Course Introduction (contd.): Operating Systems

Mainack Mondal and Saptarshi Ghosh

CS31202 / CS30002

The story so far

- What is an OS
- What are the two goals of an OS
- Two key parts of OS
- Interrupt driven functionality of OS

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking

- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

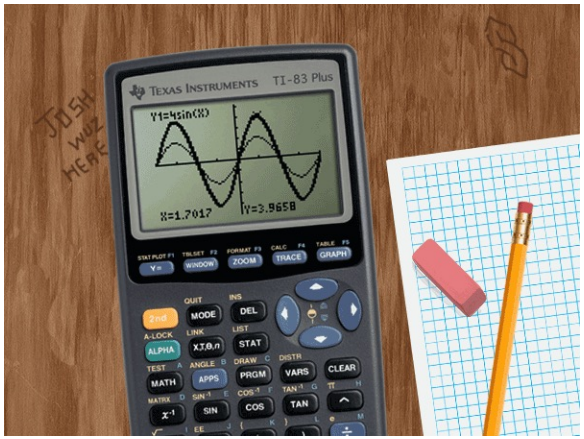
A brief history of OS

The beginning

Computers == which performs computational tasks

The beginning

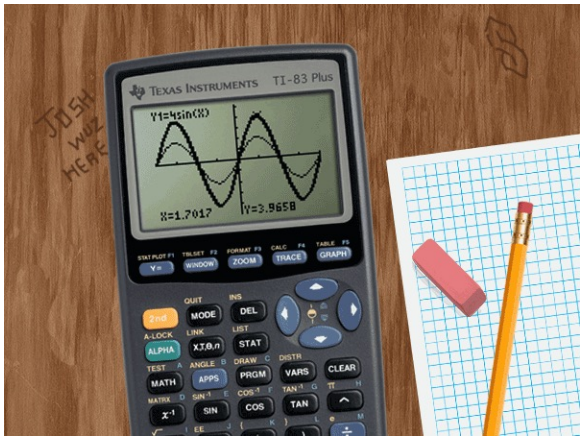
Computers == which performs computational tasks



Give a job: It will give you output

The beginning

Computers == which performs computational tasks



Give a job: It will give you output

What if you had to compute multiple jobs?

First computers were similar

- Thus the operating system was simply designed
  - Batch processing operating system
  - One job executed at a time
  - Only one job in memory at one time and executed (till completion) before the next one starts

First computers were similar

- Thus the operating system was simply designed
  - Batch processing operating system
  - One job executed at a time
  - Only one job in memory at one time and executed (till completion) before the next one starts

- https://youtu.be/YXE6HjN8heg?t=308

Problem with batch processing

A job has to wait for another to finish

- Led to very high wait times for the following jobs
- Under-utilization of CPU

Problem with batch processing

A job has to wait for another to finish

- Led to very high wait times for the following jobs
- Under-utilization of CPU

Insight: Input/Output from peripherals were very slow

- Your job has to wait forever when my job is simply reading the necessary data from peripheral devices

Problem with batch processing

A job has to wait for another to finish

- Led to very high wait times for the following jobs
- Under-utilization of CPU

Insight: Input/Output from peripherals were very slow

- Your job has to wait forever when my job is simply reading the necessary data from peripheral devices

Insight: A typical job usually has two types of phases in its lifetime - (1) when it uses CPU, (2) when it does I/O

SPOOLing

Simultaneous Peripheral Operations On-Line (SPOOL)

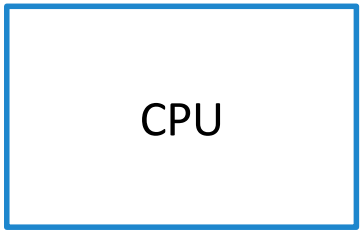
Only start jobs when all required data is read

