TLSF: a New Dynamic Memory Allocator for Real-Time Systems

M. Masmano, I. Ripoll, A. Crespo, and J. Real
Universidad Politécnica de Valencia, Spain

2004

Motivation

- Dynamic storage allocation (DSA): well studied and analysed issue for most application types
- Most DSA algorithms: good average response times, good overall performance
- In real-time scenarios rarely used: worst-case response time too high

"Developers of real-time systems avoid the use of dynamic memory management because they fear that the worst-case execution time of dynamic memory allocation routines is not bounded or is bounded with a too important bound"

(I. Puaut, 2002)

Real-Time Requirements for DSA
timing constraints

Real-time systems: schedulability analysis

▷ determine the worst-case execution time

▷ application schedulable with it’s timing constraints?

Therefore:

- Bounded response time
- Fast response time
- Memory requests need to be always satisfied

Fragmentation
memory constraints

Real-time systems: run for large periods of time
$\longrightarrow$ memory fragmentation problem

Memory exhaustion

- Application requires more memory than available
- DSA algorithm is unable to reuse memory that **is** free
    - Internal fragmentation (metadata and alignment, e.g. 10 byte memory + 8 byte header + 4/8 byte alignment)
    - External fragmentation (many small pieces)

DSA Algorithms

- *Sequential Fit* : single or double linked list (First-Fit, Next-Fit, Best-Fit)
- *Segregated Free Lists* : array of lists with blocks of free memory of the same size (Douglas Lea DSA)
- *Buddy Systems* : efficient split and merge operations (Binary Buddies, Fibonacci Buddies)
  $\longrightarrow$ good timing behaviour, large fragmentation
- *Indexed Fit* : balanced tree or Cartesian tree to index the free memory blocks (Stephenson's "Fast-Fit" allocator)
- *Bitmap Fit* : a bitmap marks which blocks are busy or free (Half-Fit algorithm)
  data structures are small (32 bit) $\rightarrow$ less cache misses

DSA Operational Model

DSA algorithm

- keeps track of which blocks are in use and which are free
- must provide at least two operations (malloc / free)

Typical management of free memory blocks

- Initially a single, large block of free memory
- First allocation requests: take blocks from the initial pool
- A previously allocated block is released: merge with other free block if possible
- New allocation requests: from free blocks or from the pool

DSA Operational Model

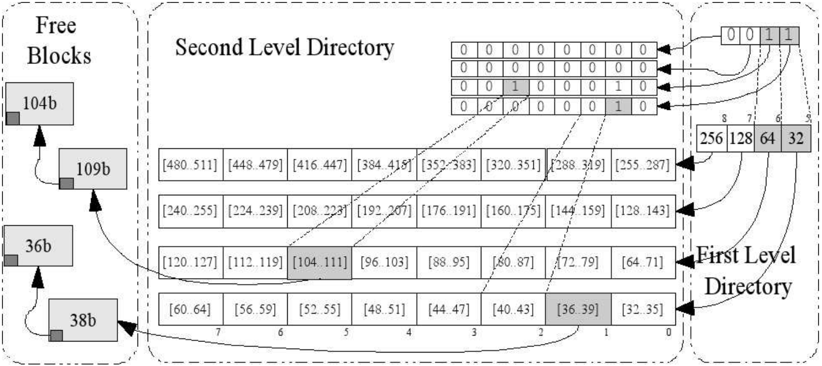
Basic operations to manage free blocks

- Insert a free block (malloc/free)
- Search for a free block of a given size or larger (malloc)
- Search for a block adjacent to another (free)
- Remove a free block (malloc/free)
- Split and merge

Two-Level Segregated Fit - Design

- Targets embedded real-time systems (trusted environment, small amount of physical memory available, no MMU)
- Immediate coalescing
- Splitting threshold (16 byte)
- Good-fit strategy
- No reallocation
- Same strategy for all block sizes
- Memory is not cleaned-up

Two-Level Segregated Fit - Example 1



memory used to manage blocks

- maximum pool of 4 GB (FLI=32, SLI=5) → 3624 bytes
- maximum pool of 32 MB (FLI=25, SLI=5) → 2856 bytes

