P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Mandya - 571 401, Karnataka

Vision/ಆಶಯ

"PESCE shall be a leading institution imparting quality engineering and management education developing creative and socially responsible professionals."

"PESCE
ಸೃಜನಶೀಲಮತ್ತುಸಾಮಾಜಿಕಜವಾಬ್ದಾರಿಯುತವೃತ್ತಿಪರರನ್ನುಅಭಿವೃದ್ಧಿಪಡಿಸುವಗುಣಮಟ್ಟದಎಂಜಿನಿಯರಿಂಗ್ಮತ್ತುನಿರ್ವಹಣಾಶಿಕ್ಷಣವನ್ನುನೀಡುವಪ್ರಮುಖಸಂಸ್ಥೆಯಾಗಿದೆ."

Mission/ ಧ್ಯೇಯ

- Provide state of the art infrastructure, motivate the faculty to be proficient in their field of specialization and adopt best teaching-learning practices.

  ಅತ್ಯಾಧುನಿಕಮೂಲಸೌಕರ್ಯಗಳನ್ನುಒದಗಿಸಿ,
  ಬೋಧಕವರ್ಗವನ್ನುತಮ್ಮವಿಶೇಷಕ್ಷೇತ್ರದಲ್ಲಿಪ್ರವೀಣರಾಗುವಂತೆಪ್ರೇರೇಪಿಸಿಮತ್ತುಅತ್ಯುತ್ತಮಬೋಧನೆ-ಕಲಿಕೆಯಅಭ್ಯಾಸಗಳನ್ನುಅಳವಡಿಸಿಕೊಳ್ಳಿ.

- Impart engineering and managerial skills through competent and committed faculty using outcome based educational curriculum.

  ಫಲಿತಾಂಶಆಧಾರಿತಶೈಕ್ಷಣಿಕಪಠ್ಯಕ್ರಮವನ್ನುಬಳಸಿಕೊಂಡುಸಮರ್ಥಮತ್ತುಬದ್ಧಅಧ್ಯಾಪಕರಮೂಲಕಎಂಜಿನಿಯರಿಂಗ್ಮತ್ತುನಿರ್ವಾಹಕಕೌಶಲ್ಯಗಳನ್ನುನೀಡಿ.

- Inculcate professional ethics, leadership qualities and entrepreneurial skills to meet the societal needs.

  ಸಾಮಾಜಿಕಅಗತ್ಯಗಳನ್ನುಪೂರೈಸಲುವೃತ್ತಿಪರನೈತಿಕತೆ,
  ನಾಯಕತ್ವಗುಣಗಳುಮತ್ತುಉದ್ಯಮಶೀಲತೆಯಕೌಶಲ್ಯಗಳನ್ನುರೂateಿಸಿಕೊಳ್ಳಿ.

- Promote research, product development and industry-institution interaction.

  ಸಂಶೋಧನೆ, ಉತ್ಪನ್ನಅಭಿವೃದ್ಧಿಮತ್ತುಉದ್ಯಮ-ಸಂಸ್ಥೆಗಳಪರಸ್ಪರಕ್ರಿಯೆಯನ್ನುಉತ್ತೇಜಿಸಿ.

P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Department of Information Science & Engineering

About the Department/ಇಲಾಖೆಯಬಗ್ಗೆ

The Department of Information science and Engineering takes pride in producing quality engineers over the past 20 years. The credit for all the flowery results goes to the highly motivating staff, from whom all students draw inspiration. The Department was started in the year 2000. The present intake of the undergraduate program is 60. The department has well equipped classrooms, computer laboratories with high-end systems, department library and good collection of software's. Also a research centre is a major credential to our department. We are proud to produce the first PhD student in our college. Faculty members of the department are involved in research activities in different fields such as Medical Image Processing, Pattern Recognition, and Data Mining etc. The department is using Outcome-based education (OBE), which is a recurring education reform model, and it is affiliated to Visvesvaraya Technological University (VTU). The department has achieved good Placement, conducted International /national Conferences and other sponsored short-term courses, workshops, National seminars and symposia. The laboratory facilities and the Internet access are available round the clock to the staff and students of the Information Science and Engineering.

