Task 1

Size of Main Memory (user space): 4MB
Size of a Page: 4KB

Organisation of memory blocks:

- 32 blocks of 8 pages = 1024KB
- 16 blocks of 16 pages = 1024KB
- 8 blocks of 32 pages = 1024KB
- 4 blocks of 64 pages = 1024KB

64 Blocks in total = 4096KB, which encompasses the entire user space

Algorithms

Free Memory Tracking

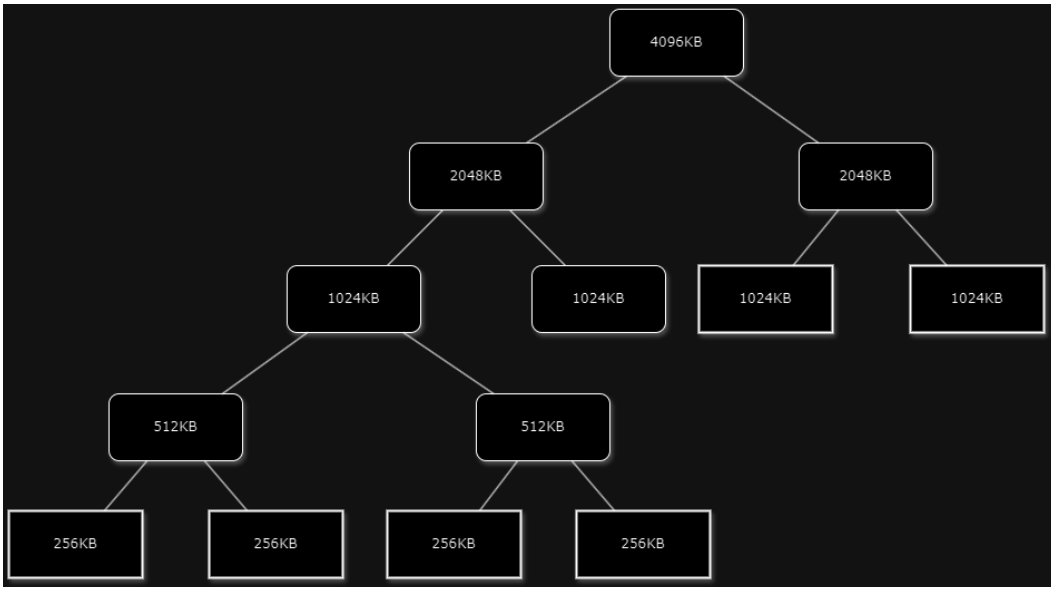
- Due to the fact that the memory is allocated in variable-sized blocks, the data structure that stores the address to the free blocks would need to be highly efficient.
- An algorithm with a fast lookup time (O(1) or O(log n)) would likely be needed to efficiently search the data structure for free blocks of memory.

Linked Lists

- Would contain objects that store:
  - First memory address
  - Size
  - Next object
- A linked list wouldn't be efficient due to its O(n) search time.
- When a memory request is made, the linked list will be searched for a block of closest fit.
- Due to its lookup and search time, I won't use a linked list, however it should be kept in mind as it could be used in addition to other data structures.

Binary Trees

- Binary trees would be far more efficient than Linked lists in terms of lookup and search times, providing O(log n) for worst-case lookup times.
- The lowest possible level of the binary tree would be a level where the block size would be the size of a page, in this case 4KB.
- This data structure would work well with the buddy system in memory allocation, as it makes it very easy to split the blocks.



- In this diagram, there are 4 blocks of 256KB and 3 blocks of 1024KB, which adds up to 4096KB.
- When a request is made for 512KB, it will be given a block of 1024KB.

Hash Maps

- Hash Maps would be much more efficient than Linked Lists, due to their fast lookup and search times.
- The keys would be the block sizes, and the values would be the blocks of that size
- Hash Maps could be even more efficient should you make the values Linked Lists containing the blocks.

- For this project, I'll be using Hash-Maps, not only for their fast look-up and search times, but also for their ease of implementation.

- Though Binary Search Trees might be faster in some cases, they're much harder to manage and more difficult to code.

