CSI3131 – Lab5
Memory Management Unit (Java)

Objectives

To implement a free space management unit using Java.

The OS keeps track of all free spaces, referred to as **holes**, using a linked list and must find a hole of appropriate size whenever a new program component is to be loaded into memory. The OS must also **coalesce** any **neighboring holes** resulting from the removal of programs from memory to prevent the memory from becoming a fragmented collection of increasingly smaller holes.

The memory management application (mmuApp) launches a simulation of many processes allocating memory blocks from a 4 GB RAM, then freeing them at a later stage.

Holes are represented as a linked list. Every node in the list has a start and end address. At the start, the system has a single hole of 4 GB. Later on, the holes are scattered everywhere.

The code provided has the following classes:

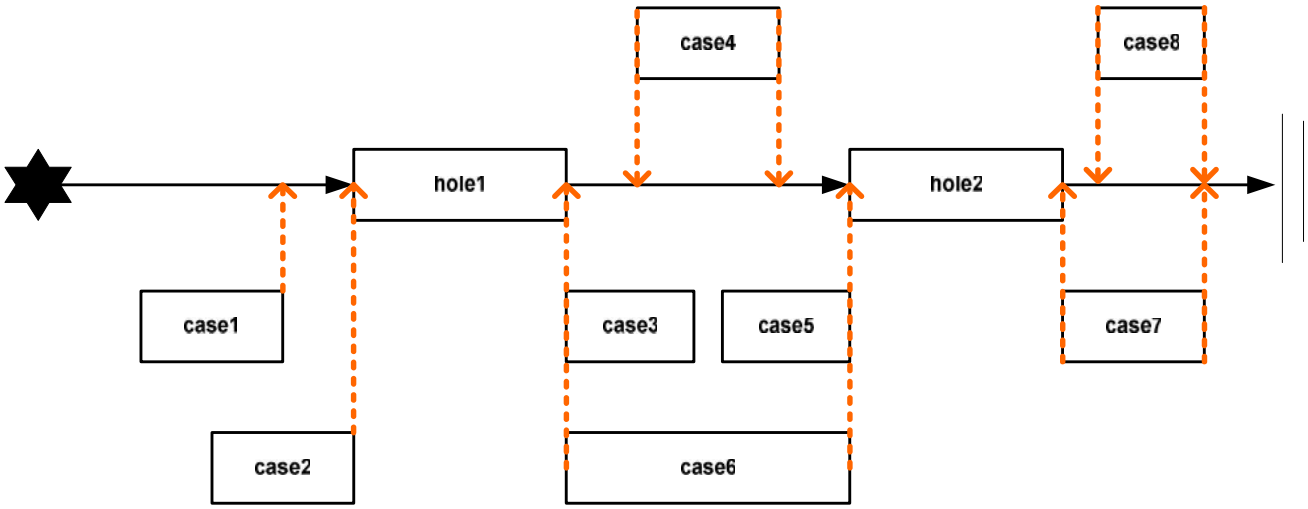
- **search_strtegy**: is an enumeration of the 4 search strategies, namely: FIRST_FIT, NEXT_FIT, BEST_FIT, WORST_FIT
- **mem_segment**: represent the nodes in the holes linked list
- **mmu_monitor**: provides 2 functions: allocate & free
- **process**: sleep – allocate – sleep – free
- **checker**: a daemon thread checking the state of the linked list – you must keep the checker happy
- **mmuApp**: The main applications managing the simulation

To run the application:

- `javac *.java: compile java code`
- `java mmuApp | tee tmp.txt: run application using first fit, with 100 processes`
- `java mmuApp 0 100| tee tmp.txt: run application using first fit, with 100 processes`
- `java mmuApp Alg nP| tee tmp.txt: run application using Alg, with nP processes`
  - `Alg: 0-first fit, 1-next fir, 2-best fit, 3-worst fit`
- **`Note you'll need to dump the log to a file to be able to trace your code`**

Part1 – Implement the free function (4 points)

The free function in mmu_monitor is provided as a stub. Your first task is to implement this function to return the freed segments back to pool. You have to be aware of coalescing issues.



Your solution have to accommodate the following cases:

1. freed segment is added before head, head is reassigned to new segment, segment is linked to previous head
2. freed segment merges with head
3. freed segment merges with a node – from the right
4. freed segment is inserted in between 2 nodes
5. freed segment merges with a node – from left
6. freed segment merges with 2 nodes
7. same as case 3
8. freed segment inserted at the tail

Note: When you get this part right your simulation should:

- ✓ not flag any checker errors
- ✓ run to completion
- ✓ deallocate all segments back to one contiguous 4 GB segment

Part2 – Implement all search strategies (6 points)

The code provided has only first fit strategy implemented. In this part you'll implement the three remaining strategies. You should implement the following methods – provided as stubs:

- public mem_segment **allocate_next_fit**(int size)
- public mem_segment **allocate_best_fit**(int size)
- public mem_segment **allocate_worst_fit**(int size)