A Tour of Computer Systems

Computer system

- **Consists of hardware and systems software that work together to run application programs**

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6       return 0;
7   }
```

The ASCII text representation of hello.c

#	i	n	c	l	u	d	e	*SP*	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	*SP*	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	*SP*	*SP*	*SP*	*SP*	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	*SP*	w	o	r	l	d	\	n	"	)	;	\n	*SP*
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	32
SP	*SP*	*SP*	r	e	t	u	r	n	*SP*	0	;	\n	}	\n	
32	32	32	114	101	116	117	114	110	32	48	59	10	125	10	

Ref: Bryant and O'Hallaron, Computer Systems: A Programmer's , Third Edition

Programs Are Translated by Other Programs into Different Forms

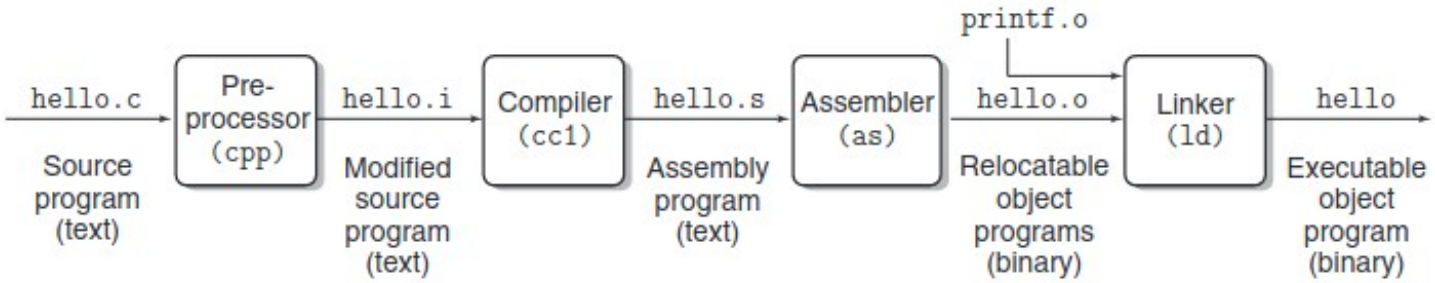


Figure 1.3 The compilation system.

ranslation from source file to object file is performed

```
linux> gcc -o hello hello.c
```

by a compiler driver:

- **Translation from source file to object file is performed by a compiler**
  - gcc compiler driver reads the source file hello.c and translates it into an executable object file hello.
- **The translation is performed in the sequence of four phases**
- **The programs that perform the four phases (preprocessor, compiler, assembler, and linker)**

- **Preprocessing Phase:**
  - The preprocessor (cpp) modifies the original C program according to directives that begin with the '#' character
  - The result is another C program, typically with the .i suffix.(hello.i)
- **Compilation phase.**
  - The compiler (cc1) translates the text file hello.i into the text file hello.s
  - I.e it contains an assembly-language program

```
1   main:
2     subq    $8, %rsp
3     movl    $.LC0, %edi
4     call    puts
5     movl    $0, %eax
6     addq    $8, %rsp
7     ret
```

- **Assembly phase.**
  - The assembler (as) translates hello.s into machine-language instructions, packages them in a form known as object program, and stores the result in the object file hello.o.

- **Linking phase.:**
  - The process of rearranging the existing code and filling missed code.
  - Example : The printf function(which is part of the standard C library) resides in a separate precompiled object file called printf.o, which must somehow be merged with our hello.o program.

