KUMARAGURU
college of technology
character is life

OPERATING SYSTEM WORKBOOK

Regulations : R18
Class : B.Tech- AI&DS
Subject Code & Name : U18AII3203 & OPERATING SYSTEM

Certificate

This is to certify that it is a bonafide record of practical work done by

Sri/Kum KAILAS P S bearing the Roll No. 20BAD014 of 2^{ND} YEAR class BTECH AI&DS branch in the OPERATING SYSTEM Laboratory during the academic year 2021-22 under our supervision.

Faculty in charge | Internal Examiner

TABLE OF CONTENTS

PYTHON PROGRAMMING - MARKS BREAK UP STATEMENT							
S.No.	**Date**	**Name of the experiment**	**Program (10)**	**Execution (10)**	**Viva (10)**	**Total (30)**	**Staff sign**
1.		Develop programs for process creation and communication					
2.		Creation of process and child process					
3.		Demonstration of inter-process communication					
4.		Creation of Zombie and Orphan process					
5.		Creation of threads					
6.		Demonstration of shared memory concept					
7.		Simulation of the CPU scheduling algorithms					
8.		Demonstration of Semaphores					
9.		Implementation of Producer-Consumer problem					
10.		Simulation of Bankers algorithm for deadlock avoidance					
11.		Creation of virtual machine in a hypervisor					

1. Develop programs for process creation and communication

OBJECTIVE:

To develop programs for process creation and communication

ALGORITHMS:

Step 1: Import os
Step 2: Define a function which takes an string argument for child process to write
Step 3: Create a pipe and returns a pair of file descriptors (r, w)

Step 4: Now create a process using fork()
a). if the process id > 0 , close file descriptor with w and print the str

b). if the process id not 0 , close file descriptor with r and print the str

Step 5: Call the function with the str

Step 6 : Print the output

PROGRAM:

```
import os

def communication(child_writes):
    r, w = os.pipe()
    processid = os.fork()
    if processid>0:
        os.close(w)
        r = os.fdopen(r)
        print ("Parent reading")
        str = r.read()
        print( "Parent reads =", str)
    else:
        os.close(r)
        w = os.fdopen(w, 'w')
        print ("Child writing")
        w.write(child_writes)
        print("Child writes = ",child_writes)
        w.close()


str1 = "Hi parent! How are You"
communication(str1)
```

OUTPUT

```
Child writing
Child writes =  Hi parent! How are You
Parent reading
Parent reads = Hi parent! How are You
```

OUTPUT SCREENSHOT:

```python
import os


def communication(child_writes):
    r, w = os.pipe()
    processid = os.fork()
    if processid > 0:
        os.close(w)
        r = os.fdopen(r)
        print("Parent reading")
        str = r.read()
        print("Parent reads =", str)
    else:
        os.close(r)
        w = os.fdopen(w, 'w')
        print("Child writing")
        w.write(child_writes)
        print("Child writes = ", child_writes)
        w.close()


str1 = "Hi parent! How are You"
communication(str1)
```

```
Child writing
Child writes =  Hi parent! How are You
Parent reading
Parent reads = Hi parent! How are You
```

RESULT:

Thus , Programs for process creation and communication is being implemented.

2. Creation of process and child process

OBJECTIVES:

To develop a program for creation of parent process and child process

ALGORITHM:

Step 1: Import os
Step 2: Define a function parent_child
Step 3: Now create a process using fork()
a). if the process id > 0 , print parent process id using os.getpid()

b). if the process id = 0 , print child process id using os.getpid()

Step 5: Call the function

Step 6 : Print the output

PROGRAM:

```
import os

def parent_child():
    n = os.fork()
    if n > 0:

      print("Parent process and id is : ", os.getpid())
    else:
      print("Child process and id is : ", os.getpid())

parent_child()
```

OUTPUT:

```
Parent process and id is :  65
Child process and id is :  124
```

OUPUT SCREENSHOT:

```python
import os

def parent_child():
    n = os.fork()
    if n > 0:

      print("Parent process and id is : ", os.getpid())
    else:
      print("Child process and id is : ", os.getpid())

parent_child()
```

```
Parent process and id is :  65
Child process and id is :  124
```

RESULT:

Thus, implemented a program for parent process and child process .

3. Demonstration of inter-process communication and Creation of Zombie and Orphan process

OBJECTIVES:

To demonstrate of inter-process communication and the Creation of Zombie and Orphan process .

