OPERATING SYSTEMS (PIPES, MESSAGE QUEUES, SHARED MEMORY,THREADS, SEMAPHORES)

PIPES:

srilatha@GESLMP22WP7T:~/Experiments/misc$ cd operating/

srilatha@GESLMP22WP7T:~/Experiments/misc/operating$ ls

pipes semaphore threads

srilatha@GESLMP22WP7T:~/Experiments/misc/operating$ cd pipes/

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ ls

1pipe 1pipe.c child.txt create create.c input.txt msgqueues openpipe.c output.txt parent.txt pipe pipe.c pipesclass shared_mem

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat 1pipe.c

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/wait.h>
//#include <avr/io.h>
#include<string.h>
#define BUF_SIZE 10

int main(int argc, char *argv[])
{
    int pfd[2]; /* Pipe file descriptors */
    char buf[BUF_SIZE];
    ssize_t numRead;
    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        printf("%s string\n", argv[0]);
    if (pipe(pfd) == -1) /* Create the pipe */
        exit(1);
```

```
    switch (fork()) {
        case -1:
            exit(1);
        case 0: /* Child - reads from pipe */
            if (close(pfd[1]) == -1) /* Write end is unused */
                exit(1);
        for (;;) { /* Read data from pipe, echo on stdout */
            numRead = read(pfd[0], buf, BUF_SIZE);
            if (numRead == -1)
                exit(1);
            if (numRead == 0)
                break; /* End-of-file */
            //if (write(STDOUT_FILENO, buf, numRead) != numRead)
            //     printf("child - partial/failed write");
        }
        //write(STDOUT_FILENO, "\n", 1);
        if (close(pfd[0]) == -1)
            exit(1);
        exit(1);
        default: /* Parent - writes to pipe */
            if (close(pfd[0]) == -1) /* Read end is unused */
                exit(1);
        if (write(pfd[1], argv[1], strlen(argv[1])) != strlen(argv[1]))
            printf("parent - partial/failed write");
        if (close(pfd[1]) == -1) /* Child will see EOF */
            exit(1);
        wait(NULL); /* Wait for child to finish */
        exit(1);
    }
}
```

```
/*
int main()
{
	int filedes[2];
	if (pipe(filedes) == -1) // Create the pipe
		exit(1);
	switch (fork()) { // Create a child process
		case -1:
			exit(1);
		case 0: // Child
			if (close(filedes[1]) == -1) // Close unused write end
				exit(1);	// Child now reads from pipe
				break;
		default:	// Parent
			if (close(filedes[0]) == -1)	// Close unused read end
				exit(1);	// Parent now writes to pipe
				break;
```

```
	}
return 1;
}

*/
```

```
/*
int main()
{
	int fd[2];
		int size;
	char buffer[100];
	pid_t child;
	pipe(fd);
	child = fork();
	if (child > 0)  {   //parent
		printf("Parent Passing value to child\n");
```

```
        write(fd[1],"hello\n",6);
        wait(NULL);
    } else {  // child
        printf("Child printing received value\n");
        size = read(fd[0],buffer,100);
        write(1,buffer,size);
    }
return 1;
}