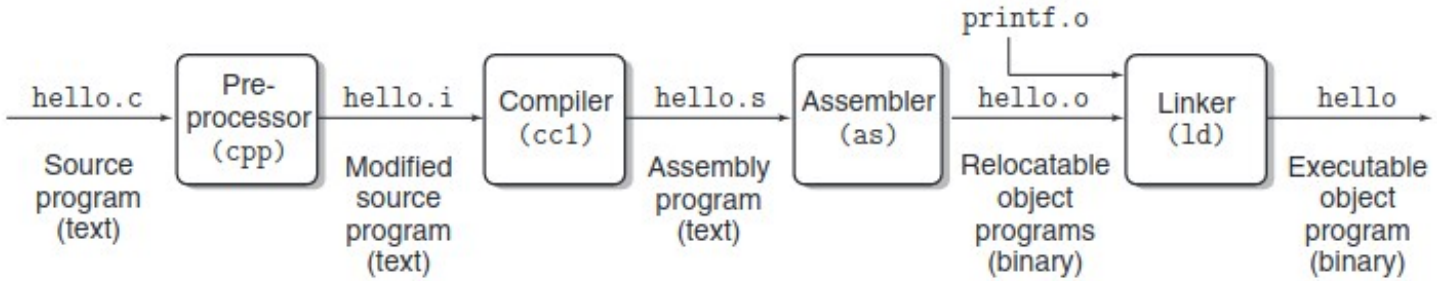


Figure 1.3 **The compilation system.**

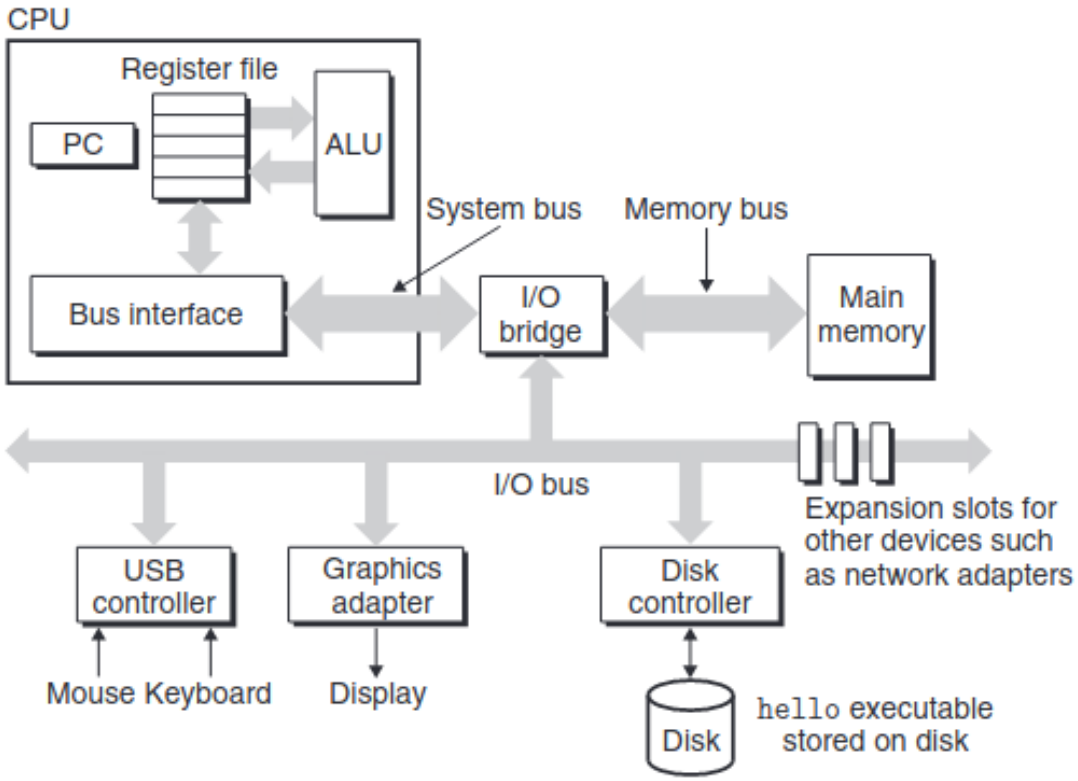
```
linux> gcc -o hello hello.c
```

- **Preprocessing**
  - cc -E abc.c
- **Compilation**
  - cc -S abc.c
    - Output : .s file
-

- **Assembly**
  - cc -c abc.c
    - Output : .o file
- **Linking**
  - cc -o abc abc.c

Processors Read and Interpret Instructions Stored in Memory

- **Hardware Organization of a System**
  - **Buses**
  - **I/O Devices**
  - **Main Memory**
  - **Processor**



Hardware Organization of a System

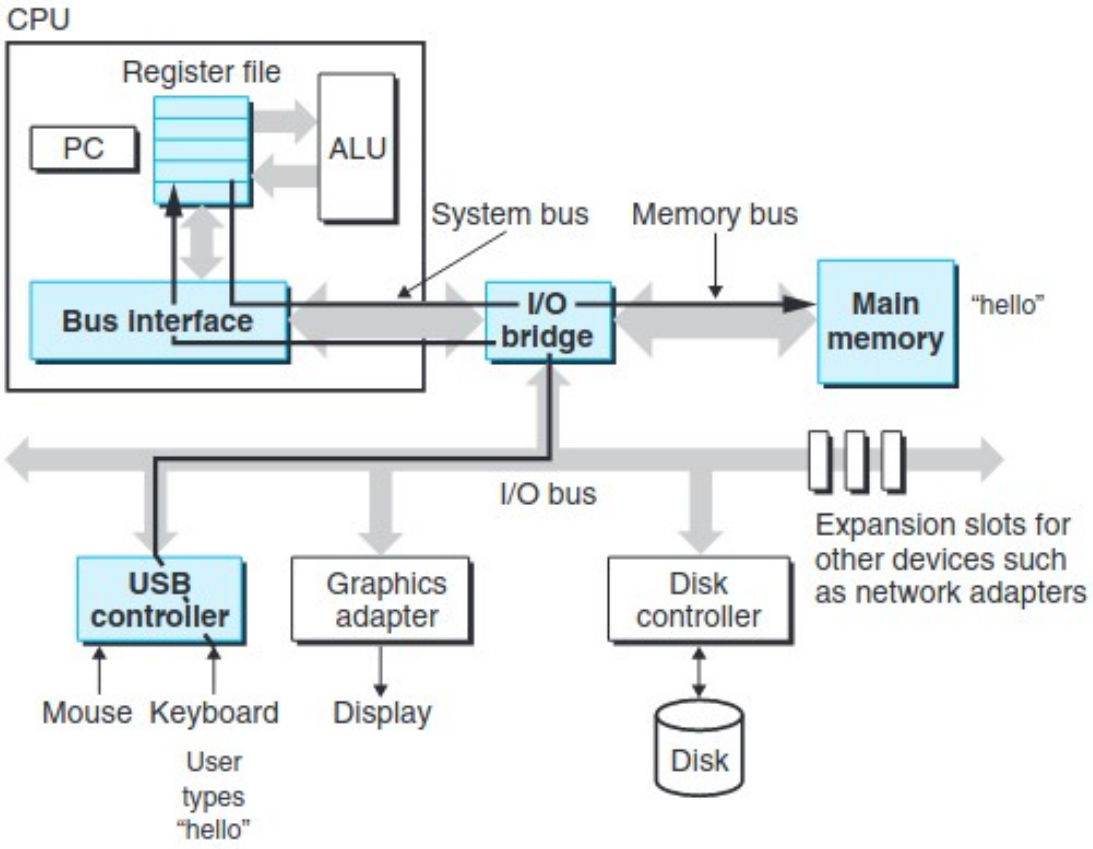
- **Buses**
  - Carry bytes of information back and forth between the components
  - Collection of electrical conduit
  - Transfer fixed-size chunks of bytes known as words.
    - Word size is either 4 bytes (32bits) or 8 bytes (64 bits)
- **I/O Devices**
  - System's connection to the external world.
  - Each I/O device is connected to the **I/O bus** by either a controller or an **adapter**
  - Example?
- **Main Memory**
  - Main memory is a temporary storage device that holds both a program