PROGRAM :

Creation of Zombie

ALGORITHM:

Step 1: Import os , sys , time
Step 2: Define a function parent_child
Step 3: Now create a process using fork()
 a). if the process id > 0 , print ("hello world")

 b). if the process id = 0 , exit the child process using exit()

Step 4 : Print the output

PROGRAM:

```
import os, sys, time

ttlForParent = 60
for i in range(0, 10):
      pid_1 = os.fork()
      print("Hello Worlds!!!")
      if pid_1 == 0:
          sys.exit()

time.sleep(ttlForParent)
os.wait()
```

OUTPUT SCREENSHOT:

```
import os, sys, time

ttlForParent = 60
for i in range(0, 10):
      pid_1 = os.fork()
      print("Hello Worlds!!!")
      if pid_1 == 0:
          sys.exit()

time.sleep(ttlForParent)
os.wait()
```

```
Hello Worlds!!!
Hello Worlds!!!
Hello Worlds!!!
An exception has occurred, use %tb to see the full traceback.

SystemExit

SEARCH STACK OVERFLOW
Hello Worlds!!!
Hello Worlds!!!
Hello Worlds!!!
Hello Worlds!!!
An exception has occurred, use %tb to see the full traceback.
```

Creation of Orphan process

ALGORITHM:

Step 1: Import os , sys , time
Step 2: Define a function parent_child
Step 3: Now create a process using fork()
a). if the process id > 0 , exit the parent process using exit()

b). if the process id = 0 , print any statement

Step 4 : Print the output

PROGRAM:

```
pid = os.fork()

if (pid ==  0):
  time.sleep(1)
```

```
  print("I am child having PID %d\n",os.getpid());
  print("My parent PID is %d\n",os.getppid())
if (pid > 0):
  print("I am parent having PID %d\n",os.getpid());
  print("My child PID is %d\n",os.getppid());
  sys.exit()
```

OUTPUT:

```
I am parent having PID %d
 65
My child PID is %d
 53

An exception has occurred, use %tb to see the full traceback.

SystemExit
/usr/local/lib/python3.7/dist-
packages/IPython/core/interactiveshell.py:2890: UserWarning: To exit: use
'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)

I am child having PID %d
 584
My parent PID is %d
 65
```

OUTPUT SCREENSHOT:

RESULT:

Thus, a program for the demonstration of inter-process communication and Creation of Zombie and Orphan process is implemented.

4. Creation of Threads

ALGORITHM:

Step 1: Import threading
Step 2: Define a function 1 which takes an argument a,b

a) Return the sum of a and b

Step 3: Define a function 2 which takes an argument a,b

a). return the difference of a and b

Step 4 : Create a thread1 object and enter the target function return the output

Step 5 : Create a thread2 object and enter the target function return the output

Step 6 : Print the outputs

PROGRAM:

```
import threading
def fun1(a, b):

  c = a + b
  print("\nthread 1",c)
def fun2(a, b):
  c = a - b
  print("\nthread 2",c)
thread1 = threading.Thread(target = fun1, args = (12, 10))
thread2 = threading.Thread(target = fun2, args = (12, 10))
thread1.start()
thread2.start()
```

OUTPUT:

```
thread 1 22

thread 2 2
```

OUTPUT SCREENSHOTS:

```python
import threading
def fun1(a, b):

  c = a + b
  print("\nthread 1",c)
def fun2(a, b):
  c = a - b
  print("\nthread 2",c)
thread1 = threading.Thread(target = fun1, args = (12, 10))
thread2 = threading.Thread(target = fun2, args = (12, 10))
thread1.start()
thread2.start()
```

```
thread 1 22

thread 2 2
```

RESULT:

Thus, a program for the creation of threads is being implemented.

5. Demonstration of shared memory concept

OBJECTIVES:

To demonstrate of shared memory concept with python program.

ALGORITHM :

- First of all, we create an Array **result** like this:
    - First argument is the **data type**. ‘i’ stands for integer whereas ‘d’ stands for float data type.
    - Second argument is the **size** of array. Here, we create an array of 4 elements.

  Similarly, we create a Value **square_sum** like this:

square_sum = multiprocessing.Value('i')

- Secondly, we pass **result** and **square_sum** as arguments while creating **Process** object.
- p1 = multiprocessing.Process(target=square_list, args=(mylist, result, square_sum))
- **result** array elements are given a value by specifying index of array element.
- for idx, num in enumerate(mylist):
- result[idx] = num * num

  square_sum is given a value by using its **value** attribute:

  square_sum.value = sum(result)
- In order to print **result** array elements, we use **result[:]** to print complete array.

