Can Demirel 21401521
Pegah Soltani 21500559

Introduction

In the third project of Operating Systems course, we are asked to manage a given memory chunk and allocate the chunk according to different given algorithms which are as follows:

First Fit: The allocated space is the first place that is found in the chunk and fits the size. The user should give the argument 0 for this allocation method.

Best Fit: The allocated space is the smallest available space greater than the allocation space. The user should give the argument 1 for this allocation method.

Worst Fit: The allocated space is the biggest available space possible in the chunk. The user should give the argument 2 for this allocation method.

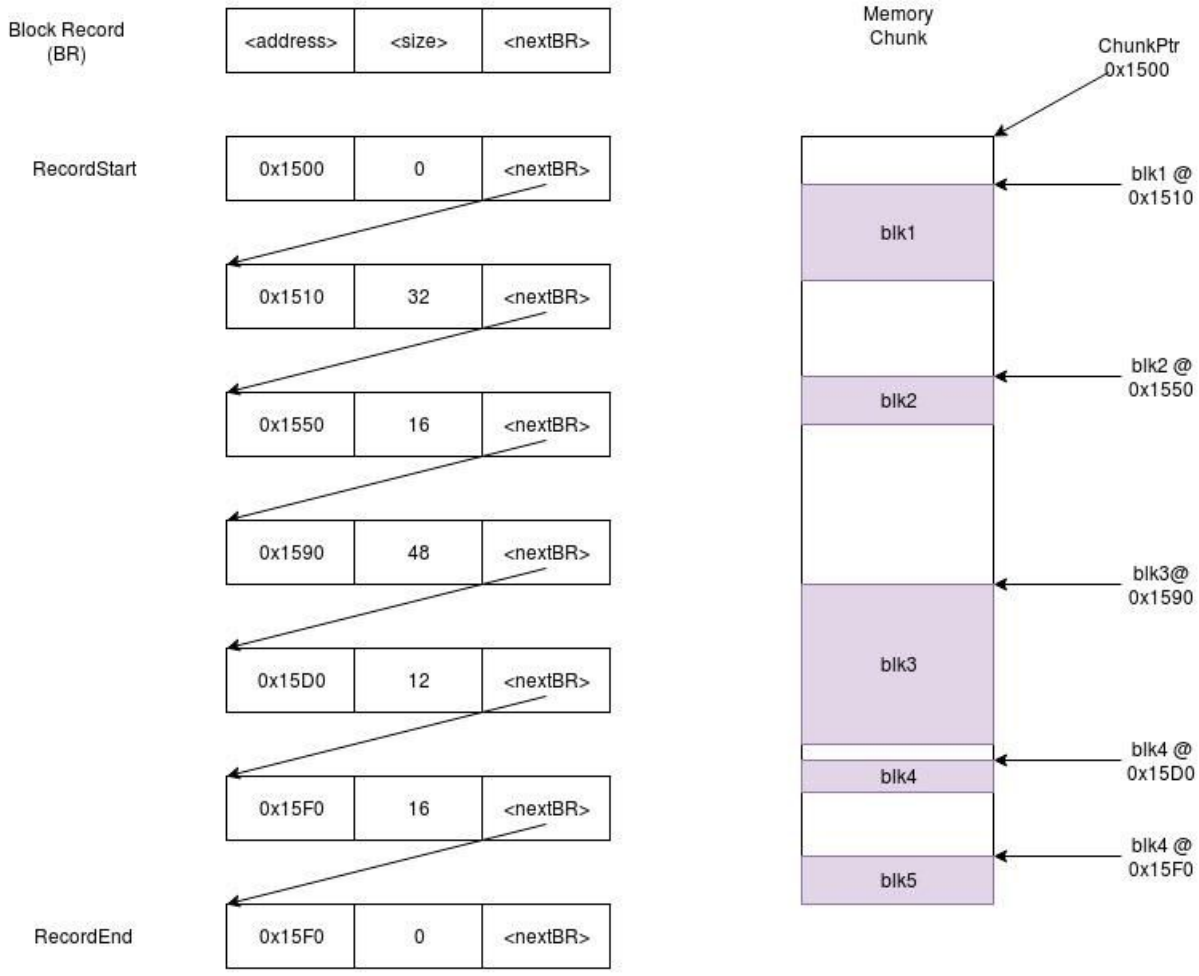
The task is to initialize a memory chunk in the given size and allocate different memory blocks with different sizes. The program is responsible for handling and managing the allocated memory blocks in the chunk according to the algorithms. The application using this program is multi-threaded so the project should handle that as well using synchronization. The user should be able to allocate, free and reallocate memory blocks.

