Operating Systems: Internals and Design Principles (Eighth Edition) notes

June 24, 2025

”An operating system exploits the hardware resources of one or more processors to provide a set of services to system users. The OS also manages secondary memory and I/O devices on behalf of its users. Accordingly, it is important to have some understanding of the underlying computer system hardware before we begin our examination of operating systems.”

architecture, interrupts, and memory

CPU - control operations, perform data processing, exchanges data with memory

- PC: typically holds the address of the next instruction to be fetched
- IR: fetched instructions loaded here
- MAR: address in memory for the next read or write
- MBR: data to be written into memoroy or receives the data read
- I/O AR
- I/O BR
- execution unit

General Purpose Processors are central processing units (CPUs) designed to handle a wide variety of tasks rather than being specialized for one. These processors are the core component of most personal computers, laptops, and servers. Their versatility allows them to run multiple types of software and handle diverse operations such as word processing, internet browsing, gaming, and programming. Because they are not tailored to a specific function, they are typically less efficient than specialized processors (like GPUs or DSPs) for narrowly defined tasks. However, their ability to adapt makes them ideal for general computing environments.

Microprocessors are a specific type of processor that integrates the entire CPU onto a single chip or integrated circuit (IC). The term is often used interchangeably with general purpose processors, especially in the context of personal computing. Microprocessors contain arithmetic, logic, control, and memory interface circuits all in one chip, which significantly reduces the physical size of the computing unit and increases processing speed. First introduced in the 1970s, microprocessors revolutionized computing by enabling the development of affordable and compact personal computers. Examples include the Intel 8085, the ARM Cortex series, and modern CPUs found in desktops and laptops.
Multiprocessors, on the other hand, refer to computer systems that use two or more processors (or cores) to work in parallel. These processors may be on the same chip (multi-core processors) or on separate chips working together. The goal of a multiprocessor system is to improve performance, reliability, and fault tolerance by dividing tasks among multiple processors. Such systems are commonly used in high-performance computing environments, servers, and modern personal computers to handle multitasking and

complex computational workloads more efficiently. Multiprocessing allows systems to run several processes simultaneously, making them highly effective for tasks that require substantial computing power, such as scientific simulations, data analytics, and video rendering.
A **core** is an individual processing unit within a CPU chip. Originally, a processor chip had just one core, meaning it could execute only one instruction thread at a time. But modern CPU chips often have multiple cores, allowing them to run multiple processes in parallel.

Each core can be thought of as a mini-CPU inside the chip, with its own arithmetic logic unit (ALU), control unit, and often its own cache memory. For example, a quad-core processor has four cores on a single chip, meaning it can handle four tasks simultaneously, improving multitasking and performance for parallelizable tasks.

A **hardware thread** refers to the ability of a single CPU core to execute multiple instruction sequences (or threads) concurrently. This is made possible by a techonlogy called **Simultaneous Multithreading (SMT)** - Intel calls their version **Hyper-Threading**.

In SMT, an single physical core is enhanced to appear as two execution units, each capable of processing its own thread. These are not full cores - they share the core's resources (like cache and execution units), but they allow the core to better utilized idl parts of its circuitry when one thread is waiting (e.g., for memory access).

So, one physical core with SMT can run two hardware threads at once.

A **logical processor** is what the operating system sees as an individual processing unit, even if it is not a full physical core. Each hardware thread is represented to the OS as a logical processor.

- If you have a CPU with 4 physical cores and each core supports 2 threads (via SMT), your system will see 8 logical processors.
- If SMT is turned off, the number of logical processors is equal to the number of physical cores.

Logical processors allow modern operating systems to schedule and manage more task simultaneously, improving responsiveness and performance, especially in multithreaded application.

graphical processing unit (GPU)

single instruction multiple data (SIMD)

digital signal processor (DSP)

codecs

SoC (handhelds)

program execution consists of repeating the process of instruction fetch and instruction execution. instruction execution may involve several operations and depends on the nature of the instruction.

an instruction contains bits that specify the action the processor is to take:

- processor-memory
- processor-i/o
- data processing
- control

consider a simple computer which has 16-bit instructions...the processor contains a register called the accumulator...the instruction format consists of 4-bits for the opcode and 12-bits that can be directly addressed

most modern processors include instructions that contain more than more than one address. also, an instruction may specify an i/o operations instead of memory reference

classes of interrupts

- program
- timer
- i/o
- hardware failure

interrupts primarily improve processor utilization
consider the following:
a computer operating at 1GHz and a HDD $\frac{7200 revolutions}{minute}$ and a half-track rotation time of 4ms

the user program does not have to contain special code to accommodate interrupts; the processor and OS are responsible for suspending the user program and then resuming it at that same point

PSW is the minimum information required, location of next instruction to be executed, other register, ect...

