Course Introduction (contd.): Operating Systems

Indranil Sengupta (odd section)
and Mainack Mondal (even section)

CS39002

Spring 2019-20

The website is up!

http://www.facweb.iitkgp.ac.in/~isg/OS/

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
  - Some practice problems

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

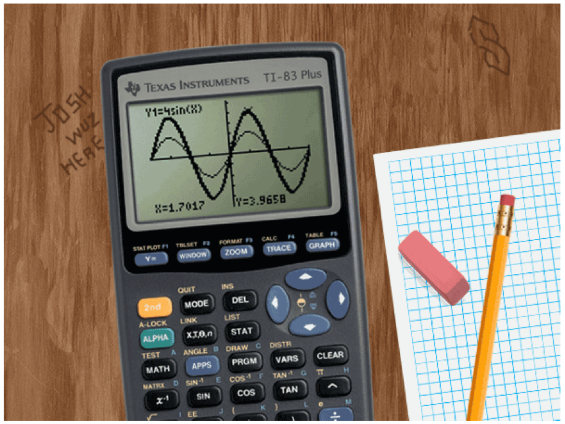
A brief history of OS

The beginning

Computers == which performs computational tasks

The beginning

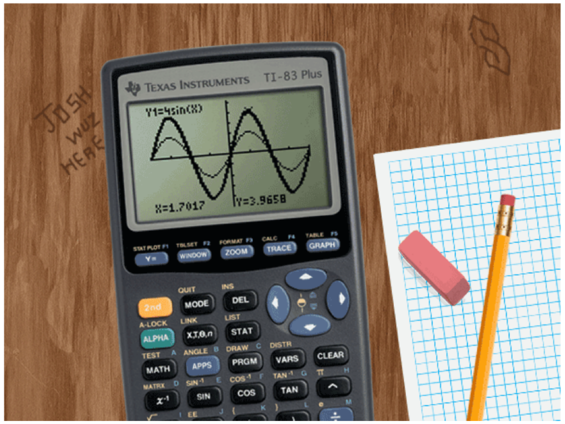
Computers == which performs computational tasks



Give a job: It will give you output

The beginning

Computers == which performs computational tasks



Give a job: It will give you output

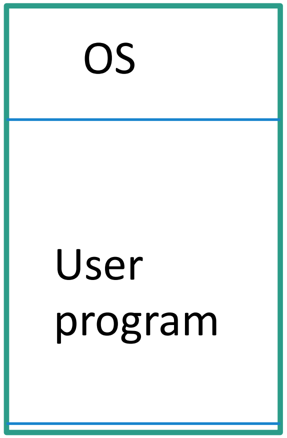
What if you had to compute multiple jobs?

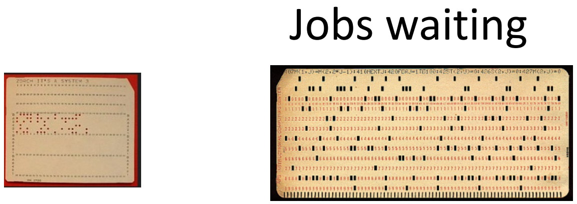
First computers were similar

- 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
  - CPU was not doing anything at that time

