RV College of Engineering®

Autonomous Institution Affiliated to Visvesveraya Technological University, Belagavi | Approved by AICTE, New Delhi, Accredited by NAAC, Bengaluru

OPERATING SYSTEMS

SUB CODE:

III SEMESTER B.E

INSTRUCTOR MANUAL

2023-24

Prepared by: Jyoti Shetty **Verified by: HOD, CSE**

Vision and Mission of the Department of Computer Science and Engineering

Vision

To achieve leadership in the field of Computer Science & Engineering by strengthening fundamentals and facilitating interdisciplinary sustainable research to meet the ever growing needs of the society.

Mission

- To evolve continually as a centre of excellence in quality education in Computers and allied fields.
- To develop state of the art infrastructure and create environment capable for interdisciplinary research and skill enhancement.
- To collaborate with industries and institutions at national and international levels to enhance research in emerging areas.
- To develop professionals having social concern to become leaders in top-notch industries and/or become entrepreneurs with good ethics.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's):

PEO1: Develop Graduates capable of applying the principles of mathematics, science, core engineering and Computer Science to solve real-world problems in interdisciplinary domains.

PEO2: To develop the ability among graduates to analyze and understand current pedagogical techniques, industry accepted computing practices and state-of-art technology.

PEO3: To develop graduates who will exhibit cultural awareness, teamwork with professional ethics, effective communication skills and appropriately apply knowledge of societal impacts of computing technology.

PEO4: To prepare graduates with a capability to successfully get employed in the right role/become entrepreneurs to achieve higher career goals or takeup higher education in pursuit of lifelong learning.

PROGRAM OUTCOMES:

Engineering Graduates will be able to:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO1: System Analysis and Design
The student will:

1. Recognize and understand the dynamic nature of developments in computer architecture, data organization and analytical methods.
2. Learn the applicability of various systems software elements for solving real-world design problems.
3. Identify the various analysis & design methodologies for facilitating development of high quality system software products with focus on performance optimization.
4. Display good team participation, communication, project management and document skills.

PSO2: Product Development
The student will:

1. Demonstrate knowledge of the ability to write programs and integrate them resulting in state-of –art hardware/software products in the domains of embedded systems, databases /data analytics, network/web systems and mobile products.
2. Participate in teams for planning and implementing solutions to cater to business – specific requirements displaying good team dynamics and professional ethics.
3. Employ state-of-art methodologies for product development and testing / validation with focus on optimization and quality related aspects

COURSE OUTCOMES

At the end of the course, the student will be able to –

1.	Demonstrate the fundamental concepts of operating system like process management, file management, memory management and issues of synchronization.
2.	Analyze and interpret operating system concepts to acquire a detailed understanding of the course.
3.	Apply the operating systems concepts to address related new problems in computer science Domain.
4.	Design or develop solutions to solve applicable problems in operating systems domain.
5.	Extend the theoretical knowledge acquired through the course to understand the real-world implementation of operating system.

LIST OF PROGRAMS

Part – A

Students are expected to identify suitable application and implement solution for the topics listed below:

Sl. No.	Program
1	Implementation of basic UNIX commands using file APIs- Write a program to implement commands ls(-l option), cp, rm and mv using UNIX file APIs.
2	Apply the concepts of Process control system calls to build applications to demonstrate use of fork, execve, wait, getpid, exit system calls
3.	Apply the pthread library to build Applications to demonstrate use of pthread library functions to create and manage threads.
4.	Apply the concepts of Process/Thread synchronization to build Applications to demonstrate process/thread synchronization using semaphores and mutex. Implement Dining philosophers problem, reader-writer and producer-consumer.
5.	Apply the concepts of Process/Thread synchronization for file access to build applications to demonstrate process/thread synchronization using file locks.
6.	Apply the concepts of Static and Shared libraries to write a program to create and use static and shared libraries. Demonstrate the advantage of shared libraries over static libraries in terms of memory usage.

