Chapter 14 Epilogue: A Look Back at the Journey
(Revision 5)

This book has been a journey "inside the box." It is worthwhile to look back at each of the topics we covered in the book and see the inter-relationship between system software and hardware.

14.1 Processor Design

We saw that high-level language constructs played a key role in the design of the instruction set and related architectural features of the processor. Table 14.1 summarizes the HLL constructs and features that give rise to the architectural decisions in a processor.

HLL Construct/Feature	Architectural features
Expressions	Arithmetic/logic instructions, addressing modes
Data types	Different operand granularity, multiple precision arithmetic and logic instructions
Conditional statements, loops	Conditional and unconditional branch instructions
Procedure call/return	Stack, link register
System calls, error conditions	Traps, exceptions
Efficient execution of HLL programs (expressions, parameter passing, etc.)	General Purpose Registers

Table 14.1: HLL and Architectural Features

Once the architecture is fully specified, we saw that there were several implementation choices such as a simple design and a pipelined design. It is fair to say that we have only scratched the surface of the exciting area of micro-architecture in this book. We hope that we have perked the readers' interest sufficiently for them to dig deeper into this fascinating area.

14.2 Process

The process concept served as a convenient abstraction for remembering all the relevant details of a running program. We know that the processor can run only one program at a time. The operating system gives the illusion of each process having its own processor to run on. To facilitate the operating system to give this illusion, the architecture provides mechanisms, namely, *trap* and *interrupt*, for taking control back from the currently running program. It is via these mechanisms that the operating system gets control of the processor to make a scheduling decision as to which user program to run next. We saw several algorithms that an operating system may employ to make such a scheduling decision. We also reviewed the data structures needed by the operating system for implementing such algorithms. The trap mechanism also allows a user level program to avail itself of system provided utilities (such as file systems and print capability) to enhance the functionality of the program, via system calls.

The processor scheduling component of a production operating system such as Linux or Windows XP is a lot more complicated than the simplistic view provided in this textbook. The intent in the book was to provide enough of an exposure for the reader so that it takes the mystery out of how to build the scheduling subsystem of an operating system.

14.3 Virtual Memory System and Memory Management

The memory system plays a crucial role in determining the performance of a computer system. Therefore, significant attention is paid in this textbook to understand the interplay between the memory management component of the operating system and the architectural assists for memory management. We reviewed a number of architectural techniques such as fence register, bounds registers, base and limit registers, and paging from the point of view of supporting the functionalities of the memory manager such as memory allocation, protection and isolation, sharing, and efficient memory utilization. We also saw the need for a privileged mode (kernel mode) of execution to be supported in the architecture for the operating system to set up the memory area before scheduling a program to run on the processor. We saw several operating systems issues (such as page replacement policies and working set maintenance) that are crucial for achieving good performance for application programs.

The main lesson we learned from both managing the processor (via scheduling algorithms) and the memory (via memory management policies) is the need for the operating system to be as quick as possible in its decision-making. The operating system should provide the resources needed by a program quickly and get out of the way!

The interesting aspect of this exploration was the revelation that very little additional hardware support was needed to realize even the most sophisticated memory management schemes (such as paging) and very efficient page replacement and working set maintenance algorithms. This serves once again to emphasize the importance of understanding the partnership between system software and hardware.

14.4 Memory Hierarchy

Related to the memory systems is the topic of memory hierarchy. Program locality (which naturally leads to the concept of working set of a program) is the key characteristic that allows programs having a large memory footprint to achieve good performance on modern computer systems. The architectural feature that exploits this property is memory hierarchy. The concept of caches pervades all aspects of system design (from web caches to processor caches). We saw how to design data caches in hardware to exploit the locality properties found in programs for instructions and data. We also saw how to design address caches (TLB) to take advantage of locality at the level of pages accessed by a program during execution.

14.5 Parallel System

Parallelism is fundamental to human thinking and therefore to the way we develop algorithms and programs. Trying to exploit this parallelism has been the quest both for the systems software and computer architects from the early days of computing. With increasing levels of integration, boxes are now housing multiple processors. In fact, the current buzz in single chip processors is *multicore*, namely, having multiple processors on a single piece of silicon.

Given these trends, it is imperative for computer scientists to understand the system software support needed for parallel programming and the corresponding hardware assists. We explored a number of topics from the operating systems point of view such as supporting multiple threads of control within a single address space, synchronization among these threads, and data sharing. Correspondingly, we investigated the architectural enhancements ranging from an atomic read-modify-write primitive in a sequential processor to cache coherence in a symmetric multiprocessor.

This topic is a fertile one and as can be imagined it is worthy of further exploration. We hope we have kindled enough interest in the reader to goad him/her to explore further.

14.6 Input/Output Systems

A computer system is useless unless it can interact with the outside world. From the hardware standpoint, we saw how we can interface devices (simple as well as sophisticated) to the computer system through techniques such as programmed I/O and DMA. The hardware mechanism of interrupt was a key to grabbing the attention of the processor. We also saw how the technique of memory mapped I/O makes integrating I/O subsystems seamless, requiring no special enhancement to the ISA of the processor. Mirroring a device controller that interfaces a device to the processor, software modules in the OS called device drivers and interrupt handlers allow manipulation of the devices connected to the system for input/output.

Two particular I/O subsystems deserve additional attention, namely, disk and network. The former is the vehicle for persistent storage of information inside a box, and the latter is the vehicle to talk to the outside world.

14.7 Persistent Storage

File system is an important component of the operating system and it is not an exaggeration to say that it most likely represents the most number of lines of code for a single subsystem in any production operating system. Due to its inherent simplicity a file is a convenient way to represent any device for the purposes of input/output from a program. In particular, files stored on media such as a disk allow persistence of information beyond the life of the program. We studied the design choices for file systems including naming and attributes related to a file. Focusing on disk as the medium, we explored space allocation strategies, disk scheduling strategies, and organization of files on the disk (including data structures in the operating system).

14.8 Network

Today, with the advent of the Internet and WWW, connection of the computer system to the outside world is taken for granted. Therefore, we devoted quite some time to understanding issues in networking from both the system software point of view and the hardware. On the hardware side, we looked at the network gear that defines our cyber environment today such as NIC, hub, switch, and routers. From the system software point of view, we understood the need for a protocol stack that comprises the transport, network, and data link layers. We learned the bells and whistles therein such as checksums, windowing, and sequence numbers. For example, to combat data loss due to packets being mangled on the wire, error correction codes are used. To decouple the application payload from hardware limitations such as fixed packet size, and to accommodate out of order delivery of packets, scatter/gather of packets is incorporated in the protocol stack. Packet loss en route from source to destination is overcome using end-to-end acknowledgement in the protocol.

Networks and network protocols are fascinating areas of further study. Therefore, we hope the network coverage in this book has whetted the appetite of the reader to learn more.

14.9 Concluding Remarks

Overall, the field of system architecture, the meeting point between hardware and software is an intriguing and fascinating area. This area is unlikely to dry up for a long time since the application of computing in everyday life is growing by leaps and bounds. To keep pace with this growth in demand, the system architect has to continuously innovate and produce new computing machines that are faster, cheaper, consume less power, and offer more services. We hope this textbook serves as a launching pad for future system architects.