  - Data is manipulates in main memory while the processor is executing the program
  - Memory is organized as a linear array of bytes(Logically)

Hardware Organization of a System

- **Processor**
  - Interprets (or executes) instructions stored in main memory.
  - At any point in time, the program counter (PC) points at (contains the address of) some machine-language instruction in main memory
  - The processor reads the instruction from memory pointed at by the program counter (PC), interprets the bits in the instruction, performs some simple operation dictated by the instruction, and then updates the PC to point to the next instruction.
    - Simple operations that the CPU might carry out at the request of an instruction
      - Load
      - Store
      - Operate
      - Jump
  -

Running the hello Program

```
:~$ cc hello.c
:~$ ./a.out
Hello, World!
```



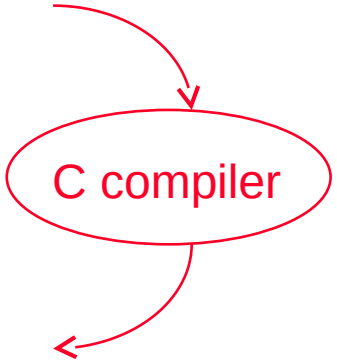
Below the Program

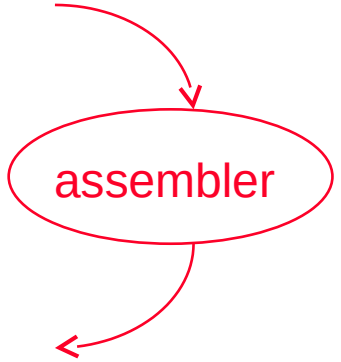
- High-level language program (in C)

```
swap (int v[], int k)
. . .
