Day-24

**Pipes:**

* pipes are oldest form of unix system IPC(inter process communication) and are provided by all unix system.
* Both ends are having descriptors (read and write).

**Limitations:**

* Historically, they have half duplex (i.e. data flow in only one direction). Some systems now provide full-duplex pipes, but for maximum portability, we should never assume that this is the case
* Pipes can be used only between processes that have a common ancestor. Normally, a pipe is created by a process, that process calls fork, and the pipe is used between the parent and the child
* A pipe is created by calling the pipe function.

#include<unistd.h>

int pipe(int fd[2]);

Returns: 0 if OK, −1 on error

* Two file descriptors are returned through the
* fd argument:
* fd[0] is open for reading, and
* fd[1] is open for writing.
* The output of fd[1] is the input for fd[0].

|  |
| --- |
| user69@trainux01:~/day24$ vi pipe.c  user69@trainux01:~/day24$ gcc pipe.c  user69@trainux01:~/day24$ ./a.out  Line Read  Hello world  End of Program  user69@trainux01:~/day24$ cat pipe.c  #include <stdio.h>  #include<stdlib.h>  #include<sys/wait.h>  #include<unistd.h>  #define BUFF 1024  int main() {  int fd[2];  pid\_t cpid;  char line[BUFF];  pipe(fd);  cpid=fork();  if(cpid==0)  {  close(fd[0]); //reading  write(fd[1],"Hello world\n",12);  \_exit(EXIT\_SUCCESS);  }  else  {  close(fd[1]); //writing  read(fd[0],line,BUFF);  printf("\nLine Read\n%s",line);  }  printf("\nEnd of Program\n\n");  return 0;  } |

|  |
| --- |
| #include <stdio.h>  #include<stdlib.h>  #include<sys/wait.h>  #include<unistd.h>  #define BUFF 1024  int main() {  int fd[2];  pid\_t cpid;  char line[BUFF];  int status;  pipe(fd);    cpid=fork();  if(cpid==0)  {  close(fd[0]); //reading  write(fd[1],"Hello world\n",12);  \_exit(EXIT\_SUCCESS);  }  else  {  wait(&status);  printf("\nchild with %d pid exited with status= %d\n",cpid,(status/256));  close(fd[1]); //writing  read(fd[0],line,BUFF);  printf("\nLine Read\n%s",line);  }    printf("\nEnd of Program\n\n");  return 0;  }  **Output:**  child with 66655 pid exited with status= 0  Line Read  Hello world  �z  End of Program |

**WRITE:**

write — send a message to another user

The write utility allows you to communicate with other users, by copying lines from your terminal to theirs.

When you run the write command, the user you are writing to gets a message of the form.

**READ:**

read - read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize\_t read(int fd, void \*buf, size\_t count);

DESCRIPTION

read () attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

* user69@trainux01:~/day24$ **ipcs**

------ Message Queues --------

key msqid owner perms used-bytes messages

------ Shared Memory Segments --------

key shmid owner perms bytes nattch status

------ Semaphore Arrays --------

key semid owner perms nsems

ipcs -q 🡪 message queues

ipcs -s 🡪 Semaphore Arrays

**MSGGET:**

NAME

msgget - get a System V message queue identifier

SYNOPSIS

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int msgget(key\_t key, int msgflg);

DESCRIPTION

The msgget() system call returns the System V message queue identifier associated with the value of the key argument. A new message queue is created if key has the value IPC\_PRIVATE or key isn't IPC\_PRIVATE, no message queue with the given key key exists, and IPC\_CREAT is specified in msgflg.

**MSGOP:**NAME

msgrcv, msgsnd - System V message queue operations

SYNOPSIS

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg);

ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp,int msgflg);

DESCRIPTION

The msgsnd() and msgrcv() system calls are used, respectively, to send messages to, and receive messages from, a System V message queue. The calling process must have write permission on the message queue in order to send a message, and read permission to receive a message.

