

PC2 - Report

Berrospi Luis, Rios Mario

July 2021

General implementation

1. To organize the blocks of allocated and unallocated memory a singly linked list was used
2. The head of the linked list represents the begin of the memory
3. Traditional linked list operations were implemented taking into account the amount of memory available and the disponibility of blocks
4. The memory was considered as a big unallocated block of memory at the beginning

The class used to implement the linked list has the following structure:

- `char* id`
The identifier of each node
- `int base_adr`
The beginning in the memory of the node
- `int top_adr`
The end in the memory of the node
- `bool free`
Wether the node is free or not
- `struct linkedList* next`
Its next node, NULL if there were none

The following methods were implemented:

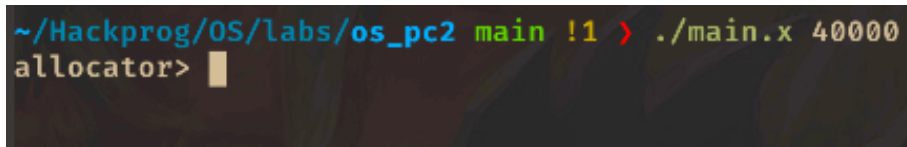
- `initMemory`
Returns a linked list head already initialized with the whole available space
- `findPrevLinkedList`
Iterates the list and returns the find's prev node

- **printMemory**
Returns all the nodes available, its corresponding bounds, its name and whether it is free
- **deallocate**
Finds the corresponding node with the provided id and sets it as free
- **allocateF (First Fit)**
Finds the first node that meets the following requirements: it's free and its size is bigger than the requested. Then splits the node and sets its correspondent bounds
- **allocateW (Worst Fit)**
Finds the free node with the biggest size (iterates all the linkedList) that can contain requested memory. Then splits and sets its correspondent bounds
- **allocateB (Best Fit)**
Finds the free node with the smallest size (iterates all the linkedList) that can contain requested memory. Then splits and sets its correspondent bounds
- **swapLinkedList**
Swaps the address of the desired nodes and its adjacent
- **compaction**
Moves all the free nodes (except the last one) after the last occupied node. It uses swap to make these movements. After all, it joins these nodes and frees the remaining ones.

Test

Allocate and Deallocate

First the program is initialized with 40000 kb



```
~/Hackprog/OS/labs/os_pc2 main !1 > ./main.x 40000
allocator> █
```

Then the next 3 requests are sent:

1. P0 1000kb First Fit
2. P1 400kb First Fit
3. P2 1400kb First Fit

The following output is produced by the STAT operation

```
allocator> RQ P0 1000 F
allocator> RQ P1 400 F
allocator> RQ P2 1400 F
allocator> STAT

Addresses [0:1000] Process P0
Addresses [1001:1401] Process P1
Addresses [1402:2802] Process P2
Addresses [2803:39999] Unused

allocator> █
```

It means that the requests were fulfilled correctly. Then deallocate process is performed with the process P1

```
allocator> RL P1
allocator> STAT

Addresses [0:1000] Process P0
Addresses [1001:1401] Unused
Addresses [1402:2802] Process P2
Addresses [2803:39999] Unused

allocator> █
```

Worst Fit request should be located at the end if its smaller enough. The following test proves it

```
allocator> RQ P3 200 W
allocator> STAT

Addresses [0:1000] Process P0
Addresses [1001:1401] Unused
Addresses [1402:2802] Process P2
Addresses [2803:3003] Process P3
Addresses [3004:39999] Unused

allocator> █
```

STAT operation shows that the request was successful. Then process P3 is deallocated and a request of 100kb with Best Fit is tested

```
allocator> RL P3
allocator> RQ P4 100 B
allocator> STAT

Addresses [0:1000] Process P0
Addresses [1001:1401] Unused
Addresses [1402:2802] Process P2
Addresses [2803:2903] Process P4
Addresses [2904:3003] Unused
Addresses [3004:39999] Unused

allocator> █
```

The STAT operations shows that the request was allocated in the smallest available space

Compaction

Following the past test. A compaction is performed

```
Addresses [0:1000] Process P0
Addresses [1001:1401] Unused
Addresses [1402:2802] Process P2
Addresses [2803:2903] Process P4
Addresses [2904:3003] Unused
Addresses [3004:39999] Unused

allocator> C
allocator> STAT

Addresses [0:1000] Process P0
Addresses [1001:2401] Process P2
Addresses [2402:2502] Process P4
Addresses [2503:39999] Unused

allocator> |
```

As expected, the free nodes were moved and compacted at the end