San José State University
Department of Computer Engineering

CMPE 142

Operating Systems

Section 1

Spring 2021
Instructor: Ron Mak

Assignment #5

Assigned: Friday, March 5
Due: Friday, March 12 at 11:30 AM
Team assignment, 105 points max

Memory allocation algorithms

In this assignment, your team will simulate three popular memory allocation algorithms, **first fit**, **next fit**, and **best fit**. You will also have another opportunity to practice multiprocessing and named pipes. You may write your programs in either C or C++.

Run the simulation at least once for each of the three algorithms.

Memory manager and requester processes

Create a single **memory manager process** that keeps track of around 3000 memory blocks. Also create several **requester processes** that each asks the memory manager to allocate and deallocate memory segments of various sizes. Give each requester process a single-character name, such as A, B, C, etc. Send each request to the memory manager via a named pipe (FIFO).

Each **allocation request** should consist of:

- A unique **request id**. If you named a requester process A, then you can use request ids like A0001, A0002, A0003, etc.
- The **name** of the requester process (not necessary if it's the first character of the request id).
- The requested **block size**.

For each request block size, randomly choose one of the following **prime number sizes**: 3, 5, 7, 11, 13, 17, 19, 23, or 29. (Put the sizes in a static array and then randomly choose one of the array elements each time.)

After making each request, the requester process must **wait** for the memory manager to fulfill the request. (Wait on a named semaphore dedicated to that process.)

Each requester process needs to keep track of the requests that it made, so that it can later make a **deallocation request** that it randomly chooses from among its remaining previous allocations. Each deallocation request should consist of:

- The **request id** of the previous allocation.
- The **name** of the requester process (if necessary).

The memory manager process can immediately grant deallocation requests, so there is no need for the requester process to wait after each deallocation request.

Each time through a loop, each requester process should make either an allocation or a deallocation request. About 60% of the times, it should be an allocation request. The rest of the times, it should be a randomly chosen deallocation request, if there are any previous allocations not yet deallocated. (Generate a random number 0 – 9. Make an allocation request if the generated number is 0 – 5, otherwise make a deallocation request.)

So that everything doesn't happen too quickly to observe, introduce a small delay (a short random sleep period) between requests.

The memory manager process must keep track of all the allocated and free segments in a **memory map**. Whenever it reads an allocation request from the named pipe, it can either immediately fulfill the request immediately using the allocation algorithm, or it must put the request on hold until a sufficiently large segment is freed from a later deallocation request. Of course, allocations in the memory map must not overlap.

When the memory manager receives a deallocation request, it should immediately service the request. Adjacent free segments should immediately merge into one larger free segment.

After granting a deallocation request, the memory manager should look at the allocation requests it has on hold to see if it can fulfill any of them. Allocation requests on hold should have priority over new incoming allocation requests.

After the memory manager process fulfills an allocation request, it must **signal** the requester process that made the request, since the requester process was waiting for its request to be fulfilled.

Since there will be more allocation requests than deallocation requests, the memory will eventually "fill up" and the memory manager can no longer fulfill any allocation requests and all the requester processes are hung waiting for their last allocation request to be fulfilled. At that point, the simulation ends.

