Assignment 2 - Operating Systems II

Task 1

The main memory size is 4MB/4096KB

The size of a page is 4KB

I am using the buddy system for memory allocation and so my blocks are of varied size, with the smallest possible block being the size of a page (4KB). Initially before any processes are allocated to memory, my memory starts of as one big block, with its size being 4MB (the size of main memory). As memory becomes allocated the block splits into smaller and smaller blocks as a result of using the buddy system. My reasoning for starting off with the memory being one block is to simply not impose any limits on the potential of memory allocation and to simply let the buddy system do what it is supposed to do; allocate memory dynamically. This idea of dynamic memory allocation is what made me choose the Buddy System as my algorithm of choice for memory allocation.

For memory allocation, I am using the Second Chance algorithm because it is a solid algorithm that gives all processes a fairly equal opportunity for allocation and to also give processes, as the name implies, a “second chance” at having memory allocated

Free Memory tracking is done using a bitmap and recursive traversals of the buddy system tree

Task 2

Free Memory Tracking - Bitmap / Tree traversal

bitmap = new dictionary

initially, the buddy tree is traversed recursively to get all leaf nodes, which are then placed in a list

for every block in the list

    it is added to the bitmap as the key and then its value set to 1 if the block is free, or set to 0 if it is not free

if a block is allocated or deallocated then

    the bitmap is updated

Memory Allocation - Buddy System

if memory is full

    execute the memory replacement algorithm

a process is dequeued from the queue of processes that require memory allocation

n = the size of memory requested by the process

if n is smaller than the size of a page, then n = the size of a page

round n to the nearest power of 2

search the leaf nodes of the buddy tree to find if any blocks match n

if true, allocate the block to the process

Otherwise, if there are any free blocks that are larger than n

    Split the first suitable block until its size matches n and then allocate that block to the process

if the process still has not been allocated a block, then execute memory replacement

Memory Replacement - Second Chance

when a process is allocated to a block, it is added to a queue for memory replacement

if the queue is not empty

a block that is allocated to a process is dequeued

if the block's access bit is 1

its access bit is set to 0 and it is enqueued back into the queue

otherwise

the process that is allocated to the block is now deallocated

if the node of the block has a buddy node that is also free

the two buddies are now merged together

Task 3

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

class Block(object):

    def __init__ (self, size):
        self._size = size
        self._allocated = False
        self._process = None
        self._pages = []
        self._accessed = 0

    def get_size(self):
        return self._size

    def set_size(self, size):
        self._size = size

    def get_pages(self):
        return self._pages

    def add_page(self, page):
        self._pages.append(page)

    def get_process(self):
        return self._process

    def set_process(self, process):
        self._process = process

    size = property(get_size, set_size)
    process = property(get_process, set_process)
```

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

class Process(object):

    def __init__(self, PID, memory):
        self._PID = PID
        self._memory = memory

    def getPID(self):
        return self._PID

    def getMemory(self):
        return self._memory

    pid = property(getPID)
    memory = property(getMemory)

    def __str__(self):
        return f"PID: {self.pid} | Memory Required: {self.memory} MB"
```

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

class QueueV0:
    def __init__(self):
        self.body = []

    def __str__(self):
        if len(self.body) == 0:
            return '<--<'
        stringlist = ['<']
        for item in self.body:
            stringlist.append('-' + str(item))
        stringlist.append('-<')
        return ''.join(stringlist)

    def enqueue(self, item):
        self.body.append(item)

    def dequeue(self):
        return self.body.pop(0)

    def length(self):
        return len(self.body)

