PRACTICAL MANUAL

OPERATING SYSTEM LAB

PROGRAM LIST:

1. Basic I/O programming. To implement CPU Scheduling Algorithms:
2. Shortest Job First Algorithm.
3. First Come First Served Algorithm.
4. Round Robin and Priority Scheduling Algorithms.
5. To implement reader/writer problem using semaphore.
6. To implement Banker's algorithm for Deadlock avoidance. Program for page replacement algorithms:
7. First In First Out Algorithm.
8. Least Recently Used Algorithm.
9. To implement first fit, best fit and worst fit algorithm for memory management.
10. Program for Inter-process Communication.

Ex. No: 01
DATE:

BASIC I/O OPERATIONS IN PYTHON

AIM:

To write PYTHON programs to simulate the basic I/O operations

DESCRIPTION:

To write basic I/O operations, python language provides various input and output functions. Some of the functions like input () and print () are widely used for standard input and output operations respectively.

ALGORITHM:

a) Create a process and print its name and ID

Step 1: Start the process to create a process and print its name and ID
Step 2: Import in python is similar to #include header file in C/C++. Python modules can get access to code from another module by importing the file/function using import. Use import statements.
Step 3: Create a process
Step 4: Using print () method process name and ID are received. Step 5: Stop the process.

b) Create a File and use file I/O concepts

Step 1: Start the process to perform file operations
Step 2: Create a file. Open it in write mode and write the contents to it
Step 3: Once again open the created file in read mode and read all the contents and print it
Step 4: In Python, seek () function is used to change the position of the File. Using this function read from a particular location.
Step 5: The readlines () method returns a list containing each line in the file as a list item. Using this function read the contents of the file and using print () method display the contents.
Step 6: Stop the process.

PROGRAM:

(a) Basic I/O Programming - Process Creation

```
import os
from multiprocessing import Process

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())
```

BASIC I/O OPERATIONS IN PYTHON

```
def f(name):
    info('function f')
    print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
    print('Child Process:', p.name)
    print("Child Process ID:", p.pid)
```

OUTPUT:

```
main line
module name:  main    parent process: 5288
process id: 13605 function f
module name:  main    parent process: 13605
process id: 13607 hello bob
```

(b) Basic I/O Programming file Operations

```
file1 = open('myfile.txt', 'w')
L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]

file1.write("Hello \n")
file1.writelines(L)
file1.close()

file1 = open("myfile.txt", "r+")
print("Output of Read function is ")
print(file1.read())
print()

file1.seek(0)
print("Output of ReadLine function is ")
print(file1.readline())
print()

file1.seek(0)
print("Output of Read(9) function is ")
print(file1.read(9))
print()
```

BASIC I/O OPERATIONS IN PYTHON

```
file1.seek(0)
print("Output of Readline(9) function is ")
print(file1.readline(9))

file1.seek(0)
print("Output of Readlines function is ")
print(file1.readlines())
print()

file1.close()
```

OUTPUT:

```
Output of Read function is
Hello
This is Delhi
This is Paris
This is London


Output of ReadLine function is
Hello

Output of Read(9) function is
Hello
Th

Output of Readline(9) function is
Hello

Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']
```

RESULT:

Thus the basic I/O operations like input() and print() are written in python code and executed successfully.

Ex. No: 02
DATE:

SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM

AIM:

To write a program to stimulate the CPU scheduling algorithm shortest job first (Non - Preemption).

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.

ALGORITHM:

Step1: Start the process.
Step2: Accept the number of processes in the ready Queue.
Step3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step5: Set the waiting time of the first processes '0' and its turn around time as its burst time.
Step6: Sort the processes names based on their Burt time.
Step7: For each process in the ready queue, calculate

a) Completion Time = Start Time + Burst Time
b) Turn - around time= Completion Time – Arrival Time
Or
Turn Around time = Burst time + Waiting time
c)Waiting Time = Turn Around Time – Burst Time.

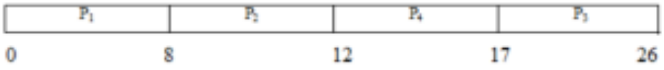
Step8: Calculate Average waiting time=Total waiting Time/ Number of process
Average Turn - around time = Total Turn - around Time/Number of processes.
Step9: Stop the process.

CALCULATION:

Consider the following 4 processes, with the length of the CPU burst time given in milliseconds

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt Chart:



0 8 12 17 26

SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM

Average time of Turn Around time: 14.25
Average time of Waiting time: 7.75

PROGRAM:

```
print ("Shortest Job First (Non-Preemptive) programming!".center(75, "~"), "\n")

P = ['p1', 'p2', 'p3', 'p4']
p = P.copy()

AT = [0, 1, 2, 3]
at = AT.copy()

BT = [8, 4, 9, 5]
bt = BT.copy()

GC = []

for i in range(len(P)):
    miv = bt.index(min(bt))
    if i == 0:
        miv = at.index(min(at))
    GC.append([at[miv], p[miv], bt[miv]])
    at.pop(miv)
    p.pop(miv)
    bt.pop(miv)

CT = [i for i in range(1, 6)]
TAT = [i for i in range(1, 6)]
WT = [i for i in range(1, 6)]

for i in range(len(P)):
    index = P.index(GC[i][1])
    CT[index] = GC[i][2]
    TAT[index] = CT[index] - AT[index]
    WT[index] = TAT[index] - BT[index]

print("*" * 75)
print("process : Arrival Time : Burst Time : Completion Time : Turn Around Time : Waiting
Time ")
for i in range(len(P)):
    print(P[i], " " * 4, ":", AT[i], " " * 10, ":", BT[i], " " * 8, ":", CT[i], " " * 14, ":", TAT[i],
          " " * 14, ":", WT[i])

print("*" * 75)
print("Average time of Turn Around time :", sum(TAT) / len(P))
print("Average time of Waiting time :", sum(WT) / len(P))
```

SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM

OUTPUT:

~~~~~~~~~~~~~~~~Shortest Job First (Non-Preemptive) programming! ~~~~~~~~~~~~~~~~

****

process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
p1	: 0	: 8	: 8	: 8	: 0
p2	: 1	: 4	: 12	: 11	: 7
p3	: 2	: 9	: 26	: 24	: 15
p4	: 3	: 5	: 17	: 14	: 9

****

Average time of Turn Around time: 14.25
Average time of Waiting time: 7.75

RESULT:

Thus the program for Shortest Job First is written using python programming language and executed successfully.
~~~~~~~~~~~~~~~~