Memory Allocation

- For memory allocation I'll be using **Best Fit**.
  - Best Fit searches the entire list and
- The reason I'll be using Best Fit is due to it working well with Hash Maps, as I can simply select the size of the block I want and instantly get a block of that size.

Task 2

Pseudocode

Page Class contains:

- Start address
- Process ID
- Access Bit (for page replacement)
- A method to allocate a process
- Deallocate a process

Block Class

- Variables:

  - List of pages it contains
  - Free Memory, used memory, total memory
  - A dictionary of process IDs and respective pages
- "largest_run_of_free_pages" method
  - Loop through all pages
  - Find the longest stretch of free pages
- "allocate_memory(PID: int, size: int)" method
  - Find the largest contiguous free space
  - Allocate the PID to the pages
  - Update memory stats and process allocations
  - Return the allocated pages
    - Or None, if there wasn't enough space
- "deallocate_memory(pid: int)" method
  - Deallocate memory assigned to the process
    - By looping through the list of pages
  - Update memory stats and process allocations
  - Return the amount of memory deallocated
- "has_amount(size: int)"
  - Returns a Boolean to indicate whether or not the block has that much free contiguous space.

MemoryRequest class
- __init__(pid: int, size: int)
  - Sets the pid and size in kb

MemoryManager class
- __init__(memory_config)
  - Initialise memory_config (how blocks are organised)
  - Initialise total_memory, all_block (list), free_blocks (dict)
    - The **free_block** dictionary will store all the free blocks, keys will be block sizes, and the values will be a linked list of blocks of that size
  - Initialise a process dictionary to keep track of what blocks a process is using
- get_closest_power_of_two(size: int) -> int
  - find the closest power of two smaller than or equal to the given size
- allocate_memory(request: MemoryRequest):
  - allocate memory to a process by choosing the smallest block with enough free space
  - Then update each memory stats and the process allocation dictionary
  - If a block is not found, call the clock_algorithm method
- deallocate_memory(pid: int):
  - By using the process allocation dictionary, we can find all the blocks being used by a process.
  - Loop these blocks are deallocate them and update the memory stats
- clock_algorithm(request: MemoryRequest):
  - Search through blocks for one with enough total space to hold the request
  - Loop through pages in the block and check the access bit
  - If it hasn't been accessed in a certain period of time, allow it to be replaced

class OperatingSystem:
- __init__():

- Initialise memory request queue
  - Using a queue implementation from another module
- Initialise a MemoryManager object

- add_memory_request(request: MemoryRequest):
  - Add the request to the memory request queue
- process_memory_requests():
  - keep dequeuing from the queue and allocate the memory through the memory manager
- finish_process_execution(pid: int):
  - Using the “deallocate_memory” method from the memory manager class, we can deallocate memory based off the pid

This pseudocode shows all 3 algorithms working together. With all of them being in the “MemoryManager” class.
The free memory tracking is done with a dictionary, with memory sizes as the keys and linked lists containing blocks of those sizes as the values.
The **best fit** algorithm is used in the “allocate_memory” function, in which it takes a size and tries to the find the closest size with the power of 2. It then searches the dictionary and finds a block of that size. If there is no block of that size it will go to the next largest size.
The clock algorithm is used in the “clock_algorithm” function, which takes in a MemoryRequest and searches through all the blocks, finding one large enough to satisfy the memory request, and it implements the clock algorithm on that block.

Task 3

The Page Class

```python
from linked_list import LinkedList
from queue_gs import Queue
from random import randint

PAGE_SIZE = 4
RAM_SIZE = 4096
BLOCK_SIZES = [8, 16, 32, 64, 128, 256, 512, 1024, 2048]

class Page:
    def __init__(self, start_address):
        self.start_address = start_address
        self.pid = None

        self.access_flag = 0

    def __repr__(self):
        return f"Page-SA:0x{self.start_address}/PID:{self.pid}"

    # assigns a process to this page
    def allocate(self, pid: int):
        if self.pid is None:
            self.pid = pid
            self.access_flag = randint(0, 1)    # to simulate it being accessed
            return True
        return False

    # removes the process
    def deallocate(self):
        self.pid = None
```