*/
```

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ ls

1pipe 1pipe.c child.txt create create.c input.txt msgqueues openpipe.c output.txt parent.txt pipe pipe.c pipesclass shared_mem

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat create.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#define MAX 128
int main()
{
    int fd[2];
    int child;
    if (pipe(fd) == -1) {
        printf("Pipe not created");
        exit(1);
    } else {
        child = fork();
```

```
    if (child == -1) {
        printf("Child not created");
        exit(1);
    } else {
        if (child == 0) {
            wait(NULL);
            close(fd[1]);
            char cstr[50];
            FILE *cptr;
            cptr = fopen("child.txt","w+");
            read(fd[0],cstr,MAX);
            printf("%s",cstr);
            fprintf(cptr,"%s",cstr);
        } else {
            close(fd[0]);
            FILE *pptr;
            char str[MAX];
            pptr = fopen("parent.txt","w");
            char name[10];
            printf("Enter name:");
            scanf("%s",name);
            fprintf(pptr,"HELLO %s\n ",name);
            fclose(pptr);
            pptr = fopen("parent.txt","r");
            fgets(str,MAX,pptr);
            write(fd[1],str,MAX);
            fclose(pptr);
        }
    }
}
```

```
return 0;
}
```

```
/*
int main()
{
	int fd[2];
   if (pipe(fd) == -1) {
		printf("error in creating file\n");
		exit(1);
	}
	int id;
	id = fork();
	if (id == -1) {
		printf("error in creating file\n");
		exit(1);
```

```
}
if (id == 0) {
	close(fd[0]);
	FILE *fptr;
	char buff[MAX];
	fptr = fopen("input.txt", "r");
	fgets(buff, MAX, fptr);
	if (write(fd[1], buff, MAX) == -1) {
		printf("error at write in pipe\n");
		exit(1);
	}
	close(fd[1]);
	fclose(fptr);



} else {
	wait(NULL);
	close(fd[1]);
	FILE *fpptr;
	char pbuff[MAX];
	fpptr = fopen("output.txt", "w+");
	if ( read(fd[0], pbuff, MAX) == -1) {
		printf("error at reading in pipe\n");
	}
	//fputs(fpptr, "Hello", 10);
	printf("%s\n", pbuff);
	fprintf(fpptr, "%s", pbuff);
	//printf("hi im parent\nRevide data from child is %d", y);
	close(fd[0]);
```

```
            fclose(fpptr);
        }



}
*/
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat o
openpipe.c  output.txt
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat openpipe.c
/* #include <fcntl.h> // Defines O_* constants
#include <sys/stat.h> // Defines mode constants
#include <mqueue.h>
struct mq_attr {
        long mq_flags; // Message queue description flags: 0 or O_NONBLOCK [mq_getattr(), mq_setattr()]
        long mq_maxmsg; // Maximum number of messages on queue [mq_open(), mq_getattr()]
        long mq_msgsize; // Maximum message size (in bytes) [mq_open(), mq_getattr()]
        long mq_curmsgs; // Number of messages currently in queue [mq_getattr()]
        };
mqd_t mq_open(const char *name, int oflag, ... // mode_t mode, struct mq_attr *attr );*/



#include<stdio.h>
#include <mqueue.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
static void usageError(const char *progName)
{
        fprintf(stderr, "Usage: %s [-cx] [-m maxmsg] [-s msgsize] mq-name ""[octal-perms]\n", progName);
```

```
        fprintf(stderr, " -c Create queue (O_CREAT)\n");
        fprintf(stderr, " -m maxmsg Set maximum # of messages\n");
        fprintf(stderr, " -s msgsize Set maximum message size\n");
        fprintf(stderr, " -x Create exclusively (O_EXCL)\n");
        exit(EXIT_FAILURE);
}
int main(int argc, char *argv[])
{
        int flags, opt;
        mode_t perms;
        mqd_t mqd;
        struct mq_attr attr, *attrp;
        attrp = NULL;
        attr.mq_maxmsg = 50;
        attr.mq_msgsize = 2048;
        flags = O_RDWR;
        //1070 Chapter 52 /* Parse command-line options */
        while ((opt = getopt(argc, argv, "cm:s:x")) != -1) {
                switch (opt) {
                        case 'c':
                                flags |= O_CREAT;
                                break;
                        case 'm':
                                attr.mq_maxmsg = atoi(optarg);
                                attrp = &attr;
                                break;
                        case 's':
                                attr.mq_msgsize = atoi(optarg);
                                attrp = &attr;
                                break;
```

```
                case 'x':
                    flags |= O_EXCL;
                    break;
                default:
                    usageError(argv[0]);
            }
        }
        if (optind >= argc)
            usageError(argv[0]);
        perms = (argc <= optind + 1) ? (S_IRUSR | S_IWUSR) :
        getInt(argv[optind + 1], GN_BASE_8, "octal-perms");
        mqd = mq_open(argv[optind], flags, perms, attrp);
        if (mqd == (mqd_t) -1)
        exit(1);
        exit(1);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat pipe.c
#if 0
#endif

#if 0
#endif

#if 0
#endif

#if 0
        /* If the parent wants to receive data from the child, it should close fd1, and the child should
close fd0.
        If the parent wants to send data to the child, it should close fd0, and the child should close fd1.
Since
```

descriptors are shared between the parent and child, we should always be sure to close the end of pipe we

