S.No.	Date	PARTICULAR	Page No.	Remarks
1.		Identifying the components of		
		the computer along with block diagram		
2.		Assembling & disassembling the		
		components of CPU.		
3.		WACP to implement cat command		
4.		WACP to implement ls command		
5.		WACP to implement mv command		
6.		WACP to implement FCFS		
		scheduling Algorithm.		
7.		WACP to implement SJF-non		
		preemptive scheduling Algorithm		
8.		WACP to print process Id &		
		parent process Id.		
9.		WACP to implement fork() system call		
10.		WACP to create child process &		
		allow the parent to print parent &		
		child to print child on the screen		
11.		WACP to implement wait() system call		
12.		WACP to implement waitpid() system call		
13.		WACP to implement exec() system call.		

Experiment-1

Input Devices

Keyboard

The keyboard stands out as one of the most prevalent input devices, allowing users to manually input data through typing. Resembling a typewriter keyboard, it features additional specialized keys, such as function keys and control keys

Mouse

The mouse functions primarily as a pointing device, serving as input tool for transmitting the coordinates of the screen's cursor position. Mouse interacts with graphical user interface (GUI) software by relaying precise location data

Joystick and Tracker ball

Joysticks and trackballs also function as pointing devices, akin to the mouse, serving a similar purpose in facilitating user interaction with graphical interfaces.

Scanner

Scanners capture comprehensive image data directly from their source, typically a page, and store it in a graphic format suitable for display on screens.

Touch pads:

Touch pads are tactile devices positioned on desktop surfaces, designed to detect and react to applied pressure.

Bar code Readers

A Barcode Reader is instrumental in this process, scanning the barcode image and converting it into an alphanumeric value. This value is then transmitted to the connected computer, facilitating efficient data entry and management.

optical Mark Reader (OMR)

OMR (optical Mark Recognition) technology is a specialized form of optical scanning utilized for identifying marks made by pens or pencils

Output Devices

video display unit

A video display unit, reminiscent of a television screen, serves as a visual interface connected to a computer. This device, also known as a raster scan device, renders digital information into visible images.

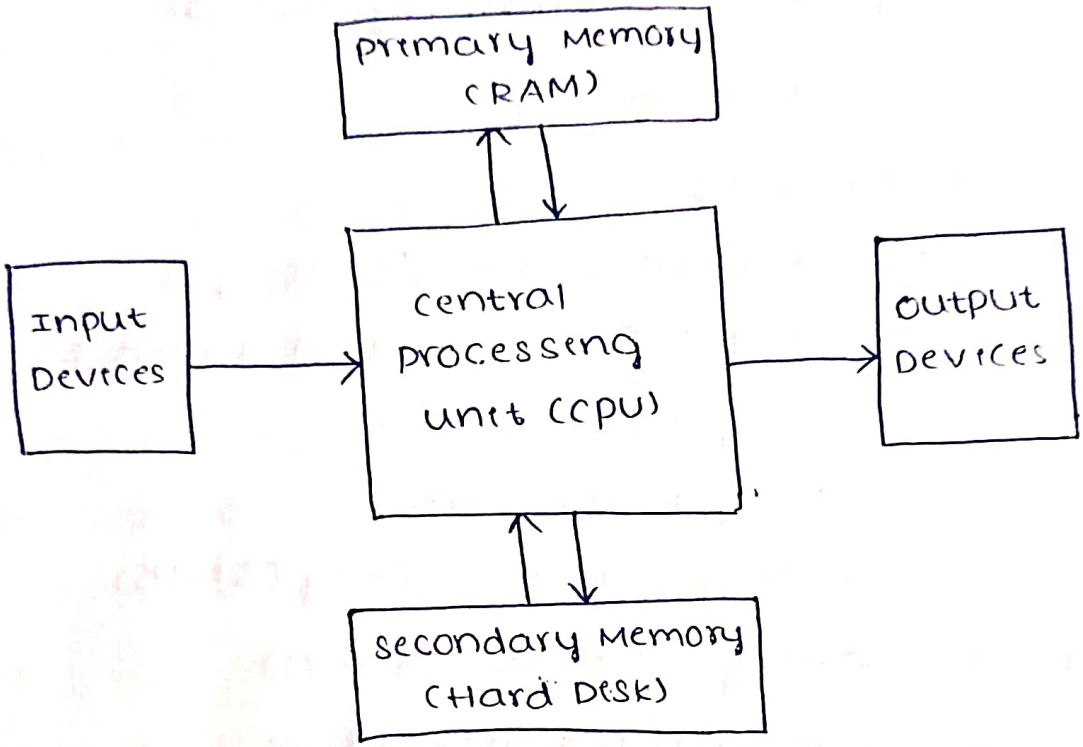
printers

There are several types of printers available, each designed for specific printing needs:
character printer: These printers operate by printing one character at a time.