    def first(self):
        return sellf.body[0]
```

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

from queues import *
from process import *
from block import *
from random import randint

class BuddySystemBST(object):

    class node(object):
        def __init__ (self, size):
            self._block = Block(size)
            self._parent = None
            self._leftchild = None
            self._rightchild = None

        def __str__ (self):
            return f"{self._block.size} KB"

        def get_block(self):
            return self._block

        def set_block(self, block):
            self._block = block

        block = property(get_block, set_block)


    def __init__(self, size):
        self._root = self.node(size)
        self._block_nodes = self._root

    def get_root(self):
        return self._root

    # checks if a node is a leaf node
    def is_leaf(self, node):
        if node._leftchild == None and node._rightchild == None:
            return True
        else:
            return False

    # returns a list of all leaf nodes in the tree
    def get_leaf_nodes(self):
        root = self.get_root()
        blocks = []
        blocks = self._get_leaf_nodes(root, blocks)
        return blocks

    def _get_leaf_nodes(self, node, blocks):
        if self.is_leaf(node):
```

```
            blocks.append(node)
        else:
            if node._leftchild != None:
                self._get_leaf_nodes(node._leftchild, blocks)

            if node._rightchild != None:
                self._get_leaf_nodes(node._rightchild, blocks)
        return blocks

    # splits a node by instantiating to node objects and setting them
    # as the original nodes children
    def split_node(self, node):
        parent_block_size = node.block.size

        left_child = self.node(parent_block_size//2)
        right_child = self.node(parent_block_size//2)

        node._leftchild = left_child
        node._rightchild = right_child

        left_child._parent = node
        right_child._parent = node

        return node

    def print_leaf_nodes(self):
        nodes = self.get_leaf_nodes()
        output = "< "
        for node in nodes:
            output += f"({node}:"
            if node.block._allocated == True:
                output += "1) "
            else:
                output += "0) "
        output += ">"
        print(output)

    # returns a list of every node in the tree
    def get_all_nodes(self):
        nodes = []
        root = self.get_root()
        nodes = self._get_all_nodes(root, nodes)
        return nodes

    def _get_all_nodes(self, node, nodes):
        nodes.append(node)

        if node._leftchild != None:
            self._get_all_nodes(node._leftchild, nodes)

        if node._rightchild != None:
            self._get_all_nodes(node._rightchild, nodes)
        return nodes

    def print_all_nodes(self):
        nodes = self.get_all_nodes()
        output = ""
        for node in nodes:
            output += f"({node.block.size}, "
            if node.block._allocated == True:
                output += "1) "
            else:
                output += "0) "
        print(output)

    # removes two children nodes of a parent node
    def merge_children_nodes(self, node):
        node._leftchild.block = None
        node._leftchild._parent = None
        node._leftchild = None

        node._rightchild.block = None
        node._rightchild._parent = None
        node._rightchild = None

    # sets a node to being deallocated. in other words the node is now free for memory allocation
    def deallocate_node(self, node):
        node.block._allocated = False
        node.block._process = None
        node.block._accessed = 0



class memory(object):

    def __init__(self):
        # in KB, 1 = 1KB, 1024 = 1024KB = 1MB
```

```
        self._user_space_mem_size = 4096
        self._page_size = 4
        self._allocation_queue = QueueV0()  #this is the queue where processes in need of memory allocation are enqueue
        self._buddy_tree = BuddySystemBST(self._user_space_mem_size)
        self._bitmap = {0:0}
