Using Named Pipes

```
EXAMPLE:
#include <stdio.h>
#include <fcntl.h>

  main()
  {

    int     pfd;
    if(mkfifo("/tmp/my_ipc", 0666) == -1){
       perror("mkfifo failed: ");
       exit(2);
    }
    printf("Successfully made FIFO  /tmp/my_ipc\n");
  }

FROM THE SHELL:
  % ls -l /tmp/my_ipc

    prw-rw-rw-   1 bill     fac   0 Dec 15 20:42 my_ipc
```

System V Interprocess Communication

Since pipe based ipc is somewhat limited due to the half duplex, stream only model, there was an early cry from the UNIX user community for a more featured set of ipc mechanism. With the advent of System V Unix in 1983, AT&T delivered a new set of ipc mechanism which included:

- A full-duplex datagram message queue facility
- Shared memory segment support
- Semaphore synchronization primitives

These facilities were the first in UNIX to use the signed integers for a name space (the file system name space had been used for all other mechanisms), and were designed to be *semi-persistent* global objects.

System V IPC (Cont'd)

The System V IPC mechanisms were designed as a set of interrelated functions which all shared a common interface style:

- Each mechanism has a kernel table for all instances
- Each mechanism uses a user-chosen numeric **key** for identification
- Each mechanism has a **get** call to translate the key into an entry identifier or **id**
- A key is an integer or the defined value IPC_PRIVATE from ipc.h
- Each get call has an integer flags field where 9 standard protection bits are or'd with the possible defined flags:
  - IPC_CREAT create if non-existent
  - IPC_EXCL fail if existent
- Each entry includes the UID and GID of the creating process, initially designated as creator and owner IDs, and owner can be redesignated
- Each entry contains status information, all of which can be read and some of which can be modified by a process with EUID of creator or owner using a control call

System V IPC (Cont'd)

```
SYNOPSIS
Get message queue identifier
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/msg.h>

       int   msgget (key, msgflg)
       key_t key;
       int   msgflg;
where:
  key      A user-defined name for the message
           queue, either an integer or the
           constant  IPC_PRIVATE
  msgflg   A set of flags indicating the requested
           permission state of the message queue,
           whether a new message queue should be
           created, and whether the message queue
           should be held exclusively:

                  possible flags values
0                              connect, do not create
IPC_CREATE | 0666              connect or create and
                               connect
IPC_CREATE | IPC_EXCL | 0666   create and connect,
                               error if already exists

returns:  msqid, a non-negative integer that
          identifies the message queue for key, or -1
```

System V IPC (Cont'd)

```
Send a message
Receive a message
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/msg.h>

       int    msgsnd (msqid, msgp, msgsz, msgflg)

       int    msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
       int    msqid;
       void  *msgp;
       size_t msgsz;
       long   msgtyp;
       int    msgflg;
   where:
     msqid    A message queue identifier
     msgp     The message buffer of the message to be
              sent
     msgsz    The size in bytes of the mtext portion of
              the message buffer
     msgtyp   Message type, 0, positive, negative
     msgflg   A set of flags modifying the action of
              msgsnd
                     possible flag values
           IPC_NOWAIT    don't wait for message
           MSG_NOERROR   truncate if buffer too
                         small
                       message object
           struct mymsg{
              int  type;
              char mtext[SOMESIZE]
           }
  returns:   0 on success, or -1
```

System V IPC (Cont'd)

```
‘
Get or set message queue attributes or destroy
a message queue
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/msg.h>

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

   where:
       msqid     A message queue identifier
       cmd       The message control operation to
                 be performed
       buf       The address of a message queue
                 attribute record (used only
                 if cmd is IPC_STAT or IPC_SET)

                      possible cmd values
      IPC_STAT  fill in buf with current attributes
      IPC_SET   use buf to modify selected attributes
      IPC_RMID  remove the msgqid from the system and
                discard all outstanding messages

  returns:   0 on success, or -1
```