ಮಾಹಿತಿವಿಜ್ಞಾನಮತ್ತುಎಂಜಿನಿಯರಿಂಗ್ವಿಭಾಗವುಕಳೆದ20ವರ್ಷಗಳಲ್ಲಿಗುಣಮಟ್ಟದಎಂಜಿನಿಯರ್‌ಗಳನ್ನುಉತ್ಪಾದಿಸುವಲ್ಲಿಹೆಮ್ಮೆಪಡುತ್ತದೆ.
ಎಲ್ಲಾಹೂವುಗಳಫಲಿತಾಂಶಗಳಕ್ರೆಡಿಟ್ಹೆಚ್ಚುಪ್ರೇರೇಪಿಸುವಸಿಬ್ಬಂದಿಗೆಸಲ್ಲುತ್ತದೆ,
ಅವರಿಂದಎಲ್ಲಾವಿದ್ಯಾರ್ಥಿಗಳುಸ್ಫೂರ್ತಿಪಡೆಯುತ್ತಾರೆ. ಇಲಾಖೆಯು 2000
ನೇವರ್ಷದಲ್ಲಿಆರಂಭವಾಯಿತು. ಪದವಿಪೂರ್ವಕಾರ್ಯಕ್ರಮದಪ್ರಸ್ತುತಸೇವನೆಯು 60.
ಇಲಾಖೆಯುಸುಸಜ್ಜಿತವಾದತರಗತಿಕೊಠಡಿಗಳು,
ಉನ್ನತಮಟ್ಟದವ್ಯವಸ್ಥೆಗಳೊಂದಿಗೆಕಂಪ್ಯೂಟರ್ಪ್ರಯೋಗಾಲಯಗಳು,
ಇಲಾಖೆಯಗ್ರಂಥಾಲಯಮತ್ತುಸಾಫ್ಟ್‌ವೇರ್‌ಗಳಉತ್ತಮಸಂಗ್ರಹವನ್ನುಹೊಂದಿದೆ.
ಅಲ್ಲದೆಒಂದುಸಂಶೋಧನಾಕೇಂದ್ರವುನಮ್ಮಇಲಾಖೆಗೆಪ್ರಮುಖರುಜುವಾತು.
ನಮ್ಮಕಾಲೇಜಿನಲ್ಲಿಮೊದಲಪಿಎಚ್‌ಡಿವಿದ್ಯಾರ್ಥಿಯನ್ನುತಯಾರಿಸಲುನಮಗೆಹೆಮ್ಮೆಇದೆ.
ಇಲಾಖೆಯಅಧ್ಯಾಪಕರುವೈದ್ಯಕೀಯಚಿತ್ರಸಂಸ್ಕರಣೆ, ಪ್ಯಾಟರ್ನ್ರೆಕಗ್ನಿಷನ್,
ಮತ್ತುಡೇಟಾಮೈನಿಂಗ್ಮುಂತಾದವಿವಿಧಕ್ಷೇತ್ರಗಳಲ್ಲಿಸಂಶೋಧನಾಚಟುವಟಿಕೆಗಳಲ್ಲಿತೊಡಗಿಸಿಕೊಂಡಿದ್ದಾರೆ. ಇಲಾಖೆಯುಫಲಿತಾಂಶಆಧಾರಿತಶಿಕ್ಷಣವನ್ನು (ಒಬಿಇ) ಬಳಸುತ್ತಿದೆ,
ಇದುಪುನರಾವರ್ತಿತಶಿಕ್ಷಣಸುಧಾರಣಾಮಾದರಿಯಾಗಿದೆ,
ಮತ್ತುಇದುವಿಶ್ವೇಶ್ವರಯ್ಯತಾಂತ್ರಿಕವಿಶ್ವವಿದ್ಯಾಲಯಕ್ಕೆ (ವಿಟಿಯು) ಸಂಯೋಜಿತವಾಗಿದೆ.
ಇಲಾಖೆಯುಉತ್ತಮಉದ್ಯೋಗವನ್ನುಸಾಧಿಸಿದೆ, ಅಂತರಾಷ್ಟ್ರೀಯ
/ರಾಷ್ಟ್ರೀಯಸಮ್ಮೇಳನಗಳನ್ನುಮತ್ತುಇತರಪ್ರಾಯೋಜಿತಅಲ್ಪಾವಧಿಕೋರ್ಸ್‌ಗಳು, ಕಾರ್ಯಾಗಾರಗಳು,
ರಾಷ್ಟ್ರೀಯವಿಚಾರಗೋಷ್ಠಿಗಳುಮತ್ತುವಿಚಾರಸಂಕಿರಣಗಳನ್ನುನಡೆಸಿದೆ.

