# Operating System

# (Real Time OS)

**A Mini-Project Report *Submitted by***

## Rishikesh Pathak ( E007 )

**Tarran Pidugu ( E012 )**

**Vaishnavi Rathod( E016 )**

**Harshit Sanga ( E021 )**

***Under the Guidance of***

**Prof. Swarnalata Bollavarapu**

## B. TECH. (COMPUTER ENGINEERING)

## At



**MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT & ENGINEERING**

**MONTH & YEAR**

**TABLE OF CONTENTS :**

|  |  |  |
| --- | --- | --- |
| **SR NO** | **INDEX** | **REMARKS** |
| 1. | Introduction |  |
| 2. | Problem Statement |  |
| 3. | Scope of the project |  |
| 4. | Literature Survey |  |
| 5. | Analysis and Design |  |
| 6. | Implementation |  |
| 7. | Results |  |
| 8. | Conclusion |  |
| 9. | References |  |

**INTRODUCTION :**

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter increments of time. A real-time system is a time-bound system which has well-defined, fixed time constraints. Processing must be done within the defined constraints or the system will fail. They either are event-driven or time-sharing. Event-driven systems switch between tasks based on their priorities, while time-sharing systems switch the task based on clock interrupts. Most RTOSs use a pre-emptive scheduling algorithm.

A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task; the variability is 'jitter'. A 'hard' real-time operating system has less jitter than a 'soft' real-time operating system. The chief design goal is not high throughput, but rather a guarantee of a soft or hard performance category. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.

An RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency; a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

**PROBLEM STATEMENT**

Develop working application simulating the working of mini version of a Real Time Operating System. The simulated OS should include the following features of an OS namely Process Management, Memory Management, I/O Management and File Management. (One algorithm for each feature)

**Functionalities:**

* Scheduling of processes assigned to processor
* Managing Memory
* Organizing of data into files and directories
* I/O Management

SCOPE OF THE PROJECT

Realtime based system performance is improved comparing to non realtime based system. Further real time system performance can be improved by implementing multi tasking preemptive RTOS algorithm. Also same design can be implement using different RTOS operating system. Performance comparisons between system with critical section and without critical section show that the average waiting time is reduced in system with critical section. But response time is increased little due to disable interrupt during execution of critical section. Further system performance can be improved by

1. Redesign Critical section algorithm to improve response time.
2. Use task synchronization can implement using semaphore.
3. Properly redesigning the scheduler to improve average response time enhancing features of scheduler, controllability over the execution can be easily achieved.

LITERATURE SURVEY

The RTOS trend is quite apparent in the embedded industry and for good reasons. Due to increasingly complex and connected applications, more and more developers rely on an RTOS. Moreover, the RTOS market is consolidating as developers gravitate towards the leading solutions.

RTOSes can be regarded as the third generation of firmware development, since multithreading brings a higher level of abstraction and less detailed control over the execution. This has significant pitfalls that call for better debugging support at the RTOS level. Common debugging tools have not, however, evolved significantly in response to the RTOS trend; instead, they typically offer only incremental improvements to the traditional debugger concept.

The debugging of RTOS-based systems can be simplified with better insight into their real-time execution. This requires RTOS-level tracing, where visualization is crucial to make sense of the data. Several tools can display an RTOS trace as a horizontal Gantt chart, but this is not ideal. More sophisticated visualization is both possible and available, optimized for RTOS traces, that makes it easier to understand the runtime system, spot important issues, and verify the solutions.

The rising popularity of freely available and open-source RTOSes has made RTOS-based design more accessible and created large user communities — a kind of “critical mass” that works as a positive feedback loop. RTOSes are no longer a hype, as they were 10-15 years ago, but are used broadly in all sorts of embedded applications. It's not only commercial RTOS vendors who are advocating RTOS-based design; many MCU vendors now include RTOSes in their software development kits. FreeRTOS is included by several MCU vendors, including ST, NXP, and Atmel. Renesas includes the ThreadX RTOS and has made it the foundation of its Synergy SDK, expecting you to use it. Moreover, Silicon Labs recently acquired Micrium, a leading RTOS vendor, most likely to integrate Micrium's µC/OS in its SDK.