aren't concerned with. On a technical note, the EOF will never be returned if the unnecessary ends of the*/

```
#include <stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
      int fd[2];
   pid_t   childpid;
      pipe(fd);
   if((childpid = fork()) == -1) {
      perror("fork");
      exit(1);
    }

    if(childpid == 0) {
                /* Child process closes up input side of pipe */
       close(fd[0]);
    } else {
                /* Parent process closes up output side of pipe */
            close(fd[1]);
    }
  return 0;
}
#endif

#if 1
```

```
#include <stdio.h>
#include <unistd.h>
#include<stdlib.h>
#include <sys/types.h>
int main()
{
      int fd[2];
   pid_t   childpid;
   pipe(fd);
   if((childpid = fork()) == -1)
   {
      perror("fork");
      exit(1);
   }
   if(childpid == 0)
   {
      /* Child process closes up input side of pipe */
      close(fd[0]);
      } else {
      /* Parent process closes up output side of pipe */
      close(fd[1]);
   }
return 1;
}
#endif
```

```
#if 0 /*opened pipe , creating child*/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include<stdlib.h>
int main()
{
      int fd[2];
   pid_t childpid;
   pipe(fd);
   if((childpid = fork()) == -1)
   {
      perror("fork");
      exit(1);
   }
      printf("%d\n",childpid);
   //printf("%d\n", pipe(fd));
return 0;
}
#endif
```

```
#if 0 /*opened pipe */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
```

```
int main()
{
      int fd[2];
   printf("%d\n", pipe(fd));
      return 0;
}
#endif
```

```
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat parent.txt
HELLO srilatha
 srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat child.txt
HELLO srilatha
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat input.txt
Hello World

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cat output.txt
Hello World
```

NAMED PIPES:

```
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cd pipesclass/
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ ls
a.out fifo fifo.c input.txt out.txt p1.txt p2.txt pipe pipe.c process1 process1.c process2 process2.c
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat fifo.c
#include<stdio.h>
#include<sys/stat.h>
int main()
{
      int m;
      m = mkfifo("fifo",0777);
      return 0;
```

```
}

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat process1
process1    process1.c
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat process1.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#define SIZE 50
int main()
{
        char buf[SIZE];
        FILE *fptr;
        int fd;
        fd = open("fifo",O_WRONLY);
        fptr = fopen("p1.txt","r");
        fgets(buf,SIZE,fptr);
        write(fd,buf, SIZE);
        printf("%s",buf);
        //write(fd,buf,SIZE);
        close(fd);
        fclose(fptr);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat process2.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/wait.h>
```

```
#include<fcntl.h>
#include<unistd.h>
#define SIZE 50
int main()
{
	char buf2[SIZE];
	FILE *fptr2;
	int fd;
	fptr2 = fopen("p2.txt","w+");
	fd = open("fifo",O_RDONLY);
	read(fd,buf2,SIZE);
	fprintf(fptr2,"%s",buf2);
	printf("%s",buf2);
	close(fd);
	fclose(fptr2);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat input.txt
HELLo
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat out.txt
-2
```

UNNAMED PIPES:

```
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat pipe.c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/wait.h>
#include<unistd.h>
#define SIZE 50
```

```
int main()
{
	int fd[2];
	if (pipe(fd) == -1) {
		printf("Error opening pipe");
		exit(1);
	} else {
		int child;
		child = fork();
		if (child == -1) {
			printf("child not created");
			exit(1);
		} else {
			if(child == 0) {
				wait(NULL);
				close(fd[0]);
				FILE *fptr;
				char cbuf[SIZE];
				fptr = fopen("out.txt","w+");
				fprintf(fptr,"%s",cbuf);
				printf("%s",cbuf);
				//fgets(cbuf,SIZE,fptr);
				//write(fd[1],cbuf,SIZE);
				close(fd[1]);
				fclose(fptr);
			} else {
				//wait(NULL);
				FILE *pptr;
				char pbuf[SIZE];
				close(fd[0]);
```

```
                pptr = fopen("input.txt","r");
                //write(fd[1],pbuf,SIZE);
                fgets(pbuf,SIZE,pptr);
                write(fd[1],pbuf,SIZE);
                //fprintf(pptr,"%s",pbuf);
                //printf("%s",pbuf);
                close(fd[0]);
                fclose(pptr);
            }
        }
    }
return 1;
}
```

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat p1.txt

HELLO world

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/pipesclass$ cat p2.txt

HELLO world

MESSAGE QUEUES:

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes$ cd msgqueues/

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ ls

3process Makefile a.out hdr.h msq msq.c receiver receiver.c receiver2 receiver2.c receiver3 receiver3.c unlink unlink.c

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ cat hdr.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include<mqueue.h>
```

```
#include <sys/msg.h>
#define MAX 100
#define NAME "/msqueue"
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ cat msq.c
#include"hdr.h"

int main()
{
	struct mq_attr attr;
	attr.mq_flags = 0;
	attr.mq_maxmsg = 10;
	attr.mq_msgsize = 100;
	attr.mq_curmsgs = 2;
	char str[100];
	char str1[100];
	char str3[100];
	int l;
	mqd_t mq;
	mqd_t mq1;
	mq = mq_open(NAME,O_CREAT | O_RDWR,0666, &attr);
	if (mq < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue created");
	printf("\nEnter the msg\n");
	fgets(str,100,stdin);
	l = strlen(str);
	mq1 = mq_send(mq,str,l,3);
	if(mq1 < 0) {
```

```
        printf("msg not sent\n");
    } else {
        printf("msg sent\n");
    }
    printf("\nEnter the msg\n");
    fgets(str1,100,stdin);
    l = strlen(str1);
    mq1 = mq_send(mq,str1,l,2);
    if(mq1 < 0) {
        printf("msg not sent\n");
    } else {
        printf("msg sent\n");

    }
    printf("\nEnter the msg\n");
    fgets(str3,100,stdin);
    l = strlen(str3);
    mq1 = mq_send(mq,str3,l,1);
    if(mq1 < 0) {
        printf("msg not sent\n");
    } else {
        printf("msg sent\n");
    }
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ cat receiver.c
#include"hdr.h"
int main()
{
    struct mq_attr attr;
```

```
	mqd_t mqd;
	mqd_t mqd1;
	int p = 2;
	char st[100];
	mqd = mq_open( NAME, O_RDONLY,0666,&attr);
	if (mqd < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue created");
	mqd1 = mq_receive(mqd,st,100,&p);
	if(mqd1 < 0) {
		printf("msg not received\n");
	} else {
		printf("msg received by receiver 1\n");
	}
	printf("%s",st);
	//mq_unlink(NAME);
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ cat unlink.c
#include"hdr.h"

int main()
{
	mqd_t mqd;
	mqd = mq_open( NAME, O_RDONLY,0666,NULL);
	if (mqd < 0) {
		printf("not created");
		exit(1);
```

```
    }
    printf("msg queue created");
    mq_unlink(NAME);
return 0;
}
```

3 PROCESSES READING DATA FROM A MESSAGE QUEUE

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues$ cd 3process/

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ ls

hdr.h input.txt receiver1 receiver1.c receiver2 receiver2.c receiver3 receiver3.c sender sender.c unlink unlink.c

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat hdr.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include<mqueue.h>
#include <sys/msg.h>
#define MAX 100
#define NAME "/msqueue"
```