Problem with batch processing

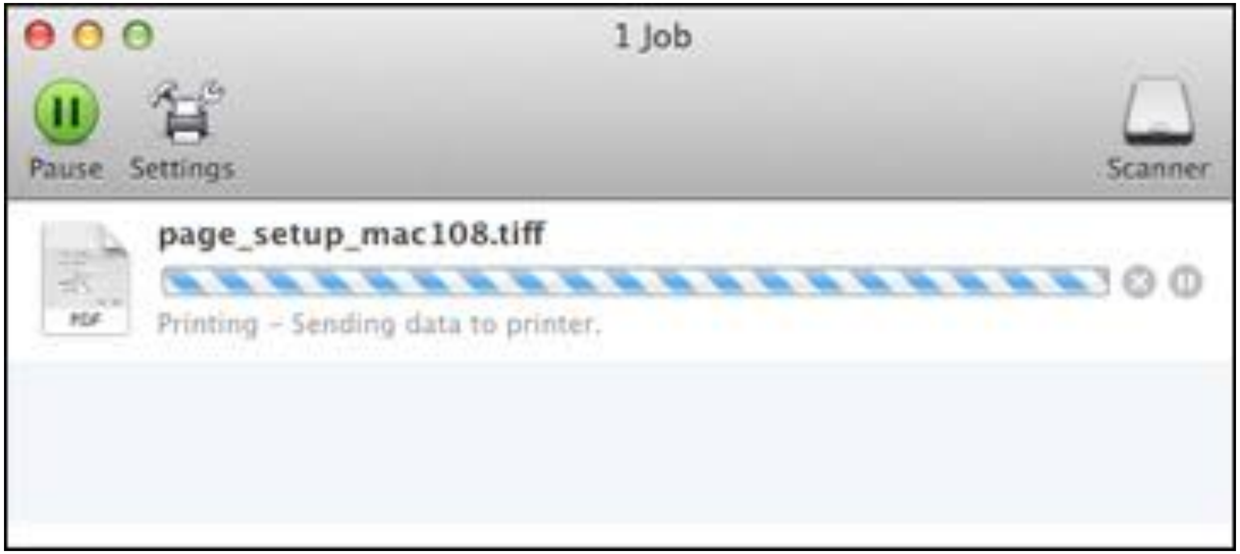
- A job has to wait for another to finish
  - Led to very high wait times for the following jobs
  - CPU was not doing anything at that time

- Insight: Input/Output from periphrals were very slow
  - Your job has to wait forever when my job is simply reading the necessary data from peripheral devices

SPOOLing

Simultaneous peripheral jobs online (SPOOL)

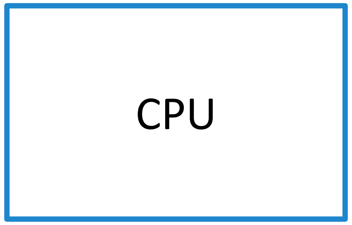
- Only start jobs when all required data is read
- OR, Send data output to a SPOOL buffer / virtual device



page_setup_mac108.tiff

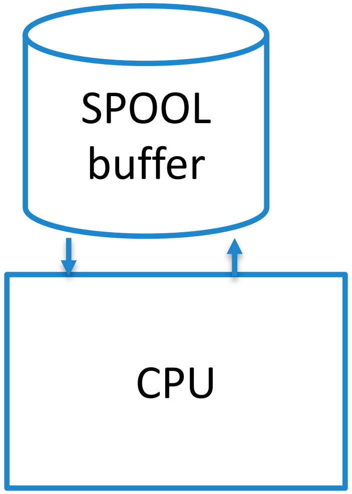
Printing - Sending data to printer.

