

Using Decision Making and Game Theory to Minimize Memory Fragmentation

Brian Carter, Ari Kassin, and Arthur Walton

February 25, 2008

1 Introduction

A common problem encountered in the design of operating systems is that, as more and more processes execute and terminate, memory becomes increasingly fragmented. Eventually, there exist only several small fragments of free memory available to the operating system. These fragments of free memory, along with large blocks of allocated memory, comprise the entire system's memory space. Therefore, when a new process requests a block of free memory that exceeds the largest possible free fragment size, the operating system is forced to assign the process a large block of memory that spans multiple such fragments. Memory fragmentation is a recognized problem in operating systems with negative effects on performance. [4]

Fragmentation of memory becomes a large problem because it reduces the potential throughput of cache. Caches rely heavily on the principle of spatial locality of memory blocks—that is, caches perform best when frequently used blocks are near each other in memory, especially when these blocks are directly adjacent to each other. This is because a lack of spatial locality tends to increase the number of cache misses (points at which a block of memory is not in cache and must be read from RAM). Cache misses take much more time than either reading blocks from the cache or directly from RAM. [6, 8]

We propose a solution to this problem of memory fragmentation in the form of a redesigned memory scheduler for computer systems, most practically implemented in the Linux kernel. In particular, we propose that our approach use the fields of decision making or game theory to make optimal decisions related to the distribution of memory to processes.

2 Considerations of Memory Management

There currently exist a tremendous number of approaches to solving the underlying problem of parceling out memory to multiple processes operating on a single system. In particular, virtual memory is crucial to enabling this process to occur in an efficient manner. Virtual memory allows each process on a system to have “its own” virtual memory space—that is, each process believes it has its own physical memory space on the system. In actuality, the operating system's responsibility is to use virtual memory to assign physical memory into multiple virtual memory spaces for multiple processes. [12] In particular, the part of the operating system that performs this memory management task is known as the memory manager.

There are several factors that should be considered by a memory manager to improve system performance: [10, 2]

Relocation. There must exist an efficient method of relocating blocks of virtual memory within physical memory to ensure maximal utilization of the physical memory space. Additionally, processes are able to request additional memory from the memory manager or “give back” memory to the memory manager. Blocks of memory must be moved within physical memory when either of these scenarios occur.

Protection. Each process must not be able to access another process' memory space. This comes with the fact that each virtual memory space is encapsulated within physical memory. The memory manager must provide independent and separate virtual memory spaces to each process. This ensures data and code security.

Sharing. A mechanism for sharing memory between processes must exist. This is different than the virtual memory spaces assigned by the memory manager. Memory sharing allows interprocess communication to occur.

Organization. A higher-level consideration of organization is the division of memory into the fastest cache, the fast RAM, and the slower disk. We are concerned only with the primary storage aspects of this system: the RAM primarily and the cache as a side effect of the way CPUs cache memory.

On a lower level, organization deals with the layout of blocks of memory within the physical memory space. This is the primary concern of our investigation into memory management using decision making and game theory. We propose that modifying memory organization using a new memory management scheme based on the principles of decision making and game theory will lead to increased cache throughput and thus higher performance.

2.1 Existing Implementations

Since the Linux kernel is freely available and highly documented, we have considered Linux's memory management implementation in our research. Linux includes a complete memory management subsystem with a virtual memory implementation. This subsystem implements memory management as described previously. In particular, memory is assigned as virtual memory to processes by the memory manager and is returned to the operating system's pool of free memory when the processes no longer need the memory. [3]

It is most important to realize that there are times when these blocks of memory (often called pages) must be moved around in physical memory to ensure a more appropriate (efficient) utilization of the physical memory space. We propose that optimal layout of pages within physical memory is key in optimizing memory performance by allowing the CPU cache to maximize throughput.

3 Definition of Problem

By considering the various possibilities of memory organization given any scenario of a set of processes with associated virtual memory spaces, we see that without a well-thought-out approach to organizing memory, this quickly becomes a very complex problem to manage. Even with a small set of processes, there are infinitely many ways to distribute the physical memory space among the processes.

To even begin solving this problem, we must define a consistent and logical algorithm for determining which memory organization will be most efficient. To assist with considering problems related to memory organization, we introduce the notion of a state:

Definition 1. A *state* is the set of all system processes, the set of all blocks of memory, and the set of all mappings between memory blocks and processes.

Therefore, our problem can be simply stated: given a set of processes, which state is the most efficient?

