PROGRAM – 3 Thread management using Pthread Library:

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

int a[4][4], b[4][4];
void *matrixeval(void *val) {
   int *thno = (int *)val;
   for (int i = 0; i < 4; i++) {
      b[*thno][i] = a[*thno][i];
      for (int j = 0; j < *thno; j++) {
         b[*thno][i] *= a[*thno][i];
      }
   }
}
int main() {
   pthread_t tid[4];
   for (int i = 0; i < 4; i++) {
      printf("Enter the elements of row %d: ", i + 1);
      for (int j = 0; j < 4; j++) {
         scanf("%d", &a[i][j]);
      }
   }
   printf("Before processing:\n");
   for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
         printf("%d ", a[i][j]);
      }
      printf("\n");
   }
   for (int i = 0; i < 4; i++) {
      pthread_create(&tid[i], NULL, matrixeval, (void *)&i);
      sleep(1);
   }
   for (int i = 0; i < 4; i++) {
      pthread_join(tid[i], NULL);
   }
   printf("After processing:\n");

   for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
         printf("%d ", b[i][j]);
      }
      printf("\n");
   }
```

```
   pthread_exit(NULL);
   return 0;
}
```

PROGRAM – 2 Process control system calls

```
# include<stdio.h>
# include<stdlib.h>
# include<unistd.h>
# include<sys/types.h>
# include<sys/wait.h>

int main(int argc, char *argv[]){
   printf("Main Function: \n");
   int retval = 1;
   pid_t pid = fork();

   if(pid<0){
      printf("Error in the fork operation.\n");
   }else if(pid == 0){
      printf("PID of the child process is %d.\n PID of the parent process is %d.\n", getpid(), getppid());
      execl("./binarysearch", argv[1], NULL);
   }else{
      printf("PID of parent process: %d\n", getpid());
      wait(&retval);

      if (WIFEXITED(retval)) {
         printf("Child terminated normally\n");
      } else {
         printf("Child terminated abnormally\n");
         exit(0);
      }
   }
   return 0;
}


#include<stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
 if (r >= l) {
 int mid = l + (r - l) / 2;
 if (arr[mid] == x)
 return 1;
 if (arr[mid] > x)
 return binarySearch(arr, l, mid - 1, x);
 return binarySearch(arr, mid + 1, r, x);
 }
 return -1;
}
void swap(int *xp, int *yp) {
```

```
 int temp = *xp;
 *xp = *yp;
 *yp = temp;
}
void sort(int arr[], int n) {
 int i, j;
 for (i = 0; i < n-1; i++)
 for (j = 0; j < n-i-1; j++)
 if (arr[j] > arr[j+1])
 swap(&arr[j], &arr[j+1]);
}
int main(void){