OR, Send data output to a SPOOL buffer / virtual device

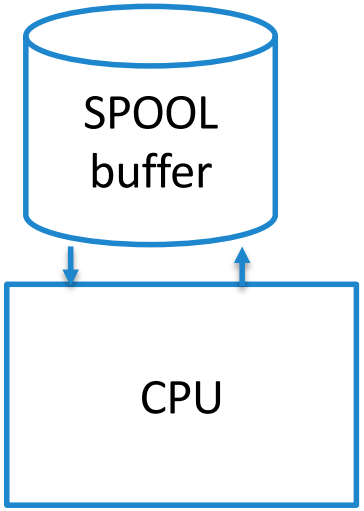
SPOOLing

- A spool is a buffer that holds output for an I/O device (usually a device such as a printer, that cannot accept interleaved data streams)
  - Read as far ahead as possible from input devices
  - Store output data until output devices are able to accept them
- A part of the disk can actually be used as a spool

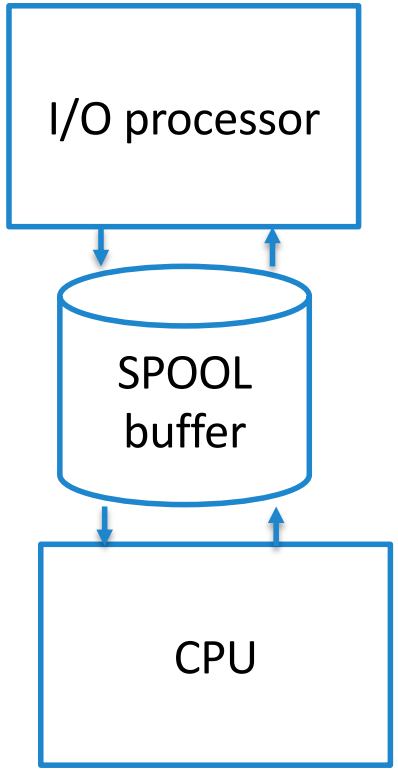
SPOOLing under the hood



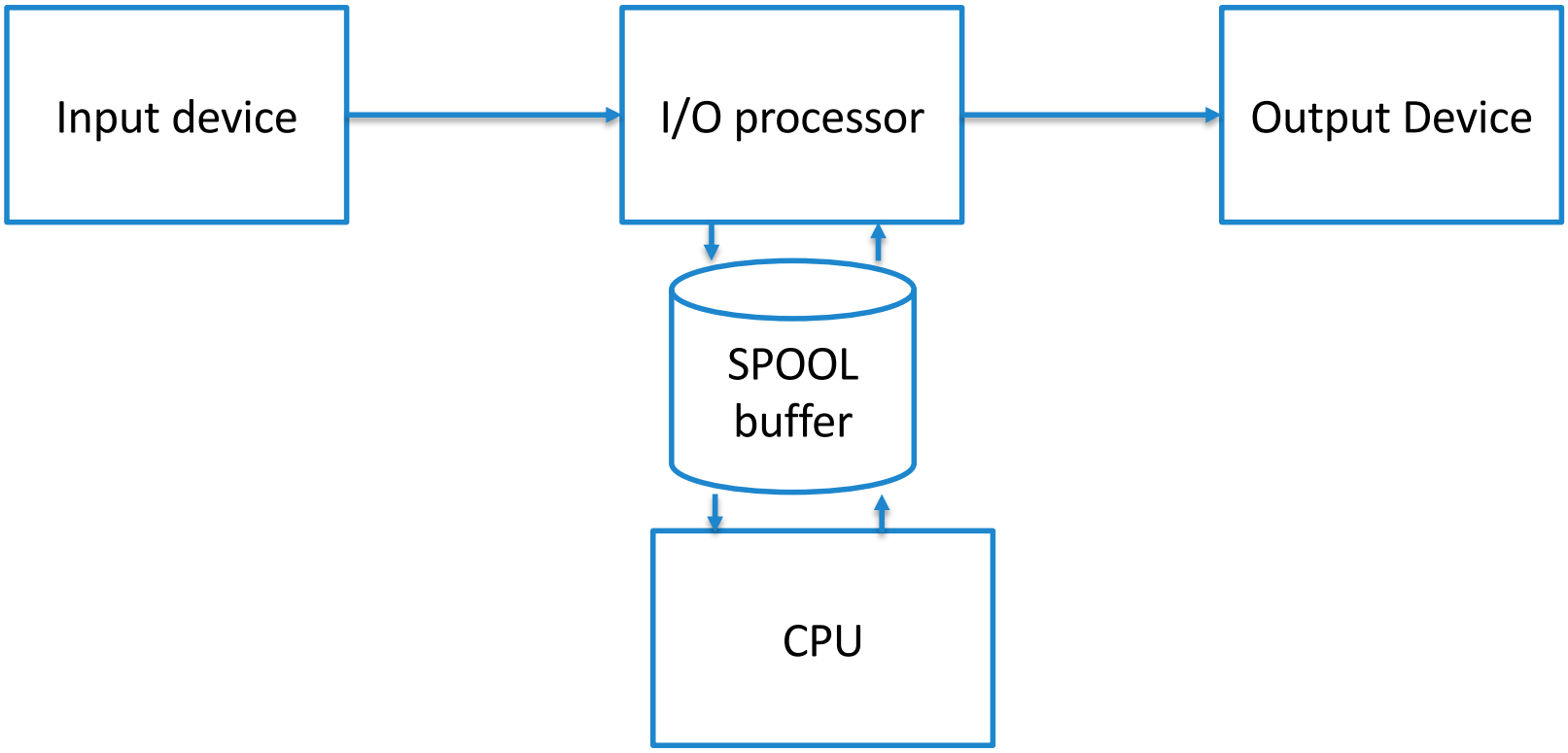
SPOOLing under the hood



SPOOLing under the hood



SPOOLing under the hood



SPOOLing brings in important concepts

- Addition of I/O processors
  - Overlap the I/O of one job with computation of other jobs
  - Better utilization of CPU
- Concept of virtual device
- Multiple jobs simultaneously reside in memory (CPU to be allocated to one of them at a time)
- CPU-bound and I/O-bound jobs

SPOOLing brings in important concepts

- Addition of I/O processors
  - Overlap the I/O of one job with computation of other jobs
  - Better utilization of CPU

- Concept of virtual device
- Multiple jobs simultaneously reside in memory (CPU to be allocated to one of them at a time)
- CPU-bound and I/O-bound jobs

Spooling is a special form of multiprogramming

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking

- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

Multiprogramming

- Multiple jobs loaded into memory at the same time and job scheduler selected a job (say job A)
  - If a big I/O request come for job A, then A's context is stored away and job B is started on the CPU
  - Once A's I/O finished, restore A

Multiprogramming: Issues

- Relies on the fact that job B can execute on the CPU when job A is doing I/O
  - Need to store context (current program state)
  - Need memory protection
  - Need privileged mode
- For multiprogramming to work: a good mix of CPU-bound and I/O-bound jobs
  - What if it is not the case?