MAIN IDEA:

- In **square_list** function. Since, this function is called by process **p1**, **result** list is changed in memory space of process **p1** only.
- After the completion of process **p1** in main program. Since main program is run by a different process, its memory space still contains the empty **result** list.

PROGRAM:

```
import multiprocessing

def square_list(mylist, result, square_sum):
  """
  function to square a given list
  """
  # append squares of mylist to result array
  for idx, num in enumerate(mylist):
    result[idx] = num * num

  # square_sum value
  square_sum.value = sum(result)

  # print result Array
  print("Result(in process p1): {}".format(result[:]))

  # print square_sum Value
  print("Sum of squares(in process p1): {}".format(square_sum.value))

if __name__ == "__main__":
  # input list
  mylist = [1,2,3,4]

  # creating Array of int data type with space for 4 integers
```

```
  result = multiprocessing.Array('i', 4)

  # creating Value of int data type
  square_sum = multiprocessing.Value('i')

  # creating new process
  p1 = multiprocessing.Process(target=square_list, args=(mylist,
result, square_sum))

  # starting process
  p1.start()

  # wait until process is finished
  p1.join()

  # print result array
  print("Result(in main program): {}".format(result[:]))

  # print square_sum Value
  print("Sum of squares(in main program): {}".format(square_sum.value))
```

<u>OUTPUT:</u>

```
Result(in process p1): [1, 4, 9, 16]
Sum of squares(in process p1): 30
Result(in main program): [1, 4, 9, 16]
Sum of squares(in main program): 30
```

OUTPUT SCREENSHOT:

```
  # print square_sum Value
  print("Sum of squares(in process p1): {}".format(square_sum.value))

if __name__ == "__main__":
  # input list
  mylist = [1,2,3,4]

  # creating Array of int data type with space for 4 integers
  result = multiprocessing.Array('i', 4)

  # creating Value of int data type
  square_sum = multiprocessing.Value('i')

  # creating new process
  p1 = multiprocessing.Process(target=square_list, args=(mylist, result, square_sum))

  # starting process
  p1.start()

  # wait until process is finished
  p1.join()

  # print result array
  print("Result(in main program): {}".format(result[:]))

  # print square_sum Value
  print("Sum of squares(in main program): {}".format(square_sum.value))
```

```
Result(in process p1): [1, 4, 9, 16]
Sum of squares(in process p1): 30
Result(in main program): [1, 4, 9, 16]
Sum of squares(in main program): 30
```

RESULT:

Thus ,demonstrated shared memory concept with python program.

6. Simulation of the CPU scheduling algorithms

FCFS

<u>OBJECTIVES:</u>

To implement FCFS CPU scheduling algorithm using Python

<u>ALGORITHM:</u>

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time Step 5: for each process in the Ready Q calculate
a). Waiting time (n) = waiting time (n-1) + Burst time (n-1) b).
Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate

a) Average waiting time = Total waiting Time / Number of process
b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

<u>PROGRAM:</u>

```
def findWaitingTime(processes, n,
          bt, wt):

  wt[0] = 0

  for i in range(1, n ):
    wt[i] = bt[i - 1] + wt[i - 1]

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

  for i in range(n):
    tat[i] = bt[i] + wt[i]

def findavgTime( processes, n, bt):
```

```
  wt = [0] * n
  tat = [0] * n
  total_wt = 0
  total_tat = 0

  findWaitingTime(processes, n, bt, wt)
  findTurnAroundTime(processes, n,
          bt, wt, tat)
  print( "Processes Burst time " +
        " Waiting time " +
        " Turn around time")
  for i in range(n):

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

  print( "Average waiting time = "+
        str(total_wt / n))
  print("Average turn around time = "+
          str(total_tat / n))

if __name__ =="__main__":
  processes = [ 1, 2, 3]
  n = len(processes)

  burst_time = [10, 5, 8]

  findavgTime(processes, n, burst_time)
```

OUTPUT:

```
Processes Burst time  Waiting time  Turn around time
 1          10     0            10
 2          5      10           15
 3          8      15           23
Average waiting time = 8.333333333333334
Average turn around time = 16.0
```