 int n,key, arr[10];
 printf("Enter the number of elements in the array: ");
 scanf("%d",&n);
 printf("Enter the elements: ");
 for(int i=0;i<n;i++)
 scanf("%d",&arr[i]);
 sort(arr,n);
 printf("Enter element to be searched: ");
 scanf("%d",&key);
 int result = binarySearch(arr, 0, n - 1, key);
 if(result==-1)
 printf("Element is not present in array");
 else
 printf("Element is present");
 return 0;
}
```

PROGRAM 4: Producer

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>

#define buffersize 10

sem_t mutex;
```

```
pthread_t tidP[20], tidC[20];
sem_t full, empty;
int counter;
int buffer[buffersize];

void * producer(void *param)
{
   int item = rand() % 100; // assuming the item is a random integer
   sem_wait(&empty);
   sem_wait(&mutex);
   buffer[counter++] = item;
   printf("Producer process produced an item %d\n", item);
   printf("The counter value is %d\n", counter);
   sem_post(&mutex);
   sem_post(&full);
   return NULL;
}

void * consumer(void *param)
{
   sem_wait(&full);
   sem_wait(&mutex);
   int item = buffer[--counter];
   printf("Consumer process consumed an item %d\n", item);
   printf("The counter value is %d\n", counter);
   sem_post(&mutex);
   sem_post(&empty);
   return NULL;
}

int main()
{
   int n1, n2, i;
   sem_init(&mutex, 0, 1);
   sem_init(&full, 0, 0);
   sem_init(&empty, 0, buffersize);
   counter = 0;

   printf("\n Enter the no. of producers : ");
   scanf("%d", &n1);
   printf("\n Enter the no. of consumers : ");
   scanf("%d", &n2);

   for(i=0; i<n1; i++)
      pthread_create(&tidP[i], NULL, producer, NULL);
```

```
    for(i=0; i<n2; i++)
        pthread_create(&tidC[i], NULL, consumer, NULL);

    for(i=0; i<n1; i++)
        pthread_join(tidP[i], NULL);
    for(i=0; i<n2; i++)
        pthread_join(tidC[i], NULL);

    sem_destroy(&mutex);
    sem_destroy(&full);
    sem_destroy(&empty);

    return 0;
}
// READER WRITER
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
int count=0,rcount=0;
sem_t mutex,wr;
void* writer(void *p){
int* i =(int*)p;
sem_wait(&wr);
printf("\nWriter %d writes page number %d",*i,++count);
sem_post(&wr);
}
void* reader(void* p){
int* i =(int*)p;
sem_wait(&mutex);
rcount++;
if(rcount==1)
sem_wait(&wr);
sem_post(&mutex);
printf("\nReader %d reads page number %d ",*i,count);
sem_wait(&mutex);
rcount--;
if(rcount==0)
sem_post(&wr);
sem_post(&mutex);
}
int main(){
sem_init(&mutex,0,1);
sem_init(&wr,0,1); int a[6]={1,2,3,1,2,3};
pthread_t p[6];
```

```
for(int i=0;i<3;i++) pthread_create(&p[i],NULL,writer,&a[i]);
for(int i=3;i<6;i++) pthread_create(&p[i],NULL,reader,&a[i]);
for(int i=0;i<6;i++) pthread_join(p[i],NULL);
}
// DINING PHILOSOPHER
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
 if (state[phnum] == HUNGRY
 && state[LEFT] != EATING
 && state[RIGHT] != EATING) {
 // state that eating
 state[phnum] = EATING;
 sleep(2);
 printf("Philosopher %d takes fork %d and %d\n",
 phnum + 1, LEFT + 1, phnum + 1);
 printf("Philosopher %d is Eating\n", phnum + 1);
 // sem_post(&S[phnum]) has no effect
 // during takefork
 // used to wake up hungry philosophers
 // during putfork
 sem_post(&S[phnum]);
 }
}
// take up chopsticks
void take_fork(int phnum)
{
 sem_wait(&mutex);
 // state that hungry
 state[phnum] = HUNGRY;
 printf("Philosopher %d is Hungry\n", phnum + 1);
 // eat if neighbours are not eating
 test(phnum);
```

```
 sem_post(&mutex);
 // if unable to eat wait to be signalled
 sem_wait(&S[phnum]);
 sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
 sem_wait(&mutex);
 // state that thinking
 state[phnum] = THINKING;
 printf("Philosopher %d putting fork %d and %d down\n",
 phnum + 1, LEFT + 1, phnum + 1);
 printf("Philosopher %d is thinking\n", phnum + 1);
 test(LEFT);
 test(RIGHT);
 sem_post(&mutex);
}
void* philospher(void* num)
{
 while (1) {
 int* i = num;
 sleep(1);
 take_fork(*i);
 sleep(0);
 put_fork(*i);
 }
}
int main()
{
 int i;
 pthread_t thread_id[N];
 // initialize the semaphores
 sem_init(&mutex, 0, 1);
 for (i = 0; i < N; i++)
 sem_init(&S[i], 0, 0);
 for (i = 0; i < N; i++) {
 // create philosopher processes
 pthread_create(&thread_id[i], NULL, philospher, &phil[i]); printf("Philosopher %d is thinking\n", i + 1);
 }
 for (i = 0; i < N; i++)
 pthread_join(thread_id[i], NULL);
}
```

PROGRAM 6: STATIC DYNAMIC:

```
cc -c ctest1.c ctest2.c
ar -cvq ctest1.o ctest2.o
cc prog.c libctest.a
./a.out
size a.out
```

```
cc -fPIC -c ctest1.c ctest2.c
cc -shared -o libctest.c ctest1.o ctest2.o
cc -L . prog.c -l ctest -o dynamic
./dynamic
size dynamic
```

PROGRAM 5: FILE LOCK

```
#include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>
 #include <fcntl.h>
 #include <errno.h>
 int main(int argc,char *argv[])
 {
 int fd;
 char buffer[255];
 struct flock fvar;
 if(argc==1)
 {
 printf("usage: %s filename\n",argv[0]);
 return -1;
 }
 if((fd=open(argv[1],O_RDWR))==-1)
 {
 perror("open");
 exit(1);
 }
 fvar.l_type=F_WRLCK;
 fvar.l_whence=SEEK_END;
 fvar.l_start=SEEK_END-100;
 fvar.l_len=100;
 printf("press enter to set lock\n");
 getchar();
 printf("trying to get lock..\n");
 if((fcntl(fd,F_SETLK,&fvar))==-1)
 { fcntl(fd,F_GETLK,&fvar);
 printf("\nFile already locked by process (pid):
\t%d\n",fvar.l_pid);
 return -1;
 }
 printf("locked\n");
 if((lseek(fd,SEEK_END-50,SEEK_END))==-1)
 {
 perror("lseek");
```

```
exit(1);
}
if((read(fd,buffer,100))==-1)
{
perror("read");
exit(1);
}
printf("data read from file..\n");
puts(buffer);
printf("press enter to release lock\n");
getchar();
fvar.l_type = F_UNLCK;
fvar.l_whence = SEEK_SET;
fvar.l_start = 0;
fvar.l_len = 0;
if((fcntl(fd,F_UNLCK,&fvar))==-1)
{
perror("fcntl");
exit(0);
}
printf("Unlocked\n");
close(fd);
return 0;
}
```