SPOOLing under the hood

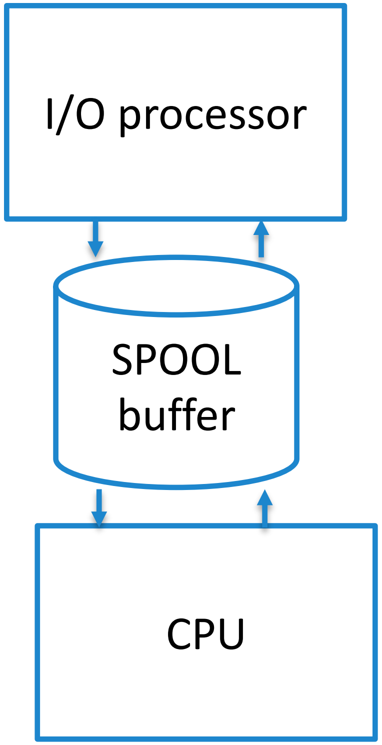


CPU

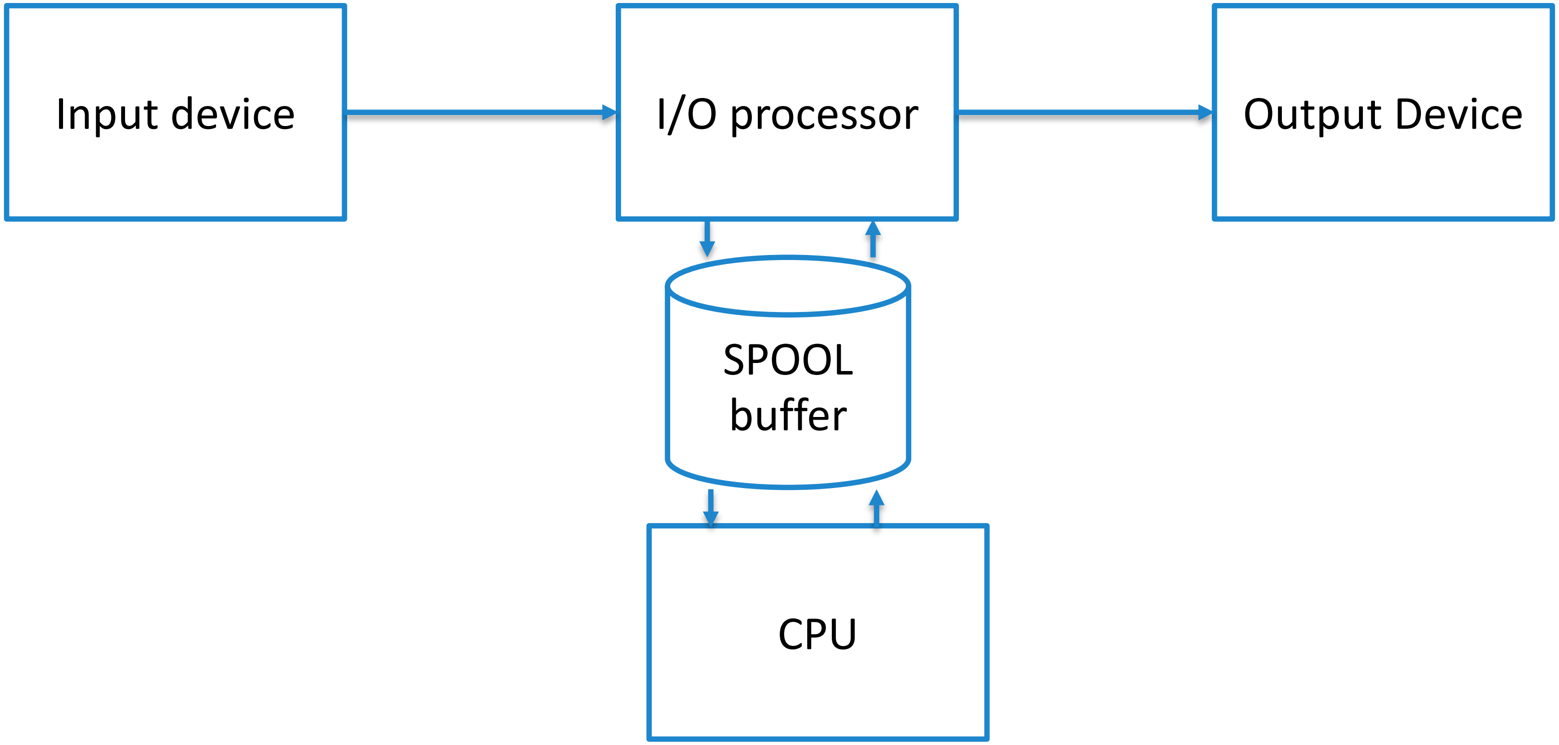
SPOOLing under the hood



SPOOLing under the hood



SPOOLing under the hood



SPOOLing bring in important concepts

- Addition of I/O processors
  - Read/Write becomes faster

- Concept of virtual device
- Batch of jobs
- CPU-bound and I/O bound jobs

SPOOLing bring in important concepts

- Addition of I/O processors
  - Read/Write becomes faster

- Concept of virtual device
- Batch of jobs
- CPU-bound and I/O bound jobs

A special form of multiprogramming

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
    - Once A's I/O finished restrore A

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
  - Once A's I/O finished restrore A

- Storing context (current program state)
  - Need memory protection
  - Need privileged mode

Multiprogramming: Issue

- Relies on the fact that job B can start when job A is doing I/O

- For multiprogramming to work: a good mix of CPU and I/O bound jobs

- What if its not the case?

