# Edith Cowan University
# CSG1102
# Operating Systems
# Assignment 1

Martin Ponce
Student 10371381

Tutor: Peter Hannay

August 26, 2015

# Contents

# 1 Annotated Bibliography

## 1.1 Virtual memory managers

**Blanchet, G., & Bertrand, D. (2012). *Computer Architecture.* Hoboken, NJ: Wiley.**

Blanchet and Bertrand's (2012) book is aimed at readers wishing to learn about the essential architecture of a computer as a whole. However, Chapter 9 (p. 175 - 204) provides insight into the functionality of virtual memory management. The authors introduce virtual management with a brief history, then explain the purpose of virtual memory. Advantages of virtual memory are listed, but on the other hand state that the cost of use is the decreased performance during the access of slower secondary storage.

They explain how physical memory and a secondary storage device interact to provide virtual memory through the use of paging, and briefly cover fetch and page fault algorithms. Page size considerations are also listed, and referred to as "conflicting criteria". The chapter also introduces the concept of multi-level paging.

Blanchet and Bertrand (2012) then conclude the chapter by provide a detailed and technical step-by-step view of virtual memory management at work, using a program execution as an example.

**Blunden, B. (2003). *Memory Management Algorithms and Implementation in C/C++.* Plano, TX: Wordware.**

Although Blunden's (2003) book is primarily for C/C++ programmers looking to implement their own memory management system, it provides an overview of operating system memory management in Chapters 1 (p. 1 - 43) and 2 (p. 45 - 126), titled "Memory Management Mechanisms" and "Memory Management Policies", respectively.

Blunden introduces memory management in Chapter 1 at the processor level, which provides the mechanisms, segmentation and paging. While explaining memory hierarchy, he outlines the purpose of the L1 and L2 cache, and as with Blanchet and Bertrand (2012), states a similar disadvantage of virtual memory: Sacrifice performance for memory space. The author then identifies the differences between page frames and pages.

Chapter 2 explores the policies of memory management while comparing virtual memory management and paging (or lack thereof) between four operating systems: MS-DOS, MMURTL, Linux, and Windows XP. Interestingly, MMURTL's use of paging was not related to virtual memory, but to provide memory protection and allocation.

While the text can be very technical at times (C/C++ code snippets), there is sufficient information in these two chapters relating to virtual memory management and paging for use in further research, particularly the comparisons between different operating systems.

**Denning, P. J. (1970). Virtual Memory. *ACM Computing Surveys, 2*(3), 153–189. doi:10.1145/356571.356573**

Denning (1970) introduces the paper by comparing static and dynamic approaches to memory management and their assumptions and requirements for implementation. He discusses the current state of programming and identifies factors which make the static approach unsuitable. Denning (1970) further subdivides the dynamic approach between those who believe that the programmer should perform memory allocation, and those who believe that memory allocation should be automatic.

He segues into describing the 1961 Atlas computer, the first to implement virtual memory. Denning (1970) explains that virtual memory creates the illusion that a large main memory is available to the programmer, and that the main concept behind it is the notion of addresses being distinct from physical location. The responsibility of passing information between secondary storage and main memory when required, becomes the responsibility of the hardware and operating system, which defines virtual memory as a "form of adaptive system". The author claims that virtual memory is potentially more efficient than a preplanned, static approach, and it is also "facilitates programming and design objectives especially important in multiprogramming and time-sharing" (Denning, 1970, p. 156).

The paper outlines some problems with virtual memory, such the loss of usable storage due to fragmentation caused by the nature of paging. Alternatively, time-sharing systems which use "pure demand paging" can cause slower performance, exacerbated by the system only being able to load a single page at a time. Denning (1970, p. 157) lists thrashing as an issue under multiprogramming environments when memory is "overcommitted".

Denning (1970) divides the main body of work into two sections, describing the "mechanisms" for implementing virtual memory, and the "policies" for using those mechanisms. These two sections are analogous to Blunden's (2003) chapter titles, and the inspiration can clearly be identified. Denning (1970) goes on to define the operation of virtual memory in every aspect, in technical and mathematical terms.

The paper concludes by iterating the history of memory management technology and the inherent need for memory management ideologies to change in order to satisfy the requirements of advancements in programming. Denning (1970) claims that the most elegant solution that satisfies these requirements is virtual memory.

**Jacob, B., Ng, S. W., & Wang, D. T. (2008). *Memory Systems - Cache, DRAM, Disk.* Burlington, MA: Morgan Kaufmann Publishers.**

Jacob, Ng, and Wang's (2008) book provides a complete description of memory systems, and dedicates a section to virtual memory in Chapter 31 (p. 883 - 920). Jacob et al. (2008) cite many academic sources throughout their book, and choose to express pseudo-code rather than code specific to a programming language (where applicable) to provide ease of use.