        self._replacement_queue = QueueV0() # this is the queue used for the second chance algorithm

    def request_memory_allocation(self, process):
        self._allocation_queue.enqueue(process)

    # processes requesting memory are dequeued on a FIFO basis and allocated memory
    def allocate_memory(self):
        if self._allocation_queue.length() != 0:
            proc = self._allocation_queue.dequeue()
            node_found = self._allocate_memory(proc)
            self._replacement_queue.enqueue(node_found)
            print(f"process {proc.pid}: requested {proc.memory}KB - allocated {node_found.block.size}KB")
            return node_found

    def _allocate_memory(self, proc):
        if self.is_memory_full():
            self.replace()

        # the amount of memory that the process is requesting
        mem_required = proc.memory

        if mem_required < self._page_size:
            mem_required = self._page_size

        # the memory requested by the process is rounded to the nearest power of 2
        mem_required -= 1
        k = 1
        while k < mem_required:
            k *= 2
        target_size = k

        # finds a free block that suits the process and allocates the block to that process
        block_nodes = self._buddy_tree.get_leaf_nodes()
        found = False
        node_found = None
        for node in block_nodes:
            if node.block.size == target_size and node.block._allocated == False:
                found = True
                node.block.process = proc
                node.block._allocated = True
                node.block._accessed = 0
                self.update_bitmap()
                node_found = node
                break

        if found == False:
            # finds if any free blocks can be split recursively to be allocated to the process
            block_node_to_split = None
            max_free_node_size = 0
            for node in block_nodes:
                if node.block.size > max_free_node_size and node.block._allocated == False and node.block.size >= target_size:
                    max_free_node_size = node.block.size
                    block_node_to_split = node
                    break

            # if a node suitable for a split is found, it is split until its size matches the target_size
            # and then allocates the left-most split leaf node to the process requesting memory
            # otherwise blocks are deallocated recursively unitl the process requesting memory
            # can have memory allocated to it
            if block_node_to_split != None:
                free_block_node = self.split_until_allocated(target_size, block_node_to_split)
                free_block_node.block.process = proc
                free_block_node.block._allocated = True
                free_block_node.block._accessed = 0
                self.update_bitmap()
                node_found = free_block_node
            else:
                self.replace()
                node_found = self._allocate_memory(proc)

        return node_found

    # split tree nodes recursively until the leaf nodes of that subtree
    # match the target size
    def split_until_allocated(self, target_size, block_node_to_split):
        free_block_node = None
        parent_node = self._buddy_tree.split_node(block_node_to_split)
        if parent_node._leftchild.block.size == target_size:
            free_block_node = parent_node._leftchild
        else:
            free_block_node = self.split_until_allocated(target_size, parent_node._leftchild)
        return free_block_node
```

```
    # does a simple check to see if the memory is completely full
    def is_memory_full(self):
        full = True
        for block in self._bitmap:
            if self._bitmap[block] == 0:
                full = False
                break
        return full

    # deallocated blocks from memory using the second chance algorithm
    def replace(self):
        if self._replacement_queue.length() != 0:
            node = self._replacement_queue.dequeue()
            if node.block._accessed == 1:
                node.block._accessed = 0