ಪ್ರಯೋಗಾಲಯದಸೌಲಭ್ಯಗಳುಮತ್ತುಇಂಟರ್ನೆಟ್ಪ್ರವೇಶವುಸಿಬ್ಬಂದಿಮತ್ತುಮಾಹಿತಿವಿಜ್ಞಾನಮತ್ತುಎಂಜಿನಿಯರಿಂಗ್ನವಿದ್ಯಾರ್ಥಿಗಳಿಗೆ 24 ಗಂಟೆಯೂಲಭ್ಯವಿದೆ.

P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Department of Information Science & Engineering

Vision/ಆಶಯ

"The department strives to equip our graduates with Knowledge and Skills to contribute significantly to Information Science & Engineering and enhance quality research for the benefit of society".

"ಮಾಹಿತಿವಿಜ್ಞಾನಮತ್ತುಎಂಜಿನಿಯರಿಂಗ್ಗೆಗಣನೀಯಕೊಡುಗೆನೀಡಲುಮತ್ತುಸಮಾಜದಪ್ರಯೋಜನಕ್ಕಾಗಿಗುಣಮಟ್ಟದಸಂಶೋಧನೆಯನ್ನುಹೆಚ್ಚಿಸಲುನಮ್ಮಪದವೀಧರರನ್ನುಜ್ಞಾನಮತ್ತುಕೌಶಲ್ಯಗಳೊಂದಿಗೆಸಜ್ಜುಗೊಳಿಸಲುಇಲಾಖೆಯುಶ್ರಮಿಸುತ್ತದೆ."

Mission/ ಧ್ಯೇಯ

- To provide students with state of art facilities and tools of Information Science & Engineering to become productive, global citizens and life-long learners.
  ವಿದ್ಯಾರ್ಥಿಗಳಿಗೆಅತ್ಯಾಧುನಿಕಸೌಲಭ್ಯಗಳುಮತ್ತುಮಾಹಿತಿವಿಜ್ಞಾನಮತ್ತುಎಂಜಿನಿಯರಿಂಗ್ಉಪಕರಣಗಳನ್ನುಉತ್ಪಾದಕ,
  ಜಾಗತಿಕನಾಗರಿಕರುಮತ್ತುಜೀವನಪರ್ಯಂತಕಲಿಯುವವರನ್ನಾಗಿಮಾಡಲು.
- To prepare students for careers in IT industry, Higher education and Research.
  ಐಟಿಉದ್ಯಮ, ಉನ್ನತಶಿಕ್ಷಣಮತ್ತುಸಂಶೋಧನೆಗಾಗಿವಿದ್ಯಾರ್ಥಿಗಳನ್ನುತಯಾರಿಸಲು.
- To inculcate leadership qualities among students to make them competent Information Science & Engineering professionals or entrepreneurs.
  ವಿದ್ಯಾರ್ಥಿಗಳಲ್ಲಿನಾಯಕತ್ವಗುಣಗಳನ್ನುಬೆಳೆಸಲುಅವರನ್ನುಸಮರ್ಥಮಾಹಿತಿವಿಜ್ಞಾನಮತ್ತುಎಂಜಿನಿಯರಿಂಗ್ವೃತ್ತಿಪರರುಅಥವಾಉದ್ಯಮಿಗಳನ್ನಾಗಿಮಾಡಲು.