First Level Index (FLI) / Second Level Index (SLI)

- Array of lists, each holding free blocks within a size class
- Each array of lists has an associated bitmap
- First-level: divides free blocks in classes (16, 32, 64, ...)
- Second-level: sub-divides each first-level class linearly

$$\text{size} = 460_d = \overset{15\,14\,13\,12\,11\,10\,9\,\mathbf{8}}{0\,0\,0\,0\,0\,0\,0\,1}\ \underbrace{\overset{7\,6\,5\,4}{1100}}_{s=12}\ \overset{3\,2\,1\,0}{1100_b} \qquad f=8$$

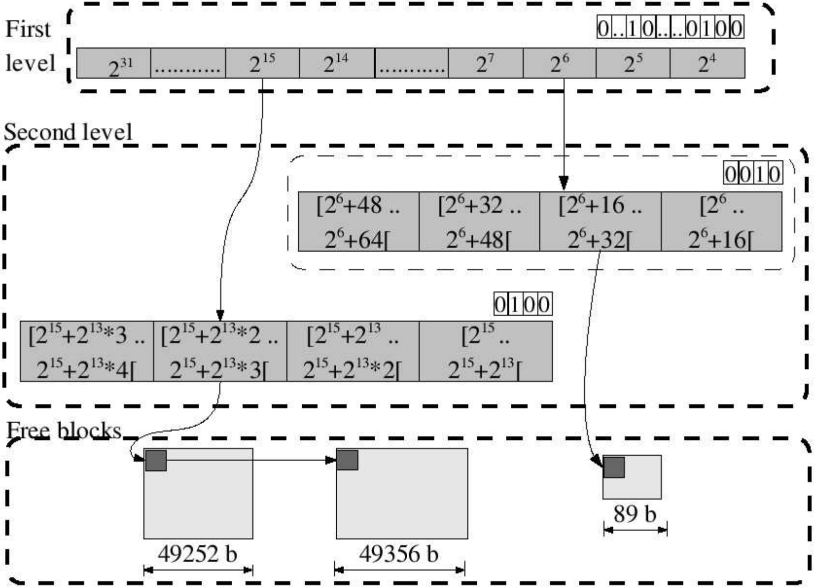
SLI = 4 $\Rightarrow$ bits 4-7

SLI = 4 $\Rightarrow$ 16 sub classes

class 8 : 256 ... 512

12^{th} sub class : $256 + 12 * (256/16) = 448$ ... 464

Two-Level Segregated Fit - Example 2



TLSF Data Structures

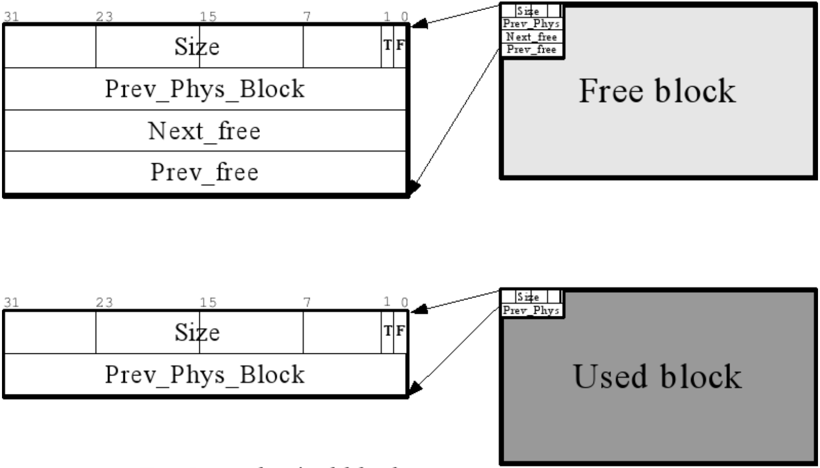
```
typedef struct TLSF_struct {
    // the TLSF's structure signature
    u32_t tlsf_signature;

    // the first-level bitmap
    // This array should have a size of REAL_FLI bits
    u32_t fl_bitmap;

    // the second-level bitmap
    u32_t sl_bitmap[REAL_FLI];

    bhdr_t *matrix[REAL_FLI][MAX_SLI];
} tlsf_t;
```