System V IPC (Cont'd)

```
EXAMPLE:
A message client sends a message to a
server and receives a reply

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

#define MSGKEY  (1234567)

typedef struct{
  int  mtype;
  char body[252];
} MSG;

int main(int argc, char * argv[]){
  MSG amessage;
  int *int_ptr, pid, msg_id, msg_type=2;

  if((msg_id = msgget((key_t)MSGKEY, 0)) == -1){
     perror("msgget failed :");
     exit(1);
  }
  pid = getpid();
  int_ptr = (int *)amessage.mtext;
  *int_ptr = pid;
  amessage.mtype = msg_type;
  if(msgsnd(msg_id, &amessage,
            sizeof(amessage.mtext), 0) == -1){
     perror("msgsnd failed :");
     exit(1);
  }
  msg_type = pid;
  if(msgrcv(msg_id, &amessage,
            sizeof(amessage.mtext), msg_type, 0) == -1){
     perror("msgrcv failed :");
     exit(1);
  }
  printf("Server pid is %d\n", *(int *)amessage.mtext);
}
```

System V IPC (Cont'd)

EXAMPLE:
A message sever receives a message from a client
and returns a reply

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#define MSGKEY  (1234567)
typedef struct{
  int  mtype;
  char body[252];
} MSG;

int msg_id;
void  sig_handler(int signum){
  printf("going down on signal %d\n", signum);
  msgctl(msg_id, IPC_RMID, 0);
  exit(1);
}

int main(int argc, char * argv[]){
  MSG amessage;
  int *int_ptr, pid, msg_type=2;
   struct sigaction   new;
   sigset_t   mask_sigs;
   int  i, nsigs;
   int sigs[] = {SIGHUP,SIGINT,SIGQUIT, SIGPIPE
                 SIGTERM, SIGBUS, SIGSEGV, SIGFPE};

   nsigs = sizeof(sigs)/sizeof(int)
   sigemptyset(&mask_sigs);
   for(i=0; i< nsigs; i++)
       sigaddset(&mask_sigs, sigs[i]);       .
   for(i=0; i< nsigs; i++){
      new.sa_handler = sig_handler;
      new.sa_mask = mask_sigs;
      new.sa_flags = SA_RESTART;
      if(sigaction(sigs[i], &new, NULL) == -1){
         perror("can't set signals: ");
         exit(1);
      }
   }
```

System V IPC (Cont'd)

EXAMPLE:
A message sever receives a message from a client
and returns a reply cont'd

```
  if((msg_id = msgget((key_t)MSGKEY, IPC_CREATE | 0666)) == -1){
     perror("msgget failed :");
     exit(1);
  }
  for(;;){
    if(msgrcv(msg_id, &amessage,
            sizeof(amessage.mtext), msg_type, 0) == -1){
       perror("msgrcv failed :");
       exit(1);
    }
    printf("Client pid is %d\n", (pid = *(int *)amessage.mtext));
    amessage.mtype = pid;
    pid = getpid();
    int_ptr = (int *)amessage.mtext;
    *int_ptr = pid;
    if(msgsnd(msg_id, &amessage,
            sizeof(amessage.mtext), 0) == -1){
       perror("msgsnd failed :");
       exit(1);
    }
  }
}
```

System V IPC (Cont'd)

```
Get shared memory segment
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/shm.h>

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

where:
  key     Key identifying shared memory segment
  size    Size in bytes of the shared memory
          segment
  shmflg  Access and option shmflgs.

                    possible flags values
0                              get id, do not create
IPC_CREATE | 0666              get id or create and
                               get id
IPC_CREATE | IPC_EXCL | 0666   create and get id, error
                               if already exists

returns:   shmid, a non-negative integer that
           identifies the shared memory segment
           associated with key or -1
```

System V IPC (Cont'd)