Program Educational Objectives (PEOs): Graduates of the program will be able to

PEO1: Establish a productive Information Science & Engineering career in industry, government or academia.

PEO2: Interact with their peers in other disciplines by exhibiting professionalism and team work to contribute to the economic growth of the country.

PEO3: Promote the development of innovative systems and solutions to the problems in Information Science using hardware and software integration.

PEO4: Pursue higher studies in Engineering, Management or Research.

Program Specific Outcomes (PSOs)

PSO1.

PSO2.

P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Department of Information Science & Engineering

Program Outcomes (POs)

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 modeling 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.

P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Department of Information Science & Engineering

Course Overview

Operating systems (OS) provide the crucial interface between a computer's hardware and the applications that run on it. It allows us to write programs without bothering much about the hardware. It also ensures that the computer's resources such as its CPU, hard disk, and memory, are appropriately utilized. In this course, we dwell into how the OS manages to do all this in an efficient manner.

Course Objectives

The objectives of this course are to make students to learn,

1. To understand the foundation knowledge in database concepts, technology and practice to prepare students into well-informed database application developers.
2. Strong practice in SQL programming through a variety of database problems.
3. Develop database applications using front-end tools and back-end DBMS.

Course Outcomes

After learning all the units of the course, students is able to:

1. Understand database language commands to create simple database
2. Analyze the database using queries to retrieve records
3. Analyze front end tools to design forms, reports and menus
4. Develop solutions using database concepts for real time requirements.

P E S Education Trust(R), Mandya

P E S College of Engineering

(An Autonomous Institution Affiliated to VTU, Belagavi)

Department of Information Science & Engineering

Syllabus

1. Program to implement the Process system calls.
2. Program to create a Process using API.
3. Program to implement Sequential file allocation method.
4. Program to simulate Single level directory file organization technique
5. Program to simulate the concept of Dining-Philosopher's problem.
6. Program to implement CPU scheduling algorithm for Shortest Job First CPU Scheduling algorithm.
7. Simulate Banker's algorithm for Dead Lock Avoidance.
8. Program to implement and simulate the MFT algorithm
9. Program to implement FIFO page replacement technique.
10. Program to simulate FCFS Disk scheduling algorithm.

1. Program to implement the Process system calls.

DESCRIPTION

The creation of processes using system calls in Linux is a fundamental concept in operating systems that enables the efficient execution of multiple tasks concurrently. System calls are essential interfaces between user-level applications and the underlying kernel. The process creation sequence involves a series of coordinated steps, each facilitated by specific system calls. The process creation begins with the fork() system call. This call duplicates the existing process, resulting in a parent process and a child process that share the same code and data. The child process is an exact copy of the parent, but they each have their unique process IDs (PIDs). Following the fork() call, the child process often invokes the exec() system call. Process communication and synchronization are achieved through other system calls like wait().

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
int pid;
pid=fork();
if(pid<0)
{
printf("fork failed");
exit(1);
}
else if(pid==0)
{
execlp("whoami","ls",NULL);
exit(0);
}
```

```
else
{
printf("\n Process id is :%d\n",getpid());
wait(NULL);
exit(0);
}
}
```

OUTPUT

```
Process id is : 20204
```

2. Program to create a Process using API.

DESCRIPTION

Creating processes using the Windows API involves the CreateProcess() function to launch new programs. Security attributes, environment variables, and synchronization mechanisms like WaitForSingleObject() are used.

SOURCE CODE

