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

Roll Number										
	0	3	9	5	1	A	6	7	B	3

Class : CSD-B Semester : IIIrd Semester

Course Code : ACSD10 Course Name : OS Laboratory

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

Exercise Number : 5 Week Number : 5 Date : 01-11-2024

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	5.1	futuristic space station	4	2	2	4	4	3	19
2	5.2	CentalAI	4	2	2	4	4	3	19
3	5.3	MetroCentral	4	2	2	4	4	4	20
4	5.4	AI Lab	4	2	2	4	4	4	20
5	5.5	MedTech Hospital	4	2	2	4	4	4	20
6									
7									
8									
9									
10									
11									
12									

Ravi Chandrika - N

Signature of the Student

Signature of the Faculty

START WRITING FROM HERE

5.1 Aim: In a futuristic space station, the central computer manages memory allocation using the worst-fit technique. The memory is divided into fixed-size blocks and whenever a program requests for memory allocation the system searches for largest available block.

Code:

```
def worst-fit-allocation(memory-blocks, program-requests):
    allocations = { }
    for program, request-size in program-requests.items():
        worst-fit-index = -1
        max-block-size = -1
        for i in range(len(memory-blocks)):
            if memory-blocks[i] >= request-size and > max-block-size:
                max-block-size = memory-blocks[i]
                worst-fit-index = i
    if worst-fit-index != -1:
        allocations [program] = request-size
        memory-blocks[worst-fit-index] -= request-size
    else:
        allocations [program] = None
    return allocations, memory-blocks
memory-blocks = [400, 250, 350, 900, 150]
program-requests = {'Program A': 150, 'Program B': 300, 'Program C': 200}
allocations, remaining-memory = worst-fit-allocation
for program, allocated-memory in allocation.items():
    if allocated-memory is not None:
        print(f"{program} allocated {allocated-memory} units of memory!")
    else:
```

ROLL NUMBER :

```
        print(f"{program} could not be allocated any memory.")
print("\n Remaining memory in memory Blocks:")
for i in range(len(memory_blocks)):
    print(f"Block {i+1}: {memory_blocks[i]} units")
```

Output:

Program A allocated 150 units of memory
Program B allocated 300 units of memory
Program C allocated 200 units of memory
Remaining Memory in Memory Blocks:
Block 1: 50 units
Block 2: 250 units
Block 3: 50 units
Block 4: 200 units
Block 5: 150 units.

5-2 Aim: In a futuristic city where advanced artificial intelligence systems govern public services, a central AI hub named as the CentralAI operates using a sophisticated memory management system using the best fit contiguous memory allocation techni

Code:

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None
```

```
    dy is_free(self):
        return not self.allocated
    dy allocate(self, process_name):
        if self.is_free():
            self.allocated = True
            self.process_name = process_name
            return True
        Return False
    dy deallocate(self):
        self.allocated = False
        self.process_name = None
    dy __str__(self):
        status = "Free" if not f"Allocated to {self.process_name}"
        return f"[start: {self.start}, size: {self.size}, status: {status}]"
clas MemoryManager:
    dy __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]
    dy allocate_memory(self, process_name, size):
        for block in self.memory_blocks:
            if block.is_free() and block.size >= size:
                if block.size > size:
                    new_block = MemoryBlock(block.start + size)
                    self.memory_blocks.append(new_block)
                    block.size = size
                block.allocate(process_name)
```

```
            return True
        print(f"Not enough memory to allocate for {process_name}.")
        return False
    def print_memory_status(self):
        print("\nCurrent Memory Status:")
        for block in self.memory_blocks:
            print(block)
memory_manager = MemoryManager(8000)
requests = [
        ("SubsysA", 1500),
        ("SubsysB", 1000),
        ("SubsysC", 700),
        ("SubsysD", 2200),
        ("SubsysE", 500),
        ("SubsysF", 1200)
    ]
for request in requests:
    process_time, size = request
    allocated = memory_manager.allocate_memory(process_name, size)
    if allocated:
        memory_manager.print_memory_status()
```

Output: Current Memory Status:

[start : 0, size : 1500, Status : Allocated to SubsysA]

[start : 1500, size : 1000, Status : Allocated to SubsysB]

[start : 2500, size : 700, status : Allocated to SubsysC]

ROLL NUMBER:

{start: 5200, size: 2200, status: Allocated to subsys2}
{start: 4400, size: 800, status: Allocated to subsys1}
{start: 1900, size: 2500, status: Allocated to subsys3}
{start: 7400, size: 900, status: Free}.

5.3 Aim: In a building metropolis where a central command centre, Metrocentral manages all public transportation systems using shared memory allocation technique to handle the memory from various subsystems.

Code:

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        return not self.allocated

    def allocate(self, process_name):
        if self.is_free():
            self.allocated = True
            self.process_name = process_name
            return True
        return False

    def deallocate(self):
        self.allocated = False
        self.process_name = None
```

```
    def __str__(self):
        status = "Free" if not self.process_name else f"Allocated to {self.process_name}"
        return f"Start: {self.start}, Size: {self.size}, Status: {status}"

class MemoryManager:
    def __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]

    def allocate_memory(self, process_name, size):
        for block in self.memory_blocks:
            if block.is_free and block.size >= size:
                if block.size > size:
                    new_block = MemoryBlock(block.start + size)
                    self.memory_blocks.append(new_block)
                    block.size = size
                block.allocate(process_name)
                return True
        print(f"Not enough memory to allocate for {process_name}")
        return False

    def print_memory_status(self):
        print("\nCurrent Memory Status:")
        for block in self.memory_blocks:
            print(block)

    def print_total_memory_used(self):
        total_used = sum(block.size for block in self.memory_blocks)
        print(f"\nTotal memory used: {total_used} units.")
```

```
memory_manager = MemoryManager(5000)
requests = [
    ("TrafficControl", 1200),
    ("RoutePlanning", 800),
    ("VechicleMonitoring", 1500),
    ("PassengerInformation", 600),
    ("MaintenanceLogistics", 700)
]
for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory
    if allocated:
        memory_manager.print_memory_status()
memory_manager.print_total_memory_used()
```

