Part -A

1.Write a python program to Get information about the operating system.

```
import os

# Get current working directory
current_directory = os.getcwd()
print("Current working directory:", current_directory)

print("\nFiles and directories in current directory:")
for item in os.listdir(current_directory):
    print(item)
# Create a new directory
new_directory = "test_directory"
os.mkdir(new_directory)
print("\nCreated directory:", new_directory)

# Rename the directory
new_directory_name = "renamed_directory"
os.rename(new_directory, new_directory_name)
print("Renamed directory to:", new_directory_name)

# Remove the directory
os.rmdir(new_directory_name)
print("Removed directory:", new_directory_name)

# Get information about the operating system
print("\nOperating system information:")
print("Name:", os.name)
print("PID:", os.getpid())
print("Parent PID:", os.getppid())
print("User ID:", os.getuid())
print("Group ID:", os.getgid())
print("Effective User ID:", os.geteuid())
print("Effective Group ID:", os.getegid())
```

Output 1:

```
Files and directories in current directory:
F5.py
 A1.py

Created directory: test_directory
Renamed directory to: renamed_directory
Removed directory: renamed_directory

Operating system information:
Name: posix
PID: 3831
Parent PID: 3818
User ID: 1000
Group ID: 1000
Effective User ID: 1000
Effective Group ID: 1000
```

2.Write a python program to get CPU and RAM usage.

```
import os
import psutil

print("The CPU usage is : ")
print(psutil.cpu_percent(0.1))

# Getting % usage of virtual_memory ( 3rd field)
print('RAM memory % used:', psutil.virtual_memory()[2])
# Getting usage of virtual_memory in GB ( 4th field)
print('RAM Used (GB):',
psutil.virtual_memory()[3]/1000000000)
```

Output 2:

```
The CPU usage is :
4.9
RAM memory % used: 63.1
RAM Used (GB): 3.448119296
```

3.Write a python program to get information about CPU core.

```
import psutil

def get_cpu_info():
    # Get the number of physical and logical CPUs
    num_physical_cpus = psutil.cpu_count(logical=False)
    num_logical_cpus = psutil.cpu_count(logical=True)

    print("Number of physical CPUs:", num_physical_cpus)
    print("Number of logical CPUs:", num_logical_cpus)

    # Get detailed information about each CPU core
    cpu_info = []
    for cpu in range(num_logical_cpus):
        cpu_details = psutil.cpu_freq(percpu=True)[cpu]
        cpu_info.append({
            "CPU": cpu,
            "Max Frequency (MHz)": cpu_details.max,
            "Min Frequency (MHz)": cpu_details.min,
            "Current Frequency (MHz)": cpu_details.current
        })

    print("\nCPU core details:")
    for core in cpu_info:
        print(core)

if __name__ == "__main__":
    get_cpu_info()
```

Output 3:

```
Number of physical CPUs: 4
Number of logical CPUs: 4
CPU core details:
{'CPU': 0, 'Max Frequency (MHz)': 3500.0, 'Min Frequency (MHz)':
1600.0, 'Current Frequency (MHz)': 1387.528}
{'CPU': 1, 'Max Frequency (MHz)': 3500.0, 'Min Frequency (MHz)':
1600.0, 'Current Frequency (MHz)': 1434.333}
```

```
{'CPU': 2, 'Max Frequency (MHz)': 3500.0, 'Min Frequency (MHz)':
1600.0, 'Current Frequency (MHz)': 2492.419}
{'CPU': 3, 'Max Frequency (MHz)': 3500.0, 'Min Frequency (MHz)':
1600.0, 'Current Frequency (MHz)': 1376.639}
```

4.Write a python program to get information about power-supply.

```
import psutil

def get_power_info():
    # Get power status
    power_status = psutil.sensors_battery()

    if power_status is None:
        print("Power status: Running on ac power   .")
    else:
        percent = power_status.percent
        power_plugged = power_status.power_plugged
        seconds_left = power_status.secsleft

        if power_plugged:
            power_plugged_str = "Plugged in"
        else:
            power_plugged_str = "Not plugged in"

        print("Battery percentage:", percent)
        print("Power status:", power_plugged_str)
        if seconds_left is not None:
            print("Time left (seconds):", seconds_left)
        else:
            print("Time left: Unknown")

if __name__ == "__main__":
     get_power_info()
```

