CS 1217

Synchronization Primitives

Concurrency, or the illusion thereof

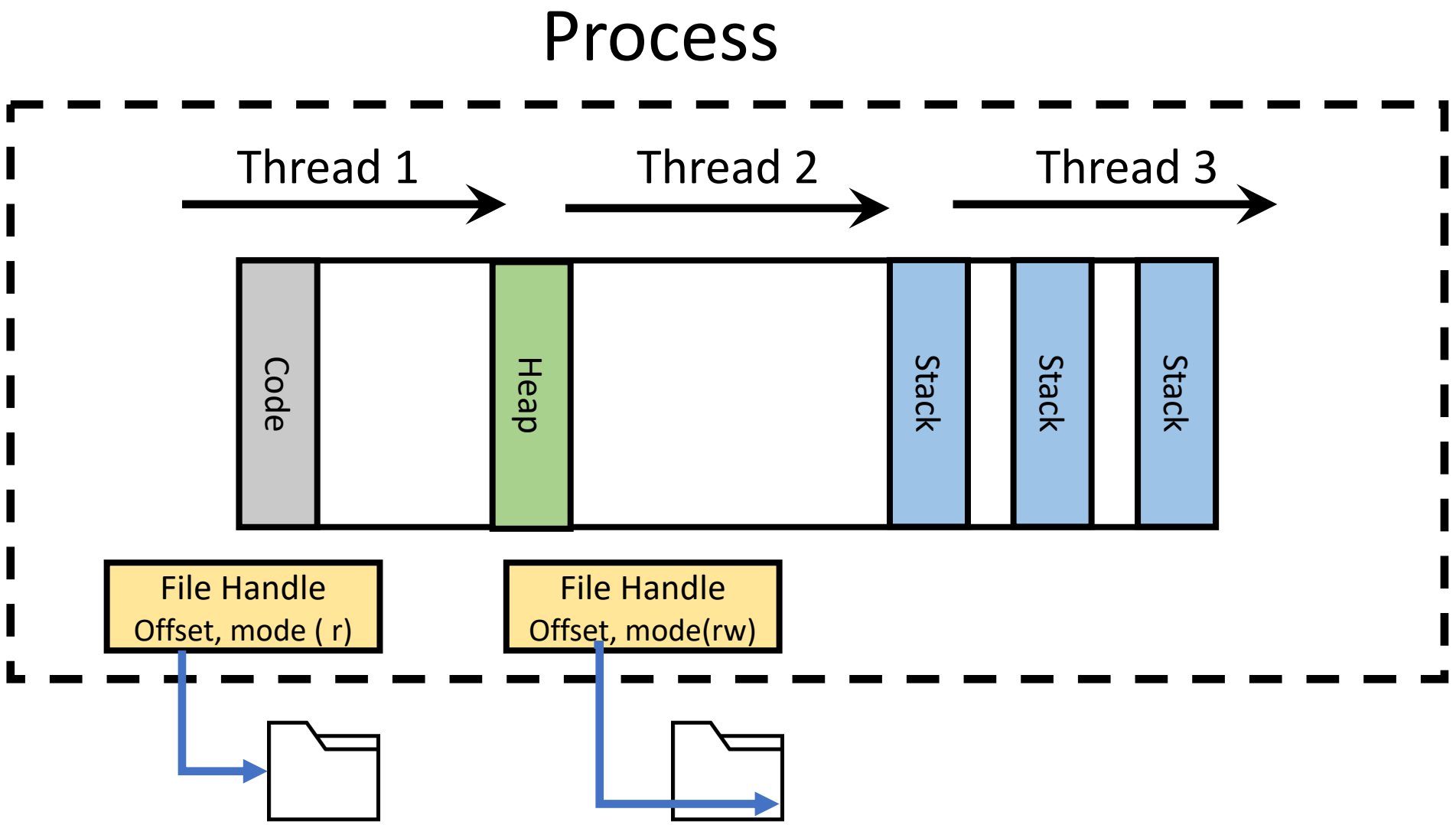
- OS creates the illusion of concurrency, even on a single core CPU. How?
- By allowing the processor to rapidly switch between threads/processes
- The system is almost always *in the process* of doing of many things.