The Block class, and largest_run method

```python
class Block:
    def __init__(self, start_address, no_pages):
        self.start_address = start_address
        self.pages = [Page(start_address+(i*PAGE_SIZE)) for i in range(no_pages)]

        self.free_memory = no_pages * PAGE_SIZE
        self.used_memory = 0
        self.total_memory = no_pages * PAGE_SIZE

        self.block_category = self.total_memory

        # a dictionary with process id's as keys and values as lists of their allo
        self.process_allocations = {}

    def __repr__(self):
        return f":::Block/SA:{self.start_address}/Memory:{self.total_memory}kb/Fre

    # gets the largest run of free pages in the block
    def _largest_run_of_free_pages(self):
        start_index = 0
        end_index = 0
        current_stretch = 0
        max_stretch = 0
        max_start = 0
        max_end = 0

        # algorithm to find the longest stretch of free pages
        for i, page in enumerate(self.pages):
            if page.pid == None:
                if current_stretch == 0:
                    start_index = i
                current_stretch += 1
                end_index = i
                if current_stretch > max_stretch:
                    max_stretch = current_stretch
                    max_start = start_index
                    max_end = end_index
            else:
                current_stretch = 0

        return max_start, max_end, max_stretch
```

Memory Allocation, Deallocation and has_amount functions

```
# takes in a process id and size and tries to find enough contiguous pages to satisfy the request
# returns None if not enough contiguous pages were found, otherwise returns a list of pages
def allocate_memory(self, pid: int, size: int):
    results = self._largest_run_of_free_pages()
    max_start = results[0]
    max_end = results[1]
    max_stretch = results[2]

    # this runs through the free stretch and allocates just enough pages
    pages = []
    if max_stretch*PAGE_SIZE >= size:
        remaining_size = size
        current_index = max_start
        while remaining_size > 0:
            page = self.pages[current_index]
            page.allocate(pid)
            current_index += 1
            remaining_size -= PAGE_SIZE
            pages.append(page)
    else:
        return None

    self.process_allocations[pid] = pages
    self.free_memory -= len(pages) * PAGE_SIZE
    self.used_memory += len(pages) * PAGE_SIZE
    return pages

```

```
## takes in a PID and deallocates it from all pages it's using
 def deallocate_memory(self, pid):
     if pid in self.process_allocations:
         total_memory_saved = 0
         for page in self.process_allocations[pid]:
             page.deallocate()
             total_memory_saved += PAGE_SIZE
         self.used_memory -= total_memory_saved
         self.free_memory += total_memory_saved
         return total_memory_saved

 ## will take in a size in kb, will return whether or not the block can hold that much
 def has_amount(self, size: int):
     results = self._largest_run_of_free_pages()
     max_stretch = results[2]

     if max_stretch*PAGE_SIZE >= size:
         return True
     return False
```

MemoryRequest class

```
class MemoryRequest:
    def __init__(self, pid, size):
        self.process_id = pid
        self.size = size
```

MemoryManager constructor

```python
class MemoryManager:
    def __init__(self, memory_config):
        self.memory_config = memory_config
        self.total_memory = sum([value*key for key, value in self.memory_config.items()])
        self.used_memory = 0

        # clock variables
        self.clock_buffer = 10

        self.all_blocks = []
        self.free_blocks = {}

        self.processes_in_blocks = {}

        for block_size in BLOCK_SIZES:
            self.free_blocks[block_size] = LinkedList()

        addr = 0
        for key, value in self.memory_config.items():
            for i in range(key):
                block = Block(addr, int(value/PAGE_SIZE))
                self.free_blocks[value].append(block)
                self.all_blocks.append(block)
                addr += value
```

These functions are to help getting the correct block size.

```python
# PRIVATE METHODS
# takes a size in kb as input and returns the closest power of two that is smaller than it
def _get_closest_power_of_two(self, size: int) -> int:
    for i, block_size in enumerate(sorted(BLOCK_SIZES)):
        if block_size >= size:
            return block_size

# the same as the previous function except for that it gets the closest that is bigger than it
def _get_closest_power_of_two_bigger(self, size: int) -> int:
    for i, block_size in enumerate(sorted(BLOCK_SIZES, reverse=True)):
        if block_size <= size:
            return block_size

