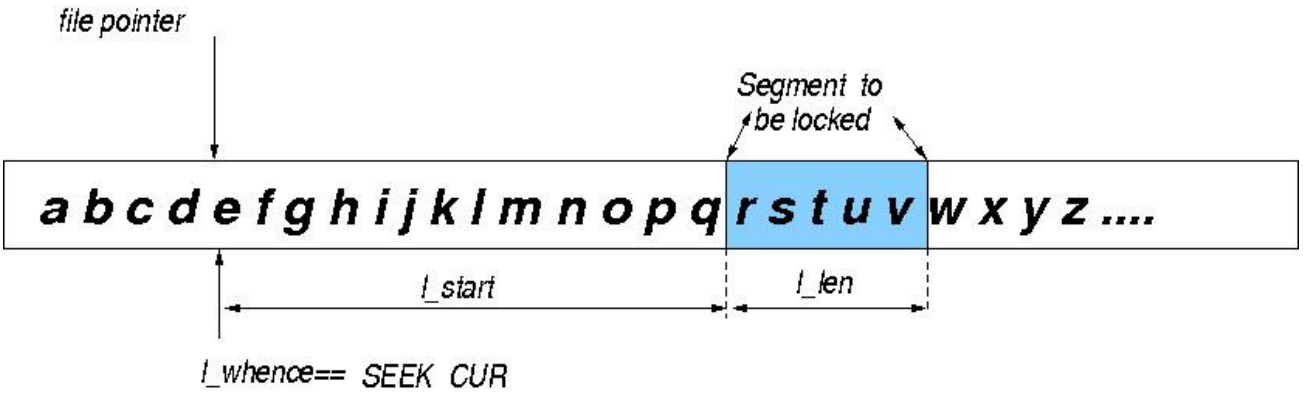
#include <fnctl.h>

int fcntl(int filedes, int cmd, struct flock *ldata)

Notes:

- The file filedes must be opened in the same mode as specified in the lock (see below).
- The cmd can be one of the three:
  - F_GETLK: get lock from data returned from ldata (i.e., check if there is a lock already on it)
  - F_SETLK: obtain lock on a file; return immediately if this is not feasible.
  - F_SETLKW: obtain lock on a file. Block, if lock held by another process.
- The flock structure defined in <fnctl.h> includes:
  - short l_type; /* describes lock type: F_RDLCK, F_WRLCK, F_UNLCK */
  - short l_whence; /*SEEK_SET, SEEK_CUR, SEEK_END (determines where l_start field starts) */
  - off_t l_start; // starting offset, relative to l_whence
  - off_t l_len; // segment size in bytes
  - pid_t l_pid; // ID of process dealing with the lock

Locking a file



l_whence: can be SEEK_SET, SEEK_CUR or SEEK_END
l_start: start position of the segment
l_len: segment in bytes

The l_type (lock type) can be:
- F_RDLCK: lock to be applied is *read*
- F_WRLCK: lock to be applied is *write*
- F_UNLCK: lock on specified segment to be removed

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main( ){
        int fd;
        struct flock my_lock;

        my_lock.l_type = F_WRLCK;
        my_lock.l_whence =  SEEK_SET;
        my_lock.l_start = 0 ;
        my_lock.l_len= 10;

        fd=open("locktest", O_RDWR);

        // lock first 10 bytes
        if ( fcntl(fd, F_SETLKW, &my_lock) == -1 ){
                perror("parent: locking");
                exit(1);
                }

        printf("parent: locked record \n");
```

```
switch(fork()){
    case -1:
        perror("fork"); exit(1);
    case 0:
        printf("child: trying to lock file \n");
        my_lock.l_len = 5 ;
        if ( (fcntl(fd, F_SETLKW, &my_lock)) == -1 ){
                perror("child: problem in locking");
                exit(1);
        }
        printf("child: locked \n"); sleep(1);
        printf("child: exiting \n");
        fflush(stdout); fflush(stderr); exit(1);
    default:
       printf("parent: just about unlocking now \n");
       sleep(5);
       my_lock.l_type = F_UNLCK;
       printf("parent: unlocking -now- \n");
       if ( fcntl(fd, F_SETLK, &my_lock) == -1 ){
              perror("parent: problem in unlocking! \n");
              exit(1); }
       printf("parent: has unlocked and is now exiting \n");
       fflush(stdout); fflush(stderr); wait(NULL);
       }
       sleep(2); }
```

Execution outcome

```
mema@bowser> ./lockit
parent: locked record
child: trying to lock file
parent: just about unlocking now
parent: unlocking -now-
child: locked
parent: has unlocked and is now exiting
child: exiting
mema@bowser>
```

Deadlock possible

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main( ){
        int fd;
        struct flock first_lock;
        struct flock second_lock;

        first_lock.l_type = F_WRLCK;
        first_lock.l_whence =  SEEK_SET;
        first_lock.l_start = 0 ;
        first_lock.l_len= 10;

        second_lock.l_type = F_WRLCK;
        second_lock.l_whence =  SEEK_SET;
        second_lock.l_start = 10;
        second_lock.l_len= 5;

        fd=open("locktest", O_RDWR);
        if ( fcntl(fd, F_SETLKW, &first_lock) == -1 )
                perror("-A:");
       printf("A: lock obtained by process %d \n",getpid());
```

Deadlock possible

```
switch(fork()) {
   case -1:
        perror("error on fork");
        exit(1);
   case 0: /* child */
        if ( fcntl(fd, F_SETLKW, &second_lock) == -1)
            perror("-B:");
        printf("B: lock obtained by process
                    %d\n",getpid());

        if ( fcntl(fd, F_SETLKW, &first_lock) == -1 ){
            perror("-C:");
            printf("Process %d terminating\n",
                     getpid());
            exit(1);
        }
        else printf("C: lock obtained by process
                      %d\n",getpid());
        printf("Process %d successfully acquired
                 BOTH locks \n", getpid());
        exit(0);
```

Deadlock possible

```
default:   /* parent */
   printf("Parent process %d sleeping \n",getpid());
   sleep(10);
   // returns immediately – F_SETLK
   if ( fcntl(fd, F_SETLK, &second_lock) == -1 ){
        perror("--D:");
        printf("Process %d about to terminate\n",getpid());
   }
   else printf("D: lock obtained by process
                    %d\n",getpid());
   sleep(1);
   printf("Process %d on its way out of here
            \n",getpid());
}
   }
```

Execution outcome

```
mema@bowser> ./deadlock
A: lock obtained by process 18979
B: lock obtained by process 18980
Parent process 18979 sleeping
--D:: Resource temporarily unavailable
Process 18979 about to terminate
Process 18979 on its way out of here
C: lock obtained by process 18980
Process 18980 successfully acquired BOTH locks
mema@bowser>
```