OUTPUT SCREENSHOT:

```
def findWaitingTime(processes, n,bt, wt):
  wt[0] = 0
  for i in range(1, n ):
    wt[i] = bt[i - 1] + wt[i - 1]
def findTurnAroundTime(processes, n,bt, wt, tat):
  for i in range(n):
    tat[i] = bt[i] + wt[i]
def findavgTime( processes, n, bt):
  wt = [0] * n
  tat = [0] * n
  total_wt = 0
  total_tat = 0
  findWaitingTime(processes, n, bt, wt)
  findTurnAroundTime(processes, n,bt, wt, tat)
  print( "Processes Burst time " +
        " Waiting time " +
        " Turn around time")
  for i in range(n):
    total_wt = total_wt + wt[i]
    total_tat = total_tat + tat[i]
    print(" " + str(i + 1) + "\t\t" + str(bt[i]) + "\t " +str(wt[i]) + "\t\t " +str(tat[i]))
  print( "Average waiting time = "+ str(total_wt / n))
  print("Average turn around time = "+ str(total_tat / n))
if __name__ =="__main__":
  processes = [ 1, 2, 3]
  n = len(processes)
  burst_time = [10, 5, 8]
  findavgTime(processes, n, burst_time)
```

```
Processes Burst time  Waiting time  Turn around time
 1              10       0               10
 2              5        10              15
 3              8        15              23
Average waiting time = 8.333333333333334
Average turn around time = 16.0
```

RESULT:

Thus, implemented FCFS CPU scheduling algorithm using Python

B). SHORTEST JOB FIRST:

AIM:

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

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as ‗0' and its turnaround time as its burst time.
Step 6: Sort the processes names based on their Burt time
Step 7: For each process in the ready queue,
calculate
a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n) Step 8: Calculate

c) Average waiting time = Total waiting Time / Number of process d) Average Turnaround time = Total Turnaround Time / Number of process
Step 9: Stop the process

PROGRAM:

```
def findWaitingTime(processes, n, wt):
  rt = [0] * n

  # Copy the burst time into rt[]
  for i in range(n):
    rt[i] = processes[i][1]
  complete = 0
  t = 0
  minm = 999999999
  short = 0
  check = False
  while (complete != n):`
    for j in range(n):
      if ((processes[j][2] <= t) and
        (rt[j] < minm) and rt[j] > 0):
        minm = rt[j]
        short = j
        check = True
```

```
        if (check == False):
          t += 1
          continue
        rt[short] -= 1
        minm = rt[short]
        if (minm == 0):
          minm = 999999999
        if (rt[short] == 0):
          complete += 1
          check = False
          fint = t + 1
          wt[short] = (fint - proc[short][1] -
                    proc[short][2])

          if (wt[short] < 0):
            wt[short] = 0
        t += 1
def findTurnAroundTime(processes, n, wt, tat):
    for i in range(n):
        tat[i] = processes[i][1] + wt[i]
def findavgTime(processes, n):
    wt = [0] * n
    tat = [0] * n
    findWaitingTime(processes, n, wt)
    findTurnAroundTime(processes, n, 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(" ", processes[i][0], "\t\t",
            processes[i][1], "\t\t",
            wt[i], "\t\t", tat[i])
    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = ", total_tat / n)
if __name__ =="__main__":
    proc = [[1, 6, 1], [2, 8, 1],
        [3, 7, 2], [4, 3, 3]]
    n = 4
    findavgTime(proc, n)
```

OUTPUT:

```
Processes Burst Time    Waiting Time     Turn-Around Time

   1         6          3          9
   2         8          16         24
   3         7          8          15
   4         3          0          3

Average waiting time = 6.75000
Average turn around time =  12.75
```

OUTPUT SCREENSHOT:

```
def findTurnAroundTime(processes, n, wt, tat):
  for i in range(n):
    tat[i] = processes[i][1] + wt[i]
def findavgTime(processes, n):
  wt = [0] * n
  tat = [0] * n
  findWaitingTime(processes, n, wt)
  findTurnAroundTime(processes, n, 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(" ", processes[i][0], "\t\t",
        processes[i][1], "\t\t",
        wt[i], "\t\t", tat[i])

  print("\nAverage waiting time = %.5f "%(total_wt /n) )
  print("Average turn around time = ", total_tat / n)
if __name__ =="__main__":
  proc = [[1, 6, 1], [2, 8, 1],
      [3, 7, 2], [4, 3, 3]]
  n = 4
  findavgTime(proc, n)
```

```
Processes Burst Time     Waiting Time    Turn-Around Time
  1             6             3             9
  2             8             16            24
  3             7             8             15
  4             3             0             3

Average waiting time = 6.75000
Average turn around time =  12.75
```

RESULT:

Thus implemented a program to stimulate the CPU scheduling algorithm Shortest job first (Non- Preemption)

C). ROUND ROBIN:

AIM:

To simulate the CPU scheduling algorithm round-robin.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where No. of time
slice for process (n) = burst time process (n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.
Step 6: Consider the ready queue is a circular Q, calculate