```

Allocate Memory function. Where best fit is used (or a modified version of it).

```python
## this method uses the _get_closest_power_of_two to search the dictionary for the smallest block that will
def allocate_memory(self, request: MemoryRequest):
    pid = request.process_id
    size = request.size
    smallest_block_size = self._get_closest_power_of_two(size)

    print(f"Attempting to allocate {size}kb of memory to process {pid}.")
    if size <= (self.total_memory - self.used_memory) and smallest_block_size != None:
        # starts with the smallest possible block for the request, works way up should they not be avaliable
        for block_size in [x for x in BLOCK_SIZES if x >= smallest_block_size]:
            if self.free_blocks[block_size].length() > 0:
                block = self.free_blocks[block_size].pop_front()

                block.allocate_memory(pid, size)

                memory_left = block.free_memory
                new_block_size = self._get_closest_power_of_two_bigger(memory_left)
                if new_block_size:
                    self.free_blocks[new_block_size].append(block)
                    block.block_category = new_block_size
                else:
                    block.block_category = None

                print(f"Successfully allocated {size}kb to process {pid}.")

                # update process id dictionary
                if pid in self.processes_in_blocks:
                    self.processes_in_blocks[pid].append(block)
                else:
                    self.processes_in_blocks[pid] = []
                    self.processes_in_blocks[pid].append(block)

                return True

        # if the algorithm gets here, it means it wasn't able to find a suitable block, therefore page replacement is needed
        self.clock_algorithm(request)
        return True
    else:
        print("Not enough memory.")
        return False
```

Deallocation Function

```python
# takes in a PID and removes it from every block that it's in
def deallocate_memory(self, pid: int):
    print(f"Attempting to deallocate Process {pid}.")
    for block in self.processes_in_blocks[pid]:
        if block.block_category:
            self.free_blocks[block.block_category].remove_by_value(block)
        block.deallocate_memory(pid)
        new_size_category = self._get_closest_power_of_two_bigger(block.free_memory)
        if new_size_category:
            self.free_blocks[new_size_category].append(block)
            block.block_category = new_size_category
        else:
            block.block_category = None

        print(f"Deallocated process {pid} from {block}.")

    del self.processes_in_blocks[pid]
```

This function simply prints each block in memory

```python
# prints every block in memory
def string_main_memory(self):
    string = ''
    for block in self.all_blocks:
        string += (block.view_block() + "\n")
    return string
```

The clock algorithm function

```python
# replaces pages by looping through blocks until it
#finds one with enough space and replaces pages based off the access flag
def clock_algorithm(self, request: MemoryRequest):
    pid = request.process_id
    size = request.size

    print(f"Attempting to replace pages for Process {pid}")
    # searches through all blocks to find suitable pages
    for block in self.all_blocks:
        if block.total_memory >= size:
            first_hand = 0
            second_hand = 0
            remaining_size = size
            possible_pages = []
            no_pages = 0

            # clock system to find pages that haven't been accessed
            while first_hand < len(block.pages)-1:
                if second_hand >= self.clock_buffer:
                    first_hand += 1

                    if block.pages[first_hand] in possible_pages and block.pages[first_hand].access_flag == 0:
                        old_pid = block.pages[first_hand].pid
                        block.pages[first_hand].pid = pid
                        remaining_size -= PAGE_SIZE

                        if pid in block.process_allocations:
                            block.process_allocations[pid].append(block.pages[first_hand])
                        else:
                            block.process_allocations[pid] = []
                            block.process_allocations[pid].append(block.pages[first_hand])

                        if old_pid:
                            block.process_allocations[old_pid].remove(block.pages[first_hand])
                        no_pages += 1

                        if remaining_size <= 0:
                            break

                if second_hand < len(block.pages)-1:
                    second_hand += 1

                    block.pages[second_hand].access_flag = 0
                    possible_pages.append(block.pages[second_hand])

            print(f"Successfully replaced {no_pages} pages for Process {pid} in {block}")

            # update process ids in the dictionary
            if pid in self.processes_in_blocks:
                self.processes_in_blocks[pid].append(block)
            else:
                self.processes_in_blocks[pid] = []
                self.processes_in_blocks[pid].append(block)

            return
```

