Operating Systems - 2021

Project 3 – Memory Management

Implemented in [Java] using [IntelliJ IDEA]

Alireza Khorshidi (9735413)

Link to the project on GitHub
Click

Table of Contents

Introduction

This documentation contains information on an implementation of the third project of the course *Operating Systems* in Spring 2021.

The goal is to practice memory management for processes on RAM. Java is used to implement the following solution.

More details on the project are specified in the assignment file & the following Project overview section.

Project overview

The goal is to simulate memory allocation to processes on the RAM using 3 different techniques: First Fit, Best Fit, and Next Fit. The program is tasked to parse incoming instructions from the Terminal with each line specifying whether requesting space on RAM for a process or removing a process from the RAM, freeing up space and making room for the upcoming processes.

In order to do so, the abstract class *Memory* contains some basic functionality a RAM must have. Corresponding to the 3 memory management techniques, 3 child classes *MemoryFirstFit* , *MemoryBestFit*, and *MemoryNextFit*, are inherited from *Memory* class.

When running the program, one instance of each of the memory managers is created and every incoming command from the Terminal will be passed to them.

Finally, their overall status (Current memory allocation distribution for each process and free spaces, the number of compactions, the number of external fragmentation blocks, and average size of external fragmentations) for each memory type will be printed to the Terminal and also saved as a file “output.txt”.

There also are some tests that can be run directly via the IDE by un-commenting their lines.

Classes

Main

Start the application from here.

- public void Main(String args[])
  Gets input from the user, initializes the cars and the carwash and runs the cars.

- private static void runMemorySimulation()
  Read commands from the Terminal and applies them to 3 different instances of memory. Then prints their overall statuses to the Terminal and also saves them to a file “output.txt”.

- private static void Test1()
  A test method for the memories that runs internally without the input from the Terminal.

- private static void Test2()
  A test method for the memories that runs internally without the input from the Terminal.

- static void printMemoryStatus(Memory memory)
  Prints a memory’s current overall status to the Terminal.

- private static String getOverallMemoryStatus(Memory memory)
  Returns a string containing a memory’s current overall status (Current memory allocation distribution for each process and free spaces, the number of compactions, the number of external fragmentation blocks, and average size of external fragmentations).

MemoryBlock

The blocks on each memory with a starting location and a size. Can be free or assigned to a process.

- private String assignedTo
  To which process this memory block is assigned to (The block is free if it’s empty).

- private int startingLocation
  The starting location of this memory block (Indexing on the memory starts from 0).

- private int size
  The size of this memory block.

- public MemoryBlock(String assignedTo, int startingLocation, int size)
  Constructor. Assigns class’s properties correspondingly.

- public String getAssignedTo()
  Returns the assignedTo.

- public int getStartingLocation()
  Returns the startingLocation.

- public int getSize()
  Returns the memory size.

- public int compareBySizeTo(MemoryBlock other)
  Returns an integer as the result of comparing two blocks by their size (0 if equal, 1 if this is bigger, and -1 if the other block is bigger).

- public int compareByStartLocationTo(MemoryBlock other)
  Returns an integer as the result of comparing two blocks by their starting location (0 if equal, 1 if this is bigger, and -1 if the other block is bigger).

Memory

An abstract class representing the model of every memory.

- private final int size
  The size of the memory. Cannot be modified after initialization.

- protected List<MemoryBlock> loadedProcesses
  The processes currently loaded on the memory.

- private int compactionCount
  The number of times the memory compacted its processes.

- public Memory(int size)
  Constructor. Makes an instance of Memory class. Must be overridden by child classes. Initializes loadedProcesses as an empty list and sets compactionCount to 0 by default.

- public int getSize()
  Returns the size of the memory.

- public List<MemoryBlock> getLoadedProcesses()
  Returns processes currently loaded on the memory.

- public int getCompactionCount()
  Returns compactionCount.

- public void request(String processID, int processSize) throws Exception
  Allocates memory to a process by its ID, and size. If successful then a new instance of MemoryBlock with values corresponding to that process, are added to loadedProcesses. If not successful then an insufficient memory space exception is thrown (Must be overriden).

- public void release(String processID)
  Removes a process from the memory by its processID accordingly.

- public void compactProcesses()
  Eliminates all free space between processes on the memory by moving all processes to the starting side of the memory, leaving only one block of free memory.

- public int getTotalAllocatedSize()
  Total amount of memory that is currently allocated to other processes.