Live memory map display

To allow you to monitor the progress of the memory manager process, the process should maintain a **live memory map display** that shows which requester process owns each memory block. For example:

```
FFFFFFF.........EE...................................AAAAAAAAAAAAAAA..............AAAAAAA........DDD
DDDDDBBBDDDD.BBBBBBBBBBBBBBBBBBBBBBC.EEEEEEEEEEEEEEEE........................................FFFFFFF
FFFFFFFFFFFFF...............EEEEEEEEEEEEEEE......EEEEEEEEEE.............BBBBBB...........AAAAAAAAAAA
AAAAAAAEE..........................CCCCCCCCCCCCCCCCC..DDDEEEEEEEEEEEE................AAAAAAAAAAAAA..
...................EEEEEEEEEEEEEEEEEEEEEE............................................EEEEEEEEEEEEEEA
.CCCCCCCCCCCCCEEEE.............DDDDDDDDDDDDDDDDDDDBBB....AAA...AAAAAAAAAAA.....BB.........EEECCCCCEE
B...............AAAAAAAAAAAAAAAAAAA.............AAAAAAAAAAAAAAAAAAABFFFFFFFFFFFFAAAAAAAAAAAAAAAAAAAA
AA...........................FFFF..............DDDDDD....................EE.........................
..................FFFFFFDDFFFFFFFFFFFFF.....DDDDDDDDDDDBBBBBBBBBBBB.......FFFFDDDDDDDDDDDDDFF....FFF
FFFFFFFFFFFFFFFFF......EEEEEEEEEEEEEEEEE.......................CCCCCCCCCCCCCCCCAAAAAAAAABB.BBBBDDDDD
.BBBBEEEEEEEEEEEEEEEEE....................CCCCCCCCCC.....................BBBBBBB..............CCCCCC
CCCCCCCEEEEEEE...................FF....................FFFFFFFFFFFF..........FBBBBBBBBBBBBBBBBBBBBBA
AAACCCCCCBBBBBBBBBBBB.....BBBBBBBBBBBBBBB...................AAAAAAAAAAAAAAAAA.......................
.....FFFFFFFFFFFFFFFFFFDDDDDDDDDDDCCCCCCCCCCCC.................CCCCCCCDDDDDDDDDDCCCCC........DDEEEE.
........................................FFFFFFFFFFFF.....................FFFFFFFBBBBBBBBBBBBBBBBBBBB
BB...EEEEEEEEEEE.AAAAAAAAAAAAAAFFDDDDDDDDDDFFFFFFFBBBBBBBBBBBBBBBBBEEEE.............................
.............................FBBBBBBBBBBBBBBBBBBBBBBBD....BBBBBBBBBBBBBBBBBBBBBDDD..........AAAAAAAA
ACCCCC..CCCCCCCCCCCCCAAAA..........AACCCCCAAAAAAAAAAAAAAFFFFFFFFFFFEEEEEEEEEEEEE....................
.......AAAAAAAAAAAAA............BBBBBB..DDD............................DDDDDDDDD....................
...............BBBBBBBBBBB....CCCCCCCC....FFFFFFFFAAAAAAAAAAAAAAAAAAAAAA.....DDDDDDDDDD.............
...EEEEEEEEFFFFFFFFFFFFFFFFFFFFCCCCAAAACCCCCCCC................CCCC.................................
.......EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEFFFFFFFF................................................
....................F.......FFFFFFFFFFFEEEEEEEEEE.....AA............................................
...................................CCCCCCCCCCCC..........CFFFFFFFFFFFF........AAAAAAAA.........CC...
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBBBBBB..DDDDDDDDDDDDCCC..................CCCCCCCCCCCC........
....EEE.....................................DDDDDDDDCCCCCCCCCCCCCCCCDDDDFFFFFFFFFFFFFF...DDDD.......
.........BBBBBBBBBB....EEBBEEEEEEEEAAAAAAAAAAAA.....................DCCDDDDDDDDDD...................
.........FFFFFFFAAAAA.AAAAAAAAAAAAAA..FFFFFFFFFFAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBAAAAAAAAA............
.CC.....AAAAAAAAAAAAAAAAADEEEEEEEEEEEEEEEEE.FFFFF................BBBBBBBBBBBBEBB....................
CCCCCCCCCCCCBBBB.BBBBBBBBBBBBBBBBBBBB.............FFFFFFD...............DEEEEEEEE...................
```

Even though your “memory” is one dimensional, you would break it up into convenient rows for the display. We see in this display that there are allocations for requester processes named A, B, C, D, E, and F. The memory manager process should update the map dynamically after each fulfilled allocation request and deallocation request.

The `ncurses` package

To create and maintain the text-based memory map display, use the **`ncurses`** package. This package may already be installed in your Ubuntu or MacOS platform. To find out, enter the following command in a terminal window (can you decipher this bash command?):

```
find / -name libncurses\.\* 2> /dev/null
```

The command’s output, if any, should tell where are your **`libncurses`** library files:

```
/usr/local/Cellar/ncurses/6.2/lib/libncurses.6.dylib
/usr/local/Cellar/ncurses/6.2/lib/libncurses.dylib
/usr/local/Cellar/ncurses/6.2/lib/libncurses.a
```

The output should come out quickly to show you the location of the library files. If the command appears to hang or terminates with no output, you don't already have **`ncurses`** installed.

To install the latest version of **`ncurses`** in Ubuntu:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

To install the latest version of **`ncurses`** in MacOS:

```
brew install ncurses
```

To compile and run the example **`DotsAndStars.c`** program:

```
cc -o das DotsAndStars.c -lncurses
./das
```

Performance statistics

During the simulation of each memory allocation algorithm, keep track of performance statistics:

- How many allocation requests in total were fulfilled?
- How many allocation requests fulfilled immediately, and how many were first put on hold?
- On average, how many requests (allocation or deallocation) were serviced before a request on hold could be fulfilled?
- At the end of each simulation, what percentage of the memory blocks were filled?

After you've performed all the simulations, what conclusions can you make about the relative performances of the algorithms?

You may want to run the simulations multiple times for each algorithm (with a different random number seed each time) and compute averages. If it's necessary to enhance the differences among the algorithms, you can adjust any of the parameters (more prime sizes, different percentage of allocation requests, etc.), and note that in your program comments.

What to submit

Submit the following to Canvas, **Assignment #5: Memory Allocation Algorithms**.

- Source files (either C or C++) of your memory manager and requester programs.
- A screen shot of the memory map at the conclusion of each simulation. Label each screen shot with the name of the algorithm (first fit, next fit, or best fit).

Rubric

Your submission will be graded according to these criteria:

Criteria	Max points
Memory manager process	**75**
• Management of allocation requests	• 10
• Management of deallocation requests	• 10
• Use of the named pipe and named semaphores	• 10
• First fit algorithm	
○ screen shot	○ 5
○ simulation statistics	○ 10
• Next fit algorithm	
○ screen shot	○ 5
○ simulation statistics	○ 10
• Best fit algorithm	
○ screen shot	○ 5
○ simulation statistics	○ 10
Requester process	**30**
• Allocation requests	• 10
• Deallocation requests	• 10
• Use of the named pipe and named semaphores	• 10