Output 4:

```
Power status: Running on ac power   .
```

(on laptop)

```
Battery percentage : 90%
Power status : Plugged in
```

5.Write a python program to get information about Display.

```
import subprocess

def get_display_info():
    try:

        xrandr_output = subprocess.check_output(['xrandr']).decode('utf-8')

        # Print the output
        print("Display information:")
        print(xrandr_output)
    except FileNotFoundError:
        print("xrandr command not found. This program requires xrandr utility which is
available on Linux systems.")

if __name__ == "__main__":
    get_display_info()
```

Output 5:

```
Display information:
Screen 0: minimum 320 x 200, current 3520 x 1080,
maximum 16384 x 16384
DVI-D-0 disconnected (normal left inverted right x
axis y axis)
HDMI-A-0 connected primary 1920x1080+1600+0 (normal
left inverted right x axis y axis) 476mm x 267mm
   1920x1080     60.00*+  50.00    59.94
   1680x1050     59.88
   1600x900      75.00    60.00
   1280x1024     75.02    70.00    60.02
```

6. Write a python program to get information about I/O.

```
import psutil

def get_io_info():
    # Get disk I/O statistics
    disk_io = psutil.disk_io_counters()
    print("Disk I/O statistics:")
    print("  Total read bytes:", disk_io.read_bytes)
    print("  Total write bytes:", disk_io.write_bytes)
    print("  Total read time:", disk_io.read_time)
    print("  Total write time:", disk_io.write_time)
    print()

    # Get network I/O statistics
    net_io = psutil.net_io_counters()
    print("Network I/O statistics:")
    print("  Total bytes sent:", net_io.bytes_sent)
    print("  Total bytes received:", net_io.bytes_recv)
    print("  Total packets sent:", net_io.packets_sent)
    print("  Total packets received:", net_io.packets_recv)

if __name__ == "__main__":
    get_io_info()
```

Output 6:

```
Disk I/O statistics:
  Total read bytes: 2024574464
  Total write bytes: 1696656384
  Total read time: 1379011
  Total write time: 1440066

Network I/O statistics:
  Total bytes sent: 23960489
```

```
  Total bytes received: 80777166
  Total packets sent: 77405
  Total packets received: 103989
```

Part -B

1.Write a python program to demonstrate FCFS scheduling Algorithm.

```
def calculateTime(n, burst_time):
      waiting_time = [0] * n
      turnaround_time = [0] * n
      avg_waiting_time = 0
      avg_turnaround_time = 0

      waiting_time[0] = 0

      # Calculating waiting time
      for i in range(1, n):
            waiting_time[i] = 0
            for j in range(i):
                  waiting_time[i] += burst_time[j]


      for i in range(n):
            turnaround_time[i] = burst_time[i] + waiting_time[i]
            avg_turnaround_time += turnaround_time[i]

      avg_turnaround_time /= n
      print(f"\nAverage Turnaround Time: {avg_turnaround_time}ms")

      # Calculating average waiting time
      for i in range(n):
            avg_waiting_time += waiting_time[i]

      avg_waiting_time /= n
      print(f"\nAverage Waiting Time: {avg_waiting_time}ms\n")

      print("Process\tBurst Time\tWaiting Time\tTurnaround Time")
      for i in range(n):

print(f"P[{i+1}]\t{burst_time[i]}\t\t{waiting_time[i]}\t\t{turnaround_ti
me[i]}")
```

```
# Driver code
if __name__ == '__main__':
      n = int(input("Enter total number of processes(maximum 20): "))

      burst_time = []
      print("\nEnter Process Burst Time")
      for i in range(n):
            burst_time.append(int(input(f"P[{i+1}]: ")))

      calculateTime(n, burst_time)
```