ANALYSIS AND DESIGN

**Algorithms used to implement functionalities:**

**Process Management :-**

Earliest Deadline First (EDF) is an optimal dynamic priority scheduling algorithm used in real-time systems. It can be used for both static and dynamic real-time scheduling uses priorities to the jobs for scheduling. It assigns priorities to the task according to the absolute deadline. The task whose deadline is closest gets the highest priority. The priorities are assigned and changed in a dynamic fashion. EDF is very efficient as compared to other scheduling algorithms in real-time systems. It can make the CPU utilization to about 100% while still guaranteeing the deadlines of all the tasks.

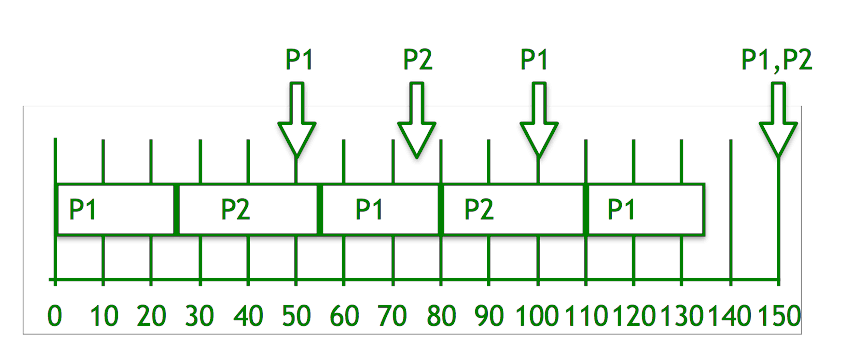
Consider two processes P1 and P2.

Let the period of P1 be p1 = 50

Let the processing time of P1 be t1 = 25

Let the period of P2 be period2 = 75

Let the processing time of P2 be t2 = 30



**Steps for solution:**

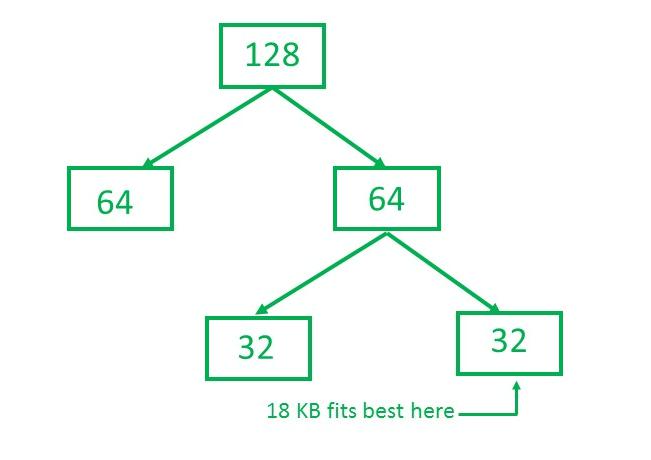
1. Deadline pf P1 is earlier, so priority of P1>P2
2. Initially P1 runs and completes its execution of 25 time.
3. After 25 times, P2 starts to execute until 50 times, when P1 executes.
4. Now, comparing the deadline of (P1, P2) = (100, 75), P2 executes.
5. P2 completes its processing at time 55.
6. P1 starts to execute until time 75, when P2 is able to execute.
7. Now, again comparing the deadline of (P1, P2) = (100, 150), P1 executes
8. Repeat the above steps…
9. Finally at time 150, both P1 and P2 have the same deadline, so P2 will continue to execute till its processing time after which P1 starts to execute.