```

- Assembly language program (for MIPS)

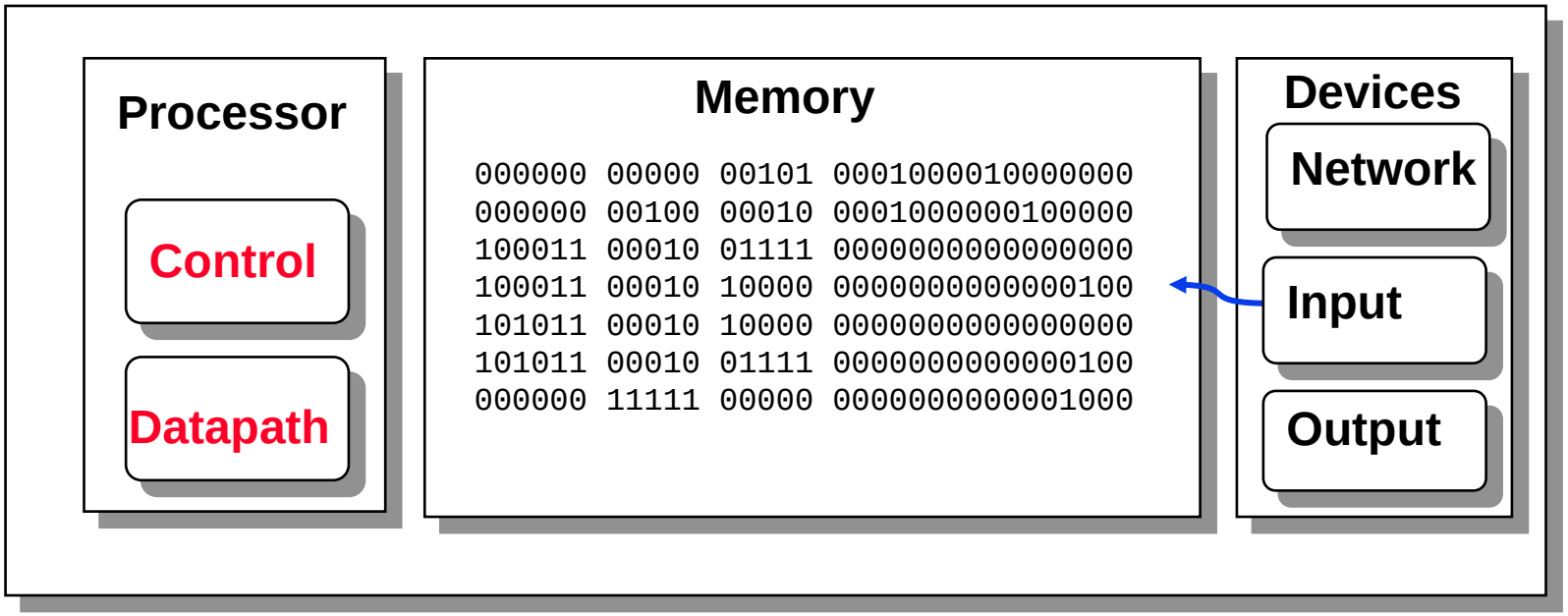
```
swap:   sll     $2, $5, 2
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
        sw      $16, 0($2)
        sw      $15, 4($2)
        jr      $31
```





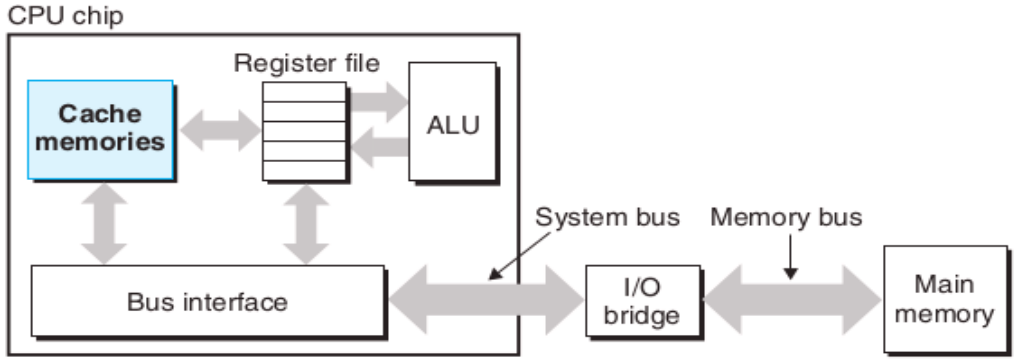
- Machine (object) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```


Processor
Control
Datapath
Memory
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
Devices
Network
Input
Output

Caches

- From a programmer's perspective, Copying is overhead that slows down the "real work" of the program.