Jacob et al. (2008) begin the chapter by describing a brief history of virtual memory management, and the technique of combining the cache, main memory and disk to provide virtual memory to a computer system. They point out the advantages of virtual memory, however, unlike Blanchet and Bertrand (2012) or Blunden (2003), they fail to mention any disadvantages.

This chapter defines the relationship between address spaces and main memory

cache. They explain the various designs of main memory cache which dictate how a virtual page is mapped to the main memory cache. Jacob et al. (2008) cite various sources attempting to improve Translation Lookaside Buffer lookup times.

The authors describe the functionality of page tables and the information they must store to enable the operating system to perform paging. Jacob et al. (2008) also compare hierarchical and inverted page tables structures and their corresponding algorithms to manage page tables.

This book functions well as a dependent source for research, with many academic citations, clear explanations and diagrams for visual representation of concepts.

**Jantz, M. R., Strickland, C., Kumar, K., Dimitrov, M., & Doshi, K. A. (2013). A framework for application guidance in virtual memory systems. In *VEE '13 Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (pp. 155–166). New York: ACM. doi:10.1145/2451512.2451543**

Jantz, Strickland, Kumar, Dimitrov, and Doshi (2013) propose an alternative virtual memory management system in which the program itself communicates information about its patterns of memory use to the operating system. They claim that this method is more efficient in terms of memory usage and power consumption, than an operating system making judgements about a program's demand of resources exclusively without guidance from the program itself.

The authors propose two new constructs to describe their methodology. The program communicates to the operating system through "colors", while main memory pages are organised at an additional level through structures called "trays" in order to encapsulate program guidance communication from low level operations of the system.

Jantz et al. (2013) implement their alternative virtual memory management system by modifying the Linux kernel to support their design. Performance is compared during several tests between the original unmodified Linux kernel, to their own custom kernel. The first test benchmarks the performance of their "colors" framework, which compares memory efficiency and local memory reads. Results indicate that their design outperforms the original management technique. The second test identifies the difference in DRAM power consumption between implementations, to identify performance of their "trays" framework. In this case, they did not observe any difference between their design and the original kernel. However, a third test comparing power consumption in a garbage collection environment yielded an almost 9% drop in DRAM power usage (Jantz et al., 2013, p. 164).

The paper concludes, reiterating the proposal, and the implementation methods. The authors state the demonstrated benefits of its application, which "meets a need for a fine-grained, power-aware, flexible provisioning of memory" (Jantz et al., 2013, p. 164).

## 1.2 Paging / swap

**Alanko, T. O., & Verkamo, A. I. (1983). Segmentation, paging and optimal page sizes in virtual memory. *Performance Evaluation, 3*(1), 13–33. doi:10.1016/0167-7136(83)90150-6**

Alanko and Verkamo's (1983) paper compares the performance between segmentation and paging, and investigates the optimal page size. They introduce the subject by identifying the differences in use, and characteristics of segmentation and paging. The authors claim, that at the time, no studies have yet been published which compare the two memory management techniques. They also identify memory systems which combine segmentation and paging techniques.

Alanko and Verkamo (1983) outline the page size and how the parameter interacts with the performance of paged systems. They consider current studies in the field and come to the conclusion that no consensus has been found in optimum page size proposals.

The paper continues with the experiments, executing programs in segmented and paged environments using the working set policy. Alanko and Verkamo's (1983) results indicate that "there is no globally optimal page size". When using "space-time integral" as the key value for measuring performance, segmentation outperforms paging, and the performance gap increases as the mean segment size increases.

**Babaoglu, Ö., & Joy, W. (1981). Converting a swap-based system to do paging in an architecture lacking page-referenced bits. In *SOSP '81 Proceedings of the eighth ACM symposium on Operating systems principles* (Vol. 15, pp. 78–86). New York. doi:10.1145/800216.806595**

Babaoglu and Joy (1981) describe the challenges and their methods converting UNIX from a segmented, swap-based system to a paging system, in a computer with architecture that does not support page-referenced bits. The authors explain the use of page-referenced bits as essential to page replacement algorithms such as clock page replacement and sampled working set (SWS). They compare various page replacement algorithms, and set out to implement their variation of clock page replacement in UNIX by simulating page-referenced bits through software.