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat sender.c

```
#include"hdr.h"
int main()
{
    FILE *fp;
    struct mq_attr attr;
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = 100;
```

```
attr.mq_curmsgs = 0;
int l;
mqd_t mq;
mqd_t mq1;
char str[MAX];
char buff1[MAX];
char buff2[MAX];
char buff3[MAX];
fp = fopen("input.txt","w+");
fgets(str,MAX,stdin);
fputs(str,fp);
fseek(fp,0,SEEK_END);
int total;
int count = 0;
int n;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
char c;
total = ftell(fp);
printf("total file size :%d",total);
count = total/3;
printf("\nEqal parts size : %d\n",count);
fseek(fp,0,SEEK_SET);
printf("set");
//while(i <= total) {
while((c = getc(fp)) != EOF) {
      //c = getc(fp);
      if (i<count) {
```

```
            buff1[m] = c;
            m++;
        } else if ((i < (2*count)) && (j < count))  {
            buff2[j] = c;
            j++;
        } else { //if(i <= (3*count)) {
            buff3[k] = c;
            k++;
        }
        i++;
        buff2[j+1] = '\0';
}
printf("%d",j);
            //c = getc(fp);
printf("\n%s",buff1);
printf("%s",buff2);
printf("%s",buff3);
// msg queue
mq = mq_open(NAME,O_CREAT | O_RDWR,0666, &attr);
if (mq < 0) {
        printf("not created");
        exit(1);
} else {
        printf("\nmsg queue created");
}
l = strlen(buff1);
mq1 = mq_send(mq,buff1,l,1);
if(mq1 < 0) {
        printf("msg1 not sent\n");
} else {
```

```
        printf("\nmsg1 sent");
    }
    l = strlen(buff2);
    mq1 = mq_send(mq,buff2,l,2);
    if(mq1 < 0) {
        printf("msg2 not sent\n");
    } else {
        printf("\nmsg2 sent");

    }
    l = strlen(buff3);
    mq1 = mq_send(mq,buff3,l,3);
    if(mq1 < 0) {
        printf("msg3 not sent\n");
    } else {
        printf("\nmsg3 sent\n");
    }
//     mq_unlink(NAME);
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat receiver1.c
#include"hdr.h"
int main()
{
    FILE *fp;
    struct mq_attr attr;
    mqd_t mqd;
    mqd_t mqd2;
    mqd_t mqd1;
    int c;
```

```
	int p = 2;
	char st[100];
	fp = fopen("ouput.txt","a+");
	c = fseek(fp,0,SEEK_END);
	ftell(fp);
	mqd = mq_open( NAME, O_RDONLY,0666,&attr);
	mqd2 = mq_getattr(mqd,&attr);
	printf("current msgs : %ld",attr.mq_curmsgs);




	if (mqd < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue created");
	mqd1 = mq_receive(mqd,st,100,&p);
	fseek(fp,0,c);
	fputs(st,fp);
	if(mqd1 < 0) {
		printf("msg not received\n");
	} else {
		printf("msg received by receiver 1\n");
	}
	printf("%s",st);
	//mq_unlink(NAME);
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat receiver2.c
```

```
#include"hdr.h"
int main()
{
	FILE *fp;
	struct mq_attr attr;
	mqd_t mqd;
	mqd_t mqd1;
	int p = 2;
	int c;
	char st[100];
	fp = fopen("ouput.txt","a+");
	c = fseek(fp,0,SEEK_END);
	ftell(fp);
	mqd = mq_open( NAME, O_RDONLY,0666,&attr);
	if (mqd < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue created");
	mqd1 = mq_receive(mqd,st,100,&p);
	fseek(fp,0,c);
	fputs(st,fp);
	if(mqd1 < 0) {
		printf("msg not received\n");
	} else {
		printf("msg received by receiver 1\n");
	}
	printf("%s",st);
	//mq_unlink(NAME);
return 0;
```

```
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat receiver3.c
#include"hdr.h"
int main()
{
	FILE *fp;
	struct mq_attr attr;
	mqd_t mqd;
	mqd_t mqd1;
	int p = 2;
	int c;
	char st[100];
	fp = fopen("ouput.txt","a+");
	c = fseek(fp,0,SEEK_END);
	ftell(fp);
	mqd = mq_open( NAME, O_RDONLY,0666,&attr);
	if (mqd < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue created");
	mqd1 = mq_receive(mqd,st,100,&p);
	fseek(fp,0,c);
	fputs(st,fp);
	if(mqd1 < 0) {
		printf("msg not received\n");
	} else {
		printf("msg received by receiver 1\n");
	}
	printf("%s",st);
```

```
	//mq_unlink(NAME);
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat unlink.c
#include"hdr.h"

int main()
{
	FILE *fp;
	fp = fopen("ouput.txt","w");
	char nam[10] = "ouput.txt";
	mqd_t mqd;
	mqd = mq_open( NAME, O_RDONLY,0666,NULL);
	if (mqd < 0) {
		printf("not created");
		exit(1);
	}
	printf("msg queue unlinked");
	unlink(nam);
	mq_unlink(NAME);
return 0;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/msgqueues/3process$ cat input.txt
asdfgh
```