OperatingSystem class, this class simply wraps everything into a neat class and implement the FIFO Queue for memory requests.

```
class OperatingSystem():
    def __init__(self):
        self.memory_requests = Queue()
        self.memory_config = {32: 64, 16: 128, 8: 256, 2: 512}
        self.memory_manager = MemoryManager(self.memory_config)

    def __str__(self):
        return self.memory_manager.string_main_memory()

    def add_memory_request(self, request: MemoryRequest):
        self.memory_requests.enqueue(request)

    def process_memory_requests(self):
        while self.memory_requests.length() > 0:
            request = self.memory_requests.dequeue()
            self.memory_manager.allocate_memory(request)

    def process_finished(self, pid: int):
        self.memory_manager.deallocate_memory(pid)

```

Task 4

```
os = OperatingSystem()

for i in range(200):
    os.add_memory_request(MemoryRequest(i, randint(15, 150)))

os.process_memory_requests()
```

This is what I will run and show the output of first. I create an OperatingSystem instance and add 200 memory requests of varying sizes to it.

```
Attempting to allocate 144kb of memory to process 0.
Successfully allocated 144kb to process 0.
Attempting to allocate 146kb of memory to process 1.
Successfully allocated 146kb to process 1.
Attempting to allocate 65kb of memory to process 2.
Successfully allocated 65kb to process 2.
Attempting to allocate 56kb of memory to process 3.
Successfully allocated 56kb to process 3.
Attempting to allocate 35kb of memory to process 4.
Successfully allocated 35kb to process 4.
Attempting to allocate 32kb of memory to process 5.
Successfully allocated 32kb to process 5.
Attempting to allocate 89kb of memory to process 6.
Successfully allocated 89kb to process 6.
Attempting to allocate 82kb of memory to process 7.
Successfully allocated 82kb to process 7.
Attempting to allocate 69kb of memory to process 8.
Successfully allocated 69kb to process 8.
Attempting to allocate 130kb of memory to process 9.
Successfully allocated 130kb to process 9.
Attempting to allocate 77kb of memory to process 10.
Successfully allocated 77kb to process 10.
Attempting to allocate 29kb of memory to process 11.
Successfully allocated 29kb to process 11.
Attempting to allocate 149kb of memory to process 12.
Successfully allocated 149kb to process 12.
Attempting to allocate 105kb of memory to process 13.
Successfully allocated 105kb to process 13.
Attempting to allocate 128kb of memory to process 14.
Successfully allocated 128kb to process 14.
Attempting to allocate 40kb of memory to process 15.
Successfully allocated 40kb to process 15.
Attempting to allocate 84kb of memory to process 16.
Successfully allocated 84kb to process 16.
Attempting to allocate 55kb of memory to process 17.
Successfully allocated 55kb to process 17.
Attempting to allocate 128kb of memory to process 18.
Successfully allocated 128kb to process 18.
Attempting to allocate 31kb of memory to process 19.
Successfully allocated 31kb to process 19.
Attempting to allocate 87kb of memory to process 20.
Successfully allocated 87kb to process 20.
Attempting to allocate 148kb of memory to process 21.
```