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking
- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

Multitasking (timesharing)

- Logical extension of multiprogramming
  - CPU switches jobs so fast that users can interact with each job while its running
  - Creates interactive computing (e.g., cancel an ongoing download, GUI)
- Characteristics
  - Real time: meeting deadline for jobs
  - Better share resources between jobs

Multitasking: Need for new tech

- Concept of CPU scheduling
  - Need hardware timers
  - Need scheduling algorithm (which task to allocate the CPU, out of all tasks that are ready to execute)
  - Have to worry about context switch overhead
  - Concept of CPU burst and I/O burst (lots of CPU operations OR lots of I/O operations in one go, so that context switches are minimized)

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking

- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

Multitasking: The tools

- For multitasking, somebody needs to schedule the tasks as time goes
  - Kernel does it
  - Dual mode of operation
  - Use of timer

Dual mode of operation

- Process / task can execute in two modes
  - User mode and kernel mode (also called privileged mode)
  - User mode: run normal tasks
  - Kernel mode: directly talk to CPU/Peripherals to schedule tasks

Dual mode of operation

- Process / task can execute in two modes
  - User mode and kernel mode (also called privileged mode)
  - User mode: run normal tasks
  - Kernel mode: directly talk to CPU/Peripherals to schedule tasks
- Mode bit provided in hardware
  - Tells CPU if it is running in user or kernel mode

Kernel mode facilities

- Can run privileged instructions on CPU
  - Only in kernel mode
  - If you try to run them in user mode, generates exceptions
  - Examples: low-level I/O operation, setting protection registers, running EI, DI instructions (Enable/Disable interrupt)

How to switch between these two modes?