INDEX

Lab Rubrics						
Sl. No	**Criteria**	**Measuring methods**	**Excellent**	**Good**	**Poor**	
1	**Understanding of problem statement and design.** **(2 Marks)** **CO1,CO3**	Observations	Student exhibits thorough understanding of concept, identifies appropriate problem, applies suitable logic and programming concepts for the problem **(2 M)**	Student has sufficient understanding of concept and applies suitable OS concept for the given problem. **(<2 M and >=1 M)**	Student does not have a clear understanding of requirements and is unable to apply suitable logic and programming concept for the problem. **(0 M)**	
2	**Execution and debugging** **(2 Marks)** **CO2, CO4**	Observations	Student demonstrates the execution of the program with efficient code. Appropriate validations with all test cases are handled. **(2 M)**	Student demonstrates the execution of the program without efficiency of the code and validates only few cases. **(1 M)**	Student has not executed the program. **(0 M)**	
3	**Results and Documentation** **(2 Marks)** **CO5**	Observations	Documentation with appropriate comments and output is covered in data sheets and manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in data sheets and manual. **(1 M)**	Documentation with no comments and no output cases is covered in data sheets and manual. **(0 M)**	
Viva Voce rubrics (Max: 4 marks)						
1	**Conceptual Understanding** **(2 Marks)** **CO1, CO2**	Viva Voce	Explains thoroughly Operating System and programming concept **(2 M)**	Adequately explains the Operating System and programming concept. **(1 M)**	Unable to explain the Operating System and programming concept.**(0 M)**	
2	**Use of Operating System Concept** **(1 Marks)** **CO3**	Viva Voce	Insightful explanation of operating system concept for the given problem to derive solution. **(1 M)**	Sufficiently explains the use of appropriate operating system concept for the given problem to derive solution. **(0.5 M)**	Unable to explain the operating system concept for the given problem. **(0 M)**	
3	**Communication of Concepts** **(1 Marks)** **CO5**	Viva Voce	Communicates the concept used in problem solving well. **(1 M)**	Sufficiently communicates the concepts used in problem solving. **(0.5 M)**	Unable to communicate the concepts used in problem. **(0 M)**	
Record Marks (6)		**Viva Marks (4)**	**Total (10)**	**Signature of Staff**		

PROGRAM – 1

Implementation of basic UNIX commands using file APIs

Write a program to implement commands ls(-l option), cp, rm and mv using UNIX file APIs.

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
```

PROGRAM – 2

Process control system calls
Application to demonstrate use of fork, execve, wait, getpid, exit system calls.

AHost Program

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
#include<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 fork operation\n");
   }
   if(pid==0){
      printf("PID for Child process: %d\nPID of its parent process: %d\n",getpid(),getppid());
      execl("./binsearch",argv[1],NULL);
   }
   else{
      printf("PID of parent process: %d\n",getpid());
      wait(&retval);
      if(WIFEXITED(retval)==1)
      {
         printf("Child terminated normally\n");
      }
      else{
      printf("Child terminated abnormally\n");
      exit(0);
      }
```

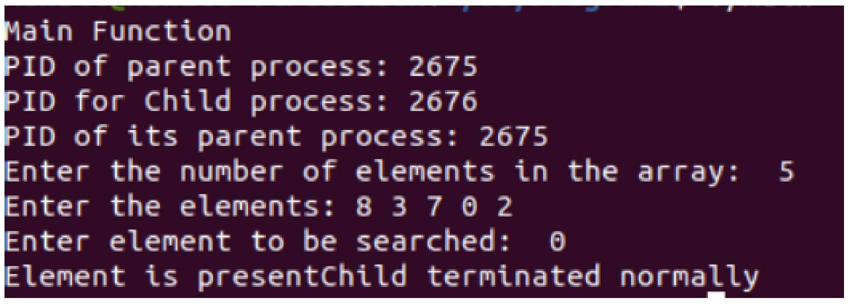
```
    }
    return 0;
}
```

Binary search application

```
#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;
}
```

Output



```
Main Function
PID of parent process: 2675
PID for Child process: 2676
PID of its parent process: 2675
Enter the number of elements in the array:  5
Enter the elements: 8 3 7 0 2
Enter element to be searched:  0
Element is presentChild terminated normally
```

PROGRAM – 3

Thread management using Pthread Library

To demonstrate use of pthread library functions to create and manage threads, using a program that raises the numbers to the power of their corresponding row in the matrix (matrix indexed from 1).