TLSF Block Header

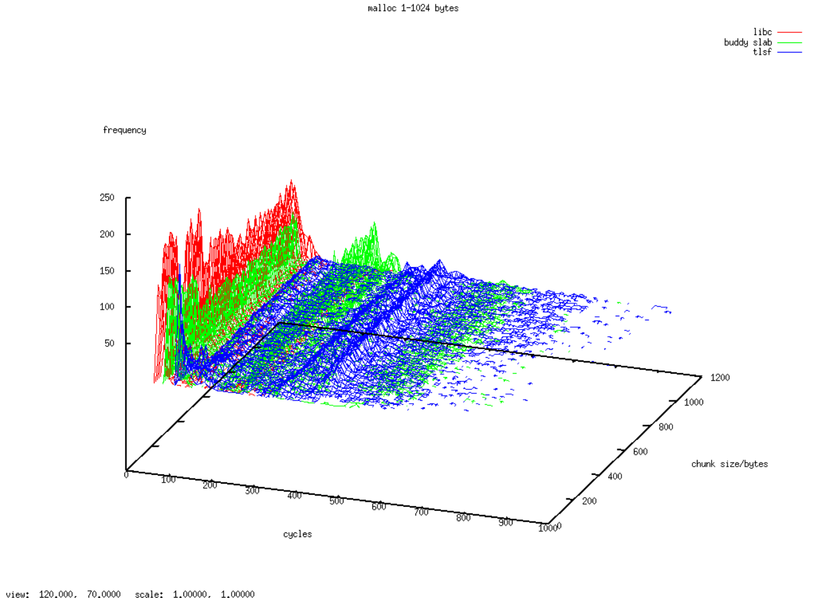


T = Last physical block
F = Free block

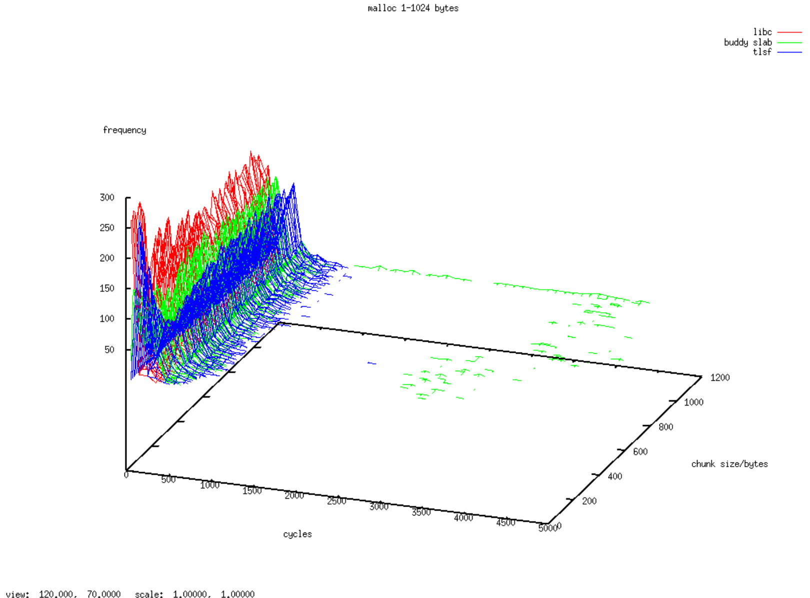
Statistics

- AMD Duron 807 MHz
- rdtsc before/after alloc()/free()
- libc vs. buddy slab vs. tlsf