```
#include<stdio.h>
#include<windows.h>
int main()
{
   HANDLE hProcess,hThread;
   STARTUPINFO si;
   PROCESS_INFORMATION pi;
   ZeroMemory(&si,sizeof(si));
   si.cb=sizeof(si);
   ZeroMemory(&pi,sizeof(pi));


if(!CreateProcess("C:\\Windows\\System32\\cmd.exe",NULL,NULL,NULL,FALSE,0,NUL
L,NULL,&si,&pi))
   {
      printf("Sorry! CreateProcess() failed\n");
      return -1;
   }
   else
   {
      printf("Well CreateProcess() looks OK\n");
      printf("exit after 5000ms\n");
   }

   WaitForSingleObject(pi.hProcess,5000);
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);
   return 0;
}
```

OUTPUT

Well CreateProcess() looks OK

exit after 5000ms

3. Program to implement Sequential file allocation method.

DESCRIPTION:

In this file organization, the records of the file are stored one after another both physically and logically. That is, record with sequence number 16 is located just after the 15th record. A record of a sequential file can only be accessed by reading all the previous records.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
int sb, nob;
}ft[30];
void main()
{
int i, j, n;
char s[20];
clrscr();
printf("Enter no of files :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d :",i+1);
scanf("%s",ft[i].name);
printf("Enter starting block of file %d :",i+1);
scanf("%d",&ft[i].sb);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
}
```

```
printf("\nEnter the file name to be searched-- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");
printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].sb+j);
}
getch();
}
```

INPUT:

Enter no of files :3
Enter file name 1: A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6
Enter file name 2: B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4
Enter file name 3: C
Enter starting block of file 3 :60
Enter no of blocks in file 3:4
Enter the file name to be searched -- B

OUTPUT:

FILE NAME	START BLOCK	NO OF BLOCKS	BLOCKS OCCUPIED
B	102	4	102, 103, 104, 105

4. Program to simulate Single level directory file organization technique

DESCRIPTION

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory.

SOURCE CODE

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch; char
f[30]; clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++; break;
case 2: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
```

```
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f); strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\nDirectory Empty");
else
{
printf("\nThe Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit (0);
}
```

```
}
getch();
}
```

OUTPUT:

Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4
The Files are -- A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3
Enter the name of the file – ABC File
ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2
Enter the name of the file – B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5

5. **Program to simulate the concept of Dining-Philosopher's problem.**

DESCRIPTION

The dining-philosophers problem is considered a classic synchronization problem because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. Obviously, she cam1ot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophers problem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence of deadlock

SOURCE CODE

```
#include<stdio.h>
int tph, philname[20], status[20], howhung, hu[20], cho;
void main()
{
int i; clrscr();
printf("\n\nDINING PHILOSOPHER PROBLEM");
printf("\nEnter the total no. of philosophers: ");
scanf("%d",&tph);
for(i=0;i<tph;i++)
{
philname[i]=(i+1); status[i]=1;
}
printf("How many are hungry : ");
scanf("%d", &howhung);
if(howhung==tph)
{
printf("\n All are hungry..\nDead lock stage will occur");
printf(\n"Exiting\n");
```

```
else{
for(i=0;i<howhung;i++){

printf(“Enterphilosopher%dposition:”,(i+1));
scanf(“%d”,&hu[i]);
status[hu[i]]=2;
}
do
{
printf("1.One can eat at a time\t2.Two can eat at a time
\t3.Exit\nEnter your choice:");
scanf("%d", &cho);
switch(cho)
{
case 1: one();
break;
case 2: two();
break;
case 3: exit(0);
default: printf("\nInvalid option..");
}
}while(1);
}
}
void one()
{
int pos=0, x, i;
printf("\nAllow one philosopher to eat at any time\n");
for(i=0;i<howhung; i++, pos++)
{
printf("\nP %d is granted to eat", philname[hu[pos]]);
for(x=pos;x<howhung;x++)
printf("\nP %d is waiting", philname[hu[x]]);
}
}
```