SHARED MEMORY

```
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem$ cat hdr.h
#include<stdio.h>
```

```
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<fcntl.h>

#define NAME "/sh_mem"

#define SIZE sizeof(int)*n
#define n 2
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem$ cat sh_open.c
#include"hdr.h"

int main()
{
	int fd;
	fd = shm_open(NAME, O_CREAT|O_EXCL|O_RDWR,0660);
	char *addr;
	int i = 0;
	if (fd < 0) {
		printf("Error creating shared memory");
		exit(1);
	}
		printf("Created\n");
		ftruncate(fd,10);
		ftruncate(fd1,10);
		printf("truncate\n");
	addr = (char*)mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
	printf("addr : %p",addr);
```

```
	//for(i = 0; i < 2 ; i++) {
	printf("\nEnter ");
	fgets((addr),100,stdin);
	//}
	int s = sizeof(addr);
	printf("%d",s);
	munmap(addr,SIZE);
	close(fd);
	return 1;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem$ cat receiver1.c
#include"hdr.h"
int main()
{
	int fd;
	fd = shm_open(NAME,O_RDONLY,0660);
	int i = 0;
	char *addr;
	if (fd < 0) {
		printf("Error creating shared memory");
		exit(1);
	} else {
		printf("Created\n");
	}
	addr = (char*)mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0);
	printf("%p\n",addr);
		printf("\ndata is :");
		printf("%s",(addr));
	//fgets(addr,100,stdout);
	close(fd);
```

```
        munmap(addr,SIZE);
        shm_unlink(NAME);
        return 1;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem$ cat unlink.c
#include "hdr.h"
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>




int main()
{
   int fd = shm_open(NAME ,  O_RDWR , 0600);
   if(fd < 0) {
      perror("shm_open()");
      return EXIT_FAILURE;
   }
   int *data = (int*)mmap(0,SIZE,PROT_READ | PROT_WRITE, MAP_SHARED , fd , 0);
      shm_unlink(NAME);
return 0;
}
```

3 PROCESS SHARING SAME MEMORY SPACE

```
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem$ cd 3process/
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ ls
hdr.h  receiver1  receiver1.c  receiver2  receiver2.c  sh_open  sh_open.c  unlink  unlink.c
```

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ cat hdr.h

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<fcntl.h>

#define NAME "/sh_mem"

#define SIZE sizeof(int)*n
#define n 2
```

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ cat sh_open.c

```
#include"hdr.h"

int main()
{
	int fd;
	fd = shm_open(NAME, O_CREAT|O_EXCL|O_RDWR,0660);
	char *addr;
	int i = 0;
	int j = 0;
	if (fd < 0) {
		printf("Error creating shared memory");
		exit(1);
	}
		printf("Created\n");
		ftruncate(fd,SIZE);
		printf("truncate\n");
```

```
	addr = (char*)mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
	//*addr = (char*)mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
	printf("addr : %p",addr);
		printf("\nEnter ");
	fgets((addr+i),100,stdin);
	i++;
	fgets((addr+i),100,stdin);
	i++;
	//munmap(addr,SIZE);
	close(fd);
	return 1;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ cat receiver1.c
#include"hdr.h"
int main()
{
	int fd;
	fd = shm_open(NAME,O_RDONLY,0660);
	int i = 0;
	char *addr;
	if (fd < 0) {
		printf("Error creating shared memory");
		exit(1);
	} else {
		printf("Created\n");
	}
	addr = (char*)mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0);
	printf("%p\n",addr);
	//for(i = 0; i < n; i++) {
		printf("\ndata is :");
```

```
        printf("%s",(addr+i));
    //}
    //fgets(addr,100,stdout);
    close(fd);
    //munmap(addr,SIZE);
    //shm_unlink(NAME);
    return 1;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ cat receiver2.c
#include"hdr.h"
int main()
{
    int fd;
    fd = shm_open(NAME,O_RDONLY,0660);
    int i = 0;
    char *addr;
    if (fd < 0) {
        printf("Error creating shared memory");
        exit(1);
    } else {
        printf("Created\n");
    }
    addr = (char*)mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0);
    printf("%p\n",addr);
    //for(i = 0; i < n; i++) {
        printf("\ndata is :");
        printf("%s",(addr+i));
    //}
    //fgets(addr,100,stdout);
    close(fd);
```

```
        munmap(addr,SIZE);
        shm_unlink(NAME);
        return 1;
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/pipes/shared_mem/3process$ cat unlink.c
#include "hdr.h"
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>




int main()
{
   int fd = shm_open(NAME ,  O_RDWR , 0600);
   if(fd < 0) {
      perror("shm_open()");
      return EXIT_FAILURE;
   }
   int *data = (int*)mmap(0,SIZE,PROT_READ | PROT_WRITE, MAP_SHARED , fd , 0);
      shm_unlink(NAME);
return 0;
}
```

THREADS:

srilatha@GESLMP22WP7T:~/Experiments/misc/operating$ cd threads/

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ ls

cond2 cond2.c conditional conditional.c create create.c evenodd evenodd.c mutx mutx.c numcreate numcreate.c threads3.c

srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat numcreate.c