Implementation:

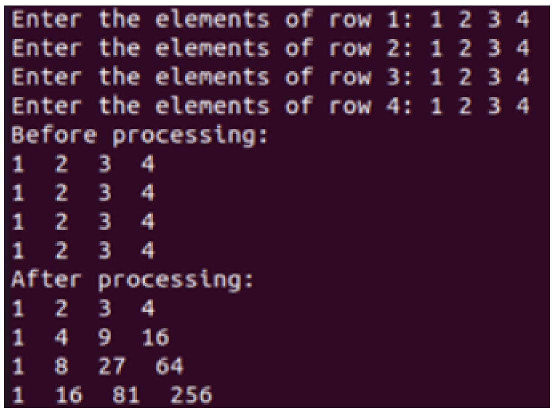
```
#include<pthread.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
#include<math.h>
#include<stdio.h>
int a[4][4],b[4][4];

void *matrixeval(void *val) {
    int *thno = (int *)val;
    for(int i=0;i<4;i++)
    b[(*thno)][i]=a[(*thno)][i];
    for(int i=0;i<4;i++)
    for(int j=0;j<(*thno);j++)
          b[(*thno)][i]*=(a[(*thno)][i]);
    //printf("(%d) thread  ",(*thno)+1);
}
```

```
int main(){

   pthread_t tid[4]; int i,j;
   for(i=0;i<4;i++){
   printf("Enter the elements of row %d: ",i+1);
     for(j=0;j<4;j++)
     scanf("%d",&a[i][j]);
     }
   printf("Before processing:\n");
   for(i=0;i<4;i++){
   for(j=0;j<4;j++)
     printf("%d  ",a[i][j]);
     printf("\n");
     }

   for (i=0; i<4; i++){
       pthread_create(&tid[i], NULL, matrixeval, (void *)&i);
       sleep(1); }
   for (i=0; i<4; i++)
       pthread_join(tid[i], NULL);
   printf("After processing:\n");
   for(i=0;i<4;i++){
   for(j=0;j<4;j++)
     printf("%d  ",b[i][j]);
     printf("\n");
     }

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

Output:

OUTPUT:

```
$ cc threads.c -pthread

$./a.out
```



```
Enter the elements of row 1: 1 2 3 4
Enter the elements of row 2: 1 2 3 4
Enter the elements of row 3: 1 2 3 4
Enter the elements of row 4: 1 2 3 4
Before processing:
1  2  3  4
1  2  3  4
1  2  3  4
1  2  3  4
After processing:
1  2  3  4
1  4  9  16
1  8  27  64
1  16  81  256
```

PROGRAM – 4

Process/Thread synchronization
Application to demonstrate process/thread synchronization using semaphores and mutex. Implement either Dining philosophers problem, reader-writer or producer-consumer.

```
#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);
}
```

Producer consumer:

```
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<stdlib.h>
#define buffersize 10
pthread_mutex_t mutex;
pthread_t tidP[20],tidC[20];
sem_t full,empty;
int counter;
int buffer[buffersize];

void initialize()
{
	pthread_mutex_init(&mutex,NULL);
	sem_init(&full,1,0);
	sem_init(&empty,1,buffersize);
	counter=0;
}

void write(int item)
{
	buffer[counter++]=item;
}

int read()
{
	return(buffer[--counter]);
}

void * producer (void * param)
{
	int waittime,item,i;
	item=rand()%5;
	waittime=rand()%5;
	sem_wait(&empty);
	pthread_mutex_lock(&mutex);
	printf("\nProducer has produced item: %d\n",item);
	write(item);
	pthread_mutex_unlock(&mutex);
	sem_post(&full);
}

void * consumer (void * param)
{
	int waittime,item;
	waittime=rand()%5;
	sem_wait(&full);
	pthread_mutex_lock(&mutex);
	item=read();
	printf("\nConsumer has consumed item: %d\n",item);
	pthread_mutex_unlock(&mutex);
	sem_post(&empty);
}

int main()
{
      int n1,n2,i;
      initialize();
      printf("\nEnter the no of producers: ");
      scanf("%d",&n1);
      printf("\nEnter 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);

