Operating System Principles

Introduction and Overview

Slides are adapted from Prof. Bhuvan Urgaonkar

Assumed background

- Algorithms and data structures
- Comfortable programming in C/C++
- Preliminary understanding of computer architecture
  - We will cover some basics in this course

Before we begin …

- Some advice
  - Speak up in class, ask questions
  - Attend all classes
    - Hint: Ideas concerning projects and exams
  - Do all projects on your own!
    - Do not copy or lift code from other sources
  - Take notes in class
  - Read text-book soon after class
    - Even better: read before class and come prepared with questions

Operating Systems: Introduction & Background

What is an operating system?

- Let us begin with an incomplete definition

Software that allows multiple programs to run on the same computer

- Why is such software needed?

Example Scenario

- Imagine a laptop running a browser and an music player
  - **Browser code (fake):**

    ```
         …
    load R1, @100
    load R2, @200
    add R3, R1, R2
         …
    ```

  - **Player code (fake):**

    ```
         …
    load R1, @700
    load R2, @900
    sub R3, R1, R2
         …
    ```

- Can you identify some of the resources that both these programs are using (i.e., sharing)?
  - CPU (including registers R1-R3), main memory, busses
- Could this cause a problem?

Example Scenario

- Consider the following “interleaving” of instructions

```
      …
load R1, @100
load R1, @700
load R2, @900
sub R3, R1, R2
load R2, @200
add R3, R1, R2
      …
```

Save registers for browser somewhere
Restore previously saved registers for player from somewhere

- What’s the problem here?
- What should happen?

Questions, questions, …

- Sharing of CPU
  - Where would the contents of the registers be saved?
    - Answer: Main memory (where data and instructions are also stored)
  - By whom?
  - When should the CPU decide to stop executing the browser and start executing the player?
- What about similar problems due to sharing of other resources – busses, h/w caches, memory, IO devices?

Option #1: Program-driven Sharing

- Let the programs take charge of dealing with sharing-related problems
  - E.g., for CPU registers: Each program saves its registers in some memory locations before the next starts executing; when it is resumed, it loads registers from these memory locations
- Will this work?
- Answer: No, unless care is taken
  - E.g., a program overwrites the memory holding registers for another

Option #1: Program-driven Sharing

- Let the programs take charge of dealing with sharing-related problems
  - E.g., for CPU cycles: Each program executes instructions after which it gives up control to another program
- Will this work?
- Answer: No!
  - What if a program is malicious and doesn’t yield the CPU -> others will “starve”

Option #2: Hardware-driven Sharing

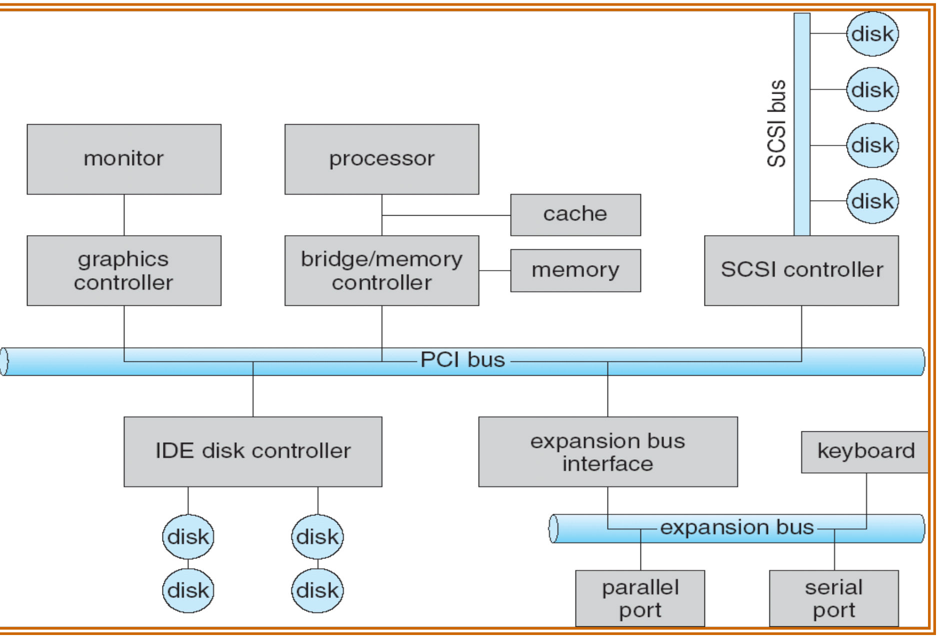
- Let the hardware take charge of dealing with sharing-related problems
    - E.g., for CPU cycles and registers: Let the CPU itself decide when to run which program and save/restore registers
- Will this work?
- Answer: Yes, but …
    - We are stuck with the policy (e.g., scheduling algorithm) implemented in hardware
    - Desirable for some resources (e.g., busses) but not for all