Output B1:

```
Enter total number of processes(maximum 20): 5

Enter Process Burst Time
P[1]: 4
P[2]: 3
P[3]: 4
P[4]: 5
P[5]: 5

Average Turnaround Time: 11.8ms

Average Waiting Time: 7.6ms

Process    Burst Time Waiting Time     Turnaround Time
P[1]  4          0          4
P[2]  3          4          7
P[3]  4          7          11
P[4]  5          11         16
P[5]  5          16         21
```

2.Write a python program to demonstrate SJF scheduling Algorithm.

```
def main():
      # Taking the number of processes
```

```
    n = int(input("Enter number of process: "))

    A = [[0 for j in range(4)] for i in range(100)]
    total, avg_wt, avg_tat = 0, 0, 0
    print("Enter Burst Time:")
    for i in range(n):
        A[i][1] = int(input(f"P{i+1}: "))
        A[i][0] = i + 1
    for i in range(n):
        index = i
        for j in range(i + 1, n):
            if A[j][1] < A[index][1]:
                index = j
        temp = A[i][1]
        A[i][1] = A[index][1]
        A[index][1] = temp
        temp = A[i][0]
        A[i][0] = A[index][0]
        A[index][0] = temp
    A[0][2] = 0
    for i in range(1, n):
        A[i][2] = 0
        for j in range(i):
            A[i][2] += A[j][1]
        total += A[i][2]
    avg_wt = total / n
    total = 0

    print("P    BT    WT    TAT")
    for i in range(n):
        A[i][3] = A[i][1] + A[i][2]
        total += A[i][3]
        print(f"P{A[i][0]}     {A[i][1]}  {A[i][2]}  {A[i][3]}")
    avg_tat = total / n
    print(f"Average Waiting Time= {avg_wt}")
    print(f"Average Turnaround Time= {avg_tat}")


if __name__ == "__main__":
    main()
```

Output b2:

Enter total number of processes(maximum 20): 3

```
Enter Process Burst Time
P[1]: 2
P[2]: 1
P[3]: 4

Average Turnaround Time: 4.0ms

Average Waiting Time: 1.6666666666666667ms

Process    Burst Time Waiting Time     Turnaround Time
P[1]  2          0          2
P[2]  1          2          3
P[3]  4          3          7
```

3.Write a python program to demonstrate Round Robin scheduling Algorithm.

```
def findWaitingTime(processes, n, bt,
                                        wt, quantum):
      rem_bt = [0] * n

      # Copy the burst time into rt[]
      for i in range(n):
            rem_bt[i] = bt[i]
      t = 0 # Current time

      while(1):
            done = True
            for i in range(n):
                  if (rem_bt[i] > 0) :
                        done = False # There is a pending process

                        if (rem_bt[i] > quantum) :

                              t += quantum
                              rem_bt[i] -= quantum

                        else:

                              t = t + rem_bt[i]
```

```
                                wt[i] = t - bt[i]

                                rem_bt[i] = 0

                # If all processes are done
                if (done == True):
                        break


def findTurnAroundTime(processes, n, bt, wt, tat):

        # Calculating turnaround time
        for i in range(n):
                tat[i] = bt[i] + wt[i]


def findavgTime(processes, n, bt, quantum):
        wt = [0] * n
        tat = [0] * n

        findWaitingTime(processes, n, bt, wt, quantum)

        findTurnAroundTime(processes, n, bt,wt, tat)


        print("Processes Burst Time       Waiting", "Time Turn-Around Time")
        total_wt = 0
        total_tat = 0
        for i in range(n):

                total_wt = total_wt + wt[i]
                total_tat = total_tat + tat[i]
                print(" ", i + 1, "\t\t", bt[i],"\t\t", wt[i], "\t\t", tat[i])

        print("\nAverage waiting time = %.5f "%(total_wt /n) )
        print("Average turn around time = %.5f "% (total_tat / n))

# Driver code
if __name__ =="__main__":

        # Process id's
        proc = [1, 2, 3]
        n = 3

        # Burst time of all processes
        burst_time = [10, 5, 8]

        # Time quantum
        quantum = 2;
        findavgTime(proc, n, burst_time, quantum)
```