      //sleep(5);
      exit(0);
}
```

```
OUTPUT:
Enter the no of producers: 3

Enter the no of consumers: 2

Producer has produced item: 3

Producer has produced item: 2

Consumer has consumed item: 2


Consumer has consumed item: 3

Producer has produced item: 0
```

Dining Philosopher's Problem :

```
#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);
}
```

OUTPUT :

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
```

PROGRAM – 5

Process/Thread synchronization for file access

Application to demonstrate process/thread synchronization using file locks.

```
#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;
}
```

Terminal 1
$./a.out foo.txt

Terminal 2
$./a.out foo.txt

PROGRAM -6

Static and Shared libraries

Write a program to create and use static and shared libraries. Demonstrate the advantage of shared libraries over static libraries in terms of memory usage.

To generate a static library (object code archive file):

- Compile: `cc -Wall -c ctest1.c ctest2.c`

- Create library "libctest.a" by archiving it:

  ```
  ar -cvq libctest.a ctest1.o ctest2.o
  ```

- List files in library:

  ```
  ar -t libctest.a
  ```

- Linking with the library:
  - Library in current directory:

    ```
    cc -o executable-name prog.c libctest.a
    ```

  - Library in not in current directory:

    ```
    cc -o executable-name prog.c
    -L/path/to/library-directory -lctest
    ```

- `size executable-name`

To generate a shared object (Dynamically linked object library file):

- Compile: `gcc -Wall -fPIC -c *.c`
  Here PIC meaning generate position independent code
- Create shared library "libctest.so"
  `gcc -shared  -o libctest.so   *.o`

- export LD_LIBRARY_PATH=.:LD_LIBRARY_PATH
- Linking with the library:

  ```
  gcc  -L . prog.c -l ctest -o prog
  ```

- `size prog`

Compare the size of shared and static libraries using size command

VIVA Questions

1)Explain the main purpose of an operating system?

Operating systems exist for two main purposes. One is that it is designed to make sure a computer system performs well by managing its computational activities. Another is that it provides an environment for the development and execution of programs.

2) What is demand paging?

Demand paging is referred when not all of a process's pages are in the RAM, then the OS brings the missing(and required) pages from the disk into the RAM.

3) What are the advantages of a multiprocessor system?

With an increased number of processors, there is a considerable increase in throughput. It can also save more money because they can share resources. Finally, overall reliability is increased as well.

4) What is kernel?

A kernel is the core of every operating system. It connects applications to the actual processing of data. It also manages all communications between software and hardware components to ensure usability and reliability.

5) What are real-time systems?

Real-time systems are used when rigid time requirements have been placed on the operation of a processor. It has well defined and fixed time constraints.

6) What is a virtual memory?

Virtual memory is a memory management technique for letting processes execute outside of memory. This is very useful especially is an executing program cannot fit in the physical memory.

7) Describe the objective of multiprogramming.

The main objective of multiprogramming is to have a process running at all times. With this design, CPU utilization is said to be maximized.

8) What is time- sharing system?

In a Time-sharing system, the CPU executes multiple jobs by switching among them, also known as multitasking. This process happens so fast that users can interact with each program while it is running.

9) What is SMP?

SMP is a short form of Symmetric Multi-Processing. It is the most common type of multiple-processor systems. In this system, each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

10) How are server systems classified?

Server systems can be classified as either computer-server systems or file server systems. In the first case, an interface is made available for clients to send requests to perform an action. In the second case, provisions are available for clients to create, access and update files.

11) What is asymmetric clustering?

In asymmetric clustering, a machine is in a state known as hot standby mode where it does nothing but to monitor the active server. That machine takes the active server's role should the server fails.

12) What is a thread?

A thread is a basic unit of CPU utilization. In general, a thread is composed of a thread ID, program counter, register set, and the stack.

13) Give some benefits of multithreaded programming.

– there is increased responsiveness to the user
– resource sharing within the process
– economy
– utilization of multiprocessing architecture

14) Briefly explain FCFS.

FCFS stands for First-come, first-served. It is one type of scheduling algorithm. In this scheme, the process that requests the CPU first is allocated the CPU first. Implementation is managed by a FIFO queue.

15) What is RR scheduling algorithm?

RR (round-robin) scheduling algorithm is primarily aimed for time-sharing systems. A circular queue is a setup in such a way that the CPU scheduler goes around that queue, allocating CPU to each process for a time interval of up to around 10 to 100 milliseconds.

