IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : N. Ravi Chandrika

Class CSD-B Semester IIIrd Semester

Course Code : ACSD10 Course Name : OS laboratory

Roll Number									
2	3	9	5	1	A	6	7	B	3

Name of the Course Faculty Ms.G.Indu Faculty ID : IARE10971

Exercise Number : 4 Week Number : 4 Date : 27/9/24

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure Performance in the Lab		Source Code Calculations and Graphs	Program Execution Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	4.1	Memory variable Technique	4	2	2	4	3	4	19
2	4.2	Best fit memory Allocation	4	2	2	4	3	4	19
3	4.3	Worst fit memory Allocation	4	2	2	4	4	4	20
4	4.4	Multiprogramming wt a fixed no.of tasks	4	2	2	4	4	4	20
5	4.5	Simulating paging memory management	4	2	2	4	4	4	20
6									
7									
8									
9									
10									
11									
12									

N. Ravi Chandrika
Signature of the Student

Signature of the Faculty

4.1 Aim: You are a system administrator responsible for managing a computer system that employs the memory variable technique for memory management. your system receives requests from multiple users to run different processes, each requiring a specific amount of memory.

Code:

```
class MemoryVariableTechnique:
    def _init_(self, participants):
        self.partitions = partitions
        self.memory_map = {
            1: [False] * participants[1],
            2: [False] * participants[2],
            3: [False] * participants[3],
        }
        self.processes = { }
    def allocate_memory(self, process_id, size):
        for partition_id, partition in self.memory_map.items():
            if len(partition) >= size and all(not partition[i] for i in range(size)):
                Allocate memory
                for i in range(size):
                    partition[i] = True
              self.processes[process_id] = (partition_id, size)
             print(f"Allocated {size} units of memory.")
            return.
      print(f"Failed to allocate {size} units of memory.")
    def deallocate_memory (self, process_id):
        if process_id in self. processes :
            partition_id, size = self.processes [process_id]
            for i in range(size):
                self. memory_map[partition_id][i] = False
            del self.processes[process_id]
            print(f" Deallocated memory for process {process_id}.")
        else:
            print(f" Process {process_id} not found.")

    def display_memory_Status(self):
        for partition_id, partition in self.memory_map.items():
            allocated_units = sum(partition)
            total_units = len(partition)
            print(f" Partition {partition_id}: units allocated.")
mvt = MemoryVariableTechnique({1:300, 2: 500, 3: 200})
mvt.allocate_memory (1, 150)
mvt.allocate_memory (2,400)
mvt.allocate_memory (3,100)
mvt.display_memory_Status()
mvt.deallocate_memory (2)
print("\n After deallocating Process 2:")
mvt.display_memory_Status()
```

<u>Output</u>: Allocated 150 units of memory for process 1 in partition 1

```
Allocated 400 units of memory for process 2 in partition 2.
Allocated 100 units of memory for process 3 in partition 3.
partition 1: 150/300 units allocated.
partition 2: 400/500 units allocated
partition 3: 100/200 units allocated
Deallocated memory for process 2.
After deallocating process 2:
Partition 1: 150/300 units allocated
partition 2: 0/500 units allocated
partition 3: 100/200 units allocated.
```

4.2 Aim: You are a software developer working on an operating system project that required implementing memory management techniques. One of the crucial algorithm is best fit algorithm, which aims to allocate memory blocks to processes in a way that minimizes wastage and fragmentation.

Code:

```
class BestFitMemoryManager:
    def _init_(self, memory_blocks):
        self.memory_blocks = memory_blocks
        self.available_blocks = {i: block for i, block in enumerate))
        self.processes = {}
    def allocate_memory(self, process_id, size):
        best_fit_index = None
        best_fit_size = float('inf')
    for index, block in self.available_blocks.items():
        if block >= size and (block < best_fit_size):
            best_fit_size = block
            best_fit_index = index
    if best_fit_index is Not None:
        self.processes[process_id] = size
        self.available_blocks[best_fit_index] -= size
        print(f"Allocated {size} units of memory.")
    else:
        print(f"failed to allocate {size} units of memory.")
def deallocate_memory (self, process_id):
    if process_id in self.processes:
        size = self.processes[process_id]
        for index, block in self.available_blocks.items():
            if block < self.memory_blocks[index]:
                self.available_blocks[index] += size
                del self.processes[process_id]
                print(f"Deallocated memory for process {process_id}.")
                return
        print(f"Process {process_id} not found.")
    else:
        print(f"Process {process_id} not found.")
def display_memory_status (self):
    print("\n Memory Status:")
    for index, block in self.available_blocks.items():
            total-size = self.memory-blocks [index]
            allocated-size = total-size - block
            print(f"Block {index}: {allocated-size}/{total-size}")
memory-manager = BestFitMemoryManager([100,200,50,150,300,80,120])
memory-manager.allocate-memory(1,70)
memory-manager.allocate-memory(2,180)
memory-manager.allocate-memory(3,250)
memory-manager.display-memory-status()
memory-manager.deallocate-memory(2)
print("\n After deallocating process 2:")
memory-manager.display-memory-status().
```