                self._replacement_queue.enqueue(node)
                print(f"process {node.block.process.pid} has been given a second chance")
            else:
                print(f"process {node.block.process.pid} has been deallocated")
                self.remove_block(node)

            self.update_bitmap()

    def update_bitmap(self):
        block_nodes = self._buddy_tree.get_leaf_nodes()
        self._bitmap = {}
        for node in block_nodes:
            if node.block._allocated == True:
                self._bitmap[block_nodes.index(node)] = 1
            else:
                self._bitmap[block_nodes.index(node)] = 0
        return self._bitmap

    # merges a block and its buddy if both are not allocated
    # othewise the block is simply deallocated and the tree stays as it is
    def remove_block(self, node):
        if self._buddy_tree.is_leaf(node) == True:
            if self._buddy_tree.is_leaf(node._parent._rightchild) == True and node._parent._rightchild.block._allocated == False:
                node._parent.block._allocated = False
                self._buddy_tree.merge_children_nodes(node._parent)
            else:
                self._buddy_tree.deallocate_node(node)
            self.update_bitmap()

    def calculate_fragmentation(self):
        nodes = self._buddy_tree.get_leaf_nodes()
        internal_memory_consumed = 0
        external_memory_consumed = 0
        for node in nodes:
            if node.block.process != None:
                internal_memory_consumed += node.block.process.memory
                external_memory_consumed += node.block.size

        # fragmentation result is the percentage of unused memory
        internal_fragmentation = ((external_memory_consumed - internal_memory_consumed) / external_memory_consumed ) * 100
        external_fragmentation = ((self._user_space_mem_size - external_memory_consumed) / self._user_space_mem_size) * 100
        print(f"-----------------\nFragmentation: \nInternal: {internal_fragmentation}% \nExternal: {external_fragmentation }%\n------




if __name__ == "__main__":
    def basic_test():
        mem = memory()
        proc1 = Process(1, 50)
        mem.request_memory_allocation(proc1)
        proc1_node = mem.allocate_memory()

        proc2 = Process(2, 254)
        mem.request_memory_allocation(proc2)
        mem.allocate_memory()

        proc4 = Process(4, 120)
        mem.request_memory_allocation(proc4)
        mem.allocate_memory()

        proc5 = Process(5, 1000)
        mem.request_memory_allocation(proc5)
        mem.allocate_memory()

        proc6 = Process(6, 500)
        mem.request_memory_allocation(proc6)
        mem.allocate_memory()

        proc7 = Process(7, 2010)
        mem.request_memory_allocation(proc7)
```

```
        mem.allocate_memory()


        print(mem._bitmap)
        mem._buddy_tree.print_leaf_nodes()

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

        #mem.remove_block(proc1_node)
        #print(mem._bitmap)
        #mem._buddy_tree.print_leaf_nodes()

        proc8 = Process(8, 60)
        mem.request_memory_allocation(proc8)
        mem.allocate_memory()
        mem._buddy_tree.print_leaf_nodes()
        mem.calculate_fragmentation()


        proc9 = Process(9, 100)
        mem.request_memory_allocation(proc9)
        mem.allocate_memory()
        mem._buddy_tree.print_leaf_nodes()

        proc10 = Process(10, 1500)
        mem.request_memory_allocation(proc10)
        mem.allocate_memory()
        mem._buddy_tree.print_leaf_nodes()

        #proc11 = Process(11, 1)
        #mem.request_memory_allocation(proc11)
        #mem.allocate_memory()
        #mem._buddy_tree.print_leaf_nodes()

        mem.calculate_fragmentation()

    def random_test():
        mem = memory()

        for i in range(250):
            proc = Process(i, randint(mem._page_size, 32))
            mem.request_memory_allocation(proc)
            node = mem.allocate_memory()

            node.block._accessed = randint(0,1)

        mem.calculate_fragmentation()
        mem._buddy_tree.print_leaf_nodes()


    #basic_test()
    random_test()
```