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking
  - Some practice problems
- 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 download)

- Characteristics
  - Real time: meeting deadline for jobs
  - Better share resources between jobs

Multitasking: Need for new tech

- Concept of CPU scheduling
  - Need hardware timers
  - Concept of CPU burst and I/O burst (lots of CPU operations OR lots of I/O operations in one go)
  - Have to worry about context switch overhead

Today's class

- A brief historical overview of OS
  - Batch processing systems
  - Multiprogramming
  - Multitasking
  - Some practice problems
- 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 can execute in two modes
  - user mode and kernel mode
  - User mode: run normal applications
  - Kernel mode: directly talk to CPU/Peripherals to schedule tasks

Dual mode of operation

- Process can execute in two modes
  - user mode and kernel mode
  - User mode: run normal applications
  - Kernel mode: directly talk to CPU/Peripherals to schedule tasks
- Mode bit in in hardware
  - Tells CPU if its 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
  - Example: low-level I/O operation, setting protection registers like, 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?

Today's class

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

How to use hardware timer?

- 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

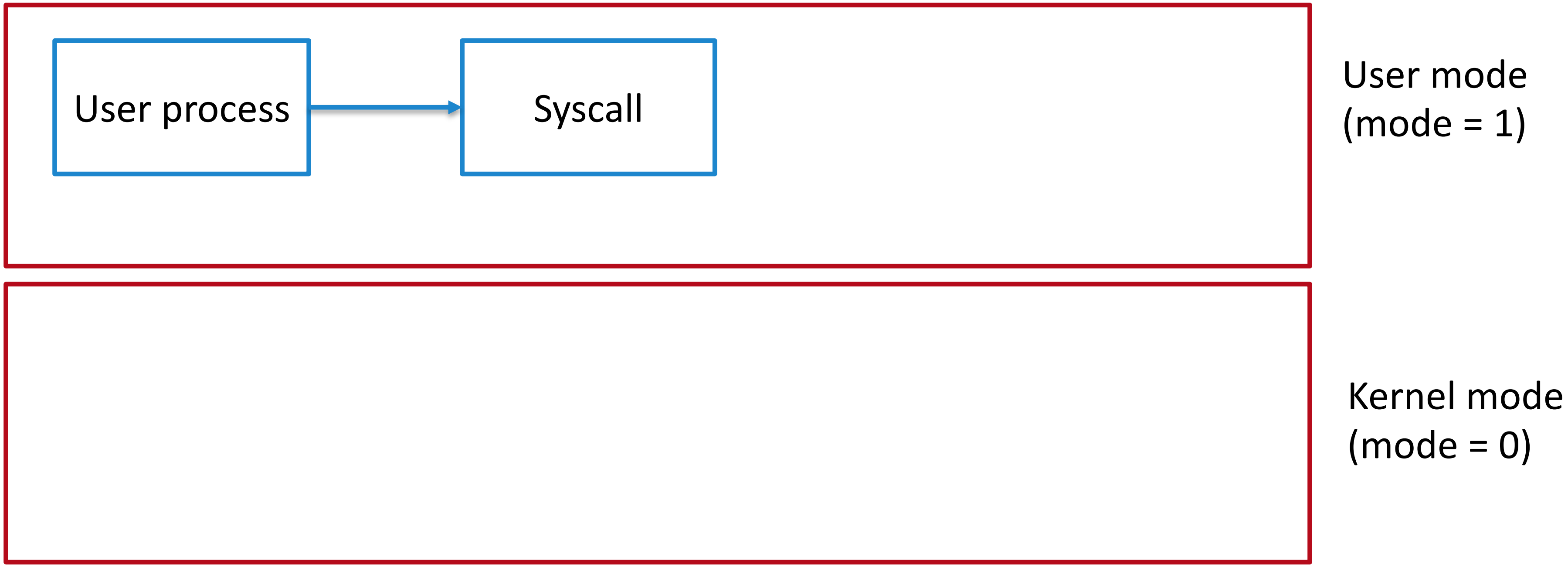
How to use hardware timer?