```
#include<stdio.h>
#include<pthread.h>
#include<string.h>



void* tfunc(char*);
int main()
{
      int t;
   int t1;
      //pthread attr_t attr;
   pthread_t eventhread;
      char d[10] = "srilatha";
   t = pthread_create(&eventhread, NULL,(void*)tfunc, d);


   if ( t == 0 ) {
      printf("done from main thread");
   } else {
      printf("error");
    }
    pthread_join(eventhread,NULL);
return 0;
}
```

```
void *tfunc(char* z )
{
      char* a = z;
   printf("10   \n %s",a);
   pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat create.c
#include<stdio.h>
#include<pthread.h>
#include<string.h>



void* tfunc(int*);
int main()
{
      int t;
      //pthread attr_t attr;
      pthread_t th;
      int d = 10;
      t = pthread_create(&th, NULL,(void *)tfunc, &d);
      if ( t == 0 ) {
            printf("done from main thread");
      } else {
            printf("error");
      }
      pthread_join(th,NULL);
return 0;
}
void* tfunc(int* z )
{
```

```
	int a = *z;
	printf("10   \n %d",a);
	pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat conditional.c
#include<stdio.h>
#include<pthread.h>
#include<string.h>

static pthread_mutex_t M = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t C = PTHREAD_COND_INITIALIZER;

static int avail = 0;

int a = 1;

void* tfunc();

int main()
{
	int t;
	int t1;
	pthread_t thread1;
	pthread_t thread2;
	t = pthread_create(&thread1,NULL,(void*)tfunc,NULL);
	//t1 = pthread_create(&thread2,NULL,tfunc,NULL);
	if (t != 0 ) {
		printf("threads not created\n");
	}
	t1 = pthread_create(&thread2,NULL,(void*)tfunc,NULL);
```

```
        if (t1 != 0 ) {
                printf("threads not created\n");
        }
        printf("Main thread\n");
        //pthread_join(t1,NULL);
}
void* tfunc()
{
        //int count = a;
        pthread_mutex_lock(&M);
        if ( a == 1) {
                printf("first\n");
                printf("%d",avail);
                a = 2;
        } else if (a = 2 ) {
                printf("second\n");
        } else {
        pthread_cond_wait(&C,&M);
        avail = avail+1;
        //printf("%d \n",a);
        //count++;
        //printf("%d \n",count);
        }
        pthread_mutex_unlock(&M);
        pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat cond2.c
#include<stdio.h>
#include<pthread.h>
#include<string.h>
```

```
static pthread_mutex_t M = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t C = PTHREAD_COND_INITIALIZER;
static pthread_cond_t C1 = PTHREAD_COND_INITIALIZER;

static int avail = 0;

int a = 1;

void* func1();
void* func2();

int main()
{
	int t1;
	int t2;
	pthread_t thread1;
	pthread_t thread2;
	t1 = pthread_create(&thread1,NULL,func1,NULL);
	if (t1 != 0 ) {
		printf("threads not created\n");
	}
	t2 = pthread_create(&thread2,NULL,func2,NULL);
	if (t2 != 0 ) {
		printf("threads not created\n");
	}
	printf("\nMain thread");
	//pthread_join(t1,NULL);
	//pthread_join(t2,NULL);
}
```

```
void* func1()
{
	int i;
	int j;
	i = pthread_mutex_lock(&M);
	if ( i != 0) {
		printf("\nMutex ia already locked");
		while(a != 2) {
			j = pthread_cond_wait(&C,&M);
			if ( j > 0)
			printf("\n 1 is waiting");
		}
	} else {
		printf("\nI am thread 1");
		printf("   %d",a);
		i = pthread_mutex_unlock(&M);
		if ( i != 0)
			printf("\nMutex ia already locked");
	}
	pthread_exit(NULL);
}
void* func2()
{
	int i;
	int j;
	i = pthread_mutex_lock(&M);
	if ( i != 0) {
		printf("\nMutex ia already locked");
		if (a == 2)
		j = pthread_cond_wait(&C1,&M);
```