Babaoglu and Joy (1981) justify their design and optimization decisions by comparing the performance of the clock page replacement, and identify opportunities for improvement, based on the given hardware and operating system. For instance, Babaoglu and Joy (1981, p. 80) state the original algorithm only seeks to replace a single page when triggered by a page fault, and through testing, found examples where page requests spiked due to UNIX's non-uniform operations. They modified the algorithm by implementing a free page pool containing page frames not currently in the clock loop, and set a minimum free page pool size as a threshold. When this threshold is reached, clock page replacement is triggered and pages are replaced until the free page pool size reaches the threshold again. As the free page pool size decreases, the scan rate of the clock page replacement implementation increases until it reaches a maximum scan rate, which is "determined by the time it takes to simulate the setting of a referenced bit" (Babaoglu & Joy, 1981, p. 80).

The authors compared their clock paging system to the original swap-based system and present their findings with graphs comparing performance between the

two. They found that under lower load levels, their clock page implementation out performed the swap-based method. However, under heavy load, while clock page replacement exhibited much lower page traffic, the overhead required to support it was higher than the swap-based method. The authors counter this finding by stating that the CPU utilization was greater for the clock paging system compared to the swap-based system.

Babaoglu and Joy (1981) conclude the article with a recommended requirement when designing a page replacement algorithm for architecture with no support for page-referenced bits, the results of their comparison of the two memory management algorithms, and limitations of their global clock replacement algorithm.

**Denning, P. J. (1967). The Working Set Model for Program Behavior. In *SOSP '67 Proceedings of the first ACM symposium on Operating System Principles* (pp. 15.1–15.12). New York: ACM.**

Although the working set model is neither a virtual memory manager or paging algorithm itself, it is essential to the efficiency of virtual memory (Silberschatz, Galvin, & Gagne, 2013). Denning's (1967) proposal is a landmark article in the field of computer science and according to Google Scholar, has been cited 1076 times.

This source introduces the concept of the working set model, which provides the ability for a system to determine what information is being used, or not being used by a program during execution. This ability allows a computer to make better decisions in allocating or deallocating resources to that program.

At the time, the user and the compiler were commonly proposed to be used as input for dynamic memory allocation. Denning (1967, p. 15.1) claims that neither sources are adequate. He argues that the user cannot possibly provide reliable estimates of resource requirements of his or her program. Additionally, due to the nature of modularised programming, it is possible that the compiler may not be able to decide which modules are required until run time, and therefore not be able to approximate the required resources appropriately.

Denning (1967) explores the current body of work in memory management strategies and states that no paging algorithms have been proposed which involve anticipating resource requirements. The author attributes the cause to the lack of reliable information that the system can use to judge allocation requirements, and returns to the point of the user and compiler as inputs.

Denning (1967, p. 15.2) proposes "new mechanisms" to monitor behaviours of programs, and allocate resources based on those observations. The proposal begins by defining the current environment, briefly explaining the process of paging and what occurs during a page fault. He defines the "core memory management" problem as deciding which pages remain resident in main memory. Denning (1967, p. 15.3) offers a strategy - to "minimize page traffic" to reduce overhead required during page swaps, and reduce processing time, which affects processor efficiency if processing pages takes too long. Thus, Denning (1967, p. 15.3) defines the working set as "the minimum collection of pages that must be loaded in main memory for a process to operate efficiently, without 'unnecessary' page faults". A more explicit definition is offered as "the set of its pages a process has referenced within the last $\tau$ seconds of its execution" (Denning, 1967, p. 15.4).

He provides hardware and software implementations of the working set model, and justifies the proposal through its practicality in resource allocation for both processor and memory demand strategies.

Denning (1967) concludes the paper with brief comparisons of current allocation strategies, the ideologies of the working set model, and iterates the various parameters and properties defined in the implementation of his proposal.

**Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). Operating System Concepts Essentials (2nd ed.). Hoboken, NJ: Wiley.**

Silberschatz et al.'s (2013) book provides overview of operating systems, and contains a chapter describing virtual memory (p. 371 - 438). The authors present a brief history of virtual memory management and explain the organisation of the virtual address space.

The chapter explains the concept of paging, while exploring the functionality of demand paging and page faults. The book offers several mathematical equations to calculate demand paging efficiency and provides considerably more detail of page replacement and frame allocation algorithms compared to other books in this annotated bibliography. Silberschatz et al. (2013) defines thrashing and the working set model. The chapter concludes by comparing the implementation of virtual memory between two operating systems, Windows and Solaris.

Similar to Jacob et al. (2008), this book provides clear and concise explanations and diagrams to visually represent concepts.

**Van Wezenbeek, A. M., & Jan Withagen, W. J. (1993). A survey of memory management. Microprocessing and Microprogramming, 36(3), 141–162. doi:10.1016/0165-6074(93)90254-I**

Van Wezenbeek and Jan Withagen's (1993) paper explores the body of literature on the topic of memory management and virtual memory to provide an overview of the research on the subject. They do not compare existing memory management technologies, however the paper serves to highlight the "state of the art research" on the topic (Van Wezenbeek & Jan Withagen, 1993, p. 141).

