Operating System Labs July-Dec-2017
Assignment 6

Exercise 1:

In *variable-partition* scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a *hole*. Eventually, as you will see, memory contains a set of holes of various sizes. When a process arrives and needs memory, system search for a hole greater than or equal to the demanded memory size. If the hole is too large, it will be splited into two parts. One part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its memory which is then placed back in the set of holes. This assignment is about the implementation of the following two strategies:

Best fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
Worst fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole.

Write a program to implement above two strategies and the program must print a nicely formatted readable summary of the memory allocation activity for each strategy at the time of each request. in particular your program must report the address of each memory allocation/free request, you must also print a summary consisting of the total number of bytes free the number of contiguous blocks free and the average size of these blocks (Hint: just divide the number of free bytes by the number of free blocks).

INPUT

The input to this program is simple: the first line contains an integer giving the size of real memory in kilobytes. Following this are pairs of integers describing memory requests. The first integer in the pair is a request id, the second is the amount(in kilobytes) of memory requested If that amount is zero, then the request is to free memory previously requested under the same request id. No request id will request additional memory (i.e a request id N for S kilobytes cannot be followed later by a request id N for any non-zero kilobytes)

EXAMPLE

As a short example, consider the following input:

256

1 64 2 64 3 32 4 16 1 0 3 0 5 32 2 0

This should produce output similar to the following:

MEMORY SIZE : 256K

REQUEST 1 : 64K

Best Fit : Allocated at address 0

192 free, 1 block(s), average size = 192K

Worst Fit : Allocated at address 0

192 free, 1 block(s), average size =192K

REQUEST 2 : 64K

Best Fit : Allocated at address 64K

128 free, 1 block(s), average size = 128K

Worst Fit : Allocated at address 64K

128 free, 1 block(s), average size =128K

REQUEST 3 : 32K

Best Fit : Allocated at address 128K

96 free, 1 block(s), average size = 96K

Worst Fit : Allocated at address 128K

96 free, 1 block(s), average size =96K

REQUEST 4 : 16K

Best Fit : Allocated at address 160K

80K free, 1 block(s), average size = 80K

Worst Fit : Allocated at address 160K

80K free, 1 block(s), average size = 80K

FREE REQUEST 1 : (64K)

Best Fit : Freed at address 0

144K free, 2 block(s), average size = 72K

Worst Fit : Freed at address 0K

144K free, 2 block(s), average size =72K

FREE REQUEST 3 (32K)

Best Fit : Freed at address 128K

176K free, 3 block(s), average size = 59K

Worst Fit : Freed at address 128K

176K free, 3 block(s), average size =59K

REQUEST 5 : 32K

Best Fit : Allocated at address 128K

144 free, 1 block(s), average size = 72K

Worst Fit : Allocated at address 176K

144 free, 3 block(s), average size =48K

FREE REQUEST 2 (64K)

Best Fit : Freed at address 64K

208K free, 3 block(s), average size = 69K (Approx.)

Worst Fit : Freed at address 64K

208K free, 3 block(s), average size = 69K (Approx.)