Available online at www.sciencedirect.com

Int. J. Electron. Commun. (AEÜ) 62 (2008) 356 – 364

International Journal of Electronics and Communications AEÜ

www.elsevier.de/aeue

# FIFO segmentation as a substitute to linked lists

Faidon C. Petreas, George I. Stasinopoulos*

*Department of Electrical Engineering and Computer Science, National Technical University of Athens (NTUA), Zographou Campus, Athens 157 73, Greece*

Received 31 July 2006; accepted 10 May 2007

**Abstract**

We present a technique for dynamic allocation of memory to different queues. Our approach lies between two well-known techniques, linked lists and the relocation of data for preserving reasonable contiguous areas of available memory space. However, we avoid the main drawbacks of both. We never allocate more than two segments of contiguous memory per flow and thus avoid the proliferation of pointers and associated memory fragmentation. Moreover, we never relocate data already admitted into memory. We thus offer a considerably simpler implementation, amenable to hardware realization. The price to pay is also twofold. We cannot guarantee total exhaustion of the available memory before overflow occurs and we can only implement the first in first out (FIFO) discipline for the flows hosted in memory. In fact, we exploit the defining feature of a FIFO: data having arrived first will also leave first, spending in memory as little time as possible and giving the opportunity for rearranging memory allocation to the advantage of future arrivals. Two segments per flow are sufficient to achieve, over time, complete memory reallocation without data movement. Simulation results exhibit this 'refreshing' feature of FIFO, as brought to light by our proposed scheme.
© 2007 Elsevier GmbH. All rights reserved.

***Keywords:*** Memory management; Shared memory; List processing

## 1. Introduction

There is much interest in dynamic allocation of memory during the last five decades. Algorithms proposed efficiently exploit the available space reducing dramatically the cost for memory resources in a variety of computer and communication systems. The problem with memory management has considerable practical importance. The widespread use of poor allocators incurs a loss of main and cache memory (and CPU cycles) which is considerable despite the dramatic price reductions in all storage technologies. It is important to note that many articles in non-refereed publications – and a number in refereed publications outside the major journals of operating systems and programming languages – are motivated by extreme concerns about the speed or memory costs of general heap allocation.

Memory requirements are continuously increasing. High bandwidth services, such as video require a robust core network consisting of high-speed embedded systems (network switches) coupled with high-capacity memory. Usage of efficient memory management techniques can reduce memory requirements, which is crucial for the reduction of system costs. More importantly multiprocessing on several incoming and outgoing flows combines large memory requirements with packetization and queue discipline tasks on a scale hitherto unencountered. One has only to think at multimedia services, all delivered on separate TCP streams and subject to complex tasks at network and application level

* Corresponding author. Tel.: +30 210 772 2531; fax: +30 210 772 2534.
*E-mail addresses:* faidonpe@central.ntua.gr (Faidon C. Petreas), stassin@cs.ntua.gr (George I. Stasinopoulos).

1434-8411/$ - see front matter © 2007 Elsevier GmbH. All rights reserved.
doi:10.1016/j.aeue.2007.05.006

(inter- and intrablock, frame encoding and decoding, etc.). Indeed, first in first out (FIFO) disciplines and link lists are inherently embedded in pure communication and/or service-related functionalities.

The main target of a memory management allocator is to place data in such way that fragmentation is minimized. Fragmentation is inevitable because blocks of different sizes are stored and retrieved. An incoming block of a size greater than the contiguous free space left by a previously retrieved block has to be fragmented, if reallocations of existing blocks have to be avoided. Else, an allocator should somehow compact memory, by moving blocks of data and produce larger contiguous regions within the available memory [1]. In our approach memory is compacted in an evolutionary manner and without extra resource being spent in an adhoc or periodic fashion.

In this light, our work has some common features with the two-hole method used in [2], in implementing efficiently the wraparound (circular buffer) technique of [3] in the presence of memory paging. However, our setting is totally different, since we consider more than one flow and a priori fixed memory size without paging, and use all available free intervals. Memory resizing indirectly amounts to dynamic memory allocation.