16) What are necessary conditions which can lead to a deadlock situation in a system?

Deadlock situations occur when four conditions occur simultaneously in a system: Mutual exclusion; Hold and Wait; No preemption; and Circular wait.

What factors determine whether a detection-algorithm must be utilized in a deadlock avoidance system?

One is that it depends on how often a deadlock is likely to occur under the implementation of this algorithm. The other has to do with how many processes will be affected by deadlock when this algorithm is applied.

20) State the main difference between logical from physical address space.

Logical address refers to the address that is generated by the CPU. On the other hand, physical address refers to the address that is seen by the memory unit.

21) How does dynamic loading aid in better memory space utilization?

With dynamic loading, a routine is not loaded until it is called. This method is especially useful when large amounts of code are needed in order to handle infrequently occurring cases such as error routines.

22) What are overlays?

Overlays are used to enable a process to be larger than the amount of memory allocated to it. The basic idea of this is that only instructions and data that are needed at any given time are kept in memory.

23) What is the basic function of paging?

Paging is a memory management scheme that permits the physical address space of a process to be noncontiguous. It avoids the considerable problem of having to fit varied sized memory chunks onto the backing store.

24) What is fragmentation?

Fragmentation is memory wasted. It can be internal if we are dealing with systems that have fixed-sized allocation units, or external if we are dealing with systems that have variable-sized allocation units.

25) How does swapping result in better memory management?

During regular intervals that are set by the operating system, processes can be copied from main memory to a backing store, and then copied back later. Swapping allows more operations to be run that can fit into memory at one time.

26) Give an example of a Process State.

– New State – means a process is being created
– Running – means instructions are being executed
– Waiting – means a process is waiting for certain conditions or events to occur
– Ready – means a process is waiting for an instruction from the main processor
– Terminate – means a process is stopped abruptly

27) What is a socket?

A socket provides a connection between two applications. Each endpoint of a communication is a socket.

28) What is Direct Access Method?

Direct Access method is based on a disk model of a file, such that it is viewed as a numbered sequence of blocks or records. It allows arbitrary blocks to be read or written. Direct access is advantageous when accessing large amounts of information.

29) When does thrashing occur?

Thrashing refers to an instance of high paging activity. This happens when it is spending more time paging instead of executing.

30) What is the best page size when designing an operating system?

The best paging size varies from system to system, so there is no single best when it comes to page size. There are different factors to consider in order to come up with a suitable page size, such as page table, paging time, and its effect on the overall efficiency of the operating system.

19) What factors determine whether a detection-algorithm must be utilized in a deadlock avoidance system?

One is that it depends on how often a deadlock is likely to occur under the implementation of this algorithm. The other has to do with how many processes will be affected by deadlock when this algorithm is applied.

20) State the main difference between logical from physical address space.

Logical address refers to the address that is generated by the CPU. On the other hand, physical address refers to the address that is seen by the memory unit.

21) How does dynamic loading aid in better memory space utilization?

With dynamic loading, a routine is not loaded until it is called. This method is especially useful when large amounts of code are needed in order to handle infrequently occurring cases such as error routines.

22) What are overlays?

Overlays are used to enable a process to be larger than the amount of memory allocated to it. The basic idea of this is that only instructions and data that are needed at any given time are kept in memory.

23) What is the basic function of paging?

Paging is a memory management scheme that permits the physical address space of a process to be noncontiguous. It avoids the considerable problem of having to fit varied sized memory chunks onto the backing store.

24) What is fragmentation?

Fragmentation is memory wasted. It can be internal if we are dealing with systems that have fixed-sized allocation units, or external if we are dealing with systems that have variable-sized allocation units.

25) How does swapping result in better memory management?

During regular intervals that are set by the operating system, processes can be copied from main memory to a backing store, and then copied back later. Swapping allows more operations to be run that can fit into memory at one time.

26) Give an example of a Process State.

– New State – means a process is being created
– Running – means instructions are being executed
– Waiting – means a process is waiting for certain conditions or events to occur
– Ready – means a process is waiting for an instruction from the main processor
– Terminate – means a process is stopped abruptly