Summary

- Sharing of resources among can cause problems related to
  - Correctness
    - E.g., a program's register contents being corrupted by another
  - Fairness
    - E.g., a program not getting enough CPU cycles
- Program-driven solutions don't suffice; Hardware-driven solutions likely to be inflexible or complex/costly
- Need for a software other than the programs that facilitates correct and fair sharing of hardware resources
  - **The operating system**

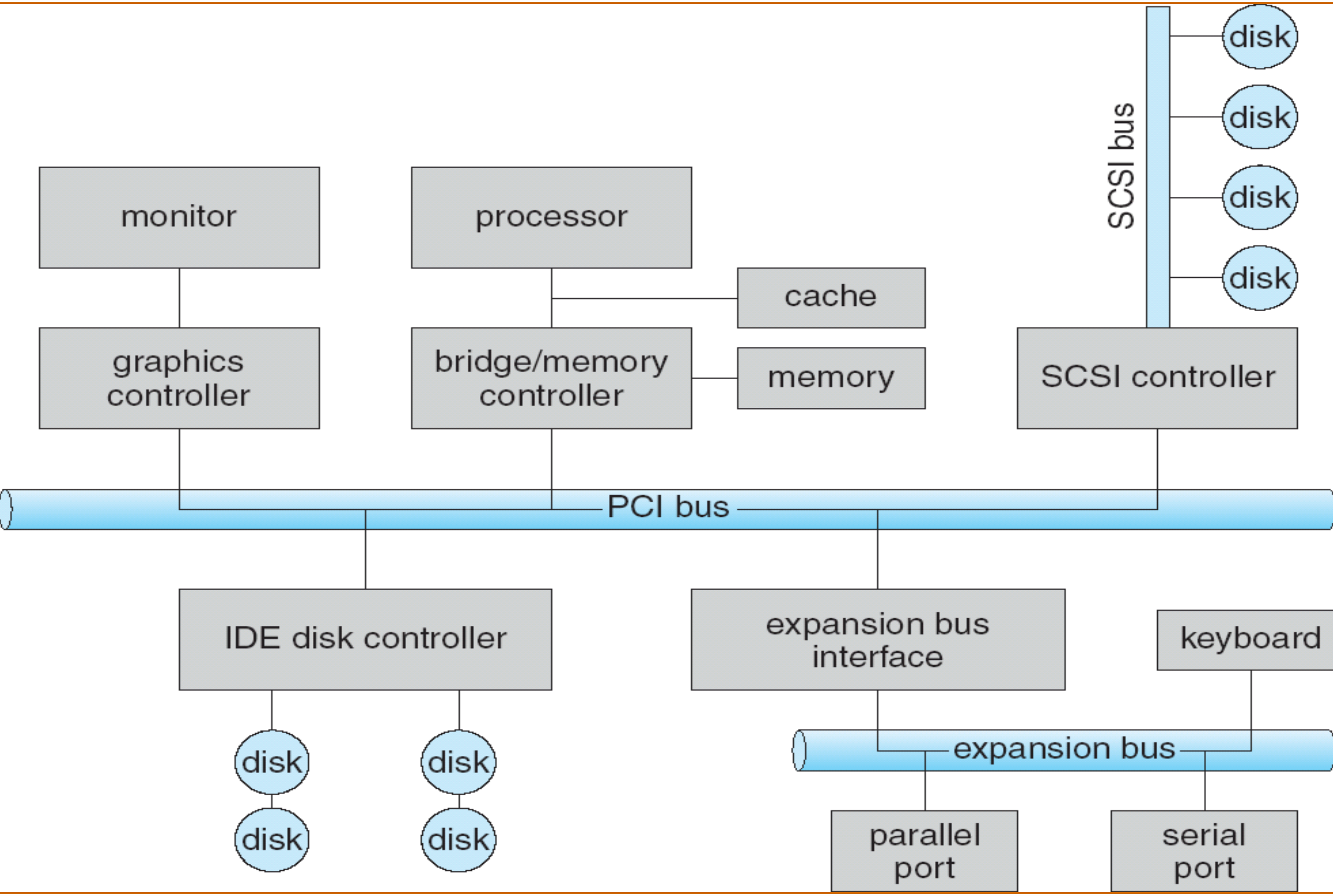
Programs

OS



Hardware

- Sharing of resources could be done by h/w, OS, or even programs
  - E.g., Sharing of busses done by h/w
  - E.g., sharing of CPU cycles done by OS
  - E.g., sharing of registers done by programs themselves
- Discuss other resources and their sharing


disk
disk
disk
disk
SCSI bus
monitor
processor
cache
graphics controller
bridge/memory controller
memory
SCSI controller
PCI bus
IDE disk controller
expansion bus interface
keyboard
disk
disk
disk
disk
expansion bus
parallel port
serial port