Multitasking

Dr. Naser Al Madi

Learning objective

- Understand the motivation for multiprogramming
- Understand the difference between Concurrency and Parallelism
- Learn multithreading and multiprocessing in Python!

*Note: Some information on multithreading and multiprocessing will be specific to Python, it does not generalize to other programming languages.

Context

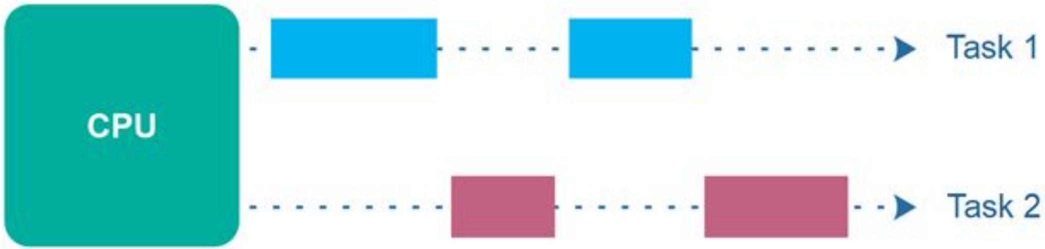
- Most of the programs we wrote so far perform a single task.
- Real-life applications typically perform multiple tasks at the same time.
- Your web browser is able run multiple webpages at the same time.
- When YouTube is loading a video, you can interact with other webpages, for example.
- How can we write such programs?

Motivating example:

Concurrency vs Parallelism

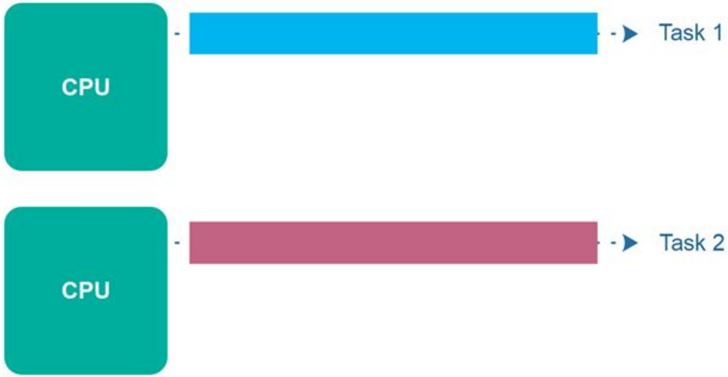
Concurrency

Concurrency means that an application is making progress on more than one task - at the same time or at least **seemingly** at the same time (concurrently).

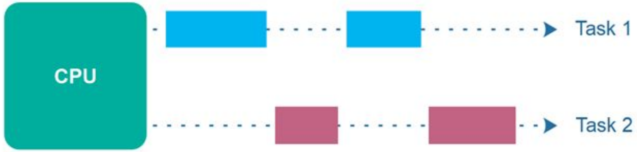


Parallel Execution

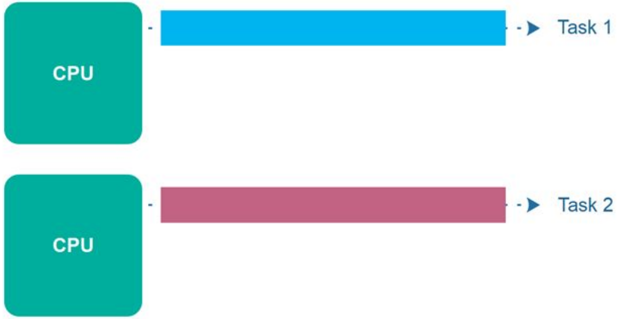
Parallel execution is when a computer has more than one CPU or CPU core, and makes progress on more than one task **simultaneously**.