interrupt processing

1. the device issues an interrupt signal to the processor.
2. the processor finishes execution of the current instruction before respongin to the interrupt.
3. the processor tests for a pending interrupt request, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. the acknowledgment allows the device to remove its interrupt signal.
4. the processor next needs to prepare to transfer control to the interrupt routine. to begin, it saves information needed to resume the current program at the point of interrupt. the minimum information required is the program status word (PSW) and the location of the next instruction to be executed, which is contained in the program counter (PC). these can be pused onto a control stack.
5. the processor then loads the program counter with the entry location of the interrupt-handling routine that will respond to this interrupt. depending on the computer architecture and OS design, there may be a single program, one for each type of interrupt, or one for each device and each type of interrupt. if there is more than one interrupt-handling routing, the processor must determine which one to invoke. this information may have been included in the original interrupt signal, or the processor may have to issue a request to the device that issued the interrupt to get a response that contains the needed information. Once the program counter has been loaded, the processor proceeds to the next instruction cycle, which begins with an instruction fetch. because the instruction fetch is detemined by the contents of the program counter, control is transferred to the interrupt-handler program. the execution of this program results in the following operations:
6. at this point, the program counter and PSW relating to the interrupted program have been saved on the control stack. however, there is other information that is considered part of the state of the executing program. in these registers may be used by the interrupt handler. so all of these values, plus any other state information, need to be saved. typically, ine interrupt handler will begin by saving the contents of all registers on the stack. other state information that must be saved is discussed later. in this case, a user program is interrupted after the instruction at location N. the contents of all of the registers plus the address of the next instruction N+1, a total of M words, are pushed onto the

control stack. the stack pointer is updated to point to the new top of stack, and the program counter is updated to point to the beginning of the interrupt service routine.

7. The interrupt handler may now proceed to process the interrupt. This includes an examination of the staus information relating to th I/o operation or other even that caused an interrupt. It may alos involve sending additional commands or acknowledgments to the I/O device.

8. When interrupt processing is complete, the saved register values are retrived from the stack and restored to the registers.

9. The final act is to restore the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

sequential vs nested interrupts
priority

design contraints of memory really boils down to how much? how fast? and how expensive?

order of magnitude (time):

- decisecond: 10^{-1}
- centisecond: 10^{-2}
- millisecond: 10^{-3}
- microsecond: 10^{-6}
- nanosecond: 10^{-9}
- picosecond: 10^{-12}
- femtosecond: 10^{-15}
- attosecond: 10^{-18}
- zeptosecond: 10^{-21}
- yoctosecond: 10^{-24}

faster access time means a greater cost per bit
greater capacity means a smaller cost per bit greater capacity means slower access speeds

so we imploy a memory hierarchy to not soley rely on one memory component

as you go down the hierarchy ↓

- decreased cost per bit
- increasing capacity
- increasing access times
- decreased frequency of access to the memory by the processor

consider the following: two levels of memory the first has 1000B and the access time is 0.1 micro seconds...the second level has 100,000B and the access time is 1 microsecond...our rule is if the data we want is in the first level we access it directly and if it is in the second level we tranfer it to level one and access it there...for simplicity ignore the time it takes for the processor to determine if the data is in level one or two.

suppose 95% of memory access are found in the cache ($H = 0.95$) then the average time to access a byte can be expressed as: $(0.95)(0.1\mu s) + (0.05)(0.1\mu s + 1\mu s) = 0.095 + 0.055 = 0.15\mu s$ the result is closer to the access time of level 1.

so the strategy of using two levels works in principle, but only if the conditions of the memory hierarchy apply...by employing a variety of technologies, a spectrum of memory systems exists that satisfies the first three conditions...fortunately the last condition is also generally valid.

the basis for the validity of the last condition (decreased frequency of access to the memory by the processor) is the principle known as **locality of reference**.

During the execution of a program, memory references by the processor, for both instructions and data, tend to cluster

consider loops and subroutines...there will be repeated references to a small set of instructions and similarly operations on arrays and tables involve access to a clustered set of data bytes.

over a long period, the clusters change...and over a short period, the processor is primarily working with fixed clusters of memory references...

main memory is usually extended with a higher-speed, small cache. the cache is not usually visible to the programmer or indeed to the processor. it is a device for staging the movement of data between main memory and processor registers to improve performance.

what is a word?
A word is a fixed-sized unit of data used by a computer's processor. It typically represents the amount of data a processor can handle in one operation - such as reading from memory, performing arithmetic, or writing to memory.
what is a block? A block is a larger unit of data primarily in memory mangement and storage systems (like hard drives, SSDs, or RAM). It refers to a chunk of contiguous memory or storage space that is read or written in one operation. Blocks are used to improve efficiency - reading or writing in block reduces overhead compared to handling data byte-by-byte.
SI Prefixes (powers of 10):

- kilo: 10^3
- mega: 10^6
- giga: 10^9
- tera: 10^{12}
- peta: 10^{15}
- exa: 10^{18}
- zetta: 10^{21}
- yotta: 10^{24}

Binary Prefixes (powers of 2):

- kibi: 2^{10}
- mebi: 2^{20}
- gibi: 2^{30}
- tebi: 2^{40}

- pebi: 2^{50}
- exbi: 2^{60}