Task 4

To simulate my memory management algorithms I have a loop that runs 1000 times. On every iteration of the loop a Process object is instantiated where it is requesting memory in the range of [4, 32]KB. The memory then allocates the memory requested to a block of a node in the tree. The block's access bit randomly given a value of either 0 or 1 in order to simulate that particular block being accessed or not.

Upon completion, the final levels of internal and external fragmentation are calculated, and all leaf nodes of the tree are printed in a bitmap format, with the block size as the key and a binary bit indicating whether that block is allocated to a process or not

```
def random_test():
    mem = memory()

    for i in range(250):
        proc = Process(i, randint(mem._page_size, 32))
        mem.request_memory_allocation(proc)
        node = mem.allocate_memory()
        node.block._accessed = randint(0,1)

        mem.calculate_fragmentation()
        mem._buddy_tree.print_leaf_nodes()
```

The output looks as follows

Since the output is quite long and screenshotting the output would be a considerate effort, I simply copy and pasted the output below

```
process 0: requested 4KB - allocated 4KB
process 1: requested 9KB - allocated 8KB
process 2: requested 13KB - allocated 16KB
process 3: requested 23KB - allocated 32KB
process 4: requested 23KB - allocated 32KB
process 5: requested 23KB - allocated 32KB
process 6: requested 8KB - allocated 8KB
process 7: requested 12KB - allocated 16KB
process 8: requested 5KB - allocated 4KB
process 9: requested 15KB - allocated 16KB
process 10: requested 31KB - allocated 32KB
process 11: requested 26KB - allocated 32KB
process 12: requested 11KB - allocated 16KB
process 13: requested 10KB - allocated 16KB
process 14: requested 4KB - allocated 4KB
process 15: requested 23KB - allocated 32KB
process 16: requested 13KB - allocated 16KB
process 17: requested 21KB - allocated 32KB
process 18: requested 28KB - allocated 32KB
process 19: requested 21KB - allocated 32KB
process 20: requested 14KB - allocated 16KB
process 21: requested 8KB - allocated 8KB
process 22: requested 18KB - allocated 32KB
process 23: requested 25KB - allocated 32KB
process 24: requested 30KB - allocated 32KB
process 25: requested 29KB - allocated 32KB
process 26: requested 4KB - allocated 4KB
process 27: requested 18KB - allocated 32KB
process 28: requested 8KB - allocated 8KB
process 29: requested 25KB - allocated 32KB
process 30: requested 20KB - allocated 32KB
process 31: requested 31KB - allocated 32KB
process 32: requested 9KB - allocated 8KB
process 33: requested 24KB - allocated 32KB
process 34: requested 24KB - allocated 32KB
process 35: requested 24KB - allocated 32KB
process 36: requested 25KB - allocated 32KB
process 37: requested 9KB - allocated 8KB
process 38: requested 27KB - allocated 32KB
process 39: requested 12KB - allocated 16KB
process 40: requested 16KB - allocated 16KB
process 41: requested 32KB - allocated 32KB
process 42: requested 6KB - allocated 8KB
process 43: requested 30KB - allocated 32KB
process 44: requested 9KB - allocated 8KB
process 45: requested 9KB - allocated 8KB
process 46: requested 27KB - allocated 32KB
process 47: requested 18KB - allocated 32KB
process 48: requested 12KB - allocated 16KB
process 49: requested 22KB - allocated 32KB
process 50: requested 13KB - allocated 16KB
process 51: requested 31KB - allocated 32KB
process 52: requested 24KB - allocated 32KB
process 53: requested 22KB - allocated 32KB
process 54: requested 26KB - allocated 32KB
process 55: requested 20KB - allocated 32KB
process 56: requested 13KB - allocated 16KB
process 57: requested 11KB - allocated 16KB
process 58: requested 20KB - allocated 32KB
process 59: requested 30KB - allocated 32KB
process 60: requested 13KB - allocated 16KB
process 61: requested 30KB - allocated 32KB
process 62: requested 8KB - allocated 8KB
process 63: requested 21KB - allocated 32KB
process 64: requested 23KB - allocated 32KB
process 65: requested 12KB - allocated 16KB
process 66: requested 4KB - allocated 4KB
process 67: requested 19KB - allocated 32KB
process 68: requested 7KB - allocated 8KB
process 69: requested 10KB - allocated 16KB
process 70: requested 22KB - allocated 32KB
process 71: requested 21KB - allocated 32KB
process 72: requested 27KB - allocated 32KB
process 73: requested 29KB - allocated 32KB
process 74: requested 28KB - allocated 32KB
process 75: requested 29KB - allocated 32KB
process 76: requested 31KB - allocated 32KB
process 77: requested 21KB - allocated 32KB
```

```
process 78: requested 4KB - allocated 4KB
process 79: requested 21KB - allocated 32KB
process 80: requested 9KB - allocated 8KB
