A Reformed Scheme of Teaching Memory Management in Operating System Courses

Jing Meng
Department of Computer Science and Engineering
Renmin University of China, Beijing, China, 100872
mengjing@mail.ruc.edu.cn

Abstract:

This paper analyzes the existing problems in the teaching of memory management of operating systems, presents a reformed teaching scheme, and introduces its adoption and effects in teaching practices. This teaching scheme consists of a head model, a main body model and a summary model, among which the four-space summary model is the most distinguishable characteristic. The scheme has been implemented in the author's textbook “Operating System principle” (in Chinese, Tsinghua University Press, Beijing, China, 2000) and is consistent with the whole teaching scheme of operating systems in this textbook.

1. The Problem and the Author's Work

Operating system principle is one of the most important courses in computer science, and memory management is one of the complex chapters with many concepts. Currently, there are several problems in the teaching practice of memory management, including:

- There are many concepts and the relationship among them is complex. But these relations are not clearly stated and compared, and there is not a unified model to summarize them.
- There lacks hierarchy of explanation and relationship between the contents.
- The interpretation of "What it is" and "Why it is necessary" is commonly ignored.

Focusing on the above problems, the author, through her more than ten years of teaching practices, designed a total set of teaching schemes and models of memory management in operating systems. This teaching scheme consists of **a** three-layer **head model, a three-step main body model and a four-space summary model**, and have the following characteristics: integrated explanation of 3W (WHAT，WHY，and HOW), step by step explanation levels, and summary and abstract models for many concepts and their relations.

The model has been implemented in the author's textbook "Operating System principle" (in Chinese, Tsinghua University Press, Beijing, China, 2000) and is consistent with the whole teaching scheme of operating systems in this textbook.

2. The three-layer head model

The three-layer head model is used to introduce **what-to-do and why** of memory management.

2.1 The Bottom layer: characteristics of the hardware interface of memory system

Firstly, memory hardware interface characters are discussed as the bottom layer, which include three machine-instruction-level characters discussed from microcosmic point of view, and three program-space-level characters discussed from macro point of view.

From microcosmic point of view, the three instruction level characters are:

- The main memory is used by the program through a series of read/write requests issued by the program to memory. Usually, each instruction will access memory for more than one times.

- The memory accessing instructions and I/O instructions accessing disks are different. In one access to memory and one access to disk, the number of instructions required, the minimum access unit, and the addressing units are all different. The deep reasons lie behind the differences is the different performance requirements and different costs.

- Besides, except for accepting read/write requests from CPU, the main memory can also serve read/write requests issued by I/O devices, such as DMA requests.

From macro point of view, after reviewing the changes of memory address formats in different stages of program life (e.g., source code, compiling, linking, and execution), three space level characters of executable binary programs are introduced:

- The program space is composed of three kinds of regions of code, data, and stack. Usually the code region is read-only and reenterable, while data and stack regions are writable.

- The start address of executable object code should be consistent with the real memory address where it is loaded and executed. In case of inconsistency, special processing, e.g., relocation or address mapping, is needed. Otherwise, the program cannot be executed correctly.

- The address space of a program should be continuous, and it is required that the program be placed in memory continuously and integratedly. If this can not be assured, special processing, e.g., address mapping or virtual memory, is needed. Otherwise, the program cannot be executed correctly, or memory space is wasted due to the large amount of memory holes.

From the above two viewpoints, the hardware interface of memory system is reviewed, which lays a solid foundation for the learning of this chapter.

2.2 Top and middle layer: User requirement, Gap analysis, the tasks of memory management in operating system

Then, the nine requirements of using memory systems by users are discussed as the top layer, which are listed as follows.

1. Physical details, such as the memory physical addressing and the size of physical memory space, should not be involved by the users.

2. The loading problem should not be considered by the users.

3. There is enough memory capacity, and the memory space should be fully utilized.

4. The memory system should be accessed with high speed.

5. User programs and their data in memory should be protected and made security.

6. User programs can dynamically scale their space requirement.

7. Processes can use memory for sharing data or communicating with each other.

8. The users should know the current memory occupation situation.

9. The cost to accomplish the above requirements should be minimized.

After that, the gaps between the memory hardware interface characters and the user requirements for using memory is discussed. In the middle layer lays the division of work and coordination between memory management (**machine-dependence and application-independence)** and other software such as compilers, libraries, and applications (**machine-independence and application-dependence)**, which fill the above gap jointly.

For example, how to make users and user programs not to directly use physical memory address? This is implemented with the cooperation of operating system, compiler, and hardware as follows. The address transform from the symbol addresses in source programs to sequential binary addresses in executable object codes is usually hardware independent, and carried out by the compiler. The address transform from logical addresses in executable object codes to the physical addresses in real execution is usually hardware dependent, and carried out by the operating system and the hardware (reallocation and address mapping).

At last, the functions and tasks of operating system memory management are given as the conclusion of the head model. The chief tasks of memory management in operating systems is correspondent to the above user requirements:

1. Transforming from logical memory addresses in executable object codes to physical memory addresses.

2. Loading the object code into main memory, with the help of compilers.

3. Using multiprogramming, noncontiguous storage allocation, sharing, swapping, virtual memory and other techniques to improve memory utilization and fulfill the requirement for large memory capacity.

4. Cooperating with hardware to match the speed gap between memory and CPU.

5. Using various software techniques to cooperate with hardware to solve the memory protection and security problem.

6. Realizing the dynamic scalability of user programs.

7. Fulfilling user's requirement in sharing and communication.

8. Providing interfaces to help users to understand the current situation of memory systems.

9. Accomplishing the above task with relatively low cost.

3. The three-step main body model

The three-step main body model is used to introduce **how-to-work** of memory management step-by-step to decompose learning difficulties and complication of memory management， and to ease understanding.