1. a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU fromprocess(n-1)
2. b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).
   Step 7: Calculate
   c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number ofprocess Step 8: Stop the process

PROGRAM:

```
def findWaitingTime(processes, n, bt,wt, quantum):
  rem_bt = [0] * n
  for i in range(n):
    rem_bt[i] = bt[i]
  t = 0
  while(1):
    done = True
    for i in range(n):
      if (rem_bt[i] > 0) :
        done = False
        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 (done == True):
      break
def findTurnAroundTime(processes, n, bt, wt, tat):
  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))

if __name__ =="__main__":
  proc = [1, 2, 3]
  n = 3
  burst_time = [10, 5, 8]
  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 turn around time = 19.66667
```

OUTPUT SCREENSHOT:

```
      if (done == True):
        break
  def findTurnAroundTime(processes, n, bt, wt, tat):
    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))

  if __name__ =="__main__":
    proc = [1, 2, 3]
    n = 3
    burst_time = [10, 5, 8]
    quantum = 2;
    findavgTime(proc, n, burst_time, quantum)
```

```
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 turn around time = 19.66667
```

RESULT:

Thus implemented the CPU scheduling algorithm round-robin using python.

D). PRIORITY:

AIM:

To write a python program to simulate the CPU scheduling priority algorithm.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time Step 6: Arrange the processes based on process priority
Step 7: For each process in the Ready Q calculate Step 8:
for each process in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n) Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number of process Print the results

in an order.

Step10: Stop

PROGRAM:

```
def findWaitingTime(processes, n, wt):
  wt[0] = 0
  for i in range(1, n):
    wt[i] = processes[i - 1][1] + wt[i - 1]
def findTurnAroundTime(processes, n, wt, tat):
  for i in range(n):
    tat[i] = processes[i][1] + wt[i]
def findavgTime(processes, n):
  wt = [0] * n
  tat = [0] * n
  findWaitingTime(processes, n, wt)
  findTurnAroundTime(processes, n, wt, tat)
  print("\nProcesses 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(" ", processes[i][0], "\t\t",
        processes[i][1], "\t\t",
        wt[i], "\t\t", tat[i])
  print("\nAverage waiting time = %.5f "%(total_wt /n))
  print("Average turn around time = ", total_tat / n)
def priorityScheduling(proc, n):
  proc = sorted(proc, key = lambda proc:proc[2],
                reverse = True);
  print("Order in which processes gets executed")
  for i in proc:
    print(i[0], end = " ")
  findavgTime(proc, n)
if __name__ =="__main__":
  proc = [[1, 10, 1],
      [2, 5, 0],
      [3, 8, 1]]
  n = 3
  priorityScheduling(proc, n)
```

OUTPUT:

```
Order in which processes gets executed 1 3 2
Processes Burst Time Waiting Time Turn-Around Time
  1          10          0           10
  3          8           10          18
  2          5           18          23

Average waiting time = 9.33333
Average turn around time =  17.0
```

OUTPUT SCREENSHOT:

```python
      "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(" ", processes[i][0], "\t\t",
        processes[i][1], "\t\t",
        wt[i], "\t\t", tat[i])
  print("\nAverage waiting time = %.5f "%(total_wt /n))
  print("Average turn around time = ", total_tat / n)
def priorityScheduling(proc, n):
  proc = sorted(proc, key = lambda proc:proc[2],
                reverse = True);
  print("Order in which processes gets executed")
  for i in proc:
    print(i[0], end = " ")
  findavgTime(proc, n)
if __name__ =="__main__":
  proc = [[1, 10, 1],
      [2, 5, 0],
      [3, 8, 1]]
  n = 3
  priorityScheduling(proc, n)
```

```
Order in which processes gets executed
1 3 2
Processes Burst Time Waiting Time Turn-Around Time
  1             10              0               10
  3             8               10              18
  2             5               18              23

Average waiting time = 9.33333
Average turn around time =  17.0
```

RESULT:

Thus implemented a python program to simulate the CPU scheduling priority algorithm.