Van Wezenbeek and Jan Withagen (1993) define the three policies of mechanisms which enable paging and segmentation. First, the fetch policy determines when information will be loaded into main memory, which can either be on demand, in advance, also known as pre-fetching, or both, called demand prefetching. Prefetching relies on the spatial locality of reference. In other words, if a page is referenced, then it is also likely that surrounding pages will also be referenced. The probability of being referenced decreases as the distance from the initially referenced page increases. Secondly, the replacement policy determines which page or segment should be replaced in order to regain required space in main memory. The third policy is the placement policy, which decides where to place new information in the free space of main memory.

The authors assert that efficiency of virtual memory is dependent on the working set, based on temporal locality of reference, and stress that good replacement policy should preserve the working set of a program. Alternatively, spatial locality, as mentioned previously, can also help to achieve virtual memory efficiency, by structuring the program so that its pages are close to each other. They cite sources stating that

this can have a greater impact on performance than the choice of a replacement policy. However, this falls under the responsibility of the programmer.

Van Wezenbeek and Jan Withagen (1993) outline the functionality of paging and due to its nature of operation, causes internal fragmentation. They explain how it works and the requirements to support paging, which includes the maintenance of various tables. For example, the authors define the information that is included in a page table, and which bits provide protection of memory in a multiprogramming environment.

Van Wezenbeek and Jan Withagen (1993) define the process of segmentation and in contrast, causes external fragmentation. Similar tables to paging are maintained for segmentation, but these tables also include extra information to keep track of segment length. They stress that a good placement policy is required for segmentation to minimize external fragmentation.

Replacement policies are explained in depth. This section is further subdivided into fixed allocation policies, such as "First In, First Out (FIFO)", "Least Recently Used (LRU)", "Most Frequently Used (MFU)", "Not Recently Used (NRU)", "Clock (CLK)", and "Second Chance (SC)". Variable allocation policies include "Working Set (WS)", "Page Fault Frequency (PFF)", "Damped Working Set (DWS)", "Sampled Working Set (SWS)", "Working Set Clock (WSC)", and "Variable Interval Sampled Working Set (VSWS)".

Van Wezenbeek and Jan Withagen (1993) explore placement policies. They state that this is operation is trivial, and works similarly for both placement and replacement policies in a paging environment. However, the process is more complicated in a segmented environment. As segmentation occurs, free blocks will be separated, and the list of these free blocks will increase as main memory is filled. A suitable block of memory must be searched for in this list when allocating memory. Van Wezenbeek and Jan Withagen (1993) outline the consequences of the order in which the blocks sit within the free space list.

# References

Alanko, T. O., & Verkamo, A. I. (1983). Segmentation, paging and optimal page sizes in virtual memory. *Performance Evaluation*, *3*(1), 13–33. doi: 10.1016/0167-7136(83)90150-6

Babaoglu, O., & Joy, W. (1981). Converting a swap-based system to do paging in an architecture lacking page-referenced bits. In *Sosp '81 proceedings of the eighth acm symposium on operating systems principles* (Vol. 15, pp. 78–86). New York: ACM. doi: 10.1145/800216.806595

Blanchet, G., & Bertrand, D. (2012). *Computer Architecture*. Hoboken, NJ: Wiley.

Blunden, B. (2003). *Memory Management Algorithms and Implementation in C/C++*. Plano, TX: Wordware.

Denning, P. J. (1967). The Working Set Model for Program Behavior. In *Sosp '67 proceedings of the first acm symposium on operating system principles* (pp. 15.1–15.12). New York: ACM.

Denning, P. J. (1970). Virtual Memory. *ACM Computing Surveys*, *2*(3), 153–189. doi: 10.1145/356571.356573

Jacob, B., Ng, S. W., & Wang, D. T. (2008). *Memory Systems - Cache, DRAM, Disk*. Burlington, MA: Morgan Kaufmann Publishers.

Jantz, M. R., Strickland, C., Kumar, K., Dimitrov, M., & Doshi, K. A. (2013). A framework for application guidance in virtual memory systems. In *Vee '13 proceedings of the 9th acm sigplan/sigops international conference on virtual execution environments* (pp. 155–166). New York: ACM. doi: 10.1145/2451512.2451543

Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating System Concepts Essentials* (2nd ed.). Hoboken, NJ: Wiley.

Van Wezenbeek, A. M., & Jan Withagen, W. J. (1993). A survey of memory management. *Microprocessing and Microprogramming*, *36*(3), 141–162. doi: 10.1016/0165-6074(93)90254-I