Output:

Current Memory Status:

[Start: 0, Size: 1200, Status: Allocated to TrafficControl]

[Start: 1200, Size: 800, Status: Allocated to RoutePlanning]

[Start: 2000, Size: 1500, Status: Allocated to VechicleMonitoring]

[Start: 3500, Size: 600, Status: Allocated to PassengerInformation]

[Start: 4100, Size: 700, Status: Allocated to MaintenanceLogistics]

[Start: 4800, Size: 200, Status: Free]

Total memory used: 4800 units of 5000 units.

ROLL NUMBER :

5.4

Aim: In a research institute focused on AI advancements, the AI manages to computational researchers using a first-fit contiguous memory allocation technique.

Code:

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        return not self.allocated

    def allocate(self, process_name):
        if self.is_free():
            self.allocated = True
            self.process_name = process_name
            return True
        return False

    def deallocate(self):
        self.allocated = False
        self.process_name = None

    def __str__(self):
        status = "Free" if not f"Allocated to {self.process_n
        return f"[start: {self.start}, size: {self.size}, Stat

class MemoryManager:
```

```
def __init__(self, total_memory):
    self.total_memory = total_memory
    self.memory_blocks = [MemoryBlock(0, total_memory)]

def deallocate_memory(self, process_name, size):
    for block in self.memory_blocks:
        if block.is_free() and block.size >= size:
            if block.size > size:
                new_block = memory_blocks
                self.memory_blocks.append(new_block)
                block.size = size
            block.allocate(process_name)
            return True
    print(f"Not enough memory to allocate for {process_name}")
    return False

def print_memory_status(self):
    print("\nCurrent Memory Status:")
    for block in self.memory_blocks:
        print(block)

def print_total_memory_used(self):
    total_used = sum(block.size)
    print(f"\nTotal memory used: {total_used} units.")

memory_manager = MemoryManager(60000)
requests = [
    ("Project A", 1500),
    ("Project B", 1000),
```

ROLL NUMBER :

```
        ("Project C", 700),
        ("Project D", 2200),
        ("Project E", 500),
        ("Project F", 1200)
    ]
for requests in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size
    if allocated:
        memory_manager.print_memory_status()
memory_manager.print_total_memory_used().
```

Output:

Current Memory Status:

[Start: 0, Size: 1500, Status: Allocated to project A]

[Start: 1500, Size: 1000, Status: Allocated to Project B]

[Start: 2500, Size: 700, Status: Allocated to Project C]

[Start: 3200, Size: 2200, Status: Allocated to Project D]

[Start: 5400, Size: 500, Status: Allocated to Project E]

[Start: 5900, Size: 100, Status: Free]

Not enough memory to allocate project F.

Total memory used: 5900 units of 6000 units.

Roll Number:

Aim: To implement a city where a central hospital, med-tech hospital manages the patient records & medical data using sophisticated computer system, efficient memory management techniques using best-fit contiguous memory.

Code:

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        return not self.allocated

    def allocate(self, process_name):
        if self.is_free():
            self.allocated = True
            self.process_name = process_name
            return True
        return False

    def deallocate(self):
        self.allocated = False
        self.process_name = None

    def __str__(self):
        return f"[Start: {self.start}, Size: {self.size}, Status: {status}]"
```

ROLL NUMBER :

```
class MemoryManager:
    def __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]

    def allocate_memory(self, process_name, size):
        for block in self.memory_blocks:
            if block.is_free() and block.size >= size:
                new_block = MemoryBlock(block.start + size,
                self.memory_blocks.append(new_block)
                block.size = size
                block.allocate(process_name)
                return True
        print(f"Not Enough memory to allocate for {process_name}.")
        return False

    def print_memory_status(self):
        print("In Current Memory Status:")
        for block in self.memory_blocks:
            print(block)

    def print_total_memory_used(self):
        total_used = sum(block.size for block in self.memory_blocks)
        print(f"\nTotal Memory used: {total_used} units.")

memory_manager = MemoryManager(8000)

requests = [
    ("Emergency Department", 2000),
```

```
    ("Cardiology Department", 1500),
    ("Laboratory Information System", 1200),
    ("Radiology Department", 1800),
    ("Patient management System", 1000),
    ("Pharmacy System", 600),
    ("Surgical Services", 2200),
]

for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size)

    if allocated:
        memory_manager.print_memory_status()

memory_manager.print_total_memory_used().
```

Output:

Current Memory Status :

[Start : 0, Size : 2000, Status : Allocated to Emergency Department]

[Start : 2000 Size : 1500, status : Allocated to Cardiology Department]

[Start : 3500, Size : 1200, Status : Allocated to Laboratory information]

[Start : 4700, Size : 1800, Status : Allocated to Radiology Department]

[Start : 6500, Size : 1000, Status : Patient management System]

[Start : 7500, Size : 500, Status : Free]

Not enough space to allocate for Pharmacy System.

ROLL NUMBER :

Not enough memory to allocate for surgical services.

Total memory used : 7500 units out of 8000 units.