7. Demonstration of Semaphores

OBJECTIVES:

To implement a python program to simulate the Semaphores

ALGORITHM:

Step 1:Import modules
Step 2: Create thread instance where count =3
Step 3:Create instance call acquire method followed by release method
Step 4: Create multiple threads
Step 5: Call all the threads
Step 6: Print the output

PROGRAM:

```
from threading import *
import time
obj = Semaphore(3)
def display(name):
    obj.acquire()
    for i in range(5):
        print('Hello, ', end = '')
        time.sleep(1)
        print(name)
        obj.release()
t1 = Thread(target = display , args = ('Thread-1',))
t2 = Thread(target = display , args = ('Thread-2',))
t3 = Thread(target = display , args = ('Thread-3',))
t4 = Thread(target = display , args = ('Thread-4',))
t5 = Thread(target = display , args = ('Thread-5',))


t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

OUTPUT :

```
Hello, Hello, Hello,
```

OUTPUT SCREENSHOT:

```
from threading import *
import time
obj = Semaphore(3)
def display(name):
    obj.acquire()
    for i in range(5):
        print('Hello, ', end = '')
        time.sleep(1)
        print(name)
        obj.release()
t1 = Thread(target = display , args = ('Thread-1',))
t2 = Thread(target = display , args = ('Thread-2',))
t3 = Thread(target = display , args = ('Thread-3',))
t4 = Thread(target = display , args = ('Thread-4',))
t5 = Thread(target = display , args = ('Thread-5',))


t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

```
Hello, Hello, Hello,
```

RESULT:

Thus implemented a python program to simulate the Semaphores

8. Implementation of Producer-Consumer problem

OBJECTIVE:

To implement a python program for Producer-Consumer problem

ALGORITHM:

Step 1: Start the program.
Step 2: Declare variable for producer & consumer as pthread-t-tid produce tid consume.
Step 3: Declare a structure to add items, semaphore variable set as struct.
Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.

Step 5: Read number the items to be produced and consumed.

Step 6: Declare and define semaphore function for creation and destroy.

Step 7: Define producer function.

Step 8 :Define consumer function.

Step 9 :Call producer and consumer.

Step 10 Stop the execution.

PROGRAM:

```
from threading import Thread, Semaphore
import time
import random
queue = []
MAX_NUM = 10
sem = Semaphore()
class ProducerThread(Thread):
    def run(self):
        nums = range(5) # [0 1 2 3 4]
        global queue
        while True:
            sem.acquire()
            if len(queue) == MAX_NUM:
                print ("List is full, producer will wait")
```

```
                sem.release()
                print ("Space in queue, Consumer notified the
producer")
            num = random.choice(nums)
            queue.append(num)
            print ("Produced", num)
            sem.release()
            time.sleep(random.random())
class ConsumerThread(Thread):
    def run(self):
        global queue
        while True:
            sem.acquire()
            if not queue:
                print ("List is empty, consumer waiting")
                sem.release()
                print ("Producer added something to queue and notified
the consumer")
            num = queue.pop(0)
            print ("Consumed", num)
            sem.release()
            time.sleep(random.random())
def main():
    ProducerThread().start()
    ConsumerThread().start()

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

<u>OUTPUT:</u>

```
Produced 3
Produced 4
Consumed 4
Produced 1
Produced 2
Produced 2
```

OUTPUT SCREENSHOT:

```
        while True:
            sem.acquire()
            if len(queue) == MAX_NUM:
                print ("List is full, producer will wait")
                sem.release()
                print ("Space in queue, Consumer notified the producer")
            num = random.choice(nums)
            queue.append(num)
            print ("Produced", num)
            sem.release()
            time.sleep(random.random())
class ConsumerThread(Thread):
    def run(self):
        global queue
        while True:
            sem.acquire()
            if not queue:
                print ("List is empty, consumer waiting")
                sem.release()
                print ("Producer added something to queue and notified the consumer")
            num = queue.pop(0)
            print ("Consumed", num)
            sem.release()
            time.sleep(random.random())
def main():
    ProducerThread().start()
    ConsumerThread().start()

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

```
Produced 3
Produced 4
Consumed 4
Produced 1
Produced 2
Produced 2
```

RESULT:

Thus implemented a python program for Producer-Consumer problem

9. Simulation of Bankers algorithm for deadlock avoidance

OBJECTIVE:

To Simulate Bankers algorithm for deadlock avoidance

ALGORITHM:

Step-1: Start the program.
Step-2: Declare the memory for the process.
Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process .