```
void two()
{
int i, j, s=0, t, r, x;

printf("\n Allow two philosophers to eat at same
time\n"); for(i=0;i<howhung;i++)
{
for(j=i+1;j<howhung;j++)
{
if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)
{
printf("\n\ncombination %d \n", (s+1));
t=hu[i];
r=hu[j]; s++;
printf("\nP %d and P %d are granted to eat", philname[hu[i]],
philname[hu[j]]);
for(x=0;x<howhung;x++)
{
if((hu[x]!=t)&&(hu[x]!=r))
printf("\nP %d is waiting", philname[hu[x]]);
}
}
}
}
}
```

INPUT

DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
How many are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

OUTPUT

1.One can eat at a time 2.Two can
eat at a time 3.Exit Enter your choice: 1
Allow one philosopher to eat at any time
P 3 is granted to eat
P 3 is waiting
P 5 is waiting
P 0 is waiting
P 5 is granted to eat
P 5 is waiting
P 0 is waiting
P 0 is granted to eat
P 0 is waiting
1.One can eat at a time 2.Two can eat at a time 3.Exit
Enter your choice: 2
Allow two philosophers to eat at same time
combination 1
P 3 and P 5 are granted to eat
P 0 is waiting
combination 2
P 3 and P 0 are granted to eat
P 5 is waiting
combination 3
P 5 and P 0 are granted to eat
P 3 is waiting
1.One can eat at a time 2.Two can
eat at a time 3.Exit Enter your choice: 3

6. Program to implement CPU scheduling algorithm for Shortest Job First CPU Scheduling algorithm.

DESCRIPTION

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

SOURCE CODE

```
#include<stdio.h>
#include<conio.h>
struct process
{
int pid;
int bt;
int wt;
int tt;
}p[10],temp;
int main()
{
int i,j,n,totwt,tottt;
float avg1,avg2;
clrscr();
printf("\nEnter the number of process:\t");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
p[i].pid=i;
printf("\nEnter the burst time:\t");
scanf("%d",&p[i].bt);
```

```
}
for(i=1;i<n;i++){

for(j=i+1;j<=n;j++)
{
if(p[i].bt>p[j].bt)
{
temp.pid=p[i].pid;
p[i].pid=p[j].pid;
p[j].pid=temp.pid;
temp.bt=p[i].bt;p[i].bt=p[j].bt;
p[j].bt=temp.bt;
}}}
p[1].wt=0;
p[1].tt=p[1].bt+p[1].wt;
i=2;
while(i<=n){
p[i].wt=p[i-1].bt+p[i-1].wt;
p[i].tt=p[i].bt+p[i].wt;
i++;
}
i=1;
totwt=tottt=0;
printf("\nProcess id \tbt \twt \ttt");
while(i<=n){
printf("\n\t%d \t%d \t%d t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
totwt=p[i].wt+totwt;
tottt=p[i].tt+tottt;
i++;
} avg1=totwt/n;
avg2=tottt/n;
```

```
printf("\n Average waiting time=%f\n Average turnaround time=%f",avg1,avg2);
getch();
return 0;

}
```

INPUT

enter the number of processes: 3

enter the burst time: 2

enter the burst time: 4

enter the burst time: 6

OUTPUT

processid	bt	wt	tt
1	2	0	2
2	4	2	6
3	6	6	12

Average waiting time =2.000000

Average turnaround time =6.000000

7. Simulate Banker's algorithm for Dead Lock Avoidance.

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the userrequest a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources. Data structures n-Number of process, m-number of resource types.

Available: Available[j]=k, k – instance of resource type Rj is available.

Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj

Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]- Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both Finish[i] =False Need<=Work If no such I exists go to step 4.
3. work= work + Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2.
   Otherwise raise an error condition.
2. if Request<=Available go to step 3.
   Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
   Available=Available-Request I;
   Allocation I=Allocation +Request I;
   Need i=Need i- Request I;

   If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

SOURCE CODE

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();
printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0; for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
```

```
A:
for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d", i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\n The given state is safe state");
getch();
}
```

OUTPUT

```
Enter the no. of processes and resources: 4 3
Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
```

0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2
Available resources are: 6 2 3
Process 2 executed?:y
All the resources can be allocated to Process 3 Available resources are: 8 3 4
Process 3 executed?:y
All the resources can be allocated to Process 4 Available resources are: 8 3 6
Process 4 executed?:y
All the resources can be allocated to Process 1
Available resources are: 9 3 6
Process 1 executed?:y
System is in safe mode
The given state is safe state

8. Program to implement and simulate the MFT algorithm

DESCRIPTION

MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MVT (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.