There are 7 different schemes of memory management, which can be listed from simplest to the most complicated as follows: no management (in earliest computers), single-partition allocation, fixed partition, variable partition, paging scheme, segmentation scheme, and paging segmentation system. The most commonly used, and hence will be explained with emphasis, is the paging scheme. For every memory management scheme there are three studying steps. The first step is the fundamental mechanism, fundamental procedures and structures of memory management. The second step is the special subject discussion, which consists of several small steps and every small step focuses on a special subject. The third step is the case study.

The whole three-step major model (and the whole teaching scheme) puts "**address**" (and the **address translation**) as the **center** and **mainline**, and at every step, the corresponding address translation procedure is given. As the course goes through these steps, the address translation procedure becomes more and more complicated (but still easy to understand because of step by step model) and approaches to real situations.

Take paging scheme as an example. In the first step (basic mechanism), the simplest address mapping process, is explained. Only real memory and paging are involved, no TLB and other performance improving mechanisms. In the second step (discussion of special topics), various techniques to improve performance, such as virtual memory, translation look-aside buffer, multi-level page table (page directory and page table page), sparse addressing, page replace and workset, etc., are introduced, and the corresponding address translation processes illustrated. At last, in the end of this step, the address mapping process is very similar to that in real systems. In the third step (case study), the memory management mechanism of Linux, Solaris and Windows 2000 are introduced and compared with each other.

4. The four-space summary model

The **four-space summary model** is used to summary all concepts and techniques of memory management, since there are large amount of concepts in memory management.

Firstly, the definitions of four spaces, i.e., memory physical space, process logical space, process physical space, and memory logical space, are given. The logical space of a process is the address space constructed from all the addresses generated from CPU to MMU during the execution of the program. The physical space of a process is constructed from all of the real physical addresses (or absolute addresses) the program occupies when it is stored in the main memory. The memory logical space is the sum of logical spaces of all processes plus the process id. The memory physical space is just the real physical memory space.

After that, the individual characteristics of the 4 spaces and their relations are analyzed from addressing mode, size, and amount aspects. At the same time, all concepts and techniques in memory management are redefined from four-space point of view.

For example, the non-continuous memory allocation technique means non-continuity of the process physical space; the sparse addressing technology and two-dimensional addressing means non-continuity of the process logical space; and partition is relative to the memory physical space. When the process logical space of a process is not consistent with the process physical space, relocation or address mapping is necessary. Multi-task means that the memory logical space includes multiple process logical spaces; multi-programming means that the memory physical space includes multiple process physical spaces; and virtual memory means the process logical space is bigger than the process physical space, etc.

At the same time, these concepts and techniques are analyzed, compared, and correlated together through the four spaces. Take the non-continuous allocation and virtual memory as examples. Non-continuity is presented relatively to the continuity of process physical space, and virtual memory is presented relatively to the size relation between process logical space and process physical space. Virtual memory definitely means non-continuous, but non-continuous system may not be virtual memory. Another example is the relation between single-/multi-programming, continuity and virtual memory. Continuity and non-continuity can both exist in the single- and multi-programming environments. In single-programming environment, if non-continuous technique is used, there definitely exists virtual memory, otherwise non-continuity will lose its value. But in multi-programming environment, non-continuity may not imply the existence of virtual memory. Maybe it is just used to fully utilize memory space (avoiding fragmentation).

At last, all the 7 memory management schemes are re-defined from the 4 space aspect, and their relations analyzed.

The four space model well summaries the concepts and techniques of memory management, and with the model, these concepts and techniques are no longer odd, but become an integrated single system.

5. Concluding Remarks

The model proposed in this paper has been implemented in the author's textbook “Operating System principle”, in Chinese, published by the Tsinghua University Press, Beijing, China, 2000, and is consistent

with the whole teaching scheme of operating systems in this textbook.

This teaching scheme has got good effects in teaching practices, and book was used as textbook by many universities and colleges in China.

References

[1] Michael Beck, et al., *Linux Kernel Internals.* Addison-Wesley, 1996.

[2] Lubomir Bic, Alan C. Shaw. The logical Design of Operating Systems. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1988

[3] Douglas Comer. Operating System Design, The Xinu Approach. Prentice-Hall,Inc., Englewood Cliffs, New Jersey, 1984.

[4] Charles Crowley. *Operating systems: A Design-Oriented Approach*. McGraw-Hill Book Co., 1999.

[5] Harvey M. Deitel. An Introduction to Operating Syetems. Addison-Wesley Publishing Company, second edition, 1990.

[6] John R.Graham. *Solaris 2.x: internals and Architecture*. McGraw-Hill, 1995.

[7] Stuart E Madnick, John J Donovan. Operating Systems. McGraw-Hill, 1974

[8] Scott Maxwell. *Linux Core Kernel Commentary*. Coriolis Group, 2000.

[9] Jing Meng. *Operating System principles* (in Chinese). Beijing, China: Tsinghua University Press, 2000.

[10] Jing Meng. *A Course of Computer Operating Systems* (in Chinese). Beijing, China: Renmin University of China, 1997.

[11] Rajeev Nagar. Windows NT File System Internals, A Developer's Guide. O'REILLY, 1997.

[12] Gary J. Nutt. *Operating Systems: A Modern Perspective*. Addison-Wesley, second edition, 2000.

[13] Abraham Silberschatz, and Peter Baer Galvin. *Operating System Concepts*. Addison-Wesley, fifth edition, 1998.

[14] William Stalling. *Operating systems: internals and Design Principles*. Prentice Hall, 1998.

[15] Andrew S. Tanenbaum, and Albert S. Woodhull. *Operating Systems Design and Implementation*.

Prentice Hall, 1997.

[16] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.