Step-6: produce the result of state of process

Step-7: Stop the program

PROGRAM:

```
import numpy as np

def check(i):
    for j in range(no_r):
        if(needed[i][j]>available[j]):
            return 0
    return 1


no_p = 5
no_r = 4

Sequence = np.zeros((no_p,),dtype=int)
visited = np.zeros((no_p,),dtype=int)

allocated =
np.array([[4,0,0,1],[1,1,0,0],[1,2,5,4],[0,6,3,3],[0,2,1,2]])
maximum = np.array([[6,0,1,2],[1,7,5,0],[2,3,5,6],[1,6,5,3],[1,6,5,6]])

needed = maximum - allocated
available = np.array([3,2,1,1])
```

```
count = 0
while( count < no_p ):
    temp=0
    for i in range( no_p ):
        if( visited[i] == 0 ):
            if(check(i)):
                Sequence[count]=i;
                count+=1
                visited[i]=1
                temp=1
                for j in range(no_r):
                    available[j] += allocated[i][j]
    if(temp == 0):
        break


if(count < no_p):
    print('The system is Unsafe')
else:
    print("The system is Safe")
    print("Safe Sequence: ",Sequence)
    print("Available Resource:",available)
```

<u>OUTPUT:</u>

```
The system is Safe
Safe Sequence:  [0 2 3 4 1]
Available Resource: [ 9 13 10 11]
```

OUTPUT SCREENSHOT:

```
Sequence = np.zeros((no_p,),dtype=int)
visited = np.zeros((no_p,),dtype=int)

allocated = np.array([[4,0,0,1],[1,1,0,0],[1,2,5,4],[0,6,3,3],[0,2,1,2]])
maximum = np.array([[6,0,1,2],[1,7,5,0],[2,3,5,6],[1,6,5,3],[1,6,5,6]])

needed = maximum - allocated
available = np.array([3,2,1,1])

count = 0
while( count < no_p ):
    temp=0
    for i in range( no_p ):
        if( visited[i] == 0 ):
            if(check(i)):
                Sequence[count]=i;
                count+=1
                visited[i]=1
                temp=1
                for j in range(no_r):
                    available[j] += allocated[i][j]
    if(temp == 0):
        break


if(count < no_p):
    print('The system is Unsafe')
else:
    print("The system is Safe")
    print("Safe Sequence: ",Sequence)
    print("Available Resource:",available)
```

```
The system is Safe
Safe Sequence:  [0 2 3 4 1]
Available Resource: [ 9 13 10 11]
```

RESULT:

Thus implemented Bankers algorithm for deadlock avoidance using python

10. Creation of virtual machine in a hypervisor

OBJECTIVE :

To Create of virtual machine in a hypervisor

ALGORITHM:

For installing Linux Ubuntu on Windows we need to perform three steps:

- Instal Virtualbox on Windows
- Create Linux Ubuntu Virtual Machine on Virtual Box
- Instal Ubuntu on Virtual Machine using Virtual Box

1- **Installing VirtualBox on Windows:**

1- Go to https://www.virtualbox.org/wiki/Downloads and click on **Windows hosts** under **VirtualBox platform packages** to download VirtualBox DMG. Once the download is completed, double click the **VirtualBox.DMG** file to see it's contents.

2- **Creating Ubuntu Linux Virtual Machine under VirtualBox on Windows:**

The steps to create Ubuntu Virtual Machine on your VirtualBox.

1) The next step is to download Ubuntu locally to our computer to use with VirtualBox. Your first step is to go to ***https://ubuntu.com/download***.

2)Once downloaded, the file will probably sit in your downloads as ubuntu-20.04.2.0-desktop-amd64.iso

3) Create a new machine in Virtual Box by clicking on **New**, or going to **Machine > New**

4) Once the prompt opens, give it a *Name* (*Ubuntu 64-bit*), and set the Memory to *2048 MB (2GB)*. Make sure you set the *Hard Disk* option to *Create a virtual hard disk now*

5) Set the ***File size*** will be set to ***10.00GB*** to give the Ubuntu file enough space to work with, the ***Hard disk file type*** to ***VDI*** to make an image formatted for VirtualBox, and allow the ***Storage on physical hard disk*** to be ***Dynamically allocated***. Once done, click the ***Create*** button.

6) Right-click on the Ubuntu Virtual Machine under VirtualBox and click on settings.