- public int getExternalFragmentationCount()
  The number of external fragmentation blocks on the memory.

- public int getTotalExternalFragmentationSize()
  The total size of all external fragmentation blocks on the memory.

- public String getMemoryStatus()
  A string containing the memory's status indicating an order of free spaces and processes in which the memory is distributed.

MemoryFirstFit

A model for a memory that uses first fit algorithm in order to allocate space to processes. The first fit algorithm, looks for the first free block of memory (starting from the beginning of it) that the process can fit into, and allocates space to that process equal to the process's size.

- public MemoryFirstFit(int size)
  Constructor.

- @Ovverride public void request(String processID, int processSize) throws Exception
  Allocates memory to a process by its ID, and size. If successful then a new instance of MemoryBlock with values corresponding to that process, are added to loadedProcesses. If not successful then an insufficient memory space exception is thrown (Overriden for child class's specifications).

- private boolean tryFirstFitAllocation(String processID, int processSize)
  The first fit algorithm that allocates space to a process. Returns true if successful and false if not.

MemoryBestFit

A model for a memory that uses best fit algorithm in order to allocate space to processes. The best fit algorithm, looks for the smallest free block of memory (starting from the beginning of it) that the process can fit into, and allocates space to that process equal to the process's size.

- public MemoryBestFit(int size)
  Constructor.

- @Ovverride public void request(String processID, int processSize) throws Exception
  Allocates memory to a process by its ID, and size. If successful then a new instance of MemoryBlock with values corresponding to that process, are added to loadedProcesses. If not successful then an insufficient memory space exception is thrown (Overriden for child class's specifications).

- private boolean tryBestFitAllocation(String processID, int processSize)
  The best fit algorithm that allocates space to a process. Returns true if successful and false if not.

- private List<MemoryBlock> getFreeMemoryBlocks()
  Returns a list of all free memory blocks on the memory, including external fragmentation blocks between the processes.

MemoryNextFit

A model for a memory that uses best fit algorithm in order to allocate space to processes. The best fit algorithm, looks for the first free block of memory (starting from the index at which the last process had allocated space on the memory) that the process can fit into, and allocates space to that process equal to the process's size.

- private int lastAllocationIndex
  The index of the location where memory was allocated to the last process.

- public MemoryBestFit(int size)
  Constructor.

- @Ovverride public void request(String processID, int processSize) throws Exception
  Allocates memory to a process by its ID, and size. If successful then a new instance of MemoryBlock with values corresponding to that process, are added to loadedProcesses. If not successful then an insufficient memory space exception is thrown (Overriden for child class's specifications).

- private boolean tryNextFitAllocation(String processID, int processSize)
  The next fit algorithm that allocates space to a process. Returns true if successful and false if not.

- @Override public void compactProcesses()
  Eliminates all free space between processes on the memory by moving all processes to the starting side of the memory, leaving only one block of free memory. Sets lastAllocationIndex to the sum of all of the loaded processes sum (Overriden).

Sample Tests

Test 1:

```
70
REQUEST A 23
REQUEST B 30
RELEASE A
REQUEST C 17
```

```
Test 1
*** First Fit ***
Compression No: 0
Memory Status: C, 6, B, 17
External Fragmentation blocks count: 1
External Fragmentation average size: 6.000000

*** Best Fit ***
Compression No: 0
Memory Status: 23, B, C,
External Fragmentation blocks count: 1
External Fragmentation average size: 23.000000

*** Next Fit ***
Compression No: 0
Memory Status: C, 6, B, 17
External Fragmentation blocks count: 1
External Fragmentation average size: 6.000000


Process finished with exit code 0
```

Test 2:

```
23
REQUEST P1 8
REQUEST P2 7
RELEASE P1
REQUEST P3 10
RELEASE P2
REQUEST P4 4
REQUEST P5 3
```

```
Test 2
*** First Fit ***
Compression No: 1
Memory Status: P4, P5, P3, 6
External Fragmentation blocks count: 0
External Fragmentation average size: NaN

*** Best Fit ***
Compression No: 1
Memory Status: P5, 4, P3, P4, 2
External Fragmentation blocks count: 1
External Fragmentation average size: 4.000000

*** Next Fit ***
Compression No: 1
Memory Status: P5, 4, P3, P4, 2
External Fragmentation blocks count: 1
External Fragmentation average size: 4.000000


Process finished with exit code 0
```