**Memory Management:-**

Buddy allocation system is an algorithm in which a larger memory block is divided into small parts to satisfy the request. This algorithm is used to give best fit. The two smaller parts of block are of equal size and called as buddies. In the same manner one of the two buddies will further divide into smaller parts until the request is fulfilled. Benefit of this technique is that the two buddies can combine to form the block of larger size according to the memory request. It is fast to allocate memory, and fast to deallocate making it efficient.

Consider a system having buddy system with physical address space 128 KB.Calculate the size of partition for 18 KB process.

**Solution –**

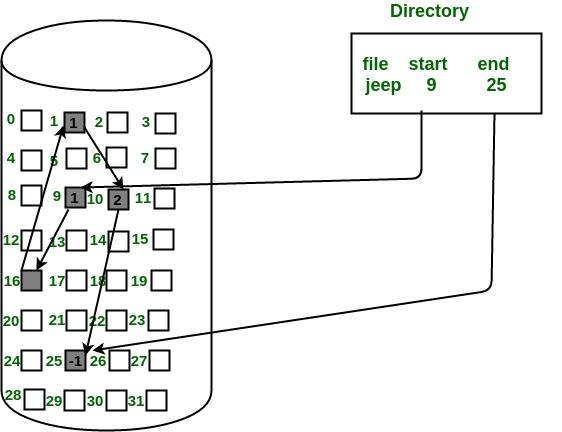


So, size of partition for 18 KB process = 32 KB. It divides by 2, till possible to get minimum block to fit 18 KB.

**File Management:-**

Linked File Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. There is no external fragmentation and any free block can be utilized in order to satisfy the file block requests.

The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



**I/O Management:-**

**C-LOOK** is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for C-LOOK.

In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

**Algorithm-**

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

IMPLEMENTATION (SOURCE CODE)

class node:

def \_\_init\_\_(self,reqd,space,flag):

self.reqd=reqd

self.space=space

self.flag=flag

self.next=None

import math

class LinkedList:

def \_\_init\_\_(self):

self.head=None

self.marker=None

def allocate(self,s):

temp=self.head

prev=self.head

t=0

n=0

k=0

while(temp is not None):

if(s<=temp.space and temp.flag =='false'):

n=math.ceil(math.log(temp.space)/math.log(2))

if(s>math.pow(2,n-1)):

temp.reqd=s

temp.flag='true'

break

else:

t=temp.space/2

add=node(0,t,'false')

add.next=temp

temp.space=t

if(k==0):

self.head=add

temp=self.head

elif (k>0):

prev.next=add

temp=add

else:

prev=temp

temp=temp.next

k+=1

if(temp is None):

print("\n")

print(s," can't be allocated!")

print("\n")

def deallocate(self,s):

self.combine()

temp=self.head

while(temp is not None):

if(temp.reqd==s and temp.flag=='true'):

temp.reqd=0

temp.flag='false'

break

temp=temp.next

if(temp is None):

print("\n")

print(s,"can't be de allocated")

print("\n")

self.combine()

def combine(self):

temp=self.head

temp2=self.head

while(temp.next is not None):

temp2=temp.next

if(temp.reqd==0 and temp2.reqd==0 and temp.flag=='false' and temp2.flag=='false'):

if(temp.space==temp2.space and temp is not self.marker):

temp.space=temp.space+temp2.space

temp.next=temp2.next

break

temp=temp.next

def display(self):

self.combine()

temp=self.head

left=0

k=0

print("M E M O R Y A L L O C A T I O N !")

