CS 370: OPERATING SYSTEMS
[PROCESSES]

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT   COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Some lingering confusion between the kernel and OS
- Kernel-user mode transitions: how expensive?
- What do computer viruses target?
- What is a GPU classified as?
  - Co-processors similar to TPUs
- If files from the OS are in the RAM, when there is a power loss will the files be gone forever?
- What is disk latency?
- What is a clock? How does it work?

COLORADO STATE UNIVERSITY   Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT   PROCESSES   L3.2

2

Topics covered in this lecture

- Processes
- A process in memory
- Process Control Blocks
- Interrupts & Context switches
- Operations on processes
  - Creation

3

INTERRUPT VECTOR TABLE

4

When an interrupt, processor exception or system call trap occurs …

- How does the processor know what code to run?
- The processor has a special register that points to an area of kernel memory called the **interrupt vector table**
- The hardware determines which device caused the interrupt, if the trap instruction was executed, or what exception condition occurred
  - Thus, the hardware can select the right entry from the interrupt vector table and invoke the appropriate handler
- The format of the interrupt vector table is processor-specific

The interrupt vector table on the x86

- Entries 0 – 31: are for different types of processor exceptions
  - anything related to arithmetic overflow, e.g.: divide-by-zero
- Entries 32 – 255 are for different types of interrupts
  - Timer, keyboard, etc.
- By convention, entry-**64 points to the system call trap handler**

What about kernel to user mode transitions? When do these happen?

- New process
- Resume after an interrupt, processor exception, or system call
- Switch to a different process
- User-level upcall
  - Most OS provide user programs with the ability to receive *asynchronous* notification of events

There are two approaches to improving performance

- Determine component **bottlenecks**
  - Replicate: Horizontal scaling
  - Improve: Vertical scaling

To replicate or improve?

"*If one ox could not do the job, they* [pioneers] *did not grow a bigger ox, but used two oxen.*"

– Admiral Grace Murray Hopper
Computer Software pioneer

"*If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?*"

— Seymour Cray
Computer Hardware pioneer

11

Process

- The oldest and most important abstraction that an operating system provides
- Supports the ability to have (*psuedo*) **concurrent** operation
  - Even if there is only 1 CPU

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA | COMPUTER SCIENCE DEPARTMENT | PROCESSES | L3.12

12

What is a process?

- A process is the **execution** of an application program with restricted rights
  - It is the abstraction for protected execution provided by the kernel

All modern computers do several things at a time

- Browsing while e-mail client is fetching data
- Printing files while burning a CD-ROM

Multiprogramming

- CPU **switches** from process-to-process quickly
- Runs each process for a few milliseconds

Multiprogramming and parallelism

- At any instant of time, the CPU is running **only one** process
- In the course of 1 second, it is working on **several** of them
- Gives the **illusion** of parallelism
  - Psuedoparallelism

A process is the unit of work in most systems

- Arose out of a need to **compartmentalize** and control *concurrent* program executions
- A process is a program in execution
- Essentially an **activity** of some kind
  - Has a program, input, output, and a state

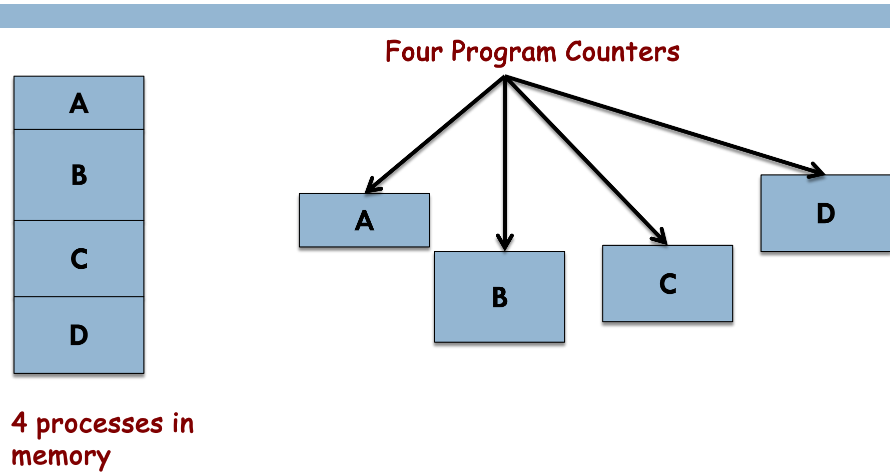
A process is just an instance of a program [1/2]

- In much the same way that an object is an instance of a class in object-oriented programming
- Each program can have zero, one or more processes executing it
- For each instance of a program, there is a process with its *own* copy of the program **in memory**