- System designers include smaller faster storage devices called **cache** memories.
- Caches serve as temporary staging areas for information that the processor is likely to need in the near future.
- An **L1 cache** on the processor chip holds tens of thousands of bytes and can be accessed nearly as fast as the register file.
- A larger L2 cache with hundreds of thousands to millions of bytes is connected to the process



Memory hierarchy.

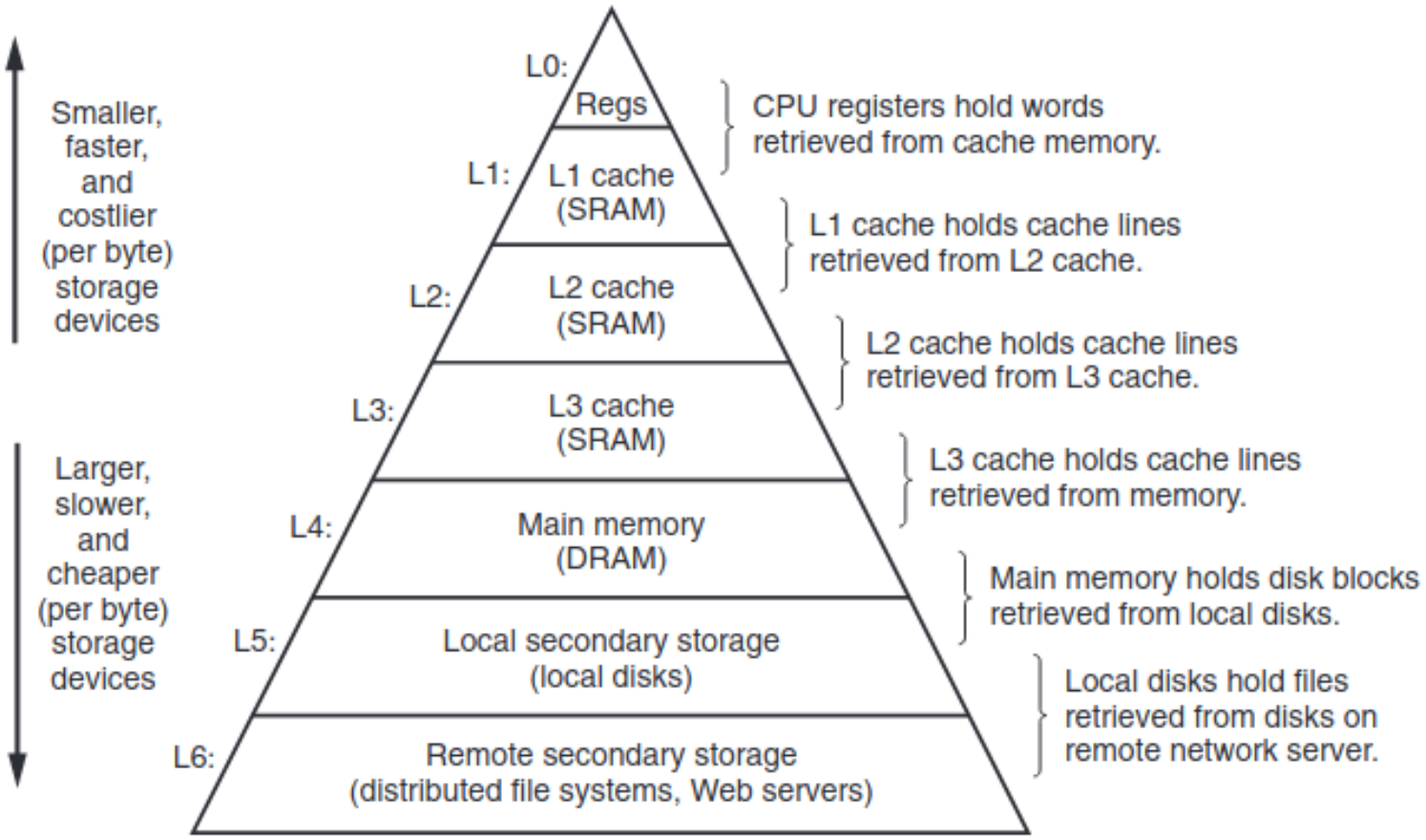
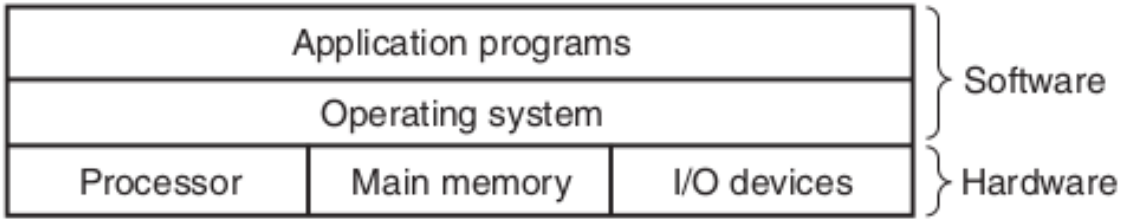
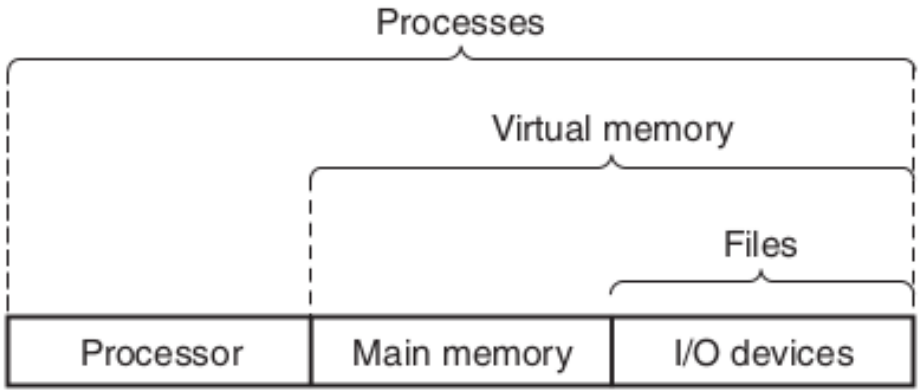


Figure 1.9 **An example of a memory hierarchy.**

The Operating System Manages the Hardware

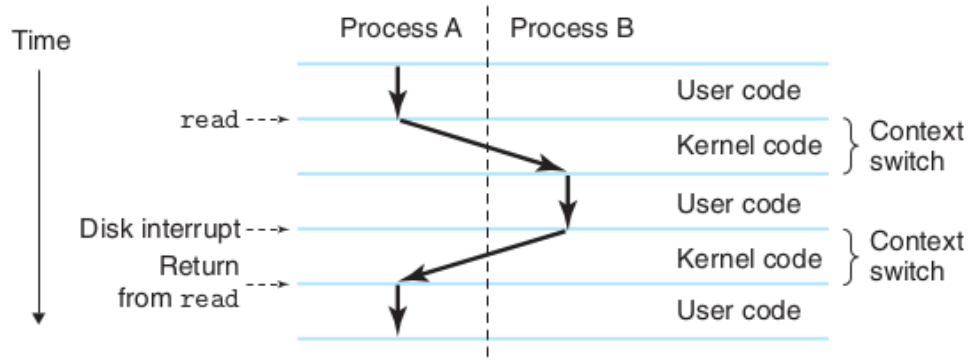
- **Operating system :** A layer of software interposed between the application program and the hardware.
- **Fundamental abstractions of OS**
  - **Processes, virtual memory, and files.**





Processes

- A **process** is the operating system's abstraction for a running program.
- **Multiple processes** can run concurrently on the same system, and each process appears to have exclusive use of the hardware
- Uniprocessor system containing a single CPU.



- **Multicore** processors can execute several programs simultaneously
  - single CPU can appear to execute multiple processes concurrently.
- A uniprocessor system can only execute the code for a single process.
- Transfer control from the current cess to some new process is **ntext switching**

Ref: Bryant and O'Hallaron, Computer Systems: A Programmer's , Third Edition

Processes

- **Threads**