The output starts out normal, allocating memory to each process without the need for page replacement because the memory still has space left in it.

```
Attempting to allocate 145kb of memory to process 82.
Attempting to replace pages for Process 82
Successfully replaced 37 pages for Process 82 in :::Block/SA:4096/Memory:256kb/Free Memory:112kb/Used Memory:144kb:::
Attempting to allocate 92kb of memory to process 83.
Attempting to replace pages for Process 83
Successfully replaced 23 pages for Process 83 in :::Block/SA:2048/Memory:128kb/Free Memory:28kb/Used Memory:100kb:::
Attempting to allocate 90kb of memory to process 84.
Attempting to replace pages for Process 84
Successfully replaced 23 pages for Process 84 in :::Block/SA:2048/Memory:128kb/Free Memory:28kb/Used Memory:100kb:::
Attempting to allocate 89kb of memory to process 85.
Attempting to replace pages for Process 85
Successfully replaced 23 pages for Process 85 in :::Block/SA:2048/Memory:128kb/Free Memory:28kb/Used Memory:100kb:::
Attempting to allocate 56kb of memory to process 86.
Successfully allocated 56kb to process 86.
Attempting to allocate 39kb of memory to process 87.
Successfully allocated 39kb to process 87.
Attempting to allocate 119kb of memory to process 88.
Attempting to replace pages for Process 88
Successfully replaced 30 pages for Process 88 in :::Block/SA:2048/Memory:128kb/Free Memory:28kb/Used Memory:100kb:::
Attempting to allocate 131kb of memory to process 89.
Attempting to replace pages for Process 89
Successfully replaced 33 pages for Process 89 in :::Block/SA:4096/Memory:256kb/Free Memory:112kb/Used Memory:144kb:::
Attempting to allocate 129kb of memory to process 90.
Attempting to replace pages for Process 90
Successfully replaced 33 pages for Process 90 in :::Block/SA:4096/Memory:256kb/Free Memory:112kb/Used Memory:144kb:::
Attempting to allocate 34kb of memory to process 91.
Successfully allocated 34kb to process 91.
Attempting to allocate 33kb of memory to process 92.
Successfully allocated 33kb to process 92.
Attempting to allocate 135kb of memory to process 93.
Attempting to replace pages for Process 93
Successfully replaced 34 pages for Process 93 in :::Block/SA:4096/Memory:256kb/Free Memory:112kb/Used Memory:144kb:::
Attempting to allocate 139kb of memory to process 94.
Attempting to replace pages for Process 94
Successfully replaced 35 pages for Process 94 in :::Block/SA:4096/Memory:256kb/Free Memory:112kb/Used Memory:144kb:::
Attempting to allocate 45kb of memory to process 95.
Successfully allocated 45kb to process 95.
Attempting to allocate 17kb of memory to process 96.
Successfully allocated 17kb to process 96.
Attempting to allocate 66kb of memory to process 97.
Attempting to replace pages for Process 97
Successfully replaced 17 pages for Process 97 in :::Block/SA:2048/Memory:128kb/Free Memory:28kb/Used Memory:100kb:::
Attempting to allocate 16kb of memory to process 98.
Successfully allocated 16kb to process 98.
Attempting to allocate 72kb of memory to process 99.
Attempting to replace pages for Process 99
```

Later on, when the memory fills up, page replacement will be needed. You can see it here searching for new pages and allocating them to processes.

After all of the allocation, the string representation of all the blocks looks like this:

```
xxxxxxxxxx------
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxx--
xxxxxxxxxx------
xxxxxxxxxxxxxxx-
xxxxxxxxxx------
xxxxxxxxx-------
xxxxxxxxxxxxx---
xxxxxxxxxxxx----
xxxxxxxxxxxx----
xxxxxxxxxxx-----
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxxx-
xxxxxxxxxxx-----
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxx--
xxxxxxxxxx------
xxxxxxxxx-------
xxxxxxxxx-------
xxxxxxxxxxxx----
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxx---
xxxxxxxxxxxxxxx-
xxxxxxxxxxxxx---
xxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx---
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxx-----
xxxxxxxxxxxxxxxxxxxxxxxxxxx-----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--
xxxxxxxxxxxxxxxxxxxxxxxxxx------
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxx-----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxx-------
xxxxxxxxxxxxxxxxxxxxxxxxxx------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx---
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx---
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Where ‘x’ represents a allocated page, and ‘-‘ is an empty page, and each line represents a block.

For the second part of the simulation, I’ll run through each process and randomly deallocate certain ones.