- 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 interrupts processor after prespecified time
  - OS initializes the count value (privileged mode)
  - Count value in timer is decremented by physical clock

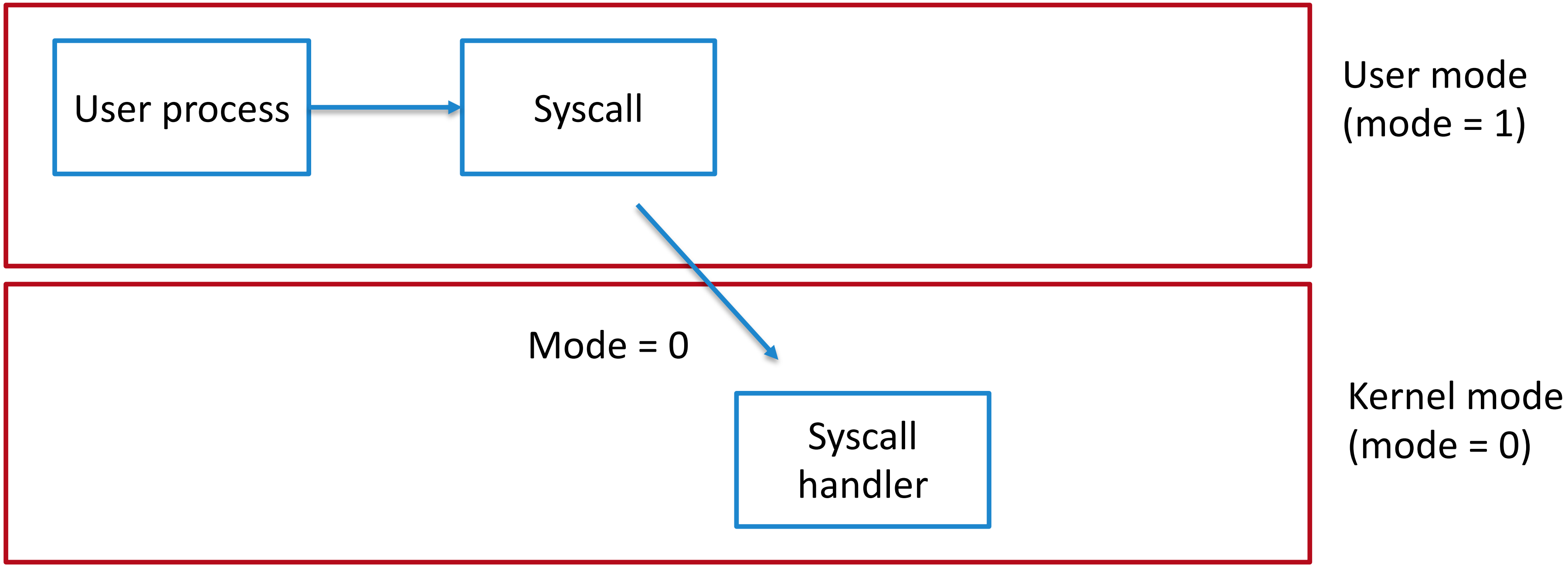
How to use hardware timer?

- 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 interrupts processor after prespecified time
  - OS initializes the count value (privileged mode)
  - Count value in timer is decremented by physical clock
  - Generates an interrupt when count value is 0