```
Attach a shared memory segment
Detach a shared memory segment
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/shm.h>

       void * shmat (shmid, shmaddr, shmflg)
       int  shmid;
       void * shmaddr;
       int  shmflg;
where:
  shmid    The shared memory identifier of the
           shared segment to attach
  shmaddr  The byte address at which to attach the
           shared segment (may be defaulted to a
           system-selected value or rounded to a
           system-specified address boundary)
  shmflg   SHM_RND and/or SHM_RDONLY, an option flag
           used to select between the various options
           for shmaddr and to choose read-only or
           read-write access to the shared memory segment
returns:   the attached address within the process address
           space or -1 (should cast to int for testing)
SYNOPSIS
       int  shmdt (shmaddr)
       void * shmaddr;
where:
  shmaddr  The byte address of the attached shared
           memory segment to be detached.  This must equal
           the value returned by shmat when the shared
           segment was attached.
returns:   0 on success, or -1
```

System V IPC (Cont'd)

```
Shared memory control operations
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/shm.h>

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

where:
  shmid  The shared memory identifier of
         the shared area to be operated on
  cmd    The specific shared memory operation
         (IPC_STAT, IPC_SET, or IPC_RMID)
  buf    The address of a shared memory
         structure to be used in the operation
         (used only if command is IPC_STAT or
         IPC_SET)

returns:   0 on success, or -1
```

System V IPC (Cont'd)

```
EXAMPLE:
A process maps a shared memory segment
at two different virtual locations

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

#define SHMKEY  (7654321)

int shm_id;
void  sig_handler(int signum){
  printf("going down on signal %d\n", signum);
  shmctl(shm_id, IPC_RMID, 0);
  exit(1);
}

int main(int argc, char * argv[]){
   int    cntr, *addr1, *addr2;
   struct sigaction   new;
   sigset_t   mask_sigs;
   int  i, nsigs;
   int sigs[] = {SIGHUP,SIGINT,SIGQUIT, SIGPIPE
                 SIGTERM, SIGBUS, SIGSEGV, SIGFPE};

   nsigs = sizeof(sigs)/sizeof(int)
   sigemptyset(&mask_sigs);
   for(i=0; i< nsigs; i++)
       sigaddset(&mask_sigs, sigs[i]);       .
   for(i=0; i< nsigs; i++){
      new.sa_handler = sig_handler;
      new.sa_mask = mask_sigs;
      new.sa_flags = SA_RESTART;
      if(sigaction(sigs[i], &new, NULL) == -1){
         perror("can't set signals: ");
         exit(1);
      }
   }
```

System V IPC (Cont'd)

EXAMPLE:
A process maps a shared memory segment
at two different virtual locations cont'd

```
  if((shm_id = shmget((key_t)SHMKEY, 1024,
                          IPC_CREATE | 0666)) == -1){
     perror("msgget failed :");
     exit(1);
  }
  if((addr1 = (int *)shmat(shm_id, NULL, 0)) == -1){
       perror("first shmat failed :");
       sig_handler(-1);
  }
  if((addr2 = (int *)shmat(shm_id, NULL, 0)) == -1){
       perror("second shmat failed :");
       sig_handler(-1);
  }
  printf("first attach at %x, second attach at %x\n",
            (int)addr1, (int)addr2);
  for(cntr=0; cntr < 256; cntr++)*(addr1+cntr)=cntr;
  *addr1=256;
  for(cntr=0; cntr < 256; cntr++)
      printf("location %x has value %d\n",
                (int)(addr2+cntr), *(addr2+cntr));
  pause();
}
```

System V IPC (Cont'd)

EXAMPLE:
A process maps a shared memory segment
created by another process

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

#define SHMKEY  (7654321)

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

  if((shm_id = shmget((key_t)SHMKEY, 0, 0)) == -1){
     perror("msgget failed :");
     exit(1);
  }
  if((addr = (int *)shmat(shm_id, NULL, 0)) == -1){
       perror("shmat failed :");
       exit(1);
  }
  while(*addr == 0);
  for(cntr=0; cntr < 256; cntr++)
      printf("location %x has value %d\n",
                (int)addr, *addr);
  }
}
```