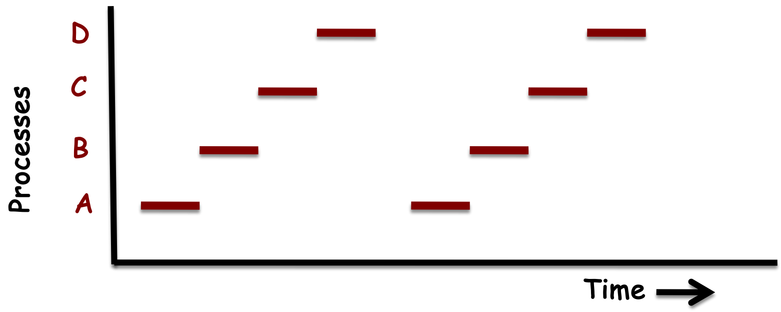
A process is just an instance of a program [2/2]

- Conceptually each process has its own **virtual CPU**
- In reality, the CPU switches back-and-forth from process to process
- Processes are <u>not affected</u> by the multiprogramming
  - Or *relative speeds* of different processes

An example scenario: 4 processes



Example scenario: 4 processes



- At any instant <u>only one</u> process executes
- *Viewed over a long time*, all processes have made **progress**

21

PROGRAMS AND PROCESSES

22

Programs and processes

- Programs are **passive**, processes are **active**
- The difference between a program and a process is subtle, but crucial

Analogy of a culinary-minded computer scientist baking cake for his daughter

Analogy	Mapping to real settings
Birthday cake recipe	Program (algorithm expressed in a suitable notation)
Well-stocked kitchen: flour, eggs, sugar, vanilla extract, etc	Input Data
Computer scientist	Processor (CPU)

- **Process is the activity of**
  1. Baker reading the recipe
  2. Fetching the ingredients
  3. Baking the cake

Scientist's son comes in screaming about a bee sting

- Scientist records *where he was* in the recipe
  - State of current process is saved
- Gets out a first aid book, follows directions in it

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PROCESSES L3.25

25

In our example, the scientist has switched to a higher priority process ...

- FROM Baking
  - Program is the cake recipe
- TO administering medical care
  - Program is the first-aid book
- When the bee sting is taken care of
  - Scientist **goes back to where he was** in the baking

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PROCESSES L3.26

26

Key concepts

- Process is an **activity** of some kind; it has a
  - Program
  - Input and Output
  - State
- Single processor may be shared among several processes
  - **Scheduling algorithm** decides when to stop work on one process, and start work on another process

HOW A PROGRAM BECOMES A PROCESS

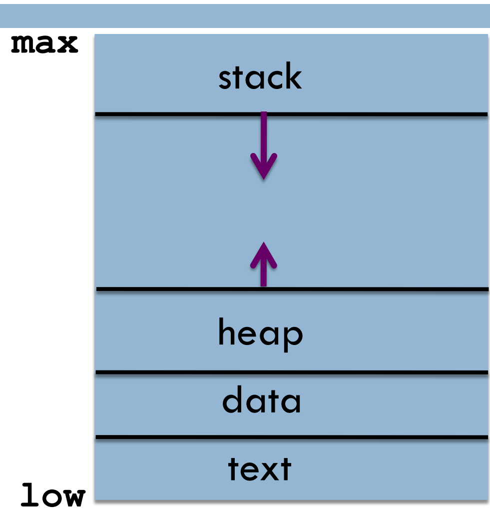
The journey from code to a becoming a process [1/2]

- Programmer types code in some high-level language
- A compiler converts that code into a sequence of machine instructions and stores those instructions in a file
  - Called the program's **executable image**
  - Compiler also defines any static data the program needs, along with its initial values, and includes them in the executable image

The journey from code to a becoming a process [2/2]

- To run the program, the kernel copies the instructions and data from the executable image into physical memory
- The kernel sets aside memory regions
  - The execution **stack**, to hold local variables during procedure calls
  - The **heap**, for any dynamically allocated data structures the program might need
- Of course, to copy the program into memory, the kernel itself must already be in memory, with its own stack and heap

A process in memory



{Function parameters, return addresses, and local variables}

{Memory allocated dynamically during runtime}

{Global variables}

{Program code}

Memory conservation

- Most operating systems reuse memory wherever possible
- The OS stores only a single copy of a program's instructions
  - Even when multiple copies of the program are executed at the same time
- Even so, a **separate copy** of the program's data, heap, and stack are needed

How a program becomes a process

- Allocation of memory is *not enough* to make a program into a process
- Must have a process ID
- OS tracks IDs and process **state**s to orchestrate system resources

Program in memory [1/2]

- Program image *appears* to occupy **contiguous** blocks of memory
- OS **maps** programs into non-contiguous blocks

Program in memory [2/2]

- Mapping divides the program into equal-sized pieces: **pages**
- OS loads pages into memory
- When processor references memory on page
  - OS looks up page in table, and loads into memory