- System call or interrupt changes mode to kernel
- Special “return” instruction changes mode to user

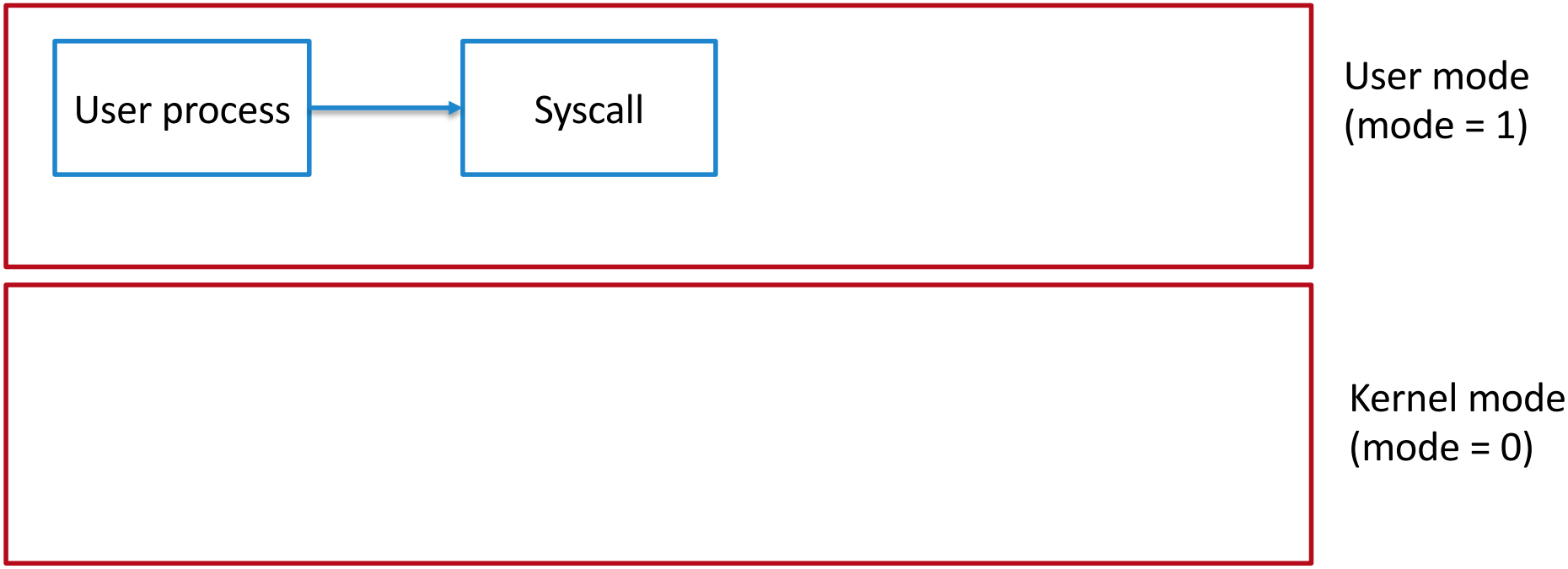
How to switch between these two modes?