Output:

```
Allocated 70 units of memory for process 1 in Block 5.
Allocated 180 units of memory for process 2 in Block 1.
Allocated 250 units of memory for process 3 in Block 4.
Memory Status:
Block 0: 0/100 units allocated
Block 1: 180/200 units allocated
Block 2: 0/50 units allocated
Block 3: 0/150 units allocated
Block 4: 250/300 units allocated
Block 5: 70/80 units allocated
Block 6: 0/120 units allocated
Block 7: 0/200 units allocated
Deallocated memory for process 2.
```

```
After deallocating process 2:
Block 0: 0/100 units ~~deallocated~~
Block 1: 0/200 units allocated
Block 2: 0/50 units allocated
Block 3: 0/150 units allocated
Block4: 250/300 units allocated
Block5: 70/80 units allocated
Block6: 0/120 units allocated
Block7: 0/200 units allocated.
```

4.3 Aim: You are a system analyst tasked with designing the memory management techniques for a new operating system. One of the memory allocation algorith techniques to implement is WorstFit memory block to t which priortizes allocating the largest available memory processes.

code:

```
class BestFitMemoryManager:
  def -init- (self, memory-blocks):
    self.memory-blocks = memory-blocks
    self.available_blocks = {i: block for i, block in enumerate(memory
    self.processes = {}
  def allocate-memory(self, process_id, size):
    best_fit_index = None
   best-fit-size = float('inf')
   for index, Block in self.available-blocks.items():
      if block>= size and (block < best-fit-size):
            best_fit_size = block
            best_fit_index = index
    if best_fit_index is not None:
        self.processes[process_id] = (best_fit_index, size)
        self.available_blocks = [best_fit_index] - = size
        print(f" Allocated {size} units of memory.")
    else:
        print(f" Failed to allocate {size} units of memory.")

def deallocate_memory(self, process_id):
    if process_id in self.processes:
        best_fit_index, size = self.processes[process_id]
        self.available_blocks[best_fit_index] + = size
        del self.processes[process_id],
        print(f" Deallocated memory for process {process_id}.")
    else:
        print(f"process {process_id} not found.")
                   Worst
memory_manager = ~~Best~~FitMemoryManager([100,200,50,150,300,80,120,200])
memory_manager.allocate_memory(1,70)
memory_manager.allocate_memory(2,180)
memory_manager.allocate_memory(3,250)
memory_manager.display_memory_status()
memory_manager.deallocate_memory(2)
print("\nafter deallocating process 2:")
memor_manager.display_memory_status()
```

output:

```
Allocated 180 units of memory for block 1 in Block 6.
Allocation failed for process 2 : Not Enough space
Allocated 180 units of memory for process 3 in Block 1.
Memory Status:
Block 0: 0/150 units allocated
Block 1: 120/300 units allocated
Block 2: 0/200 units allocated
Block 3: 0/800 units allocated
Block 4 : 0/250 units allocated
:
Block 9: 0/200 units allocated
Process 2 not found.
deallocated memory for process 1.
After deallocating memory process 1:
Memory Status:
Block 0: 0/150 units allocated
Block 1: 120/300 units allocated.
Block 2: 0/100 units allocated
Block 3: 0/200 units allocated:
Block 4: 0/250 units allocated.
:
Block 9: 0/200 units allocated-
```

4.4 Aim: you are a system engineer working ona legacy mainfram System that employs the MFT memory management technique the System is designed to handle multiple process Simultaneou by dividing memory into fixed-size partitions.