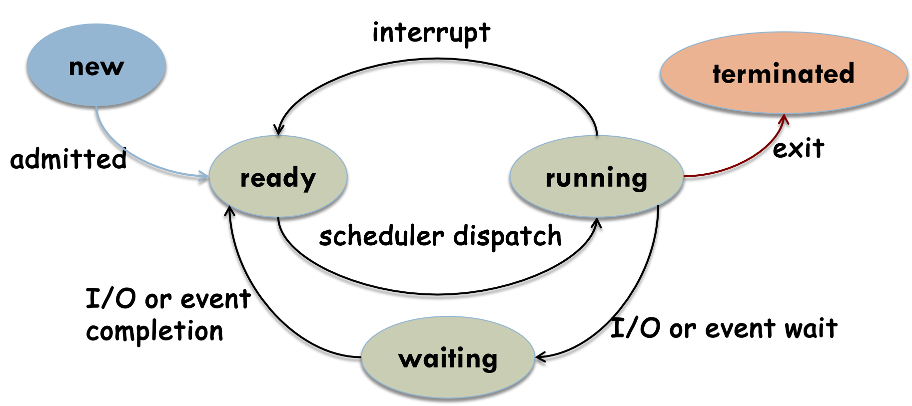
Advantages of the mapping process

- Allows **large** logical address space for stack and heap
  - **No physical memory used** unless actually needed
- OS hides the mapping process
  - Programmer views program image as **logically contiguous**
  - Some pages may not reside in memory

Finite State Machine

- An initial **state**
- A set of possible **input** events
- A finite number of states
- **Transitions** between these states
- Actions

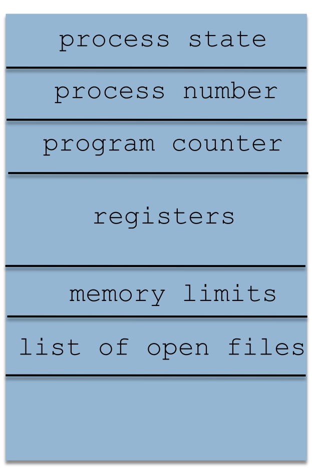
Process state transition diagram: When a process executes it changes state



How does the OS track processes?

- Via a data structure called the **process control block**, or **PCB**
- The PCB stores all the information the OS needs about a particular process
  - Where it is stored in memory, where its executable image resides on disk, which user asked it to execute, what privileges it has, etc.
- The set of the PCBs defines the current state of the OS

Each process is represented by a process control block (PCB)



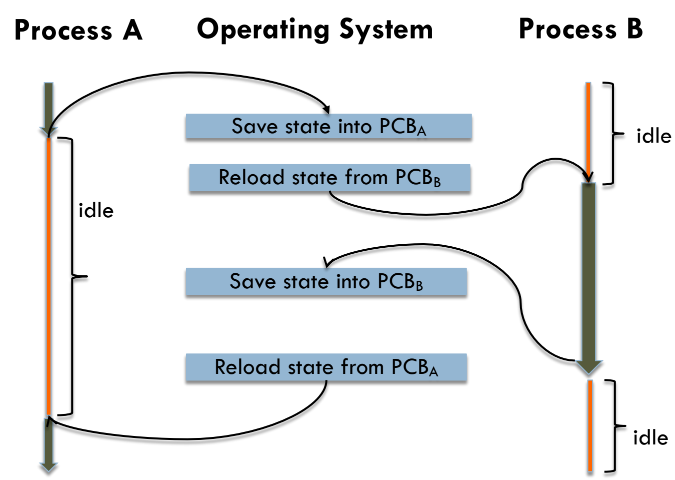
PCB is a **repository** for *any* information that *varies* from process to process.

Where is the PCB stored?

- Since PCB contains the critical information for the process
  - It must be kept in an area of memory protected from normal user access
- Maintained in kernel memory

41

An example of CPU switching between processes



42

THERE'S AN APP FOR THAT!

What can be at the user level, should be.

- Allow user programs to create and manage their own processes
- If creating a process is something a process can do, then anyone can build a new version of any of these applications
  - **Without recompiling the kernel** or forcing anyone else to use it
- Instead of a single program that does everything, we can create specialized programs for each task, and mix-and-match what we need
  - There's an app for that!

INTERRUPTS & CONTEXTS

Interrupts and Contexts

- Interrupt causes the OS to **change** CPU from its current task to run a kernel routine
- Save current context so that ***suspend*** and ***resume*** are possible
- Context is represented in the **PCB**
  - Value of CPU registers
  - Process state
  - Memory management information

Context switch refers to switching from one process to another

① **Save** state of current process

② **Restore** state of a different process

- Context switch time is pure **overhead**
  - No useful work done while switching

Factors that impact the speed of the context switch

- Memory speed
- Number of registers to copy
- Special instructions for loading/storing registers
- Memory management: Preservation of address space

The contents of this slide-set are based on the following references

- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620.* [Chapter 2].
- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles and Practice, 2nd Edition. Recursive Books. ISBN: 0985673524/978-0985673529. [Chapters 1-2]*
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 3]