Line printer: Line printers are capable of printing one line of text at a time

page printer: page printers are designed to print an entire page at once

components of CPU and it's functions



central processing unit (CPU):-

a) Memory unit

Memory unit plays a crucial role in storing and retrieving data. It significantly influences the performance of the computer. Memory is of 2 types. They are primary Memory and secondary Memory.

b) Arithmetic and Logic unit (ALU)

The ALU serves as the computational powerhouse within the CPU, responsible for executing a wide range of arithmetic and logical operations.

c) control unit

The control unit serves as the central coordinator within the computer system, directing and orchestrating the activities of other units.

Experiment-2.

Disassembling and assembling the pc

→ Steps for Disassembling

step-1: Unplugging
step-2: Removing outer shell/casing.
step-3: Removing TOP panel and Front panel.
step-4: Removing system Fan.
step-5: Removing & CPU Fan.
step-6: Removing power supply (SMPS)
step-7: Removing CD/DVD Drive(s).
step-8: Removing Hard Disk Drive.
step-9: Removing Expansion cards
step-10: Removing connectivity centre cables (front and back panel)
step-11: Removing RAM (Random Access Memory)
step-12: Removing Power Button, Power LED + HDD LED
step-13: Removing Motherboard
step-14: Done!

→ steps for Assembling

step-1: Get Ready.

step-2: put in the power supply.

step-3: Attach the Motherboard

step-4: Install the processor (CPU)

step-5: Attach the CPU cooler

step-6: Add Memory (RAM)

step-7: Install storage Dreves.

step-8: connect Front panel wires

step-9: put in Expansion cards

step-10: connect case Fans

step-11: organize cables

step-12: check everything

step-13: close the case.

step-14: Turn it ON.

step-15: Install software.

step-16: Install Drivers.

3) Write a C program to view the contents of a given file, similar to the cat command in the linux environment.

```
#include <stdio.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
   char ch;
   int fd;
   fd=open(argv[1], O_RDONLY);
  if(fd==-1)
  {
    printf("\n File not found");
  }
  while(read(fd, &ch, 1))
     putchar(ch);
 close(fd);
 return 0;
}
```

```
O/P:- gcc catc.c -o h
      ./h b.txt
      Hello, World!
```

4) Write a C program to implement ls command

```
#include<stdio.h>
#include<dirent.h>
int main(int argc, char *argv[])
{
  DIR *dirop;
  struct dirent *dired;
  if(argc!=2)
  {
  printf("\n Invalid no. of arguments");
  }
if((dirop=opendir(argv[1]))==NULL)
   printf("\n cannot open the directory");
while(dired=readdir(dirop))!=NULL)
    printf("%s\n", dired->d_name);
 closedir(dirop);
}
```

cd sample
touch c.txt
touch s.txt
gcc isprog.c -o n
./n sample

o/p:-
:
c.txt
s.txt

5) Write a c program to implement mv command.

→ copying the contents from source file to destenation file and deleting the source file.

```
#include<stdio.h>
int main(int argc, char *argv[])
{
   int r;
   if(argc<3)
   { printf("wrong arguments\n");
   }
  else if(argc==3)
  { printf("\n %s %s \n", argv[1], argv[2]);
    r=link(argv[1], argv[2]);
    printf("%d\n", r);
    unlink(argv[1]);
  }
 return 0;
}
```

O/p:- gcc mvprog.c -o mvprog↵

./mvprog mvprog.c mvprog1.c↵

mvprog.c mvprog1.c

0

1/4/22

6) Write a c program to implement FCFS scheduling algorithm to calculate average WT and average TAT.

```
#include<stdio.h>
struct process {
  int id;
  int arrival_time;
  int burst_time;
  int completion_time;
  int turnaround_time;
  int waiting_time;
};
int main()
{
   int n,i;
   float total_waiting_time =0, total_turnaround_
   time =0;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   struct process p[n];
   printf("Enter arrival time and burst time
   for each process:\n");
   for(i=0;i<n;++i) {
    printf("For process %d:\n", i+1);
    printf("Arrival time:");
    scanf("%d", &p[i].arrival_time);
    printf("Burst time:");
```

```
scanf("%d", &p[i].burst_time);
p[i].id = i+1;
}
p[0].completion_time = p[0].arrival_time+
p[0].burst_time;
p[0].turnaround_time = p[0].completion_time-
p[0].arrival_time;
p[0].waiting_time=0;
for(i=1;i<n;++i)
{
  p[i].completion_time = p[i-1].completion_time+
  p[i].burst_time;
  p[i].turnaround_time = p[i].completion_time-
  p[i].arrival_time;
  p[i].waiting_time = p[i].turnaround_time-
  p[i].burst_time;
}
printf("\n process\t Arrival Time\t Burst Time
\t completion Time\t Turnaround Time\t
Waiting Time\n");
for(i=0;i<n;++i)
{ printf("%d\t%d\t\t%d\t\t%d\t\t%d\t
    \t\t%d\n", p[i].id, p[i].arrival_time,
  p[i].burst_time, p[i].completion_time,
  p[i].turnaround_time, p[i].waiting_time);
total_waiting_time+ =p[i].waiting_time;
total_turnaround_time+ =p[i].turnaround_time;
}
```

```
float avg_waiting_time = total_waiting_time/n;
float avg_turnaround_time = total_turnaround_
  time/n;
printf("\nAverage Waiting Time: %.2f\n",
    avg_waiting_time);
printf("\nAverage Turnaround Time: %.2f\n",
    avg_turnaround_time);
  return 0;
}
```