Concurrency

- The illusion of concurrency is **powerful** and **useful**:
  - It helps us think about how to structure our applications
  - It hides latencies caused by slow hardware devices

- Also creates some issues:
  - **Coordination**: enabling efficient communication between the **multiple threads** involved in **performing a single task**.
  - **Correctness:** ensuring consistent state when being accessed by multiple threads, concurrently**.**

Threads and CPU Virtualization



Why use threads?

- Parallelism

- Concurrency

Concurrency

- The illusion of concurrency is **powerful** and **useful**:
  - It helps us think about how to structure our applications
  - It hides latencies caused by slow hardware devices

- Also creates some issues:
  - **Coordination**: enabling efficient communication between the **multiple threads** involved in **performing a single task**.
  - **Correctness:** ensuring consistent state when being accessed by multiple threads, concurrently**.**

OS and Concurrency

- Why study concurrency in an OS class?
- Tradition: why mess with it?

- OS was the first concurrent program, even before multi-threaded applications
  - **Multiplexes** state across multiple processes
  - Lots of **shared state** (shared data structures for processes, memory, storage etc.)
  - State needs to be kept consistent: page tables, process lists, file system structures etc.
  - Uses **many threads** to hide hardware delays while servicing devices and application requests
  - What happens if the kernel gets synchronization wrong?

OS and Concurrency

- OS is a difficult concurrent program itself
- Multiplexes many hardware resources among threads, leading to a great deal of shared resources between multiple processes
- Uses some threads to make forward progress, while servicing slow I/O requests from other threads
- Also, OS **cannot** get synchronization incorrect. If if does, many processes might die

Parallelism vs. Concurrency

*…when people hear the word **concurrency** they often think of **parallelism**, a related but quite distinct concept. In programming, concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations. Concurrency is about dealing with **lots of things at o**nce. Parallelism is about **doing lots of things at once**.*

Rob Pike (https://vimeo.com/49718712)

https://en.wikipedia.org/wiki/Rob_Pike

Concurrent Threads

- Concurrency forces us to **relax some assumptions** that we may want to make about how any particular thread executes.
- Unless explicitly synchronized, threads may:
  - Be run in **any order**,
  - Be stopped and restarted at **any time**,
  - Remain stopped for **arbitrary lengths of time**.

- Is this good or bad, in general?

The Bank Example

```
void UpdateTheMoolah (account_t account, int largeAmount) {
  int currBal = get_balance (account);
  currBal = currBal + largeAmount;
  put_balance (account, currBal);
  return; }
```

- Assume, there are ₹ 10000 initially in the account
- Two people are simultaneously trying to deposit money
  - One is depositing ₹ 10000 (Person B)
  - Another one is depositing ₹ 20000 (Person A)

What you want to happen

Person A	Person B	Balance
		₹ 10000

`int currBal = get_balance (account);`
`currBal = currBal + 20000;`
`put_balance (account, currBal);`		

		₹ 30000

	`int currBal = get_balance (account);`
	`currBal = currBal + 10000;`
	`put_balance (account, currBal);`	

		₹ 40000

What you do NOT want

Person A	Person B	Balance
		₹ 10000
`int currBal = get_balance (account);` `currBal = currBal + 20000;`		
	`int currBal = get_balance (account);` `currBal = currBal + 10000;`	
	`put_balance (account, currBal);`	₹ 20000
`put_balance (account, currBal);`		₹ 30000

What you REALLY do NOT want

Person A	Person B	Balance
		₹ 10000
`int currBal = get_balance (account);` `currBal = currBal + 20000;`		
	`int currBal = get_balance (account);` `currBal = currBal + 10000;`	
`put_balance (account, currBal);`		₹ 30000
	`put_balance (account, currBal);`	₹ 20000

Race Conditions

- A **race condition** happens "when the output of a process is unexpectedly dependent on timing or other events.”

- Note that the definition of a race depends on what we **expected** to happen:
  - We **expected** to have ₹ **40000** after both deposits