System V IPC (Cont'd)

```
Get a set of semaphores
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>

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

where:
 key     Key identifying  a semaphore set
 nsems   The requested number of semaphores in
         the semaphore set
 semflg  Access and option flags

                    possible flags values
0                                get id, do not create
IPC_CREATE | 0666                get id or create and
                                 get id
IPC_CREATE | IPC_EXCL | 0666     create and get id, error
                                 if already exists

returns:   semid, a non-negative integer that
              identifies the set of semaphores
              associated with key or -1
```

System V IPC (Cont'd)

```
Semaphore operations
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>

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

where:
    semid     A semaphore set identifier
    sops      An array of semaphore operations
              to perform
    nsops     The number of semaphore operation
              records in sops

struct sembuf {
     short   sem_num;    /* semaphore # */
     short   sem_op;     /* semaphore operation */
     short   sem_flg;    /* operation flags */
};
                possible flags values
IPC_NOWAIT  don't wait for a synchronous operation
SEM_UNDO    keep a balance on semaphore operations

                possible operations
sem_op      positive: always succeeds, add to counter
sem_op      zero: block until counter is zero
sem_op      negative: block until counter can be
            subtracted without becoming negative

returns:   0 on success, or -1
```

System V IPC (Cont'd)

```
Semaphore control operations
SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>
       #include <sys/sem.h>

       int   semctl (semid, semnum, cmd, arg)
       int   semid;
       int   semnum;
       int   cmd;
       union semun {
           int val;
           struct semid_ds *buf;
           unsigned short *array;
       }arg;

where:
  semid    A semaphore set identifier
  semnum   The subject semaphore (used only if cmd
           is GETVAL, SETVAL, GETNCNT, GETZCNT or GETPID)
  cmd      The semaphore operation to be performed
  arg      An argument (used only if cmd is SETVAL,
           GETALL, SETALL, IPC_STAT, OR IPC_SET)

returns:    0 or value on success, or -1
```

System V IPC (Cont'd)

Semaphore control operations cont'd

possible commands

SETVAL	Set the value of semaphore number semnum to arg.val. If an error occurs, the semaphore set is unchanged.
GETVAL	Return the value of semaphore number semnum.
GETPID	Return the process id of the last process to perform an operation on semaphore number semnum.
GETNCNT	Return the number of processes waiting for the value of semaphore number semnum to increase.
GETZCNT	Return the number of processes waiting for the value of semaphore number semnum to become zero.
SETALL	Set the value of all semaphores in the specified semaphore set to the values contained in the array pointed to by arg.array. If an error occurs, the semaphore set is unchanged.
GETALL	Return the value of all semaphores in the specified semaphore set using the array pointed to by arg.array.
IPC_STAT	The current semaphore set attributes are stored in the structure pointed to by arg.buf.
IPC_SET	The following semaphore set attributes are set to the values found in the structure pointed to by arg.buf: user id (sem_perm.uid), group id (sem_perm.gid), and permission rights (in sem_perm.mode).
IPC_RMID	The semaphore set is destroyed.

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
Header file:

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

#define SEMKEY (key_t)5763
#define MEMKEY (key_t)5763
#define NUMSLOTS       50
#define PROD           0
#define CONSUMER       1
#define OUTPTR         2

struct donut_ring{
    int  donut_slot[NUMSLOTS];
    int  outptr;
    int  empty_slots;
};

struct donut_ring *shared_ring;
int    shmid, semid;

void   handler();
void   semsetval();
void   p();
void   v();
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
The utility file:

```
#include "donuts.h"

void    p(semid)
        int semid;
{
        struct sembuf semopbuf;
        semopbuf.sem_num=0;
        semopbuf.sem_op=(-1);
        semopbuf.sem_flg=0;
        if(semop(semid, &semopbuf,1) == -1){
                perror("p operation failed: ");
                handler(-2);
                exit(2);
        }
}

void    v(semid)
        int semid;
{
        struct sembuf semopbuf;
        semopbuf.sem_num=0;
        semopbuf.sem_op= 1;
        semopbuf.sem_flg=0;
        if(semop(semid, &semopbuf,1) == -1){
                perror("p operation failed: ");
                handler(-2);
                exit(2);
        }
}
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
The utility file cont'd:

```
void    semsetval(semid, value)
        int semid, value;
{
          union semun arg;

          arg.val = value;
          if(semctl(semid, 0, SETVAL, arg.val) == -1){
                perror("semset failed");
                handler(-3);
                exit(3);
          }
}

void    handler(sig)
        int     sig;
{
        int     i,j,k;

        printf("In signal handler with signal # %d\n",sig);
        if(shmctl(shmid, IPC_RMID, 0) == -1){
                perror("handler failed shm RMID: ");
                exit(4);
        }
        if(semctl(semid, 0, IPC_RMID, 0) == -1){
                perror("handler failed sem RMID: ");
                exit(4);
        }
}
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
The producer:

```
int   main(int argc, char *argv[])
{
        int     in_ptr, serial;
        int     i,j,k;
        int     nsigs=15;
        struct sigaction new;
        sigset_t mtset;

        in_ptr = serial = 0;
        sigemptyset(&mtset);
        new.sa_handler = handler;
        new.sa_mask = mtset;
        new.sa_flags = 0;
        for(i=1; i<nsigs; i++)
                sigaction(i, &new, NULL);
        if((shmid=shmget(MEMKEY, sizeof(struct donut_ring),
                        IPC_CREAT | 0666)) == -1){
                perror("shared get failed: ");
                exit(1);
        }
        if((shared_ring=(struct donut_ring *)shmat(shmid, 0, 0)) ==
                                        (struct donut_ring *)-1){
                perror("shared attach failed: ");
                handler(-1);
        }
        shared_ring->empty_slots = 50;
        shared_ring->outptr = 0;
        if((semid = semget(SEMKEY, 1 , IPC_CREAT | 0666)) == -1){
                perror("semaphore allocation failed: ");
                handler(-1);
        }
        semsetval(semid, 1);

        printf("just before while\n");
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
The producer cont'd:

```
        while(1){
          printf("in producer with serial = %d\n",serial);
          p(semid);
          if(shared_ring->empty_slots == 0){
                v(semid);
                continue;
          }
          shared_ring->donut_slot[in_ptr] = serial;
          in_ptr = (in_ptr+1) % NUMSLOTS;
          serial++;
          shared_ring->empty_slots--;
          printf("prod serial %d\n",serial);
          v(semid);
        }
}
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer problem
The consumer:

```
#include "donuts.h"

main(int argc, char *argv[]){
        int     i,j,k,donut;

        if((shmid=shmget(SEMKEY, sizeof(struct donut_ring),
                                0)) == -1){
                perror("shared get failed: ");
                exit(1);
        }
        if((shared_ring=(struct donut_ring *)shmat(shmid, 0, 0)) ==
                                                (struct donut_ring *)-1){
                perror("shared attach failed: ");
                exit(1);
        }
        if((semid=semget(SEMKEY, 1, 0)) == -1){
                perror("semaphore allocation failed: ");
                exit(1);
        }
        for(i=0; i<10; i++){
           for(k=0; k<12; k++){
                printf("in consumer with i = %d and k = %d\n", i, k);
                p(semid);
                  if(shared_ring->empty_slots == NUMSLOTS){
                        v(semid);
                        k--;
                        continue;
                  }
                donut=shared_ring->donut_slot[shared_ring->outptr];
                shared_ring->outptr =
                        (shared_ring->outptr+1) % NUMSLOTS;
                shared_ring->empty_slots++;
                v(semid);
                printf("donut serial number %d\n",donut);
           }
        printf("dozen number %d completed \n\n",i);
        }
}
```