```python
for i in range(200):
    if randint(0, 2) == 0:
        os.process_finished(i)

print(os)
```

```
Attempting to deallocate Process 0.
Deallocated process 0 from :::Block/SA:4096/Memory:256kb/Free Memory:168kb/Used Memory:88kb:::.
Attempting to deallocate Process 1.
Deallocated process 1 from :::Block/SA:4352/Memory:256kb/Free Memory:172kb/Used Memory:84kb:::.
Attempting to deallocate Process 2.
Deallocated process 2 from :::Block/SA:2048/Memory:128kb/Free Memory:32kb/Used Memory:96kb:::.
Attempting to deallocate Process 6.
Deallocated process 6 from :::Block/SA:2176/Memory:128kb/Free Memory:96kb/Used Memory:32kb:::.
Attempting to deallocate Process 13.
Deallocated process 13 from :::Block/SA:2688/Memory:128kb/Free Memory:128kb/Used Memory:0kb:::.
Attempting to deallocate Process 20.
Deallocated process 20 from :::Block/SA:3200/Memory:128kb/Free Memory:96kb/Used Memory:32kb:::.
Attempting to deallocate Process 25.
Deallocated process 25 from :::Block/SA:3584/Memory:128kb/Free Memory:100kb/Used Memory:28kb:::.
Attempting to deallocate Process 26.
Deallocated process 26 from :::Block/SA:5376/Memory:256kb/Free Memory:160kb/Used Memory:96kb:::.
Attempting to deallocate Process 27.
Deallocated process 27 from :::Block/SA:5632/Memory:256kb/Free Memory:176kb/Used Memory:80kb:::.
Attempting to deallocate Process 30.
Deallocated process 30 from :::Block/SA:3840/Memory:128kb/Free Memory:128kb/Used Memory:0kb:::.
Attempting to deallocate Process 34.
Deallocated process 34 from :::Block/SA:6144/Memory:512kb/Free Memory:120kb/Used Memory:392kb:::
Attempting to deallocate Process 35.
Deallocated process 35 from :::Block/SA:6144/Memory:512kb/Free Memory:208kb/Used Memory:304kb:::
Attempting to deallocate Process 43.
Deallocated process 43 from :::Block/SA:6656/Memory:512kb/Free Memory:96kb/Used Memory:416kb:::.
Attempting to deallocate Process 44.
Deallocated process 44 from :::Block/SA:2560/Memory:128kb/Free Memory:48kb/Used Memory:80kb:::.
Attempting to deallocate Process 46.
Deallocated process 46 from :::Block/SA:4096/Memory:256kb/Free Memory:168kb/Used Memory:88kb:::.
Attempting to deallocate Process 54.
Deallocated process 54 from :::Block/SA:4096/Memory:256kb/Free Memory:168kb/Used Memory:88kb:::.
Attempting to deallocate Process 56.
Deallocated process 56 from :::Block/SA:640/Memory:64kb/Free Memory:64kb/Used Memory:0kb:::.
Attempting to deallocate Process 59.
```

It runs through and deallocates certain processes, which would simulate certain processes finishing their execution.

The string representation of the blocks now looks like this:

```
xxxxxxxxxx------
xxxxxxxxx-------
xxxxxxxxxxxxx---
----------------
----------------
xxxxxxxxxxx-----
xxxxxxxxxxxxxx--
----------------
xxxxxxxxxxxxxxxx
----------------
xxxxxxxxxxxxxxx-
xxxxxxxxxxx-----
xxxxxxxxxxxxxx--
xxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxx
----------------
----------------
----------------
xxxxxxxxx-------
xxxxxxxxxxxx----
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxx---
xxxxxxxxxxxxxxx-
xxxxxxxxxxxxx---
xxxxxxxxxxxx----
-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
-----------------------xxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx---
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxx------------
--------------------------------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
----------------------xxxxxxxx--
xxxxxxxxxxxxxxxxxxxxxxxxxx------
xxxxxxxxxxxxxxxxxxxxxxxxxxxx----
--------------------xxxxxxx-----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
--------------------------------
xxxxxxxxxxxxxxxxxxxxxxxxxx------
-------------------------------------xxxxxxxxxxxxxxxxxx---------
-------------------------------------xxxxxxxxxxxxxx-------------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-------------------------------
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--------------------------
```

This would simulate what a real system’s memory would look like as new processes are added and some are finished their execution.