while(temp is not None):

print("Memory space of: ",temp.space,end=" ")

if(temp.flag =='true'):

k=temp.space-temp.reqd

left+=k

print("is occupied with:",temp.reqd)

else:

left+=temp.space

print("is empty")

temp=temp.next

print("--------------------------------------------")

print("Total unused spaces are->",left)

print("\n")

def control(self):

total=0

total=int(input("Enter the TOTAL MEMORY space in the system:"))

print("--------------------------------------------")

if(self.head is None):

add=node(0,total,'false')

add.next=None

self.head=add

self.marker=self.head

n=math.ceil(math.log(total)/math.log(2))

ch=0

while(ch!=4):

print("Make a choice from the following menu->")

print("1.Allocation to memory")

print("2.De-allocation from memory")

print("3.Display")

print("4.Exit")

print("--------------------------------------------")

ch=int(input("Enter choice-->"))

print("--------------------------------------------")

if(ch==1):

c=int(input("Enter size of the memory to be allocated:"))

self.allocate(c)

elif(ch==2):

c=int(input("Enter the size of the memory to be de-allocated:"))

self.deallocate(c)

self.combine()

elif(ch==3):

self.combine()

self.display()

elif(ch==4):

print("The program is exited")

break

else:

print("Your choice is invalid")

def edf():

tasks=[]

print("Please Enter the number of tasks:")

n=int(input())

for i in range (n):

tasks.append([])

tasks[i].append( i+1)

print("Please Enter the Execution time of task:",i+1)

tasks[i].append(int(input()))

print("Please Enter the Periodicity time of task:",i+1)

tasks[i].append(int(input()))

print("Please Enter the Deadline time of task:",i+1)

tasks[i].append(int(input()))

print("Please Enter the Arrival time of task:",i+1)

tasks[i].append(int(input()))

print (tasks)

u=0

for i in range(n):

u+=float(tasks[i][1]/tasks[i][2])

print("Utilization: ",u)

if u>1:

print("The tasks are not feasible")

else:

lcm=1

temp\_p=[]

for i in range(n):

temp\_p.append(tasks[i][2])

print(temp\_p)

i=2

while i <= max(temp\_p):

counter=0

for j in range(n):

if temp\_p[j]%i==0:

counter=1

temp\_p[j]/=i

if counter==1:

lcm=lcm\*i

else:

i+=1

print("LCM: ",lcm)

i=0

instances=[]

for i in range(n):

j=1

while 1:

if j\*tasks[i][2]<=lcm:

instances.append([tasks[i],j\*tasks[i][2], (j\*tasks[i][2])-(tasks[i][2]-tasks[i][3])])

j+=1

else:

break

print()

print("Processes are divided into time periods")

print()

for i in range(len(instances)):

print(instances[i])

for i in range(len(instances)):

tmp = instances[i].copy()

k = i

while k > 0 and tmp[2] < instances[k-1][2]:

instances[k] = instances[k - 1].copy()

k -= 1

instances[k] = tmp.copy()

print()

print()

print("Process are sorted according to their deadlines : ")

print()

for i in range(len(instances)):

print(instances[i])

timeLeft=[]

for i in range(n):

if tasks[i][4]==0:

timeLeft.append(tasks[i][1])

else:

timeLeft.append(int(0))

timeLine=[]

time=0

while time<lcm:

for i in range(n):

if time>1 and ((time%tasks[i][2]==0 and time>tasks[i][4]) or time==tasks[i][4]):

timeLeft[i]=tasks[i][1]

anyrun=0

for j in range(len(instances)):

if j==0 and timeLeft[instances[j][0][0]-1]>0:

timeLine.append(instances[j][0][0])

timeLeft[instances[j][0][0]-1]-=1

anyrun=1

if timeLeft[instances[j][0][0]-1]==0:

instances.pop(j)

break

elif j>0 and instances[j][1]==instances[0][1]:

if timeLeft[instances[j][0][0]-1]>0:

tmp=instances[j].copy()

instances[j]=instances[0].copy()

instances[0]=tmp.copy()

time-=1

anyrun=1

break

elif j>0 and timeLeft[instances[j][0][0]-1]>0:

timeLine.append(instances[j][0][0])

timeLeft[instances[j][0][0]-1]-=1

anyrun=1

if timeLeft[instances[j][0][0]-1]==0:

instances.pop(j)

break

if anyrun==0:

timeLine.append(0)

time+=1

print()

print()

print(" GANTT CHART :")

mn=0

mx=0

for i in range(lcm):

if i>0 and timeLine[i]!=timeLine[i-1]:

mx=i

if timeLine[i-1] == 0:

print(mn," - ",mx," : ", "[ IDLE ]")

else :

print(mn," - ",mx," : ", "[ Process "+str(timeLine[i-1])+"]")

mn=i

if i==lcm-1:

mx=lcm

if timeLine[i] == 0:

print(mn," - ",mx," : ", "[ IDLE ]")

else:

print(mn," - ",mx," : ", "[ Process"+str(timeLine[i])+"]")

def c\_look():

n=int(input("Enter total requests "))

l=list()

print("Enter Request Sequence")

for i in range(n):

x=int(input())

l.append(x)

head = int(input("Enter head position "))

sum=0

sum1=0

sum2=0

l.sort()

l1=list()

for x in l:

if(x<head):

l1.append(x)

l3=list()

l3 = [x for x in l if x not in l1]

sum=l3[-1]-head

sum1=l3[-1]-l1[0]

sum2=l1[-1]-l1[0]

total=sum+sum1+sum2

print("Total seek time = ",total)

print("Seek sequence : ")

for x in l1:

l3.append(x)

for x in l3:

print(x)

def linked\_file\_allocation():

f = [0 for i in range(50)]

p = int(input("Enter how many blocks that are already allocated\n"))

print("Enter the blocks no.s that are already allocated")

for i in range(p):

a = int(input())

f[a] = 1

def X(f):

print("Enter the starting index block & length")

inp = input().split()

inp = list(map(int,inp))

st = inp[0]

length = inp[1]

k = length

j = st

while j<(k+st):

if f[j] == 0:

f[j] = 1

print(j,"->",f[j])

else:

print("{0}->file is already allocated".format(j))

k+=1

j+=1

c = 1

while c != 0:

X(f)

c = int(input("If u want to enter one more file? (yes-1/no-0)\n"))

while 1:

choice = int(input("Enter 1 for process management \nEnter 2 for memory management \nEnter 3 for file management \nEnter 4 for I/O management\nEnter 5 to quit\n"))

if choice == 1:

edf()

elif choice == 2:

l=LinkedList()

l.control()

elif choice == 3:

linked\_file\_allocation()

elif choice == 4:

c\_look()

elif choice == 5:

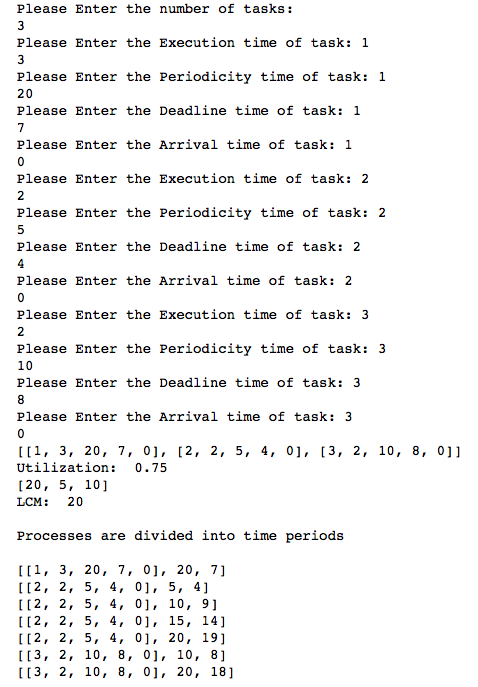
break

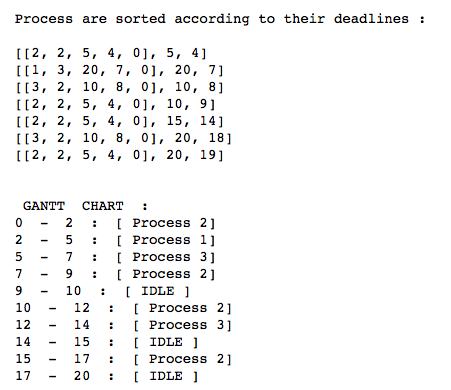
else :

print("Invalid choice . Enter again")

