CS342

OPERATING SYSTEMS

2019-2020 FALL

PROJECT 3 REPORT

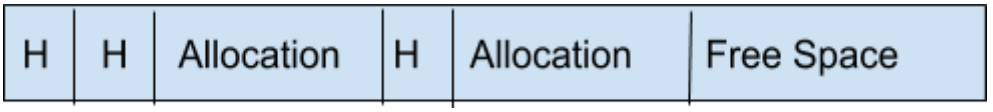
Nergiz Ünal - 21301027
Elif Kevser Arslan - 21400763

CODE EXPLANATION

- For this project, to allocate requested size of block, we added a header at the very beginning of the block itself. This header holds the information about the size of the block and the address of the next block. More clearly:

  struct header
  {
  int size; → 8 byte
  struct header*, next; → 8 byte
  }

- In total, header's size is 16 byte. Since we need to hold 16 byte for each header, we need to allocate extra 16 additional byte for every requested block.
- There is also one general header at the beginning of whole memory chunk for indicating the situation of it.



In app.c, the program allocates 6 blocks with size of (chunksize in kb) *16 byte, for every allocation method these blocks allocated consecutively. And then the program frees second, third and fifth blocks.

|----------|----------|----------|----------|----------|----------|------------------|
Alloc Free Free Alloc Free Alloc Free

And then the program creates 6 threads for allocating memory with size of (chunksize in kb) *8, this scenario gives different results for every different allocation algorithms. You can see how these 3 algorithms allocate memory by outputs given above.

EXPERIMENTATION ON DIFFERENT ALGORITHMS

FIRST FIT:

BEST FIT

```
chunkstart=560387504000, chunkend=56038750c000, chunksize=32768 bytes
---starting testing chunk
---chunk test ended - success
Implementation alloc: from 0x560387504000 to: 0x560387504010 size: 16
Alloc from 0x560387504000 to: 0x560387504210, block + header size: 528
Free  from 0x560387504220 to: 0x560387504640,                size: 1056
Alloc from 0x560387504630 to: 0x560387504840, block + header size: 528
Free  from 0x560387504850 to: 0x560387504a60,                size: 528
Alloc from 0x560387504a50 to: 0x560387504c60, block + header size: 528
Alloc from 0x560387504c60 to: 0x560387504d70, block + header size: 272
Alloc from 0x560387504d70 to: 0x560387504e80, block + header size: 272
Alloc from 0x560387504e80 to: 0x560387504f90, block + header size: 272
Alloc from 0x560387504f90 to: 0x5603875050a0, block + header size: 272
Alloc from 0x5603875050a0 to: 0x5603875051b0, block + header size: 272
Alloc from 0x5603875051b0 to: 0x5603875052c0, block + header size: 272
Free  from 0x5603875052d0 to: 0x56038750c000,                size: 27952
```

WORS FIT

```
chunkstart=56163306c000, chunkend=561633074000, chunksize=32768 bytes
---starting testing chunk
---chunk test ended - success
Implementation alloc: from 0x56163306c000 to: 0x56163306c010 size: 16
Alloc from 0x56163306c000 to: 0x56163306c210, block + header size: 528
Free  from 0x56163306c220 to: 0x56163306c640,                size: 1056
Alloc from 0x56163306c630 to: 0x56163306c840, block + header size: 528
Free  from 0x56163306c850 to: 0x56163306ca60,                size: 528
Alloc from 0x56163306ca50 to: 0x56163306cc60, block + header size: 528
Alloc from 0x56163306cc60 to: 0x56163306cd70, block + header size: 272
Alloc from 0x56163306cd70 to: 0x56163306ce80, block + header size: 272
Alloc from 0x56163306ce80 to: 0x56163306cf90, block + header size: 272
Alloc from 0x56163306cf90 to: 0x56163306d0a0, block + header size: 272
Alloc from 0x56163306d0a0 to: 0x56163306d1b0, block + header size: 272
Alloc from 0x56163306d1b0 to: 0x56163306d2c0, block + header size: 272
Free  from 0x56163306d2d0 to: 0x561633074000,                size: 27952
```

First Fit
Best Fit
Worst Fit

time/ #thread	200	150	100	50	30	10
real	0m0.039s	0m0.033s	0m0.024s	0m0.011s	0m0.006s	0m0.005s
user	0m0.005s	0m0.005s	0m0.004s	0m0.000s	0m0.001s	0m0.004s
sys	0m0.028s	0m0.014s	0m0.014s	0m0.010s	0m0.006s	0m0.000s
real	0m0.049s	0m0.039s	0m0.026s	0m0.009s	0m0.008s	0m0.004s
user	0m0.007s	0m0.003s	0m0.004s	0m0.003s	0m0.006s	0m0.000s
sys	0m0.026s	0m0.021s	0m0.011s	0m0.009s	0m0.000s	0m0.003s
real	0m0.039s	0m0.037s	0m0.038s	0m0.008s	0m0.006s	0m0.008s
user	0m0.008s	0m0.005s	0m0.000s	0m0.000s	0m0.000s	0m0.000s
sys	0m0.018s	0m0.017s	0m0.017s	0m0.009s	0m0.007s	0m0.005s

INTERPRETATION OF RESULTS

First fit algorithm is the fastest algorithm among them because it does searching in less number of time for free space.
Although best fit algorithm has advantage on memory utilization, it has disadvantage on timing because it searches more.
Last but not the least, worst fit aims reducing the ratio of small gaps, and due to this, it has similar timing results with first fit.