Resizing a circular FIFO allocated to a flow can dynamically update the amount of memory available to adjacent flows [4]. Successive application of the resize procedure can lead to memory reallocation for all flows. However the relative placement of flows in memory remains the same. In our work, we do not resort to memory resizing and do not stipulate that flows have to be stored in a fixed order inside memory. Hence we are more dynamic.

We have drawn some ideas from [5], who present a scheme adequate for accommodating many LIFO flows (stacks). Each flow is represented by a segment that is placed in the middle of the most appropriate (largest) free interval. When one segment flows into another, [5] resorts to relocation of data. Our technique also allows segments to grow towards each other. To avoid an overflow, we continue elsewhere with a new segment. So we do not sacrifice resources for moving content to other areas in memory. We totally prevent relocation, however not for LIFO, but exclusively for the FIFO discipline. Our development and numerical results show the crucial difference between the FIFO and LIFO discipline. The first has the 'refreshing' property, which is totally absent from the second. It is indeed this difference which has enabled us to go further than [5] and achieve our results.

Finally, our proposition is related and comparable to linked lists. We consider our work as a valid alternative to link lists, but again only for the case of FIFOs and give arguments and numerical results showing our advantages. Linked lists is the established method, where memory usage is crucial. An incoming block is allowed to be split in multiple segments filling empty blocks. However, measures have to be taken to avoid extreme fragmentation, even with a small number of distinct flows.

Modern handling of linked lists uses sophisticated ways (a) to select ('first fit', 'best fit', 'next fit' and 'lazy fits' in [6,7], 'segregated lists' in [6,7], 'indexed fits' in [8]) or construct ('buddy' method in [9,1]) the appropriate free interval, and (b) to coalesce free intervals ('address ordering' in [6,7,9,1]).

In the first fit method, the first empty block large enough to accommodate a segment is used. Best-fit policy selects the empty block which leaves the minimum empty space after the insertion of the incoming segment. Next fit method is an optimization of first fit method. The only difference is that at the beginning of a new search, a pointer shows the next available free interval in the list of the free intervals. This is done to decrease the average search time in the list, but has some serious negative effects regarding memory reuse. Objects from different phases from memory may be interspersed in different locations in memory increasing fragmentation when these blocks are emptied later. Several experiments have shown that next fit causes more fragmentation than address ordered first fit and best fit method. Lazy fits is a relatively new variant, which combines some of the techniques presented above. An allocation and a boundary pointer define the size of the empty interval. Each incoming block that fits in the free interval, increments the allocation pointer. If the block does not fit, the remaining empty block is returned to the list of free intervals, and a new free interval is selected via first fit, best fit or address ordered method. Lazy fits are faster than segregated lists, but not necessarily better. They do not need to compute the size of the object class every time like segregated lists do. Lazy fits have the potential of performing faster than conventional fits and segregated lists, without increasing considerably the memory fragmentation [6,7].

The segregated lists method consists of separate free lists for each group of similar object sizes. It is an alternative to best fit and first fit, and produces better execution speed [6,7]. This method is very competitive regarding the fragmentation of memory in a broad range of environments. It requires larger memory than best fit and first fit in order to perform well [9]. Buddy systems are a variant of segregated lists. The entire memory is split into two areas, and progressively every single area is split into further two, until the target area fits the size of the segment to be allocated. Free intervals of the same binary hierarchical division can be coalesced and create a single free interval of the next larger size class. When the splitting is binary and into two equal areas, we have a binary buddy system [1,9]. The buddy or binary splitting algorithms suffer from internal fragmentation of space [9]. Indexed fits use a tree that orders the empty blocks by size. The algorithm searches the tree for the empty block that satisfies the best-fit policy.

These methods are used to minimize the number of split segments and hence reduce the memory required for pointer