Output:-

```
Enter the number of processes: 4
Enter arrival time and burst time for
each process:
For process 1:
Arrival time: 0
Burst time: 6
For process 2:
Arrival time: 0
Burst time: 8
For process 3:
Arrival time: 0
Burst time: 2
For process 4:
Arrival time: 0
Burst time: 1
```

Process	Arrival Time	Burst Time	completion Time	Turnaround Time	Waiting Time
1	0	6	6	6	0
2	0	8	14	14	6
3	0	2	16	16	14
4	0	1	17	17	16

Average Waiting Time: 9.00

Average Turnaround Time: 13.25

7) Write a c program to emplement SJF scheduling algorithm to calculate avg WT and avg TAT.

```
#include<stdio.h>
struct process{
int process_id;
int arrival_time;
int burst_time;
};
void sjf_non_preemptive(process p[], int n)
{
  for(int i=0;i<n-1;i++)
  {
    for(int j=i+1;j<n;j++)
    {
    if(p[i].burst_time > p[j].burst_time)
    {
        Process temp = p[i];
        p[i] = p[j];
        p[j] = temp;
    }
  }
}
```

```
int completion_time[n];
int turnaround_time[n];
int waiting_time[n];
float total_waiting_time=0;
float total_turnaround_time=0;
completion_time[0] = p[0].burst_time;
turnaround_time[0] = completion_time[0] -
    p[0].arrival_time;
waiting_time[0] = turnaround_time[0] -
    p[0].burst_time;
total_waiting_time += waiting_time[0];
total_turnaround_time += turnaround_time[0];
for(int i=1; i<n; i++)
{
    completion_time[i] = completion_time[i-1] +
    p[i].burst_time;
    turnaround_time[i] = completion_time[i] -
    p[i].arrival_time;
    waiting_time[i] = turnaround_time[i] -
    p[i].burst_time;
    total_waiting_time += waiting_time[i];
    total_turnaround_time += turnaround_time[i];
}
printf("Process\t Arrival Time\t Burst Time\t
completion Time\t Turnaround Time\t Waiting
Time\n");
for(int i=0; i<n; i++)
{ printf("p%d\t %d\t\t %d\t\t %d\t\t
    %d\t\t\t %d\n", p[i].process_id, p[i].arrival_time,
```

```
p[i].burst_time, completion_time[i], turnaround_time[i], waiting_time[i]);
}
printf("\n Average turnaround time: %.2f\n",
total_turnaround_time/n);
printf("\n Average waiting Time: %.2f\n",
total_waiting_time/n);
}
int main()
{ int n;
  printf("Enter number of processes:");
  scanf("%d", &n);
 process p[n];
 printf("Enter burst time:\n");
 for(int i=0; i<n; i++)
 { printf("p%d:", i+1);
   scanf("%d", &p[i].burst_time);
   p[i].process_id = i+1;
   p[i].arrival_time = 0;
 }
 sjf_non_preemptive(p, n);
 return 0;
}
```

Output:-

Enter number of processes: 4

Enter burst time:

P1: 6

P2: 8

P3: 7

P4: 3

process	Arrival Time	Burst Time	completion Time	Turnaround Time	Waiting Time
P4	0	3	3	3	0
P1	0	6	9	9	3
P3	0	7	16	16	9
P2	0	8	24	24	16

Average Turnaround Time: 13.00

Average Waiting Time: 7.00

2) Write a c program to print process id & Parent process id.