process 81: requested 30KB - allocated 32KB
process 82: requested 10KB - allocated 16KB
process 83: requested 6KB - allocated 8KB
process 84: requested 21KB - allocated 32KB
process 85: requested 8KB - allocated 8KB
process 86: requested 4KB - allocated 4KB
process 87: requested 26KB - allocated 32KB
process 88: requested 18KB - allocated 32KB
process 89: requested 12KB - allocated 16KB
process 90: requested 23KB - allocated 32KB
process 91: requested 17KB - allocated 16KB
process 92: requested 20KB - allocated 32KB
process 93: requested 22KB - allocated 32KB
process 94: requested 17KB - allocated 16KB
process 95: requested 9KB - allocated 8KB
process 96: requested 17KB - allocated 16KB
process 97: requested 12KB - allocated 16KB
process 98: requested 8KB - allocated 8KB
process 99: requested 32KB - allocated 32KB
process 100: requested 16KB - allocated 16KB
process 101: requested 19KB - allocated 32KB
process 102: requested 11KB - allocated 16KB
process 103: requested 7KB - allocated 8KB
process 104: requested 32KB - allocated 32KB
process 105: requested 22KB - allocated 32KB
process 106: requested 12KB - allocated 16KB
process 107: requested 31KB - allocated 32KB
process 108: requested 22KB - allocated 32KB
process 109: requested 29KB - allocated 32KB
process 110: requested 16KB - allocated 16KB
process 111: requested 26KB - allocated 32KB
process 112: requested 24KB - allocated 32KB
process 113: requested 18KB - allocated 32KB
process 114: requested 7KB - allocated 8KB
process 115: requested 9KB - allocated 8KB
process 116: requested 29KB - allocated 32KB
process 117: requested 6KB - allocated 8KB
process 118: requested 15KB - allocated 16KB
process 119: requested 19KB - allocated 32KB
process 120: requested 14KB - allocated 16KB
process 121: requested 8KB - allocated 8KB
process 122: requested 23KB - allocated 32KB
process 123: requested 16KB - allocated 16KB
process 124: requested 16KB - allocated 16KB
process 125: requested 31KB - allocated 32KB
process 126: requested 29KB - allocated 32KB
process 127: requested 20KB - allocated 32KB
process 128: requested 23KB - allocated 32KB
process 129: requested 25KB - allocated 32KB
process 130: requested 4KB - allocated 4KB
process 131: requested 6KB - allocated 8KB
process 132: requested 23KB - allocated 32KB
process 133: requested 32KB - allocated 32KB
process 134: requested 5KB - allocated 4KB
process 135: requested 21KB - allocated 32KB
process 136: requested 21KB - allocated 32KB
process 137: requested 17KB - allocated 16KB
process 138: requested 9KB - allocated 8KB
process 139: requested 26KB - allocated 32KB
process 140: requested 21KB - allocated 32KB
process 141: requested 6KB - allocated 8KB
process 142: requested 13KB - allocated 16KB
process 143: requested 19KB - allocated 32KB
process 144: requested 14KB - allocated 16KB
process 145: requested 16KB - allocated 16KB
process 146: requested 26KB - allocated 32KB
process 147: requested 13KB - allocated 16KB
process 148: requested 28KB - allocated 32KB
process 149: requested 25KB - allocated 32KB
process 150: requested 29KB - allocated 32KB
process 151: requested 13KB - allocated 16KB
process 152: requested 29KB - allocated 32KB
process 153: requested 14KB - allocated 16KB
process 154: requested 17KB - allocated 16KB
process 155: requested 28KB - allocated 32KB
process 156: requested 11KB - allocated 16KB
process 157: requested 4KB - allocated 4KB
process 158: requested 6KB - allocated 8KB
process 159: requested 25KB - allocated 32KB
process 160: requested 12KB - allocated 16KB
process 161: requested 18KB - allocated 32KB
process 162: requested 30KB - allocated 32KB
process 163: requested 17KB - allocated 16KB
process 164: requested 27KB - allocated 32KB
```

```
process 165: requested 14KB - allocated 16KB
process 166: requested 26KB - allocated 32KB
process 167: requested 9KB - allocated 8KB
process 168: requested 11KB - allocated 16KB
process 169: requested 25KB - allocated 32KB
process 170: requested 16KB - allocated 16KB
process 171: requested 17KB - allocated 16KB
process 172: requested 4KB - allocated 4KB
process 173: requested 29KB - allocated 32KB
process 174: requested 30KB - allocated 32KB
process 175: requested 25KB - allocated 32KB
process 176: requested 17KB - allocated 16KB
process 177: requested 26KB - allocated 32KB
process 178: requested 9KB - allocated 8KB
process 179: requested 14KB - allocated 16KB
process 180: requested 30KB - allocated 32KB
process 0 has been given a second chance
process 1 has been deallocated
process 2 has been deallocated
process 3 has been given a second chance
process 4 has been deallocated
process 181: requested 19KB - allocated 32KB
process 182: requested 4KB - allocated 4KB