```
            if ( j > 0)
            printf("\n thread 2 is waiting");
        } else {
            printf("\nI am thread 2");
            printf("   %d",a++);
            i = pthread_mutex_unlock(&M);
            if ( i != 0)
                printf("\nMutex ia already locked");
        }
        pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat mutx.c
#include<stdio.h>
#include<pthread.h>
#include<string.h>

static pthread_mutex_t M = PTHREAD_MUTEX_INITIALIZER;
int a = 1;

void* tfunc();

int main()
{
        int t;
        int t1;
        pthread_t thread1;
        pthread_t thread2;
        t = pthread_create(&thread1,NULL,tfunc,NULL);
        //t1 = pthread_create(&thread2,NULL,tfunc,NULL);
        if (t != 0 ) {
```

```
        printf("threads not created\n");
    }
    t1 = pthread_create(&thread2,NULL,tfunc,NULL);
    if (t1 != 0 ) {
        printf("threads not created\n");
    }
    printf("Main thread\n");
    pthread_join(t,NULL);
    pthread_join(t1,NULL);
}
void* tfunc()
{
    int c;
    c = pthread_mutex_lock(&M);
    if (c != 0) {
        printf("Mutex not locked");
    }
    a = ++a;
    printf("%d \n",a);
    c = pthread_mutex_unlock(&M);
    if (c != 0) {
        printf("Mutex not unlocked");
    }
    pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat threads3.c
#include <stdio.h>
#include <pthread.h> static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
static int data = 0; void* fun1();
```

```
void* fun2();
void* fun3(); int main()
{
	pthread_t subt1;
	pthread_t subt2;
	pthread_t subt3;
	int res;
	res = pthread_create(&subt1, NULL, (void*) fun1, NULL);
	if (res != 0) {
		printf("ERROR in 1\n");
	}
	res = pthread_create(&subt2, NULL, (void*) fun2, NULL);
	if(res != 0) {
		printf("ERROR in 2\n");
	}
	res = pthread_create(&subt3, NULL, (void*) fun3, NULL);
	if(res != 0) {
		printf("ERROR in 3\n");
	}
	pthread_join(subt1, NULL);
	printf("Im main thread\n");
} void* fun2()
{
	pthread_mutex_lock(&mtx);
	while (data != 0) {
		pthread_cond_wait(&cond, &mtx);
	}
	printf("Im subthread 2 %d\n", data);
	data++;
	pthread_mutex_unlock(&mtx);
```

```
	pthread_exit(NULL);
} void* fun3()
{
	pthread_mutex_lock(&mtx);
	while (data != 1) {
		pthread_cond_wait(&cond, &mtx);
	}
	printf("Im subthread 3 %d \n", data);
	data++;
	pthread_cond_signal(&cond);
	pthread_mutex_unlock(&mtx);
	pthread_exit(NULL);
} void* fun1()
{
	pthread_mutex_lock(&mtx);
	while(data != 2) {
		pthread_cond_wait(&cond, &mtx);
	}
	data = 0;
	printf("Im sunthread 1 %d \n", data);
//	 pthread_cond_signal(&cond);
	pthread_mutex_unlock(&mtx);
	pthread_exit(NULL);
}
srilatha@GESLMP22WP7T:~/Experiments/misc/operating/threads$ cat evenodd.c
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<string.h>
```

```
void* evenfunc(int*);
void* oddfunc(int*);
int main()
{
      int t;
   int t1;
      //int num = rand();
   int num;
      int i = 10;
      pthread_t eventhread;
      pthread_t oddthread;
      while(i) {
            num = random();
      t = pthread_create(&eventhread, NULL,(void*)evenfunc,&num);
      if ( t != 0 ) {
            printf("\nhii main thread");
            }
      t1 = pthread_create(&oddthread, NULL,(void*)oddfunc, &num);
      if ( t1 != 0) {
            printf("\nhii main thread");
        }
            i--;
      }
       pthread_join(eventhread,NULL);
       pthread_join(oddthread,NULL);
            printf("\n");
return 0;
}
void* evenfunc (int *z)
```

```
{
        int a = *z;
        if (a % 2 == 0) {
        printf("\nnum is even %d",a);
        }
    pthread_exit(NULL);
}
void* oddfunc(int *z )
{
        int a = *z;
        if (a % 2 != 0) {
        printf("\nnum is odd %d",a);
        }
    pthread_exit(NULL);
}
```