Memory Allocation Lab

This lab project addresses the various memory-management schemes in an operating system.

We model the memory managed in an operating system as a list of memory blocks. Each block of memory is modeled by a data structure of MEMORY_BLOCK defined below:

```
struct MEMORY_BLOCK {
    int start_address;
    int end_address;
    int segment_size;
    int process_id; //0 indicates a free block
}
```

Thus, the operating system code can set up memory blocks for the available memory and keep track of which blocks are allocated to which processes. In particular, note that for each memory block, the system tracks the starting and ending addresses, along with the size of the block and the process to which the block is currently allocated. For free blocks, the process is indicated as 0.

The system manages the memory as an array of memory blocks, stored in an array. As memory is allocated and released, the list of blocks in the memory map changes.

In this lab project, we address the various ways in which memory can be allocated (based on different schemes - Best Fit, First Fit, Next Fit, Worst (Largest) Fit). In addition, we also develop a method to manage the release of memory blocks by processes.

For the purposes of these labs we define the NULLBLOCK as [start_address: 0, end_address: 0, segment_size: 0, process_id: 0]

Implement the following five methods in a file called **memory.c**. You should include the oslabs.h file.

best_fit_allocate:

This method allocates memory according to the Best Fit scheme. The method is given the process id of the requesting process, size of the memory being requested, and the memory map. It finds the candidate memory blocks that can be allocated and chooses the one whose size is closest to the requested size. If the free block found is exactly of the same size as the requested size, the method updates the process id to allocate it and returns this memory block. If the free block found is larger than the requested size, the block is split into two pieces - the first piece allocated and the second piece becoming a free block in the memory map. Thus, the method may alter the memory map appropriately. Note that if there is no free block of memory (in the memory map) that is at least as large as the requested size, the method returns the NULLBLOCK.

The signature of the method is as follows:

struct MEMORY_BLOCK best_fit_allocate(int request_size, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt, int process_id);

A sample execution input and output:

input/output	parameter	value
input	memory_map	[start_address: 0, end_address: 1023, segment_size: 1024, process_id: 0]
input	map_cnt	1
input	request_size	10
input	process_id	32
output	memory_map	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32], [start_address: 10, end_address: 1023, segment_size: 1014, process_id: 0]
output	map_cnt	2
output	MEMORY_BLOCK	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32]

Please refer to Section 3.2 of the Modern Operating Systems book for a detailed discussion of

the Best Fit algorithm.

first_fit_allocate:

This method allocates memory according to the First Fit scheme. The method is given the process id of the requesting process, size of the memory being requested, and the memory map. It finds the first (lowest starting address) free memory block whose size is at least as large as the requested size. If the free block found is exactly of the same size as the requested size, the method updates the process id to allocate it and returns this memory block. If the free block found is larger than the requested size, the block is split into two pieces - the first piece allocated and the second piece becoming a free block in the memory map. Thus, the method may alter the memory map appropriately. Note that if there is no free block of memory (in the memory map) that is at least as large as the requested size, the method returns the NULLBLOCK.

The signature of the method is as follows:

struct MEMORY_BLOCK first_fit_allocate(int request_size, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt, int process_id);

A sample execution input and output:

input/output	parameter	value
input	memory_map	[start_address: 0, end_address: 1023, segment_size: 1024, process_id: 0]
input	map_cnt	1
input	request_size	10
input	process_id	32
output	memory_map	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32], [start_address: 10, end_address: 1023, segment_size: 1014, process_id: 0]
output	map_cnt	2
output	MEMORY_BLOCK	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32]

Please refer to Section 3.2 of the Modern Operating Systems book for a detailed discussion of the First Fit algorithm.

worst_fit_allocate:

This method allocates memory according to the Worst Fit scheme. The method is given the process id of the requesting process, size of the memory being requested, and the memory map. It finds the candidate memory blocks that can be allocated and chooses the largest among these blocks. If the free block found is exactly of the same size as the requested size, the method updates the process id to allocate it and returns this memory block. If the free block found is larger than the requested size, the block is split into two pieces - the first piece allocated and the second piece becoming a free block in the memory map. Thus, the method may alter the memory map appropriately. Note that if there is no free block of memory (in the memory map) that is at least as large as the requested size, the method returns the NULLBLOCK.

The signature of the method is as follows:

struct MEMORY_BLOCK worst_fit_allocate(int request_size, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt, int process_id);

A sample execution input and output:

input/output	parameter	value
input	memory_map	[start_address: 0, end_address: 1023, segment_size: 1024, process_id: 0]
input	map_cnt	1
input	request_size	10
input	process_id	32
output	memory_map	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32], [start_address: 10, end_address: 1023, segment_size: 1014, process_id: 0]
output	map_cnt	2
output	MEMORY_BLOCK	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32]

Please refer to Section 3.2 of the Modern Operating Systems book for a detailed discussion of the Worst Fit algorithm.

next_fit_allocate:

This method allocates memory according to the Next Fit scheme. The method is given the process id of the requesting process, size of the memory requested, the memory map, and the address of the last block allocated. It finds the first (lowest starting address) free memory block, greater than or equal to the previously allocated block address, whose size is at least as the requested size. If the free block found is exactly of the same size as the requested size, the method updates the process id to allocate it and returns this memory block. If the free block found is larger than the requested size, the block is split into two pieces - the first piece allocated and the second piece becoming a free block in the memory map. Thus, the method may alter the memory map appropriately. Note that if there is no free block of memory (in the memory map) that is at least as large as the requested size, the method returns the NULLBLOCK.

The signature of the method is as follows:

struct MEMORY_BLOCK next_fit_allocate(int request_size, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt, int process_id, int last_address);

A sample execution input and output:

input/output	parameter	value
input	memory_map	[start_address: 0, end_address: 1023, segment_size: 1024, process_id: 0]
input	map_cnt	1
input	request_size	10
input	process_id	32
input	last_address	0
output	memory_map	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32], [start_address: 10, end_address: 1023, segment_size: 1014, process_id: 0]
output	map_cnt	2
output	MEMORY_BLOCK	[start_address: 0, end_address: 9, segment_size: 10, process_id: 32]

Please refer to Section 3.2 of the Modern Operating Systems book for a detailed discussion of the Next Fit algorithm.

release_memory:

This method releases a memory block. Accordingly, it modifies the memory map passed in. Specifically, it marks the released block of memory as free and then it merges that block with adjacent free blocks if any. That is, if the memory block adjacent to the newly released block is free, the memory map is altered to reduce the number of memory blocks by one and the ending address (and the size) of the previous free block extended. Note that the method does not have any explicit return value and instead modifies the memory map passed in.

The signature of the method is as follows:

void release_memory(struct MEMORY_BLOCK freed_block, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt);

A sample execution input and output:

input/output	parameter	value
input	memory_map	[start_address: 0, end_address: 7, segment_size: 8, process_id: 12], [start_address: 8, end_address: 15, segment_size: 8, process_id: 0], [start_address: 16, end_address: 23, segment_size: 8, process_id: 13], [start_address: 24, end_address: 27, segment_size: 4, process_id: 0], [start_address: 28, end_address: 29, segment_size: 2, process_id: 11]
input	map_cnt	5
input	freed_block	[start_address: 16, end_address: 23, segment_size: 8, process_id: 13]
output	memory_map	[start_address: 0, end_address: 7, segment_size: 8, process_id: 12], [start_address: 8, end_address: 27, segment_size: 20, process_id: 0], [start_address: 28, end_address: 29, segment_size: 2, process_id: 11]
output	map_cnt	3