PROGRAM 1: commands

ls –l command

```
#include<stdio.h>
#include<dirent.h>
#include<sys/stat.h>
#include<pwd.h>
#include<grp.h>
#include<time.h>
int main()
{
DIR *d;
 struct dirent *de;
 struct stat buf;
 int i,j;
 char P[10]="rwxrwxrwx",AP[10]=" ";
 struct passwd *p;
 struct group *g;
 struct tm *t;
 char time[26];
 d=opendir(".");
 readdir(d);
```

```
readdir(d);
while( (de=readdir(d))!=NULL)
{
stat(de->d_name,&buf);

// File Type
if(S_ISDIR(buf.st_mode))
printf("d");
else if(S_ISREG(buf.st_mode))
printf("-");
else if(S_ISCHR(buf.st_mode))
printf("c");
else if(S_ISBLK(buf.st_mode))
printf("b");
else if(S_ISLNK(buf.st_mode))
printf("l");
else if(S_ISFIFO(buf.st_mode))
printf("p");
else if(S_ISSOCK(buf.st_mode))
printf("s");
//File Permissions P-Full Permissions AP-Actual Permissions
for(i=0,j=(1<<8);i<9;i++,j>>=1)
AP[i]= (buf.st_mode & j ) ? P[i] : '-' ;
printf("%s",AP);
//No. of Hard Links
printf("%5d",buf.st_nlink);
//User Name
p=getpwuid(buf.st_uid);
printf(" %.8s",p->pw_name);
//Group Name
g=getgrgid(buf.st_gid);
printf(" %-8.8s",g->gr_name);
//File Size
printf(" %8d",buf.st_size);
//Date and Time of modification
t=localtime(&buf.st_mtime);
strftime(time,sizeof(time),"%b %d %H:%M",t);
printf(" %s",time);
//File Name
printf(" %s\n",de->d_name);
}
}
```

cp command

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

```
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#define BUF_SIZE 8192
int main(int argc, char* argv[]) {
 int input_fd, output_fd; /* Input and output file descriptors */
 ssize_t ret_in, ret_out; /* Number of bytes returned by read() and write() */
 char buffer[BUF_SIZE]; /* Character buffer */
 /* Are src and dest file name arguments missing */
 if(argc != 3){
 printf ("Usage: cp file1 file2");
 return 1;
 }
 /* Create input file descriptor */
 input_fd = open (argv [1], O_RDONLY);
 if (input_fd == -1) {
 perror ("open");
 return 2;
 }
 /* Create output file descriptor */
 output_fd = open(argv[2], O_WRONLY | O_CREAT, 0644);
 if(output_fd == -1){
 perror("open");
 return 3;
 }
 /* Copy process */
 while((ret_in = read (input_fd, &buffer, BUF_SIZE)) > 0){
 ret_out = write (output_fd, &buffer, (ssize_t) ret_in); if(ret_out != ret_in){
 /* Write error */
 perror("write");
 return 4;
 }
 }
 /* Close file descriptors */
 close (input_fd);
 close (output_fd);
 return (EXIT_SUCCESS);
}
```

mv command

```
int main(int argc, char* argv[]) {
 int input_fd, output_fd; /* Input and output file descriptors */
 /* Are src and dest file name arguments missing */
 if(argc != 3){
```

```
 printf ("Usage: mv file1 file2");
 return 1;
 }
 /* Create input file descriptor */
 input_fd = link(argv [1], argv[2]);
 if (input_fd == -1) {
 perror ("link error");
 return 2;
 }
 /* Create output file descriptor */
 output_fd = unlink(argv[1]);
 if(output_fd == -1){
 perror("unlink");
 return 3;
 }
```

rm command

```
int main(int argc, char* argv[]) {
 output_fd = unlink(argv[1]);
 if(output_fd == -1){
 perror("unlink error");
 return 3;
 }
}
$gcc -o myls ls.c
$./myls
$gcc -o mycp cp.c
$./mycp a.c b.c
$gcc -o mymv mv.c
$./mymv a.c b.c
$gcc -o myrm rm.c
$./myrm a.c
```