SOURCE CODE

```
#include<stdio.h>
#include<conio.h>
main()
{
int ms, bs, nob, ef,n,
mp[10],tif=0; int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
```

```
scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory--%d",nob);
printf("\n\nPROCESS\tMEMORYREQUIRED\tALLOCATED\tINTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t%d",bs-mp[i]);
tif = tif + bs-mp[i];

p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
getch();
}
```

INPUT

Enter the total memory available (in Bytes) -- 1000

Enter the block size (in Bytes)-- 300

Enter the number of processes – 5

Enter memory required for process 1 (in Bytes) -- 275

Enter memory required for process 2 (in Bytes) -- 400

Enter memory required for process 3 (in Bytes) -- 290

Enter memory required for process 4 (in Bytes) -- 293

Enter memory required for process 5 (in Bytes) -- 100

No. of Blocks available in memory – 3

OUTPUT

PROCESS	MEMORY REQUIRED	ALLOCATED	INTERNAL FRAGMENTATION
1	275	YES	25
2	400	NO	-----
3	290	YES	10
4	293	YES	7

Memory is Full, Remaining Processes cannot be accommodated

Internal Fragmentation is 42

Total External Fragmentation is 100

9. Program to implement FIFO page replacement technique.

DESCRIPTION

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

SOURCE CODE

```
#include<stdio.h>
#include<conio.h> int fr[3];
void main() {
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int
flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{fr[i]=-1;
}
for(j=0;j<12;j++) {
flag1=0; flag2=0; for(i=0;i<12;i++) {
if(fr[i]==page[j]) {
flag1=1; flag2=1; break; }}
if(flag1==0) {
for(i=0;i<frsize;i++)
{if(fr[i] = = -1)
{
fr[i]=page[j]; flag2=1; break; }}}
```

```
if(flag2==0) {
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0; }
display(); }
Page 31
printf("Number of page faults : %d ",pf+frsize);
getch();
}
void display()
{
int i; printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2 3 -1
2 3 -1
2 3 1
5 3 1
5 2 1
5 2 4
5 2 4
3 2 4
3 2 4
3 5 4
3 5 2
Number of page faults: 9
```

10. Program to simulate FCFS Disk scheduling algorithm.

DESCRIPTION

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service.

SOURCE CODE

```
#include<stdio.h>
void main()
{
int t[20], n, I, j, tohm[20], tot=0;
 float avhm;
 clrscr();
 printf("enter the no.of tracks");
 scanf("%d",&n);
 printf("enter the tracks to be traversed");
 for(i=2;i<n+2;i++)
 scanf("%d",&t*i+);
 for(i=1;i<n+1;i++)
 {
 tohm[i]=t[i+1]-t[i];
 if(tohm[i]<0)
 tohm[i]=tohm[i]*(-1);
 }
 for(i=1;i<n+1;i++)
 tot+=tohm[i];
 avhm=(float)tot/n;
```

```
printf("Tracks traversed\tDifference between tracks\n");
 for(i=1;i<n+1;i++)
 printf("%d\t\t\t%d\n",t*i+,tohm*i+);
 printf("\nAverage header movements:%f",avhm);
 getch();
}
```

INPUT

Enter no of tracks:9

Enter track position:55 58 60 70 18 90 150 160 184

OUTPUT

Tracks traversed	Difference between tracks
55	45
58	3
60	2
70	10
18	52
90	72
150	60
160	10
184	24

Average header movements:30.888889