Pid.c

```
#include<stdio.h>
#include <unistd.h>
#include<sys/types.h>
int main()
{
  printf("The process id PID=%d \n parent
  process id PPID=%d \n", getpid(), getppid());
  return 0;
}
```

O/P:-

```
~$ gcc pid.c
~$ ./a.out
The process id PID=51
The process
parent process id PPID=13
```

9) Write a c program to implement fork() system call.

fork.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
  fork();
  printf("The process id PID = %d \n parent
  process id PPID = %d \n", getpid(), getPPid());
  return 0;
}
```

O/P:-

```
~$ gcc fork.c
~$ ./a.out
The process id PID = 61
The parent process id PPID = 13
The process id PID = 62
 Parent process id PPID = 61
```

10) Write a c program to create child process and allow the parent to print parent & child to print the child on the screen.

parent.c

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
  pid_t p;
  p=fork();
  if(p==0)
  {
    printf("Thes is child process\n");
  }
  else if(p>0)
  {
    printf("Thes is parent process\n");
  }
  else
  {
    printf("Thes is error\n");
  }
  return 0;
}
```

o/p:-

```
~$ gcc parent.c
~$ ./a.out
Thes is parent process
Thes is child process
```

11) Write a C program to implement wait() system call.

wait.c

```
#include<stdio.h>
#include<unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include<sys/wait.h>
int main()
{ pid_t p,z;
  printf("This is message before fork()\n");
  p=fork();
  if(p==0)
  { printf("\n I am 1st child with PID=%d",
      getpid());
    printf("\n parent of 1st child with PID=%d",
     getppid());
  }
  else
  {
    z=forkc();
    if(z==0)
    {
      printf("\nI am 2nd child with PID=%d",
        getpid());
      printf("\n parent of 2nd child with PID=%d",
        getppid());
    }
    else
    { wait(NULL);
      printf("\n I am parent PID=%d",getpid());
```

```
printf("\n My 1st child PID = %d", p);
printf("\n My 2nd child PID = %d\n", 2);
  }
}
return 0;
}
```

O/p:-

```
~$ gcc wait.c
~$ ./a.out
```

Thes is message before fork()

I am 1st child with PID = 87

parent of 1st child with PID = 86

I am 2nd child with PID = 88

parent of 2nd child with PID = 86

I am parent PID = 86

My 1st child PID = 87

My 2nd child PID = 88

12) Write a c program to implement Waitpid() System call.

waitpid.c

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
  pid_t p,z;
  printf("This is message before fork()\n");
  p=fork();
  if(p==0)
  {
   sleep(5);
   printf("I am 1st child with PID=%d",
        getpid());
  printf("\n parent of 1st child with PID=%d\n",
        getppid());
   }
  else
  {
    z=fork();
   if(z==0)
  {
   printf("I am 2nd child with PID=%d",
     getpid());
  printf(" parent of 2nd child with PID=%d",
     getppid());
}
```

```
else
{
  waitpid(2, NULL, 0);
printf("\n I am parent PID = %d", getpid());
printf("\n My 1st child PID = %d", p);
printf("\n My 2nd child PID = %d", 2);
 }
}
return 0;
}
```

o/p:-

```
~$ gcc waitpid.c
~$ ./a.out
```

This is message before fork()

I am 2nd child with PID=104

parent of 2nd child with PID=102

I am parent with PID=102

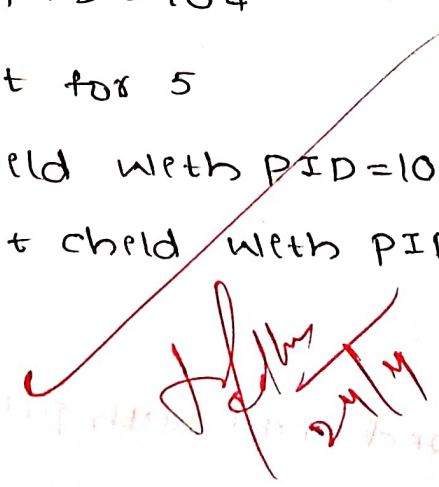
My 1st child PID = 103

My 2nd child PID = 104

---- wait for 5

I am 1st child with PID=103

parent of 1st child with PID=12



13) Write a c program to implement execl() system call.

file1.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
  printf("PID of file1 = %d\n", getpid());
  char *args[] = {"./file2", "./file3", NULL};
  execv(args[1], args);
  printf("\n Back to file1.c\n");
  return 0;
}
```

file2.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{ printf("\n we are in file2.c\n");
  printf("\n PID of file2.c = %d\n", getpid());
  return 0;
}
```

file3.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("\n we are in file3.c\n");
    printf("\n PID of file3.c = %d\n", getPid());
    return 0;
}
```

O/P:-

```
~$ gcc file3.c -o file3
~$ gcc file2.c -o file2
~$ gcc file1.c -o file1
~$ ./a.out
PID of file1 = 131
we are in file3.c
PID of file3.c = 131
```