The msgp argument is a pointer to a caller-defined structure of the following general form:

struct msgbuf {

long mtype; /\* message type, must be > 0 \*/

char mtext[1]; /\* message data \*/

};

$ ./a.out -s

sent: a message at Wed Mar 4 16:25:45 2015

$ ./a.out -r

message received: a message at Wed Mar 4 16:25:45 2015

**FTOK:**

ftok - convert a pathname and a project identifier to a System V IPC key

SYNOPSIS

#include <sys/types.h>

#include <sys/ipc.h>

key\_t ftok(const char \*pathname, int proj\_id);

DESCRIPTION

The ftok() function uses the identity of the file named by the given pathname (which must refer to an existing, accessible file) and the least significant 8 bits of proj\_id (which must be nonzero) to generate a key\_t type System V IPC key, suitable for use with msgget(2), semget(2), or shmget(2).

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <unistd.h>  int main() {  key\_t key = ftok("/home2/user69/day24", 10010);    printf("\n%u", key); //printf(“\n%X”,key) => output : 1A31325C  printf("\n\n");  return 0;  }  Output:  439431772 |

|  |
| --- |
| user69@trainux01:~/day24$ vi t2.c  user69@trainux01:~/day24$ gcc t2.c  user69@trainux01:~/day24$ ./a.out  MSG queue created with 6 id  user69@trainux01:~/day24$ cat t2.c  #include <stdio.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include<sys/msg.h>  #include <unistd.h>  int main() {  int msgid;  msgid =msgget((key\_t)11,IPC\_CREAT);  printf("\nMSG queue created with %d id\n",msgid);  printf("\n\n");  return 0;  } |

Receive : ./Rcv

|  |
| --- |
| user69@trainux01:~/day24$ ./a.out  MSG queue created with 35 id  TO recv msg from msgid = 35  ^C  user69@trainux01:~/day24$ cat t3.c  #include <stdio.h>  #include <stdlib.h>  #include <sys/wait.h>  #include <unistd.h>  #include <signal.h>  #include <sys/ipc.h>  #include <sys/types.h>  #include <sys/msg.h>  #include <string.h>  #define BUFF 1024  #define PERMS 0666  typedef struct msgText{  long msgtype;  char txtMsg[1024];  }MSG;  int main()  {  int msgid, len=0;  MSG msg1;  msgid = msgget((key\_t)15,IPC\_CREAT|PERMS);  if(msgid < 0){  perror("msgget ");  \_exit(EXIT\_SUCCESS);  }  printf("\nMSG queue created with %d id\n",msgid);  printf("\nTO recv msg from msgid = %d\n",msgid);  if(msgrcv(msgid,&msg1,BUFF,0,0)<0)  {  perror("msgrcv ");  return (EXIT\_FAILURE);  }  printf("\nReceived message: Type: %ld\n",msg1.msgtype);  printf("\nReceived message: Msg: %s\n",msg1.txtMsg);  printf("\n\n");  return 0;  } |

Chat application using msg queue

………………………….

**FIFO:**

* FIFOs are sometimes called named pipes.
* Unnamed pipes can be used only between related processes when a common ancestor has created the pipe.
* With FIFOs, however, unrelated processes can exchange data.

**Shared memory:**

* Shared memory allows two or more processes to share a given region of memory.
* This is the fastest form of IPC, because the data does not need to be copied between the client and the server.

|  |
| --- |
| user69@trainux01:~/day24$ ./a.out  Shared mem id =16  Msg = Hello Bhima, WellCome to C Programmming class  user69@trainux01:~/day24$ cat p5.c  #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <stdlib.h>  #include <string.h>  #define PERMS 0666  #define MAXBUF 1024  int main()  {  key\_t key = 10003;  int shmid;  char msg[] = "Hello Bhima, WellCome to C Programmming class";  NULL;  (char \*)malloc(MAXBUF);  shmid = shmget(key,46,PERMS|IPC\_CREAT);  char \*ptr = NULL;  if(shmid < 0)  {  perror("shmget()");  printf("\nDirectly use this shm mem to read and write");  exit(EXIT\_FAILURE);  }  printf("\nShared mem id =%d\n",shmid);  ptr = (char \*)shmat(shmid,(void \*)0,0);  printf("\nMsg = %s\n",ptr);  shmdt(ptr);  return 0;  }  user69@trainux01:~/day24$ |

**SEMGET:**

NAME

semget - get a System V semaphore set identifier

SYNOPSIS

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

int semget(key\_t key, int nsems, int semflg);

DESCRIPTION

The semget() system call returns the System V 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.

………………………………………………………………………………………………….