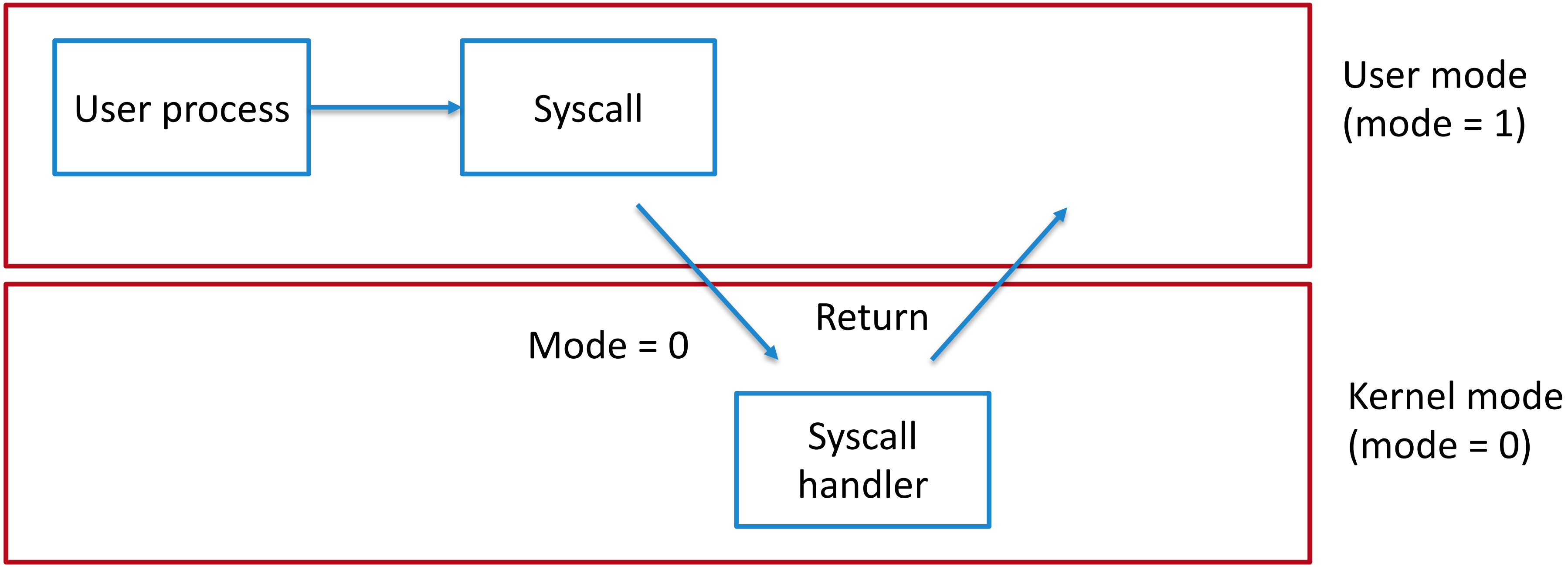
Putting it all together: the multitasking basic in two modes



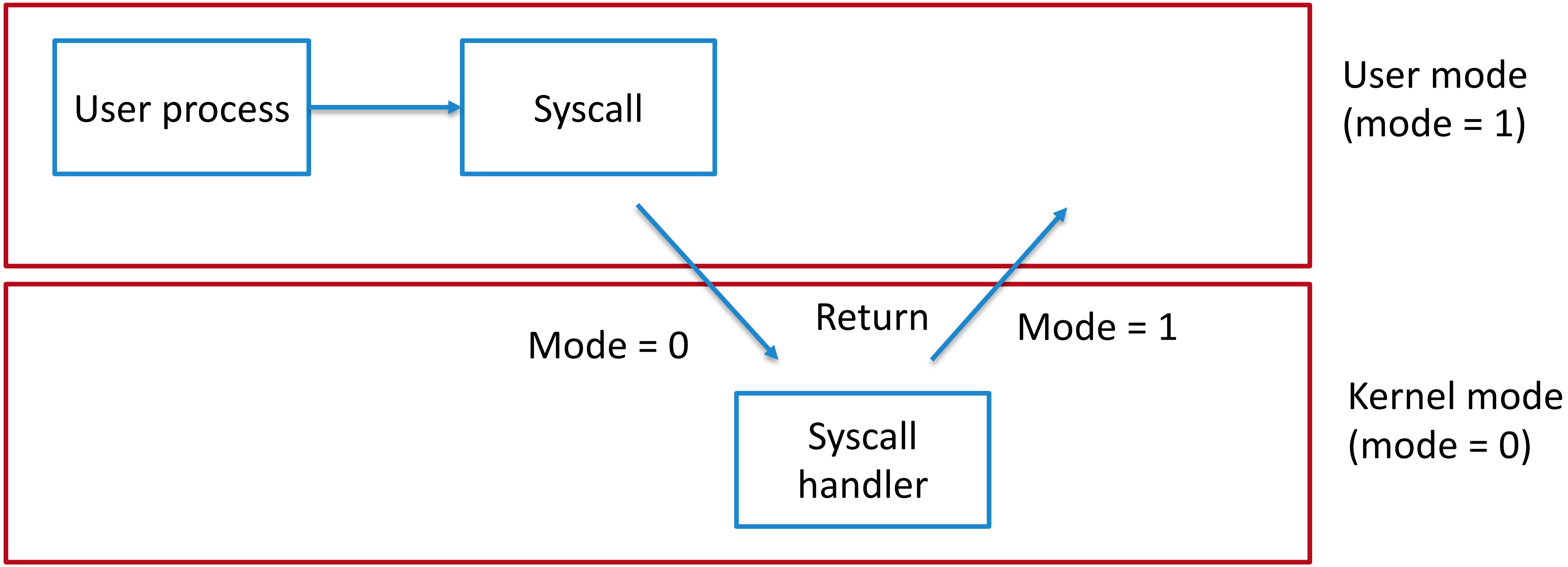
Putting it all together: the multitasking basic in two modes



