Operating Systems Coursework Report

6606598

Mouaz Ramadan Abdelsamad

There are 6 classes (java files) in my course work.

Detailed comments have been written throughout the code.

PhysicalMemory class:

This class houses the core functionality of allocating/ deallocating a process.

This class contains a list of memory allocated spaces (that contain segments of a process or can be a free space representing the memory left), a TLB table and a list of the processes in the memory.

This class also contains a set of helper methods to easy the simulation of a process's allocation/ deallocation.

Process Allocation:

When allocating a process to the memory the 'allocateProcess' method will be used, and it will take an input of a 2D array, which will represent a process. (Example; int[][] p1 = {{1}, {100}, {200}}) In this example we would like to take this 2D array and create a process object out of it, assign the segments and then allocate it to the memory, the 'setUpProcess' method will be called to construct a Process object with its segments. Since the process is inputted in the form of a 2D array, we will need to deconstruct this array into sections;

1. p1[0][0] will represent the process number, which in the example provided above is 1
2. The remaining part of the array will represent the segments and their size. For this part, a loop will be used to iterate through the 2D array after the index p1[0][0]. When looping through this array and extracting the size and the segment number at which this segment occurred, a segment object will be created by taking the segment number and size and putting them into the Segment constructor. All these segment objects that are created will be added to a list of segments that will then be used to initialize a Process object.

In the case that this is a shared segment (segment type of 0),

example int[][] p3 = {{3}, {100, 0, 1, 2}}

Meaning that this segment of size 100 will be shared (hence the zero) with process 1 and 2. These 2 process Ids will be added to the segments list of 'shared with processes' for that shared segment.

After all the segments have been created and added to a list of segments, a Process object will then be created. This Process object will be defined using a process id (which will be int[][] process {{1}, {100}} -> process[0][0]) and the list of segments that was defined earlier in the creation of the segments. When the process object is created the segments will be added to the process and will be assigned a segment number in the process, this will all be automatically, as the 'addSegments' method from the Process class which will be called in the constructor of the Process class.

After initializing the process, all the segments in this process will be assigned a memory allocated space (this memory allocated space is defined in the 'MemoryAllocation' class). This memory allocated space contains a segment from the process and the starting and end position in the physical memory. The free memory allocated space (remaining memory left) will be updated, every time a new segment is allocated. If the segment added fills up the whole memory, then the free memory allocated space will no longer be empty and will be occupied fully by the segment (the remaining memory left will be zero). All this will be done in the 'normalAllocation' method that will be called by the 'allocateProcess' method after the process is set up.

If a process already exists in the list of processes in memory, then the process's segments will need to be updated, this will be done in the 'updateSegments' method called by the 'allocateProcess' method. In the 'updateSegments' method, it will take the previous process and the new process in the parameter and will update each segments size, number (in case a segment was deleted from the process) and the position of the memory allocated space that contains the segment we would like to update. The memory allocated space of the other segments following this segment, that is being updated, will also be updated. When updating the memory allocation spaces if there is any space or gap created by the updated segment (example segment size was 100 and is now 90, there is a 10-byte gap), this gap will be added to the memory left to allow other process, we want to allocate, to be allocated without any issues of fragmentation. This will be done by calling the 'shuffle' method in the 'updateSegments' method.

The purpose of this 'shuffle' method is to implement compaction by combining all the free spaces between the memory allocated segments and adding these free spaces to the left-over memory (the last free memory allocation space) to avoid any fragmentation and to allow the processes to be allocated to the memory in a 'first fit' fashion.

Process deallocation:

A process will be deallocated by calling the 'deallocateProcess'. A 2D array input will be taken in as the parameter, the process id of the input will be retrieved by 'input[0][0]'. The process id will be used to get a Process object by calling the 'getProcess' method. After getting the process object, a loop will be used to iterate through all the segments in this process, setting the memory allocated spaces that reference these segments to an empty/ free memory allocated space.

After emptying these memory allocated spaces, they will be shuffled using the 'shuffle' method to remove the empty spaces (gaps) between the memory segments and add these gaps to the remaining memory left. Again, this is done to avoid any fragmentation and to allow the processes to be allocated in a 'first fit' fashion.

Translation look-aside buffer (TLB)

The TLB list in the 'PhysicalMemory' class consists of 'ProcessSegment' objects. These ProcessSegment objects represent a process and a segment specified at a certain segment number.

The purpose of the TLB is to get the location of a certain segment in a process. The simulation of the TLB will be done using the 'access' method. This method will take an input of a process number and a segment number from that process. If the Process and segment number do not exist in the TLB, then a TLB miss will occur and a 'ProcessSegment' object, that contains the Process number and the segment, will be created and added to the TLB. After the ‘ProcessSegment’ object is created and added to the TLB, the next time we try to access the location of the Process number and segment number, the position of this segment will be returned.

If the TLB is full, the oldest ‘ProcessSegment’ object will be removed automatically to allow space for a new ‘ProcessSegment’ object. Here first in first out algorithm is used in the implementation of the TLB.