This report will first give a brief information about the methodology used in implementing the code. Also

Methodology

In order to solve the memory management problem manually, me and my partner decided to implement a structure similar to a linked lists (called BlockRecord in our code). The nodes in the list should keep two types of information. One is the address of the beginning of allocated memory from the chunk, the other is the size of the allocated space. As a result, we will have a chunk of memory which is managed by the linked-list nodes pointing to the different places of the chunk. mem_allocate function will create a node (BlockRecord) and insert it in the linked-list. On the other hand, mem_free will remove one node from the linkedlist (frees the allocated space).

Below is a graphical representation of the described data structure.



Issues

We faced a lot of issues while designing the BlockRecords linked-list and handling memory arithmetics while traversing the BlockRecords linked-list. However, in the end we were able to do carry out memory address arithmetics by casting pointer addresses to char in one case and simply adding the integers to the pointer addresses to obtain the amount of empty

spaces (holes) in the chunk. Traversing through the linked list lead us to infinite loops many times since somehow the value of the head pointer (for traversal) would not be updated. As a solution, we made an external pointer to the end of the chunk and resolved the issue.

Experimentation & Analysis

Experiment #	Method	Size	Time
#1	0 (First Fit)	1024	Real: 0m0.002s User: 0m0.000s Sys: 0m0.000s
#2	1(Best Fit)	1024	Real: 0m0.002s User: 0m0.000s Sys: 0m0.000s
#3	2(Worst Fit)	1024	Real: 0m0.002s User: 0m0.000s Sys: 0m0.000s

Analysis

As our experiments find and as shown in full details in Appendices A, B and C for First Fit, Best Fit and Worst Fit hole search methods, First-Fit Methodology is the most time efficient algorithm because it does not need to traverse whole memory chunk to make an allocation except the case that there is no space left in the chunk. There might be processes left unallocated because of First Fit algorithm because a process can be allocated to a memory space which is much larger than its required size. Despite that, best Fit algorithm focuses on placing processes to the best possible space they can fit. This process requires a whole traversal along memory chunk therefore it is more time consuming than First Fit algorithm but it provides a more precise memory allocation. Worst Fit algorithm as its name suggest is the worst algorithm in both

aspects. It does not run as fast as First Fit algorithm also it does not provide an effective memory allocation like Best-Fit algorithm.

APPENDIX A

A 1024 kB memory chunk is created at 0x7f07453a7010 ending at 0x7f07454a7010

Looking for holes of size 524288 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f07453a7010 of size 1048576
Created BlockRecord for block 0x7f07453a7010 of size 524288.

Looking for holes of size 262144 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f0745427010 of size 524288
Created BlockRecord for block 0x7f0745427010 of size 262144.

Looking for holes of size 131072 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f0745467010 of size 262144
Created BlockRecord for block 0x7f0745467010 of size 131072.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f0745487010 of size 131072
Created BlockRecord for block 0x7f0745487010 of size 65536.

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f0745497010 of size 65536
Created BlockRecord for block 0x7f0745497010 of size 32768.

>> Printing Block Records
****
**** Block Records ****

```
************************
Block Record: Size of 524288 Bytes, at 0x7f07453a7010.
Block Record: Size of 262144 Bytes, at 0x7f0745427010.
Block Record: Size of 131072 Bytes, at 0x7f0745467010.
Block Record: Size of 65536 Bytes, at 0x7f0745487010.
Block Record: Size of 32768 Bytes, at 0x7f0745497010.
********** END **********

Deleting Block Record at 0x7f0745427010.
Memory at 0x7f0745427010 is freed up.
Deleting Block Record at 0x7f0745487010.
Memory at 0x7f0745487010 is freed up.

>> Printing Block Records
************************
****  Block Records  ****
************************
Block Record: Size of 524288 Bytes, at 0x7f07453a7010.
Block Record: Size of 131072 Bytes, at 0x7f0745467010.
Block Record: Size of 32768 Bytes, at 0x7f0745497010.
********** END **********

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f0745427010 of size 262144
Created BlockRecord for block 0x7f0745427010 of size 32768.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using FirstFit Method.
Found an available spot at 0x7f074542f010 of size 229376
Created BlockRecord for block 0x7f074542f010 of size 65536.

>> Printing Block Records
```

****

**** Block Records ****

****

Block Record: Size of 524288 Bytes, at 0x7f07453a7010.

Block Record: Size of 32768 Bytes, at 0x7f0745427010.

Block Record: Size of 65536 Bytes, at 0x7f074542f010.

Block Record: Size of 131072 Bytes, at 0x7f0745467010.

Block Record: Size of 32768 Bytes, at 0x7f0745497010.

********** END **********

APPENDIX B

A 1024 kB memory chunk is created at 0x7fea4f1d0010 ending at 0x7fea4f2d0010

Looking for holes of size 524288 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f1d0010 of size 1048576
Created BlockRecord for block 0x7fea4f1d0010 of size 524288.