Output:

```
Processes Burst Time   Waiting Time Turn-Around Time
  1         10         13         23
  2         5          10         15
  3         8          13         21

Average waiting time = 12.00000
Average turnaround time = 19.66667
```

4.Write a python program to demonstrate Deadlock using threads.

```
import threading

# Define two locks
lock1 = threading.Lock()
lock2 = threading.Lock()

def thread1():
    print("Thread 1 trying to acquire lock 1")
    lock1.acquire()
    print("Thread 1 acquired lock 1")

    print("Thread 1 trying to acquire lock 2")
    lock2.acquire()
    print("Thread 1 acquired lock 2")

    lock2.release()
    lock1.release()

def thread2():
    print("Thread 2 trying to acquire lock 2")
    lock2.acquire()
    print("Thread 2 acquired lock 2")

    print("Thread 2 trying to acquire lock 1")
    lock1.acquire()
    print("Thread 2 acquired lock 1")

    lock1.release()
```

```
    lock2.release()

if __name__ == "__main__":
    # Create two threads
    t1 = threading.Thread(target=thread1)
    t2 = threading.Thread(target=thread2)

    # Start the threads
    t1.start()
    t2.start()

    # Wait for threads to finish
    t1.join()
    t2.join()

    print("Main thread exiting")
```

Output:

```
Thread 1 trying to acquire lock 1
Thread 2 trying to acquire lock 2
Thread 2 acquired lock 2
Thread 1 acquired lock 1
Thread 2 trying to acquire lock 1
Thread 1 trying to acquire lock 2
```

5.Write a python program to demonstrate Best Fit- Memory Management Technique.

```
class MemoryManager:
    def __init__(self, memory_size):
        self.memory_size = memory_size
        self.memory = [0] * memory_size
        self.processes = {}



    def allocate_best_fit(self, process_id, size):
        best_fit_start = None
        best_fit_size = float('inf')
```

```
        for i in range(self.memory_size - size + 1):
            if all(self.memory[i + j] == 0 for j in range(size)):
                hole_size = sum(1 for j in range(self.memory_size)
if self.memory[j] == 0 and j >= i and j < i + size)
                if hole_size < best_fit_size:
                    best_fit_start = i
                    best_fit_size = hole_size

        if best_fit_start is not None:
            for j in range(size):
                self.memory[best_fit_start + j] = process_id
            self.processes[process_id] = (best_fit_start, size)
            return True
        return False



    def print_memory(self):
        print("Memory:")
        for i, val in enumerate(self.memory):
            print(val, end=" ")
            if (i + 1) % 20 == 0:
                print()
        print()

if __name__ == "__main__":
    memory_manager = MemoryManager(100)



    # Best fit allocation
    print("Best Fit Allocation:")
    memory_manager.allocate_best_fit("P4", 25)
    memory_manager.print_memory()
```

OUTPUT:

```
P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4
```

```
P4 P4 P4 P4 P4 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

6. Write a python program to demonstrate First Fit- Memory Management Technique

```
class MemoryManager:
    def __init__(self, memory_size):
        self.memory_size = memory_size
        self.memory = [0] * memory_size
        self.processes = {}

    def allocate_first_fit(self, process_id, size):
        for i in range(self.memory_size - size + 1):
            if all(self.memory[i + j] == 0 for j in range(size)):
                for j in range(size):
                    self.memory[i + j] = process_id
                self.processes[process_id] = (i, size)
                return True
        return False


    def print_memory(self):
        print("Memory:")
        for i, val in enumerate(self.memory):
            print(val, end=" ")
            if (i + 1) % 20 == 0:
                print()
        print()

if __name__ == "__main__":
    memory_manager = MemoryManager(100)

    # First fit allocation
    print("First Fit Allocation:")
```

```
    memory_manager.allocate_first_fit("P1", 20)
    memory_manager.print_memory()
```

Output:

```
First Fit Allocation:
Memory:
P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```