- System call or interrupt changes mode to kernel

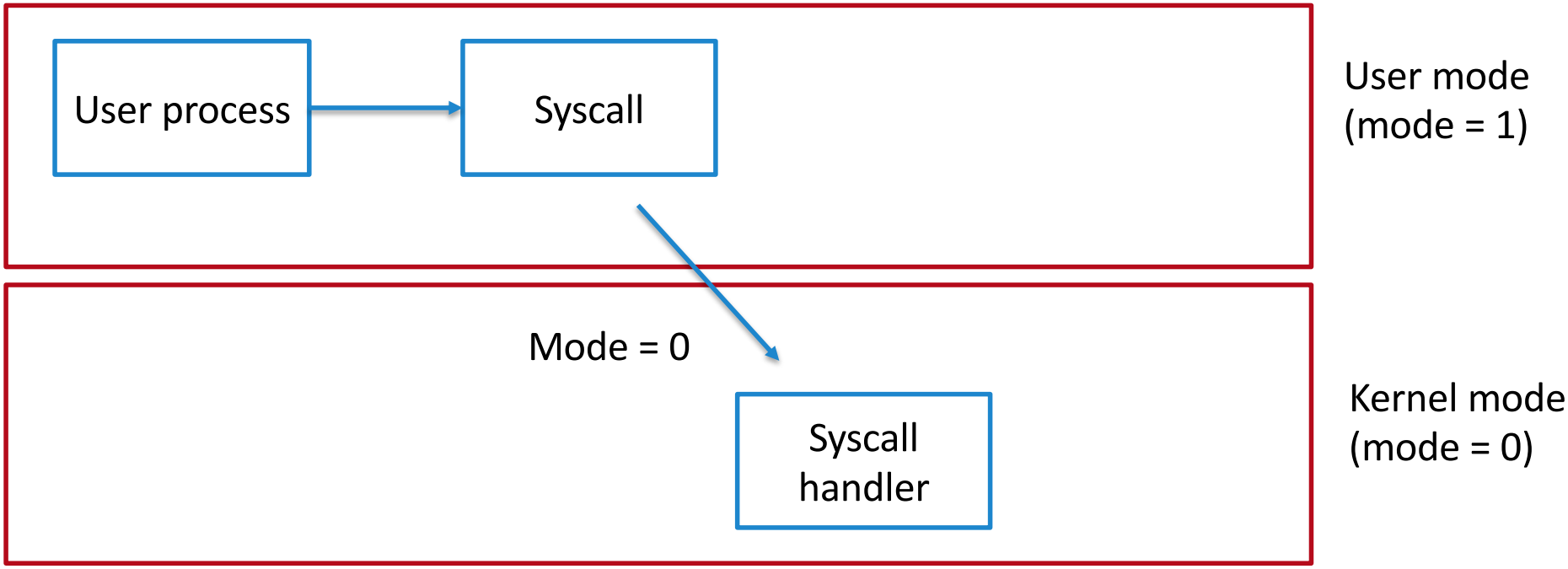
- Special “return” instruction changes mode to user

But when to change modes when applications are running?

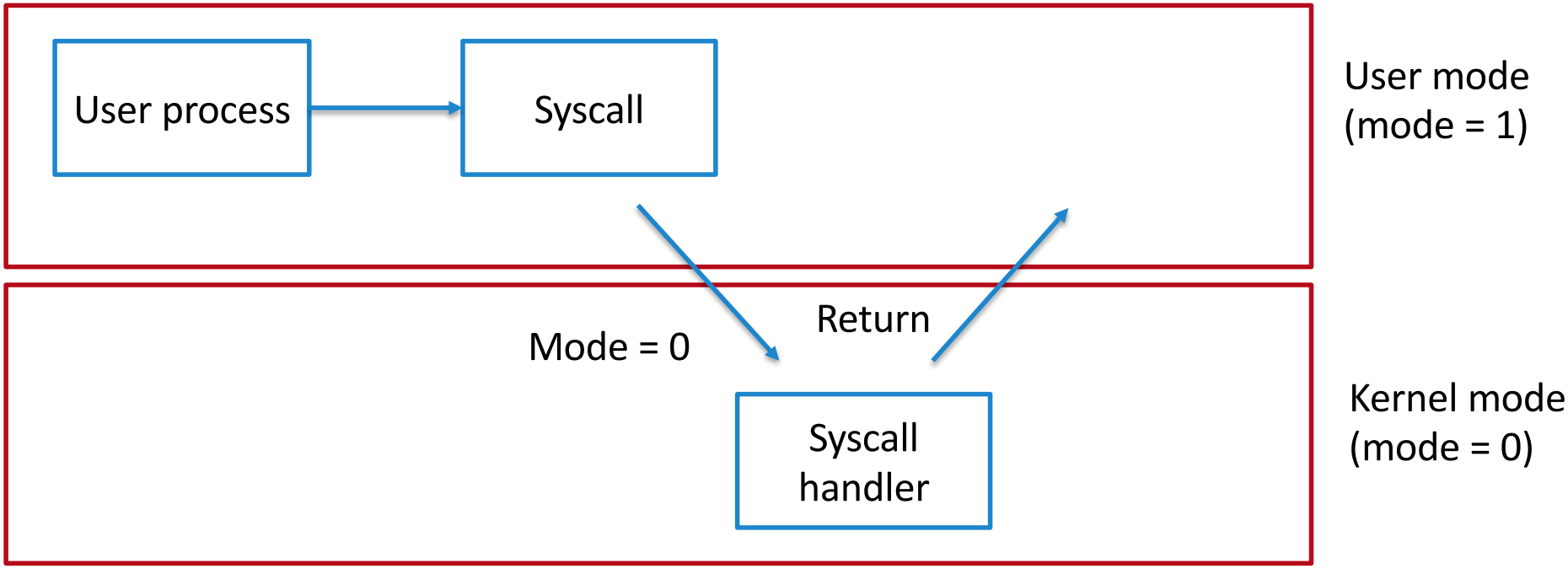
Putting it all together: the multitasking basic in two modes