code:

```
class MFTMemoryManager:
  def _init_(self, num_partitions, partition_size):
      self.num_partitions = num_partitions
      self.partition_size = partition_size
    self.available_partitions = [True] * num_partitions
    self.processes = {}
 def allocate_memory(self, process_id, size):
   if size > self.partition_size:
     print(f"Failed to allocate {size}")
     return.
 for i in range(self.num_partitions):
     if self.available_partitions[i]:
       self.available_partions[i] = false
       self.processes[process_id] = i.
       print(f"Failed to allocate {size}")
       return
print(f"failed to allocate {size} units")

def deallocate_memory(self, process_id):
  if process_id in self.processes:
    partition_index = self.processes-[process_id]
    self.available_partitions[partition_index] = True
    del self.processes[process_id]
```

```
            print(f"Deallocated memory for process {process_id}.")
        else:
            print(f"Process {process_id} not found.")
    def display_memory_status(self):
        print("\n Memory Status:")
        for i in range(self.num_partitions):
            status = "Allocated" if not self.available_partitions[i] else "Free
            print(f" Partition {i}: {status}")
memory_manager = MFTMemoryManager(num_partitions = 3,
                                          partition_size = 800)

memory_manager.allocate_memory(1,600)
memory_manager.allocate_memory(2,900)
memory_manager.allocate_memory (3,400)
memory_manager.display_memory_status()
memory_manager.deallocate_memory(2).
memory_manager.deallocate_memory(1)
memory_manager.display_memory_status().
```

__Output:__

```
Allocated 600 units of memory for process 1 in partition 0.
Failed to allocate 900 units of memory for process 2: Size excee
Allocated 400 units of memory for process 3 in partition 1

Memory Status:
                                        Partition 4: Free
Partition 0: Allocated
                                        Partition 5: Free
Partition 1: Allocated
                                        Partition 6: Free
Partition 2: Free
                                        Partition 7: Free
```

```
process 2 not found
Deallocated memory for process 1.
After deallocating Process 1:
Memory Status:
Partition 0: Free
Partition 1: Allocated
Partition 2: Free
Partition 3: Free
Partition 4: Free
Partition 5: Free
Partition 6: Free
Partition 7: Free.
```

4.5 Aim: You are a software engineer tasked with implementing memory management for a new operating system that utilizes the paging technique.

Code:

```
class PagingMemoryManager:
    def _init_(self, num_pages, page_size):
        self.num_pages = num_pages
        self.page_size = page_size
        self.available_pages = [True]* num_pages
        self.processes = { }
    def allocatememory (self, process_id, size):
        required_pages = (size + self.page_size - 1) // self.page_size
        allocated_pages = [ ]
        for i in range(self.num_pages):
            if self.available_pages[i]:
                allocated_pages.append(i)
                self.available_pages[i] = False
```

```
          if len(allocated_pages) == required_pages:
              break
      if len(allocated_pages) == required_pages:
          self.processes[process_id] = allocated_pages
          print(f"Allocated {size} bytes")
    else:
        print(f"Allocation failed")

def deallocate_memory(self, process_id):
    if process_id in self.processes:
        allocated_pages = self.processes[process_id]
        for page in deallocated_pages:
            self.available_pages[page] = True
        del self.processes[process_id]
        print(f"deallocated memory")
    else:
        print(f"Process {process_id} not found.")

def display_memory_status(self):
    print("\nMemory Status:").
    for i in range(self.num_pages):
        status = "Allocated" if not "Free"
        print(f"page {i} : {status}")
memory_manager = PagingMemoryManager(num_pages=20,
                                      page_size = 200)
memory_manager.allocate_memory(1, 400)
memory_manager. allocate_memory (2, 600)
memory_manager.allocate_memory (3, 300)
memory_manager.display_memory_status()
memory_manager.deallocate_memory(1)
memory_manager.display_memory_status()
```

Output:

```
Allocated 400 bytes for process 1: Pages[0,1]
Allocated 600 bytes for process 2: Pages[2,3,4]
Allocated 500 bytes for process 3: Pages[5,6]

Memory Status:
page 0: Allocated          page 10: Free
page 1: Allocated          page 11: Free
page 2: Allocated          page 12: Free
page 3: Allocated          page 13: Free
page 4: Allocated          page 14: Free
page 5: Allocated          page 15: Free
page 6: Allocated          page 16: Free
page 7: Free               page 17: Free
page 8: Free               page 18: Free
page 9: Free               page 19: Free
```