4 Approach to Solution

We present a simple but effective solution to this problem by assigning a utility value μ to each state of memory, which we will define below.

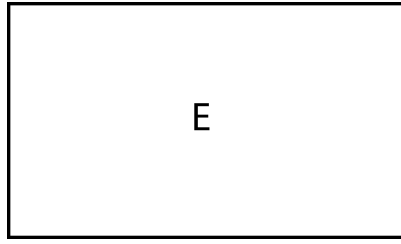


Figure 1: An illustration of a physical memory space composed entirely of free blocks. Consider that the memory space consists of five sub-blocks of memory which together comprise a single empty block. Thus, $\mu = 1 - \frac{1}{5} = 0.8$.

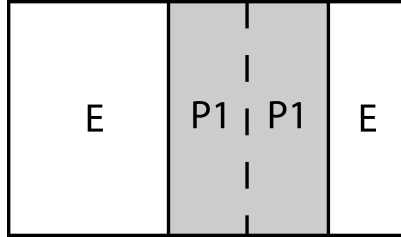


Figure 2: An illustration of a physical memory space with three contiguous segments of blocks of memory. Note that the middle block is two adjacent sub-blocks—our algorithm considers these to be a single block. Thus, $\mu = 1 - \frac{2}{5} = 0.4$.

Memory, of course, is divided into blocks. For the purpose of considering our proposed solution, the size of each block is irrelevant as long as the total physical memory size is divisible by the block size. A block is the minimum unit of memory a process can obtain from the memory manager. Let us, then, denote the total number of blocks of physical memory as α .

Now we must consider the number of contiguous blocks of memory that belong to the same process. However, there is a slight complication here: blocks may be composed of adjacent sub-blocks of the same process. Consider, for example Figure 1. In this case, there is one large block composed of five sub-blocks. In the case of Figure 2, there are three blocks. The middle block is composed of two sub-blocks, both of P1. However, Figure 3 has no adjacent sub-blocks of the same process; therefore, it has five blocks.

We may define the utility μ of a state of memory as follows:

$$\mu = 1 - \frac{c}{\alpha}.$$

$\mu \in [0, 1)$. Values of μ nearer 1 indicate a better state; conversely, values nearer 0 indicate a worse state.

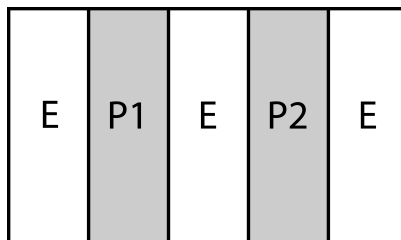


Figure 3: An illustration of a physical memory space composed of some blocks of memory being used by processes (P1 and P2) and some free blocks of memory. Since no blocks are composed of adjacent sub-blocks, $\mu = 1 - \frac{5}{5} = 0$.

4.1 Methodology

The challenge of our problem lies in developing an algorithm that may be used to have the largest number of regions of contiguous blocks of free memory at any given time. The field of game theory provides an excellent framework of concepts that may allow us to solve our problem. At a high level, we consider processes to be players of a game, trying to maximize their memory usage. The game, then, of course, is processes obtaining and freeing blocks of memory.

We investigated several game theory-related approaches to help solve our problem. A particularly interesting approach is the extensive form game. This type of game uses a tree to represent every possible state of the game as it is played. [7] Play begins at a unique root node and flows along a path of the tree until a leaf node is reached. The utility of a given path for each player is determined strictly by the utility of the state at the path's leaf node. To apply extensive form game theory to our problem, memory requests will be divided into rounds. Each round, then, is a play of the extensive form game with the result being a state of memory such that every process is satisfied with the allocation of memory it has received.

Play begins at the root node of the tree with the layout of memory just as it was the end of the previous turn but with additional free memory blocks added from any processes that returned memory. Processes then take turns choosing blocks of memory with each possible state of memory represented by a tree node. The utility of each leaf state is determined by the utility function previously defines. Then, this becomes a repeated game with each iteration being a round of the previous game. This repetition continues until there are no more processes running on the system. [11]

We can draw a few analogues between our tree and our system: nodes are equivalent to states. The root node (and terminal node) is the initial state of the system—that is, the state during which no processes have any memory reserved. Edges (transitions between nodes) represent a request for free memory blocks from the memory manager or a return of used memory blocks to the memory manager. Using this model, our problem can be stated as follows: which path of the tree results in the most efficient execution while ensuring optimal utilization of memory blocks?