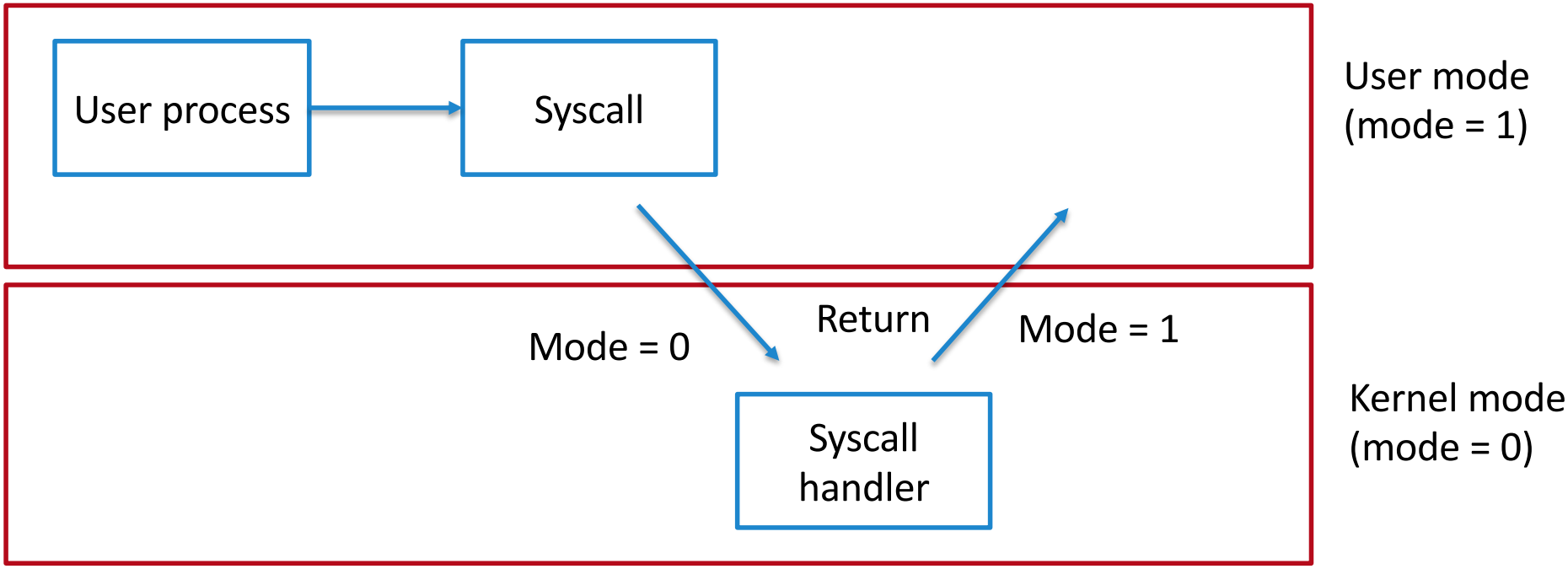
Putting it all together: the multitasking basic in two modes