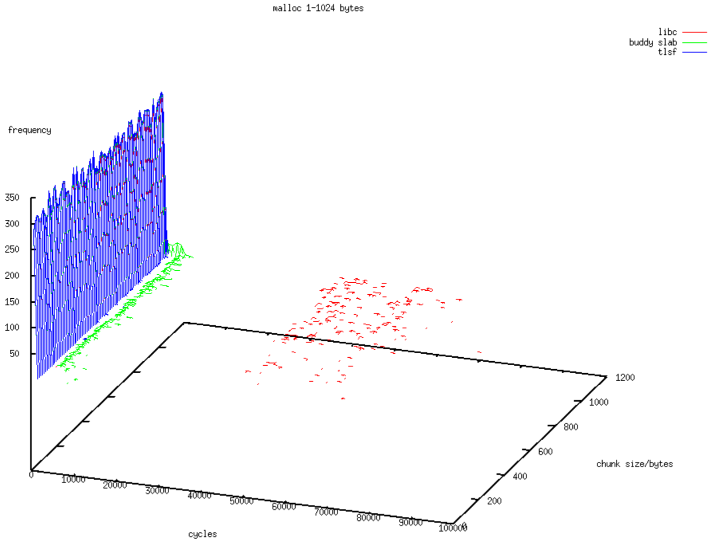
Statistics : malloc 1-1024 bytes



Statistics : malloc 1-1024 bytes

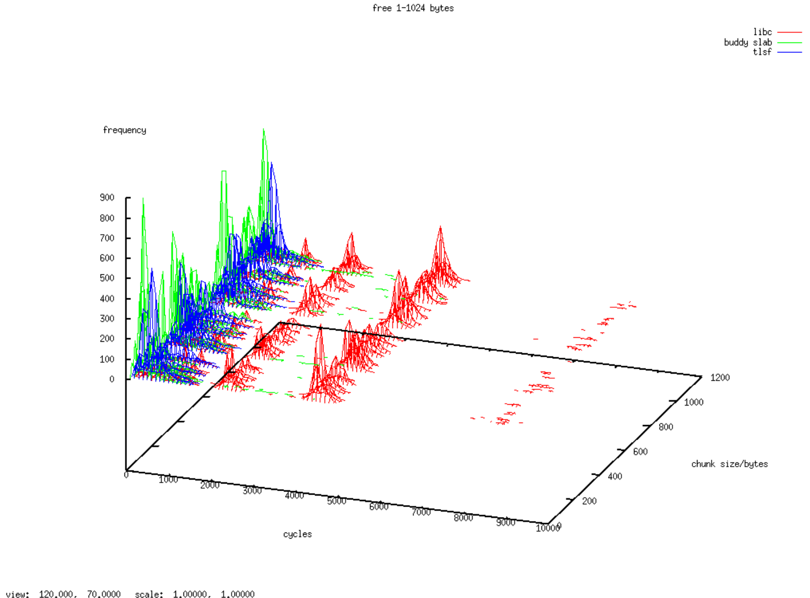


Statistics : malloc 1-1024 bytes

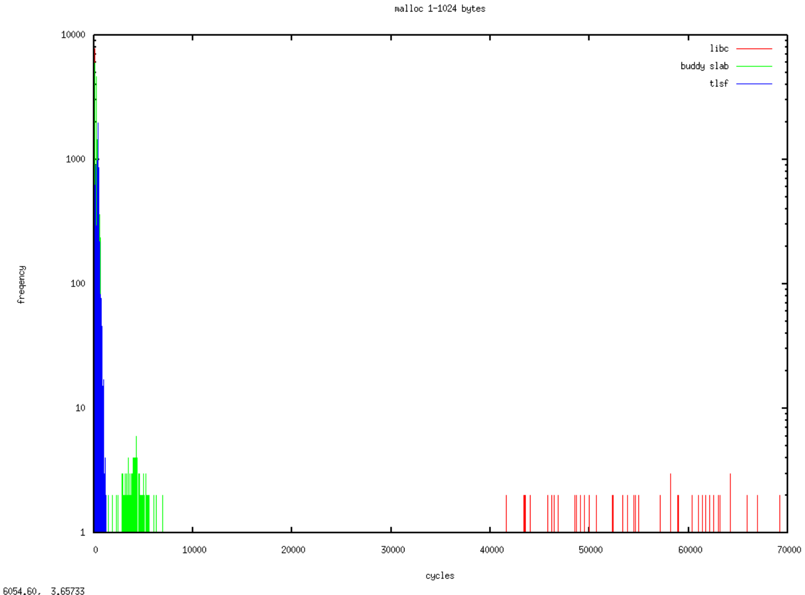


view: 120.000, 70.0000 scale: 1.00000, 1.00000

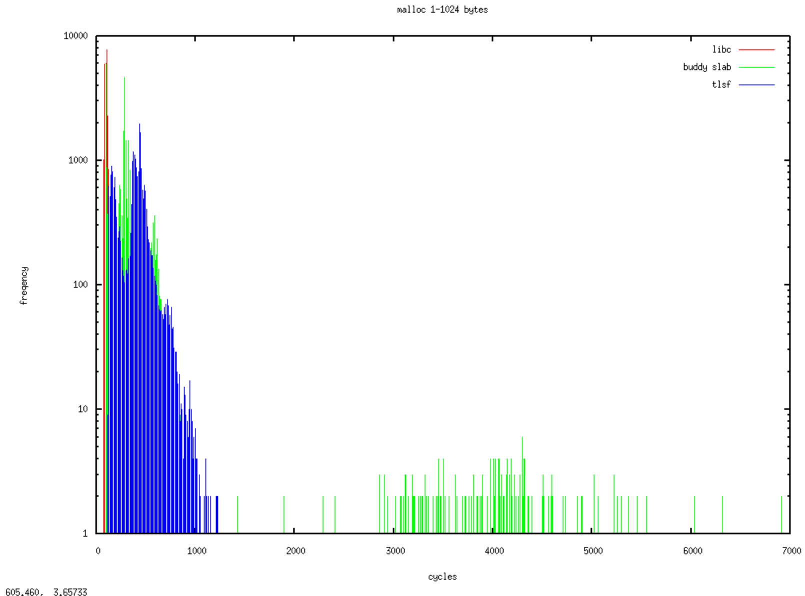
Statistics : free 1-1024 bytes



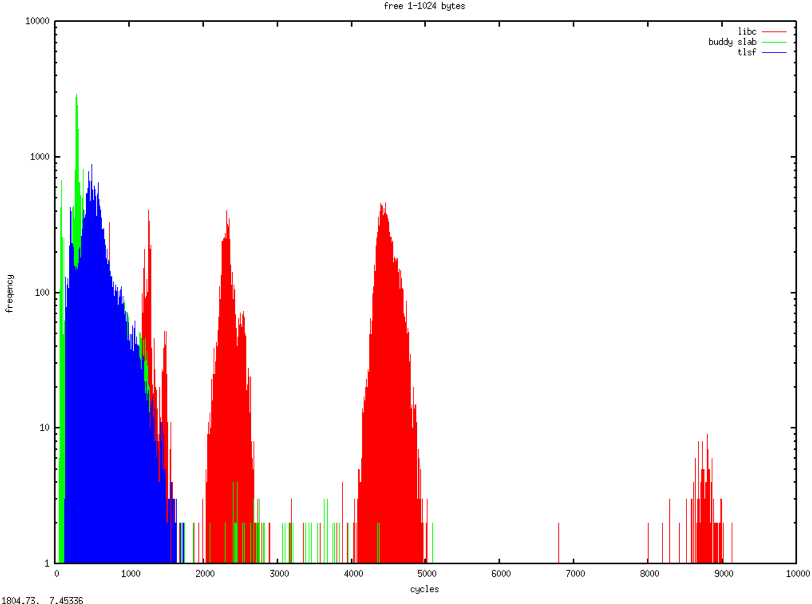
Statistics : malloc 1-1024 distribution



Statistics : malloc 1-1024 distribution



Statistics : free 1-1024 distribution



Remarks / Open Questions

- Synthetic workload? ("Dynamic Storage Allocation: A Survey and Critical Review")
- Do real-time applications really need such a general purpose DSA?
- Usable for non-real-time applications? (higher response time vs. low upper bound)