To achieve our solution, it would be best to allow players (processes) to interact with each other and share information about each other's state.¹ Players may form groups known as coalitions. These coalitions may work against each other. Therefore, the game becomes a competition between coalitions, not just between independent players. In order to achieve this, we consider that players may make decisions individually but that the decisions must be approved by consensus among members of the player's coalition before taking effect.

We assume the following conditions before decisions may be made:

- All pre-play messages formulated by one player are transmitted, without distortion, to the other players;
- all agreements are binding and are enforceable by the game's rules; and
- a player's evaluation of the outcomes of the game are not disturbed by these pre-play negotiations.

A cooperative game is described by specifying a value for every coalition. Then, we describe each coalition the game as a mapping from the set of coalitions to the set of payments: [5, 1]

$$v : \mathcal{P}(N) \rightarrow \mathbb{R}.$$

v , known as the characteristic function, then, describes the amount of collective payoff a set of players will gain by forming a coalition.

Clearly, by allowing processes to form coalitions, it may be possible to substantially increase the efficiency of the system by improving efficiency among sets of processes (coalitions).

¹Note that this will be done via a third party, the memory manager. Allowing this type of exchange directly would violate one of the principles of virtual memory management.

4.2 Application of Solution

Implementation of the approach presented previously is not being considered at this time. As of now, the approach is purely theoretical. However, the natural way to apply the approach to an implementation is to consider, for each state, upon some event occurring (such as a process requesting from or returning to the memory manager blocks of memory), the utility of the current state μ . Next, the set of possible future states would be constructed. This is relatively simple. Consider, for example, that a process requests additional memory. The set of future states is simple:

- The state in which the process was allocated the requested memory; or
- the state in which the process was not allocated the requested memory.

Then, μ would be calculated for each state in this set. Naturally, then, the state with the higher value of μ would be selected to become the next state. Additionally, we must consider the benefit of combining processes into coalitions. Exactly how this will be accomplished has not been considered as of this time.

Clearly, there are some problems with this approach. For example, what if a process is never able to allocate memory because the future state corresponding to this event always has a lower μ than other states? Questions such as these will be important to consider as implementation nears; however, at this point, we feel that the brief description presented in this section is a good overview of how our proposed approach may be used in a “real” system.

5 Conclusion

We will continue to investigate further game theory-related approaches to solving this problem as we learn more about game theory.² A promising game theory approach appears to be an extension to the field of cooperative games. Using this model, each process may be able to communicate important details about itself to other processes—this will, in turn, allow the memory scheduler to produce an optimal plan for memory distribution to each process. Consider, for example, that each process could anticipate future memory requirements. Upon learning this, the memory scheduler could plan for the upcoming requirements and distribute memory differently in the short term.

Clearly, game theory provides tremendously powerful tools that may be used for solving problems such as the one we raise. By applying a combination of a utility function for each state of virtual memory and the notion of extensive form games, it will be possible to achieve highly accurate predictions of expected efficiency based on the state of virtual memory. With additional research and time, it appears that it will be possible to produce an elegant and simple algorithm for efficient memory management.

²Note, however, that we will not discount decision making as a potential approach to solution.

References

- [1] R. D. Luce and H. Raiffa, *Games and Decisions*, Dover Publications, New York, 1957.
- [2] The Linux Documentation Project, *The Linux Kernel*, Chapter 3, *Memory Management*, <http://www.tldp.org/LDP/tlk/mm/memory.html>, 1996.
- [3] linux-mm.org, *VirtualMemory*, <http://linux-mm.org/VirtualMemory>.
- [4] LWN.net, *Avoiding - and fixing - memory fragmentation*, <http://lwn.net/Articles/211505/>, November 28, 2006.
- [5] Wikipedia, *Cooperative game*, http://en.wikipedia.org/wiki/Cooperative_game.
- [6] Wikipedia, *CPU cache*, http://en.wikipedia.org/wiki/CPU_cache.
- [7] Wikipedia, *Extensive form game*, http://en.wikipedia.org/wiki/Extensive_form_game.
- [8] Wikipedia, *Fragmentation (computer)*, http://en.wikipedia.org/wiki/Fragmentation_%28computer%29.
- [9] Wikipedia, *Game theory*, http://en.wikipedia.org/wiki/Game_theory.
- [10] Wikipedia, *Memory management*, http://en.wikipedia.org/wiki/Memory_management.
- [11] Wikipedia, *Repeated game*, http://en.wikipedia.org/wiki/Repeated_game.
- [12] Wikipedia, *Virtual memory*, http://en.wikipedia.org/wiki/Virtual_memory.