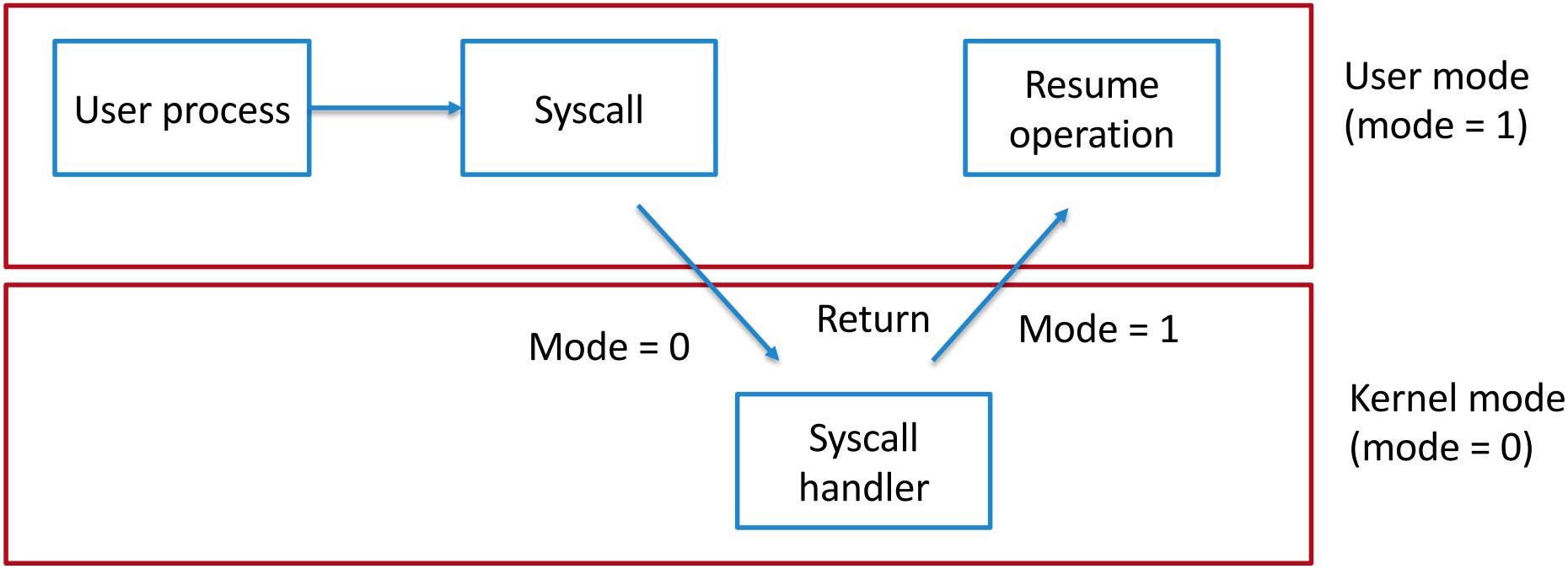
Putting it all together: the multitasking basic in two modes



Putting it all together: the multitasking basic in two modes



Putting it all together: the multitasking basic in two modes



Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking
- Today's OS (multitasking, like Unix)
  - Dual mode of operation
  - Uses of timer

How is hardware timer used?

- Recall that OS divide tasks into micro tasks and then schedule them in CPU
  - Uses a hardware timer to prevent infinite loop or resource hogging

How is hardware timer used?

- Recall that OS divide tasks into micro tasks and then schedule them in CPU
  - Uses a hardware timer to prevent infinite loop or resource hogging
- Timer is set such that it interrupts the processor after prespecified time
  - OS initializes the count value (privileged mode)
  - Count value in timer is decremented by physical clock
  - Timer generates an interrupt when count value is 0
  - Kernel "wakes up" upon interrupt and schedules next task