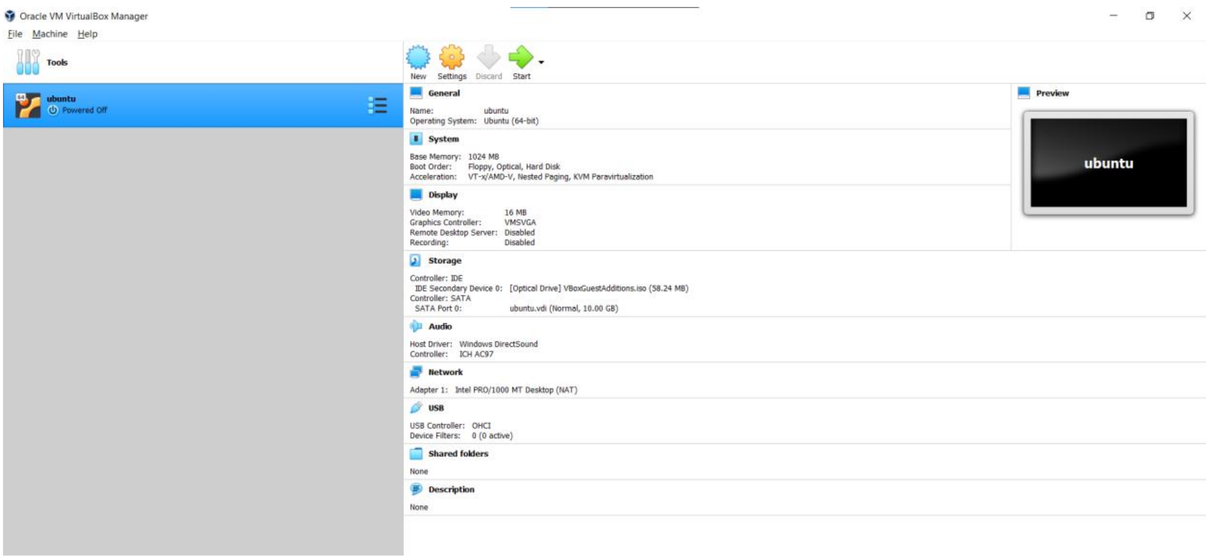
7) Under settings navigate to storage and click on disk icon under attributes and choose the location where ubuntu-desktop iso file has been saved.

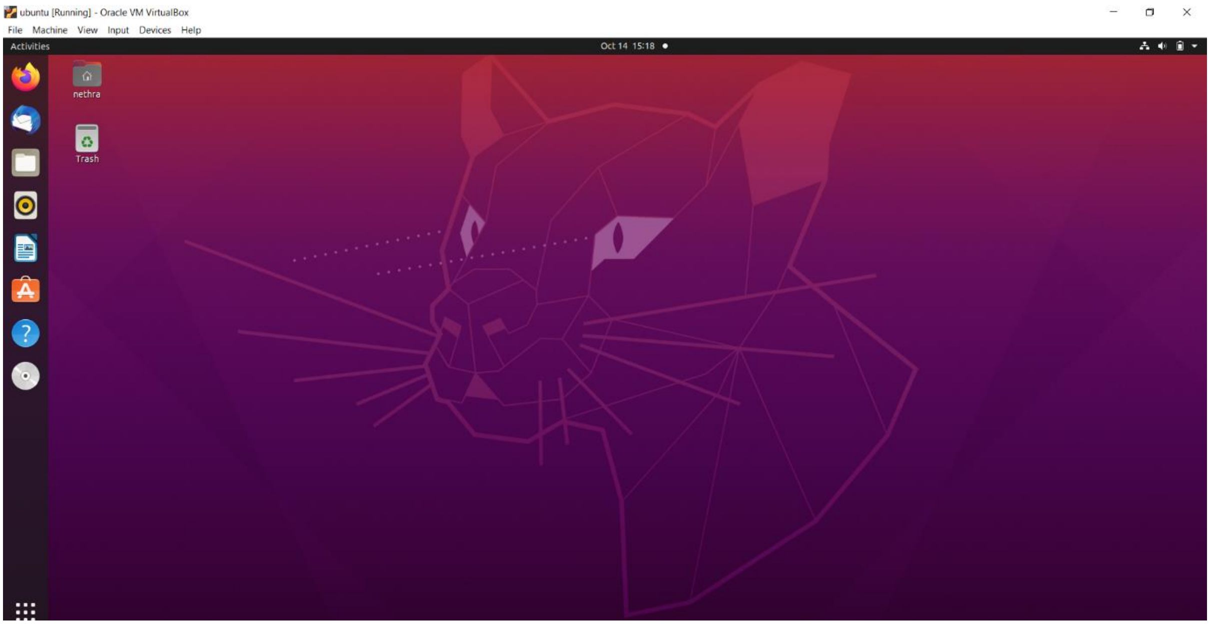
8) Select your VM under VirtualBox and click start to start installation.

9) On successful completion of basic setup, you should receive a message indicating your installation is successful.

10) Restart the computer and to use Ubuntu on your Windows.

OUTPUT SCREENSHOTS:





RESULT:

Thus , the virtual machine in a hypevisor is created.