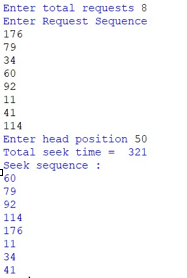
**RESULTS (OUTPUT SCREENSHOTS)**

**PROCESS MANAGEMENT :**

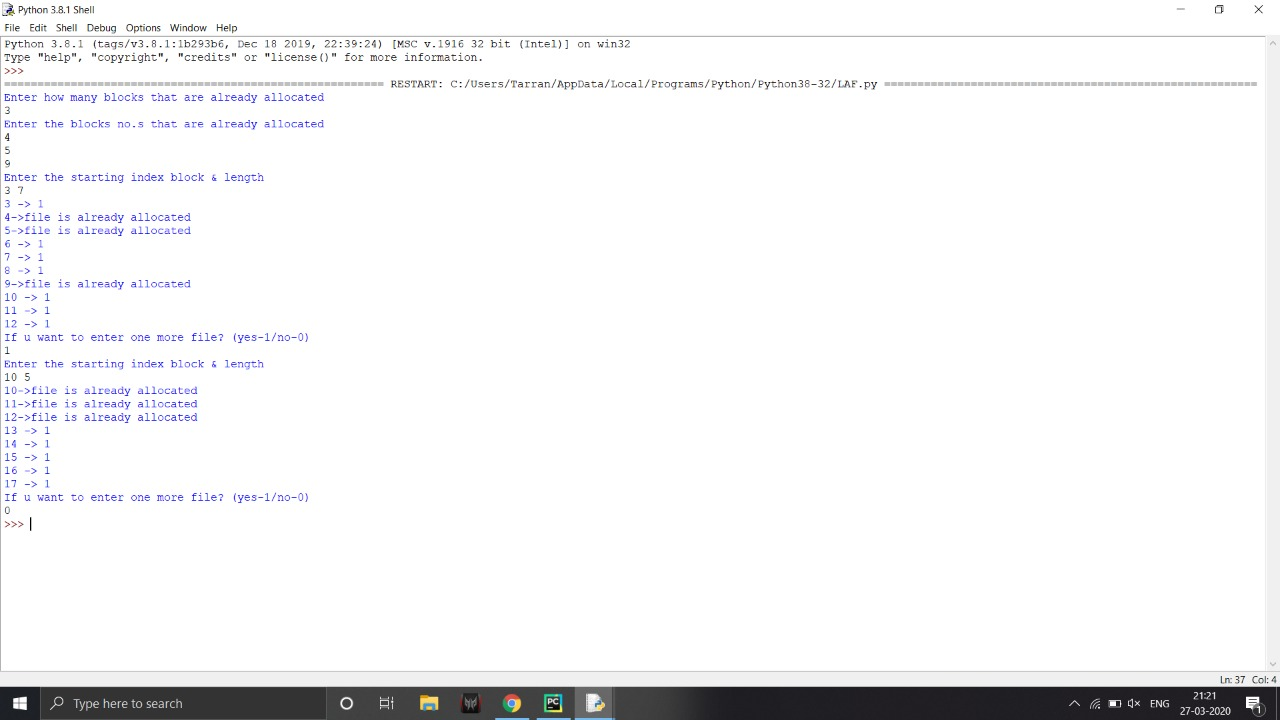




**I/O MANAGEMENT:**

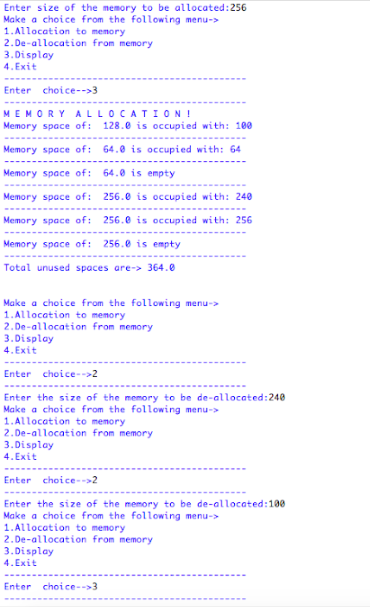
****

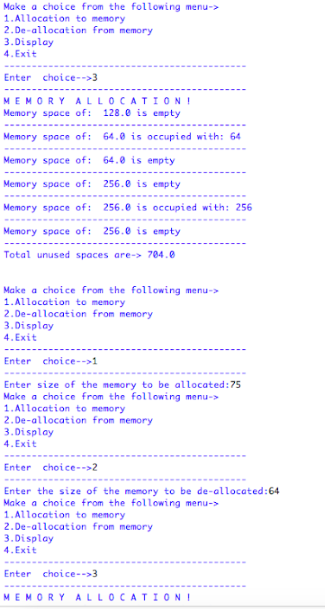
**FILE MANAGEMENT:**

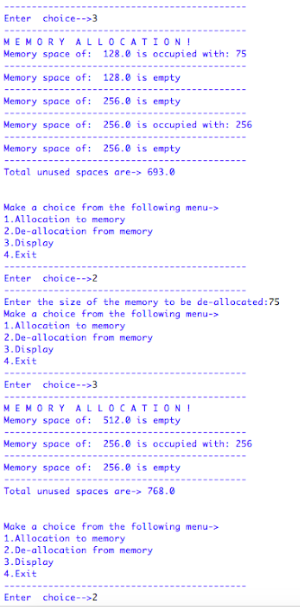
****

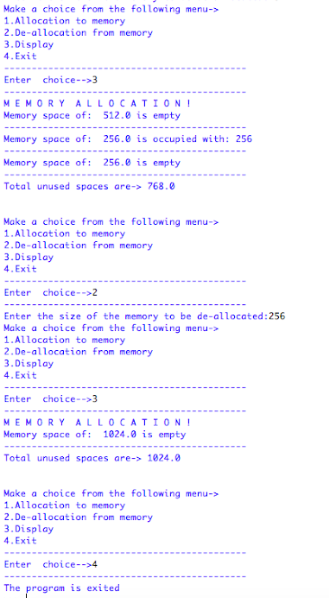
**MEMORY MANAGEMENT:**











CONCLUSION

Hence, we designed a RTOS which includes the most efficient algorithm in each of the 4 functionalities : Process Management, I/O Management, Memory Management and File Management. This is a very simple model of an OS but we came to know about its basic components and how to devise it using a programming language like Python. We hope to learn more in depth about OS so that at the end of our graduation, we should be well versed with a fully functional and operational model of OS.

It has given us immense learning. We have completely enjoyed the process of first learning the algorithm through youtube videos and then writing a code for it ourselves without any external intervention. This project has been a fruitful thing to do because we came to know about the deeper aspects of OS. We even came to know about the different fields of work of OS and helped us dived into the deeper realms of it.

REFERENCES :

Youtube videos which we used for understanding the algorithm:

1. <https://www.youtube.com/watch?v=j9sOpKm5goQ>

2) <https://www.youtube.com/watch?v=gK6L3v1b8AM>

3) <https://www.youtube.com/watch?v=ejPXTOcMRPA>

4) <https://www.youtube.com/watch?v=gwCgG5ORXW8>

Other References :

<https://www.geeksforgeeks.org/>

<https://www.researchgate.net/>