Summary:

The driver program summarizes the performance of your allocator by computing a performance index , P, which is a weighted sum of the space utilization and throughput P=wU+ (1−w) min■ 1,T Tlibc■ where Uis your space utilization, Tis your throughput, and Tlibcis the estimated throughput of libc malloc on your system on the default traces.1The performance index favors space utilization over throughput, with a default of w= 0.6. Using this as a starting place, modify these functions (and possibly define other private static functions), so that they obey the following semantics: 1•mminit: Before calling mmmalloc mm realloc ormmfree , the application program (i.e., the trace-driven driver program that you will use to evaluate your implementation) calls mminit to perform any necessary initializations, such as allocating the initial heap area. CS 230, Fall 2022 Malloc Lab: Writing a Dynamic Storage Allocator Assigned: Wednesday Nov 16, Due: Tuesday Dec 6, 11:59PM 1 Introduction In this lab you will be writing a dynamic storage allocator for C programs, i.e., your own version of the malloc ,free andrealloc routines. Two performance metrics will be used to evaluate your solution: –Space utilization : The peak ratio between the aggregate amount of memory used by the driver (i.e., allocated via mmmalloc ormmrealloc but not yet freed via mmfree ) and the size of the heap used by your allocator. int mm_init(void); void *mm_malloc(size_t size); void mm_free(void *ptr); void *mm_realloc(void *ptr, size_t size); Themm.c file we have given you implements the simplest but still functionally correct malloc package that we could think of. You can invoke the following functions in memlib.c : •void *memsbrk(int incr) : Expands the heap by incr bytes, where incr is a positive non-zero integer and returns a generic pointer to the first byte of the newly allocated heap area. Notice that the address of the new block might be the same as the old block, or it might be different, depending on your imple- mentation, the amount of internal fragmentation in the old block, and the size of the realloc request. –ifptr is NULL, the call is equivalent to mmmalloc(size) ; –ifsize is equal to zero, the call is equivalent to mmfree(ptr) ; –ifptr is not NULL, it must have been returned by an earlier call to mmmalloc ormmrealloc . For example, if the old block is 8 bytes and the new block is 12 bytes, then the first 8 bytes of the new block are identical to the first 8 bytes of the old block and the last 4 bytes are uninitialized. The driver mdriver.c accepts the following command line arguments: 3•-t <tracedir> : Look for the default trace files in directory tracedir instead of the default directory defined in config.h . Since each metric will contribute at most wand 1−wto the performance index, respectively, you should not go to extremes to optimize either the memory utilization or the throughput only. • For consistency with the libc malloc package, which returns blocks aligned on 8-byte boundaries, your allocator must always return pointers that are aligned to 8-byte boundaries. Each trace file contains a sequence of allocate, reallocate, and free directions that instruct the driver to call your mmmalloc ,mmrealloc , and mmfree routines in some sequence. Observing that both memory and CPU cycles are expensive system resources, we adopt this formula to encourage balanced optimization of both memory utilization and throughput. Since the libc malloc always returns payload pointers that are aligned to 8 bytes, your malloc implementation should do likewise and always return 8-byte aligned pointers.

Extracted Keywords:

compound
application
allocated
–Compile
Tuesday
mentation
sophisticated
corresponding
throughput
ormmfree
initialization
addition

different
fragmentation
standard
trace-driven
uninitialized
equivalent
•Performance
utilization
simplest
constant
memory-management
requirement
contiguous
expensive
weighted
Wednesday
Program
The-voption
implementation
•Correctness
preprocessor
helpful
allocator
profiler
complexity
mmmalloc ormmrealloc
positive non-zero
untyped
starting
stand-alone
balanced optimization
textbook
manipulation
mm_realloc
attention
Instructions
departure
information
additional diagnostic information
performance
complete documentation
sequence
reallocate
tracedir
argument
•mmmalloc
through-
detailed
mm_malloc
Dynamic
Allocator
Everything
directory tracedir
internal fragmentation
tracefile
correctness
mmrealloc
solution
performance breakdown
address

Themmrealloc
pagesize
Consistency
heapsize
difficult
Evaluation
arithmetic
error-prone
debugger
identical
consistency
Tlibcis
development
andrealloc
Trace-driven
possible
efficient
aggregate
Otherwise
allocate
Themmmalloc
Introduction
minimum
following
consistent
necessary
function
memsbrk
everything
mmmalloc
debugging
instructor
•mmrealloc
Support
implicit
particular tracefile
directory