Looking for holes of size 262144 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f250010 of size 524288
Created BlockRecord for block 0x7fea4f250010 of size 262144.

Looking for holes of size 131072 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f290010 of size 262144
Created BlockRecord for block 0x7fea4f290010 of size 131072.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f2b0010 of size 131072
Created BlockRecord for block 0x7fea4f2b0010 of size 65536.

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f2c0010 of size 65536
Created BlockRecord for block 0x7fea4f2c0010 of size 32768.

****

**** Block Records ****

****

Block Record: Size of 524288 Bytes, at 0x7fea4f1d0010.
Block Record: Size of 262144 Bytes, at 0x7fea4f250010.
Block Record: Size of 131072 Bytes, at 0x7fea4f290010.
Block Record: Size of 65536 Bytes, at 0x7fea4f2b0010.
Block Record: Size of 32768 Bytes, at 0x7fea4f2c0010.
********** END **********

Deleting Block Record at 0x7fea4f250010.
Memory at 0x7fea4f250010 is freed up.

Deleting Block Record at 0x7fea4f2b0010.
Memory at 0x7fea4f2b0010 is freed up.

****
**** Block Records ****
****
Block Record: Size of 524288 Bytes, at 0x7fea4f1d0010.
Block Record: Size of 131072 Bytes, at 0x7fea4f290010.
Block Record: Size of 32768 Bytes, at 0x7fea4f2c0010.
********** END **********

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f2c8010 of size 32768
Created BlockRecord for block 0x7fea4f2c8010 of size 32768.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using BestFit Method.
Found an available spot at 0x7fea4f2b0010 of size 0
Created BlockRecord for block 0x7fea4f2b0010 of size 65536.

****
**** Block Records ****

****

Block Record: Size of 524288 Bytes, at 0x7fea4f1d0010.

Block Record: Size of 131072 Bytes, at 0x7fea4f290010.

Block Record: Size of 65536 Bytes, at 0x7fea4f2b0010.

Block Record: Size of 32768 Bytes, at 0x7fea4f2c0010.

Block Record: Size of 32768 Bytes, at 0x7fea4f2c8010.

********** END **********

APPENDIX C

A 1024 kB memory chunk is created at 0x7f116a918010 ending at 0x7f116aa18010

Looking for holes of size 524288 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a918010 of size 1048576
Created BlockRecord for block 0x7f116a918010 of size 524288.

Looking for holes of size 262144 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a998010 of size 524288
Created BlockRecord for block 0x7f116a998010 of size 262144.

Looking for holes of size 131072 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a9d8010 of size 262144
Created BlockRecord for block 0x7f116a9d8010 of size 131072.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a9f8010 of size 131072
Created BlockRecord for block 0x7f116a9f8010 of size 65536.

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116aa08010 of size 65536
Created BlockRecord for block 0x7f116aa08010 of size 32768.

>> Printing Block Records
****
**** Block Records ****

```
************************
Block Record: Size of 524288 Bytes, at 0x7f116a918010.
Block Record: Size of 262144 Bytes, at 0x7f116a998010.
Block Record: Size of 131072 Bytes, at 0x7f116a9d8010.
Block Record: Size of 65536 Bytes, at 0x7f116a9f8010.
Block Record: Size of 32768 Bytes, at 0x7f116aa08010.
********** END **********

Deleting Block Record at 0x7f116a998010.
Memory at 0x7f116a998010 is freed up.

Deleting Block Record at 0x7f116a9f8010.
Memory at 0x7f116a9f8010 is freed up.

>> Printing Block Records
************************
****  Block Records  ****
************************
Block Record: Size of 524288 Bytes, at 0x7f116a918010.
Block Record: Size of 131072 Bytes, at 0x7f116a9d8010.
Block Record: Size of 32768 Bytes, at 0x7f116aa08010.
********** END **********

Looking for holes of size 32768 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a998010 of size 32768
Created BlockRecord for block 0x7f116a998010 of size 32768.

Looking for holes of size 65536 and greater.
Looking for holes in Chunk using WorstFit Method.
Found an available spot at 0x7f116a9a0010 of size 32768
Created BlockRecord for block 0x7f116a9a0010 of size 65536.
```

>> Printing Block Records

****

**** Block Records ****

****

Block Record: Size of 524288 Bytes, at 0x7f116a918010.

Block Record: Size of 32768 Bytes, at 0x7f116a998010.

Block Record: Size of 65536 Bytes, at 0x7f116a9a0010.

Block Record: Size of 131072 Bytes, at 0x7f116a9d8010.

Block Record: Size of 32768 Bytes, at 0x7f116aa08010.

********** END **********