Process class:

Note: as mentioned before, Process objects are created from 2D array inputs in the ‘setUpProcess’ private method that will be called by the ‘allocateProcess’ in the PhysicalMemory class.

A Process object is represented by a Process id and a list of Segment objects that the process contains.

The main functionality in the class is in the ‘addSegments’ method. This method is called in the constructor of the Process class, meaning that the list of Segments passed in the parameter of the constructor will be added automatically to the process.

The ‘addSegments’ method will check if a segment does previously exist, if it does not exist then the segment will be added normally, otherwise the ‘editSegments’ method will be called. In this method, if the difference between the update segment’s size and the old segment size is zero, it means that this segment is being deleted, and therefore the segments that come after this removed segment must have their segment number updated. If the difference is not zero, then the size of the new segment must be added to the size of the existing segment.

If we are updating a shared segment, then the previous processes that share the segment, will be overridden by the new processes or process that share the segment. The size of the segment will be update as stated before.

Segment class:

In this class, the properties of the segment are defined. The segment has a number, size, base, limit and type (by default it will ‘1’, meaning it can’t be shared). If the segment type is set to 0, then this segment can be shared by other processes. When it is shared by other processes then these processes are added to a list called ‘IncludedInProcesses’. A segments property is updated if it is repeated more than once in a process.

MemoryAllocation class:

This class is used as a helper class to assign memory allocated spaces in the physical memory. These memory allocations have bases, limits (starting and ending location), size and segment reference. The segment reference can be empty, which represents empty space in the physical memory, or it could be assigned a segment that has been allocated to the memory.

ProcessSegment class:

This class is used as a helper class to implement the TLB. This class consists of a process segment combination. These objects will be created, filled up (with a process and a segment) and added to the TLB, so that when we want to retrieve a certain segment from a process the location of this segment will be printed to the console.

Allocation_Test class:

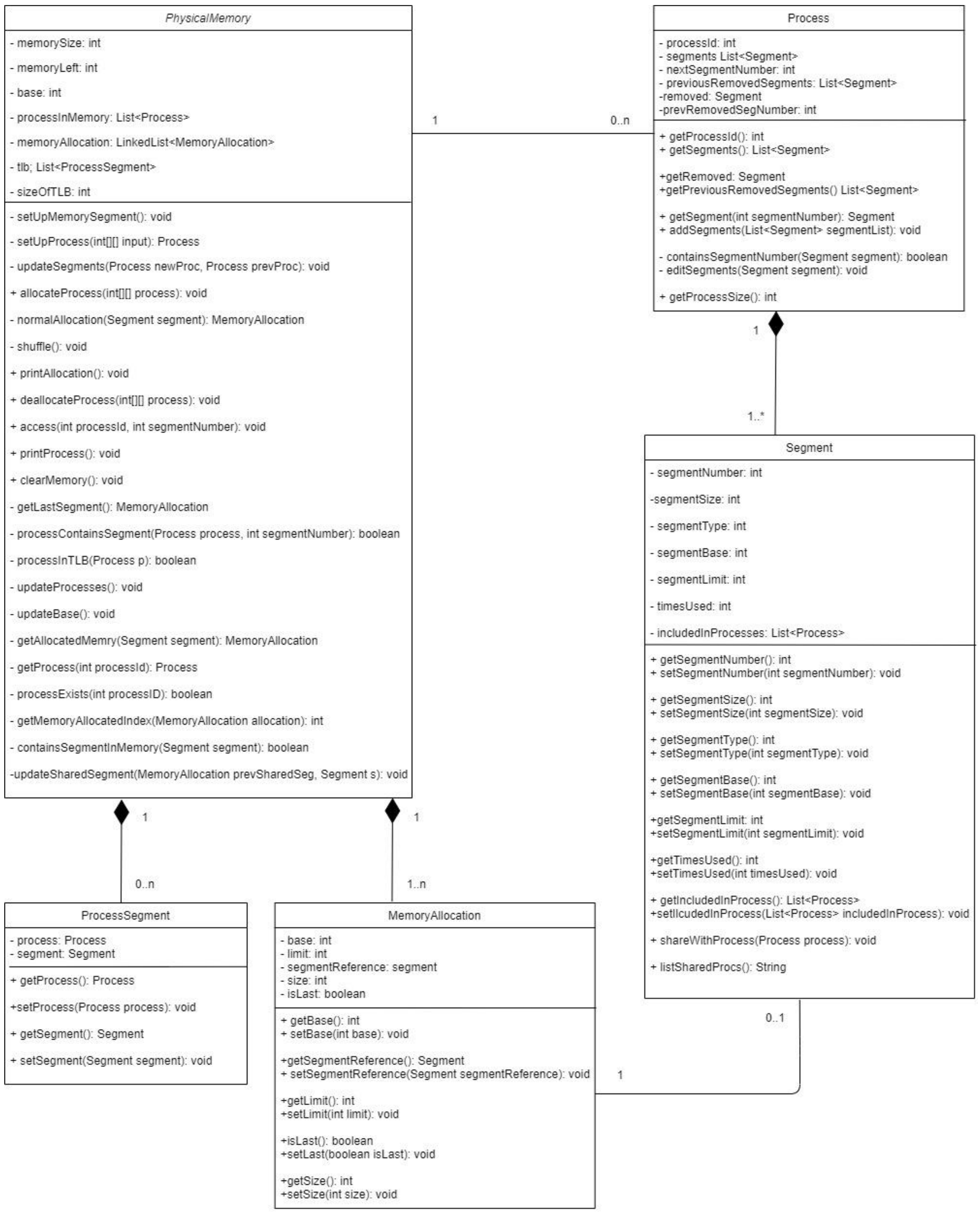
This class houses the main method to run all the examples that will be used to test the simulation of the physical memory.