process 183: requested 12KB - allocated 16KB
process 5 has been given a second chance
process 6 has been given a second chance
process 7 has been deallocated
process 184: requested 11KB - allocated 16KB
process 8 has been given a second chance
process 9 has been given a second chance
process 10 has been deallocated
process 185: requested 13KB - allocated 16KB
process 11 has been deallocated
process 186: requested 22KB - allocated 32KB
process 12 has been deallocated
process 13 has been given a second chance
process 14 has been given a second chance
process 15 has been given a second chance
process 16 has been deallocated
process 17 has been deallocated
process 187: requested 18KB - allocated 32KB
process 188: requested 13KB - allocated 16KB
process 189: requested 10KB - allocated 16KB
process 190: requested 8KB - allocated 8KB
process 18 has been given a second chance
process 19 has been deallocated
process 191: requested 29KB - allocated 32KB
process 192: requested 14KB - allocated 16KB
process 193: requested 8KB - allocated 8KB
process 20 has been given a second chance
process 21 has been deallocated
process 22 has been deallocated
process 194: requested 25KB - allocated 32KB
process 23 has been given a second chance
process 24 has been deallocated
process 195: requested 28KB - allocated 32KB
process 196: requested 6KB - allocated 8KB
process 25 has been deallocated
process 197: requested 26KB - allocated 32KB
process 26 has been given a second chance
process 27 has been deallocated
process 198: requested 10KB - allocated 16KB
process 199: requested 4KB - allocated 4KB
process 28 has been deallocated
process 29 has been deallocated
process 200: requested 16KB - allocated 16KB
process 201: requested 17KB - allocated 16KB
process 30 has been deallocated
process 202: requested 27KB - allocated 32KB
process 31 has been given a second chance
process 32 has been given a second chance
process 33 has been deallocated
process 203: requested 29KB - allocated 32KB
process 34 has been deallocated
process 204: requested 14KB - allocated 16KB
process 35 has been deallocated
process 205: requested 27KB - allocated 32KB
process 36 has been deallocated
process 206: requested 23KB - allocated 32KB
process 207: requested 13KB - allocated 16KB
process 37 has been deallocated
process 38 has been deallocated
process 208: requested 15KB - allocated 16KB
process 209: requested 9KB - allocated 8KB
process 39 has been deallocated
process 40 has been deallocated
process 41 has been deallocated
```

```
process 210: requested 25KB - allocated 32KB
process 211: requested 13KB - allocated 16KB
process 212: requested 14KB - allocated 16KB
process 213: requested 9KB - allocated 8KB
process 42 has been deallocated
process 43 has been deallocated
process 214: requested 23KB - allocated 32KB
process 44 has been deallocated
process 45 has been given a second chance
process 46 has been deallocated
process 215: requested 21KB - allocated 32KB
process 216: requested 14KB - allocated 16KB
process 47 has been given a second chance
process 48 has been given a second chance
process 49 has been given a second chance
process 50 has been deallocated
process 217: requested 17KB - allocated 16KB
process 51 has been deallocated
process 218: requested 21KB - allocated 32KB
process 52 has been deallocated
process 219: requested 28KB - allocated 32KB
process 220: requested 4KB - allocated 4KB
process 53 has been deallocated
process 221: requested 13KB - allocated 16KB
process 222: requested 6KB - allocated 8KB
process 223: requested 15KB - allocated 16KB
process 54 has been given a second chance
process 55 has been deallocated
process 224: requested 29KB - allocated 32KB
process 56 has been deallocated
process 57 has been given a second chance
process 58 has been given a second chance
process 59 has been given a second chance
process 60 has been given a second chance
process 61 has been given a second chance
process 62 has been given a second chance
process 63 has been given a second chance
process 64 has been deallocated
process 225: requested 28KB - allocated 32KB
process 65 has been given a second chance
process 66 has been given a second chance
process 67 has been given a second chance
process 68 has been given a second chance
process 69 has been given a second chance
process 70 has been deallocated
process 226: requested 31KB - allocated 32KB
process 71 has been deallocated
process 227: requested 25KB - allocated 32KB
process 72 has been deallocated
process 228: requested 28KB - allocated 32KB
process 229: requested 15KB - allocated 16KB
process 73 has been deallocated
process 230: requested 22KB - allocated 32KB
process 74 has been deallocated
process 231: requested 14KB - allocated 16KB