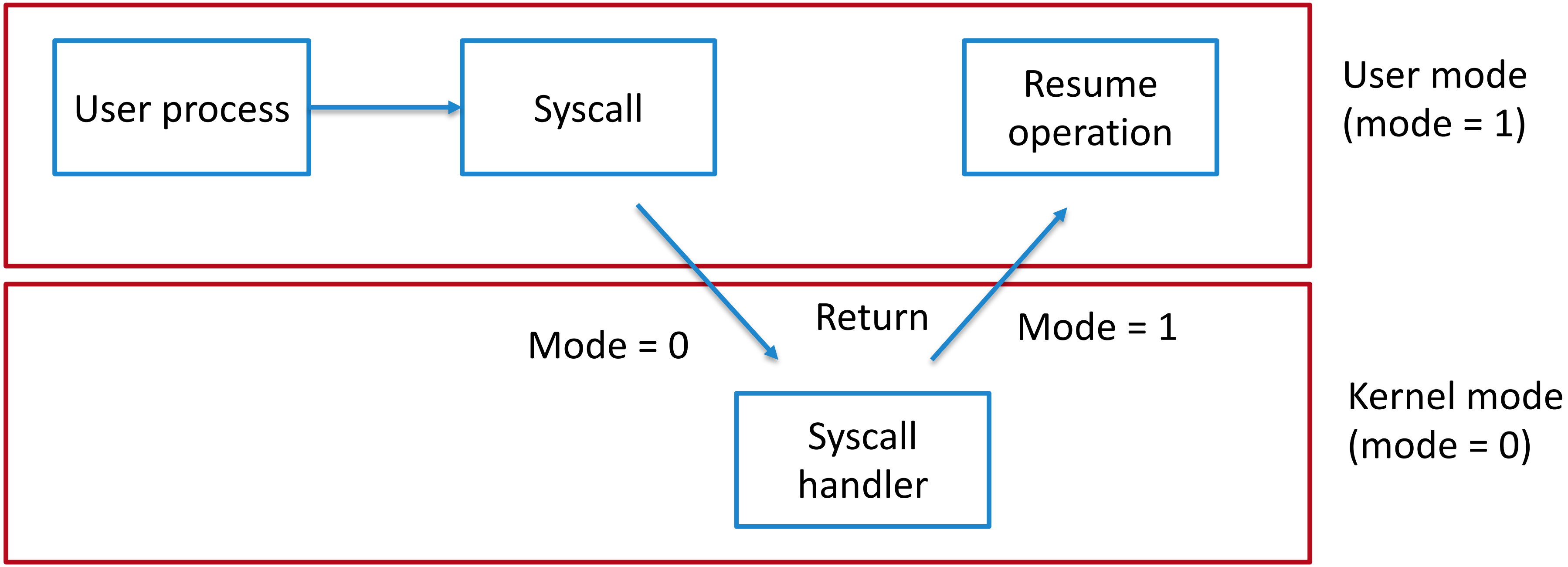
Putting it all together: the multitasking basic in two modes



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