Benha University

Faculty of Engineering at Shoubra

Computer program – Third year

Data structure and algorithms

Memory Allocation Project

This project implements a simulation of a basic memory allocation scheduling mechanism. Each process requires a specific amount of memory to initiate execution. If the required memory is not readily available, the process is placed in a waiting list until sufficient memory becomes available. Throughout the simulation, the status of memory allocation, waiting processes, and running processes is maintained until all processes have completed execution.

Classes:

Class	***Description***
MemBlock	Defines a block of memory with size, start address and end address
Process	Defines a process with id(should be assigned automatically), size of memory, time needed to finish and MemBlock reference to block assigned to it It has toString() function to return a string with process id and its size
MemAlloc	Drive class which has: • main method • All lists of processes and available memory • Method employs an algorithm to identify a free memory block within the available memory pool that is sufficient to accommodate the memory requirements of a specific process. Upon successful allocation, the method modifies the list of available memory blocks to reflect the newly occupied space. • Method to show all processes in waiting and running lists • Method to update lists:

	method governs the dynamic evolution of both the waiting and running process queues as follows: ➢ **Process completion:** When a running process reaches its stipulated completion time, it is removed from the running list, and a notification with relevant information is displayed to show that this process is ended. ➢ **Memory allocation:** If a free memory block of adequate size becomes available, it is allocated to a suitable process currently waiting in the waiting list. The selected process is then transferred from the waiting list to the running list. • **Memory Block Reclamation:** Upon the completion of a process, its allocated memory block is released. The released block is then merged with adjacent blocks, if possible, to form larger contiguous memory blocks.

Lists:

List	***Description***
Memory block list	List of all available memory blocks (starting with one block with the full size of all memory).
Starting process list	List to initiate processes and take one process from it each one second.
Running process list	List of processes which assigned a memory block and start running.
Waiting process list	List of processes which wait a suitable memory block to run.

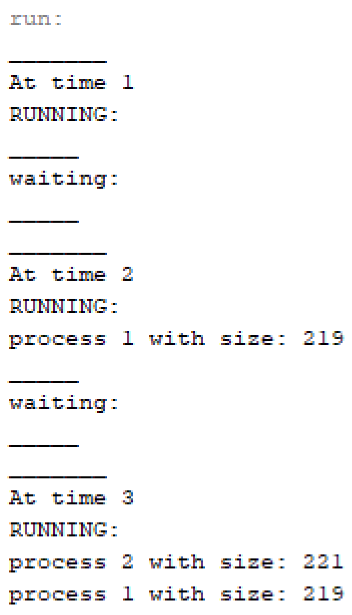
For each list, you can use any of basic data structures we studied in our course: Linked List, Queue, priority Queue, stack. (You should choose the suitable data structure).

If you need a priority queue, you can implement its data structure in a new class to define how to order processes (i.e. process memory block size, time out).

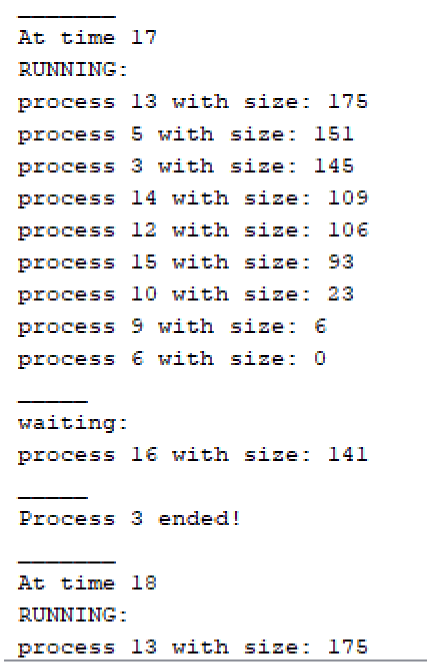
Project steps:

1. Memory is created with all available size of 1024(from address 0 to 1023).
2. 20 processes are created with random size (max 256) and random timeout (max 20000 ms). All these processes are inserted in the starting list.
3. At each second:
    - ➢ One process is dispatched from the starting list and assigned to the running or waiting list based on its needed memory.
    - ➢ All running processes timeout is decreased by 1000 ms.
    - ➢ All lists are updated (running process with timeout less than or equal zero is released and release used memory, waiting process that has suitable block size is transferred to the running list).
    - ➢ Show all processes in the running and waiting lists.
    - ➢ The program is stopped when all running and waiting lists are empty.

<u>Sample Output:</u>



```
run:
_______
At time 1
RUNNING:
_____
waiting:
_____
_______
At time 2
RUNNING:
process 1 with size: 219
_____
waiting:
_____
_______
At time 3
RUNNING:
process 2 with size: 221
process 1 with size: 219
```

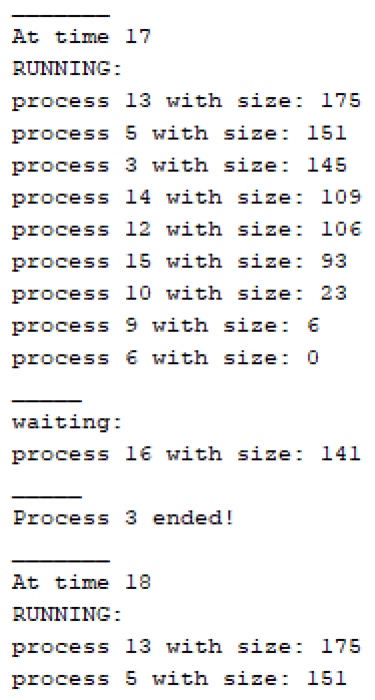


```
_______
At time 17
RUNNING:
process 13 with size: 175
process 5 with size: 151
process 3 with size: 145
process 14 with size: 109
process 12 with size: 106
process 15 with size: 93
process 10 with size: 23
process 9 with size: 6
process 6 with size: 0

_____
waiting:
process 16 with size: 141

_____
Process 3 ended!

_______
At time 18
RUNNING:
process 13 with size: 175
```



```
_______
At time 17
RUNNING:
process 13 with size: 175
process 5 with size: 151
process 3 with size: 145
process 14 with size: 109
process 12 with size: 106
process 15 with size: 93
process 10 with size: 23
process 9 with size: 6
process 6 with size: 0

_____
waiting:
process 16 with size: 141

_____
Process 3 ended!

_______
At time 18
RUNNING:
process 13 with size: 175
process 5 with size: 151
```

Instructions:

1. Each team is consists of 5-6 members.
2. Your code should be UNIQUE any copy from any source will result in ZERO grade.(a plagiarism checker is used to test all projects).
3. Each team should deliver a list of members and show participation of each member in this project.

Appendix:

1. Random values can be created using Random library " java.util.Random" and create an object like scanner
   Random r=new Random();
   int x=r.nextInt(1000); -> assign variable x with a random value between 1 and 1000
2. To make your program wait for some time you can use :
   Thread.sleep(*time to wait in milliseconds*);
   ex. Thread.sleep(1000); ->make the program wait for 1 sec
3. To use built-in linked list / queue implementation you can identify the type of object in <>
   ex.
   LinkedList<process> m=new LinkedList<process>();
   Queue<process> m = new LinkedList<> ();