process 75 has been given a second chance
process 76 has been given a second chance
process 77 has been given a second chance
process 78 has been given a second chance
process 79 has been deallocated
process 232: requested 24KB - allocated 32KB
process 233: requested 4KB - allocated 4KB
process 234: requested 10KB - allocated 16KB
process 80 has been given a second chance
process 81 has been given a second chance
process 82 has been deallocated
process 235: requested 17KB - allocated 16KB
process 83 has been given a second chance
process 84 has been given a second chance
process 85 has been given a second chance
process 86 has been deallocated
process 87 has been deallocated
process 236: requested 28KB - allocated 32KB
process 88 has been deallocated
process 237: requested 26KB - allocated 32KB
process 89 has been given a second chance
process 90 has been given a second chance
process 91 has been given a second chance
process 92 has been deallocated
process 238: requested 18KB - allocated 32KB
process 93 has been deallocated
process 239: requested 14KB - allocated 16KB
process 94 has been deallocated
process 95 has been given a second chance
process 96 has been deallocated
process 97 has been given a second chance
process 98 has been given a second chance
```

```
process 99 has been given a second chance
process 100 has been given a second chance
process 101 has been deallocated
process 240: requested 28KB - allocated 32KB
process 241: requested 5KB - allocated 4KB
process 102 has been given a second chance
process 103 has been given a second chance
process 104 has been given a second chance
process 105 has been given a second chance
process 106 has been given a second chance
process 107 has been deallocated
process 242: requested 28KB - allocated 32KB
process 108 has been deallocated
process 243: requested 18KB - allocated 32KB
process 244: requested 11KB - allocated 16KB
process 245: requested 6KB - allocated 8KB
process 109 has been given a second chance
process 110 has been deallocated
process 111 has been given a second chance
process 112 has been deallocated
process 246: requested 31KB - allocated 32KB
process 113 has been given a second chance
process 114 has been deallocated
process 115 has been given a second chance
process 116 has been deallocated
process 247: requested 32KB - allocated 32KB
process 248: requested 11KB - allocated 16KB
process 117 has been given a second chance
process 118 has been deallocated
process 119 has been deallocated
process 249: requested 31KB - allocated 32KB
-----------------
Fragmentation:
Internal: 17.814667988107036%
External: 1.46484375%
-----------------
< (4 MB:1) (4 MB:1) (8 MB:1) (16 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (8 MB:1) (4 MB:1) (4 MB:1) (16 MB:1) (16 MB:1) (16 MB:1) (16 MB:1
(16 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (8 MB:1) (8 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (4 MB:
(16 MB:1) (32 MB:1) (32 MB:1) (8 MB:1) (8 MB:1) (16 MB:1) (32 MB:1) (16 MB:1) (16 MB:1) (32 MB:1) (32 MB:1)(16 MB:1) (16 MB:1) (16 MB:
(32 MB:1) (32 MB:1) (8 MB:1) (8 MB:1) (16 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (16 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (16 MB
(16 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (4 MB:1) (4 MB:1) (8 MB:1) (32 MB:1) (16 MB:1) (8 MB:1) (8 MB:1)
(32 MB:1) (16 MB:1) (16 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:1) (8 MB:1) (4 MB:0) (4 MB:1) (32 MB:1)(32 MB:1
(32 MB:1) (32 MB:1) (16 MB:1) (16 MB:1) (16 MB:1) (8 MB:1) (8 MB:1) (16 MB:0) (16 MB:1) (32 MB:1) (16 MB:1) (16 MB:1) (32 MB:1) (8 MB:
(32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:0) (8 MB:1) (8 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (32 MB:1) (16 MB:0) (16 MB:1) (32 MB
(8 MB:1) (16 MB:1) (32 MB:1) >
```

As we can see in the output above, initially the program begins allocating memory to processes without any replacement. Once the memory becomes full we see processes having memory deallocated to them, some processes getting a second chance before becoming replaced and also new processes getting memory allocated to them.

Running the code multiple times produces different results and fragmentation levels but the fragmentation levels more or less stay consistent for output. The external fragmentation is almost always fairly low ~2%, but at times it may be as low as 0.3% or as high as 5%. Internal fragmentation on the other hand is quite significant and stays around 20%. If the potential request size is increased to a range of [4, 64]KB or [4,128]KB for example, then the internal and fragmentation levels also increase and produce a wider distribution of fragmentation upon every instance of the code running