<u>Exercise 1</u>: the allocation/ deallocation will be done by calling the 'allocateProcess' / 'deallocateProcess' and the 'printProcesses'   will be used to print out the processes in the memory with their segments and the location of each of these segments

<u>Exercise 2</u>: the same methods will be used to test the examples of the shared segments.

<u>Exercise 3</u>: the 'allocateProcess' / 'deallocateProcess' method will be used as normal. For the TLB example, the 'access' method will be used to print the location of a segment in a certain process to the console. For the compaction example, after allocating or deallocating a process to the memory the 'printAllocation' method will be used to print out the order of the segments that have been allocated in the memory in this format; [A100] [A20] [A30] [H350].

UML Diagram for coursework


PhysicalMemory
- memorySize: int
- memoryLeft: int
- base: int
- processInMemory: List<Process>
- memoryAllocation: LinkedList<MemoryAllocation>
- tlb; List<ProcessSegment>
- sizeOfTLB: int
- setUpMemorySegment(): void
- setUpProcess(int[][] input): Process
- updateSegments(Process newProc, Process prevProc): void
+ allocateProcess(int[][] process): void
- normalAllocation(Segment segment): MemoryAllocation
- shuffle(): void
+ printAllocation(): void
+ deallocateProcess(int[][] process): void
+ access(int processId, int segmentNumber): void
+ printProcess(): void
+ clearMemory(): void
- getLastSegment(): MemoryAllocation
- processContainsSegment(Process process, int segmentNumber): boolean
- processInTLB(Process p): boolean
- updateProcesses(): void
- updateBase(): void
- getAllocatedMemry(Segment segment): MemoryAllocation
- getProcess(int processId): Process
- processExists(int processID): boolean
- getMemoryAllocatedIndex(MemoryAllocation allocation): int
- containsSegmentInMemory(Segment segment): boolean
-updateSharedSegment(MemoryAllocation prevSharedSeg, Segment s): void
1
0..n
Process
- processId: int
- segments List<Segment>
- nextSegmentNumber: int
- previousRemovedSegments: List<Segment>
-removed: Segment
-prevRemovedSegNumber: int
+ getProcessId(): int
+ getSegments(): List<Segment>
+getRemoved: Segment
+getPreviousRemovedSegments() List<Segment>
+ getSegment(int segmentNumber): Segment
+ addSegments(List<Segment> segmentList): void
- containsSegmentNumber(Segment segment): boolean
- editSegments(Segment segment): void
+ getProcessSize(): int
1
1..*
Segment
- segmentNumber: int
-segmentSize: int
- segmentType: int
- segmentBase: int
- segmentLimit: int
- timesUsed: int
- includedInProcesses: List<Process>
+ getSegmentNumber(): int
+ setSegmentNumber(int segmentNumber): void
+ getSegmentSize(): int
+ setSegmentSize(int segmentSize): void
+ getSegmentType(): int
+ setSegmentType(int segmentType): void
+ getSegmentBase(): int
+ setSegmentBase(int segmentBase): void
+getSegmentLimit: int
+setSegmentLimit(int segmentLimit): void
+getTimesUsed(): int
+setTimesUsed(int timesUsed): void
+ getIncludedInProcess(): List<Process>
+setIlcudedInProcess(List<Process> includedInProcess): void
+ shareWithProcess(Process process): void
+ listSharedProcs(): String
1
0..n
1
1..n
ProcessSegment
- process: Process
- segment: Segment
+ getProcess(): Process
+setProcess(Process process): void
+ getSegment(): Segment
+ setSegment(Segment segment): void
MemoryAllocation
- base: int
- limit: int
- segmentReference: segment
- size: int
- isLast: boolean
+ getBase(): int
+ setBase(int base): void
+getSegmentReference(): Segment
+ setSegmentReference(Segment segmentReference): void
+getLimit(): int
+setLimit(int limit): void
+isLast(): boolean
+setLast(boolean isLast): void
+getSize(): int
+setSize(int size): void
0..1
1