Concurrency vs Parallel



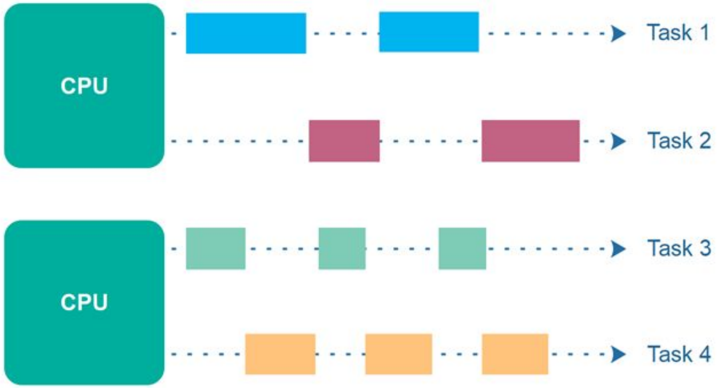
Concurrency



Parallel

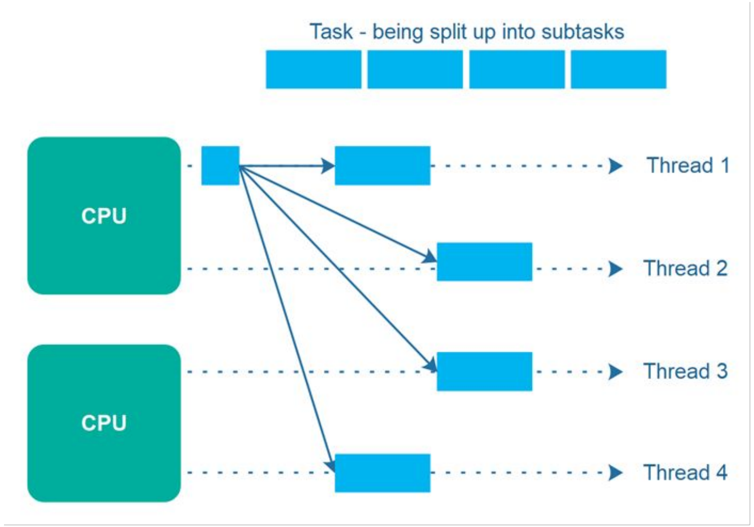
Parallel Concurrent Execution

It is possible to have parallel concurrent execution, where threads are distributed among multiple CPUs. Thus, the threads executed on the same CPU are executed concurrently, whereas threads executed on different CPUs are executed in parallel.



Parallelism

The term parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel



Multithreading vs Multiprocessing

Multitasking

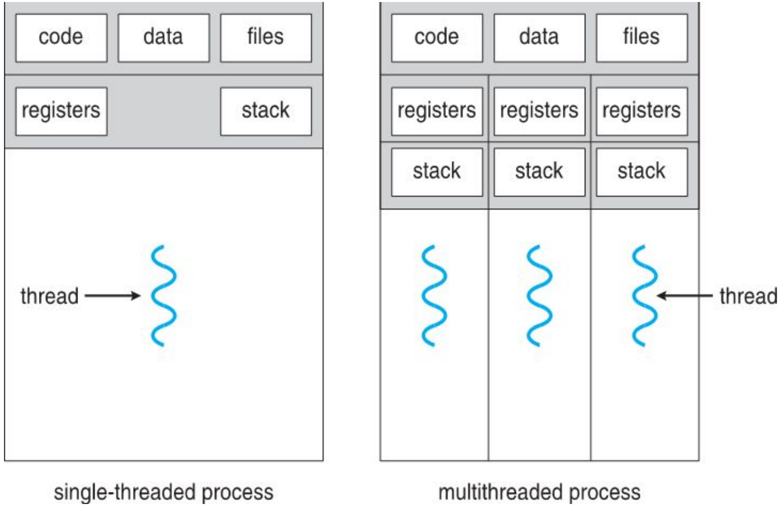
- The goal is to write code that performs multiple tasks at the same time.
- This can be achieved in multiple ways:

Multiprocessing

Multithreading

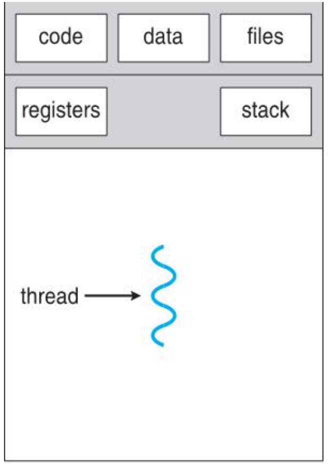
Multithreading

- So far, process has a single thread of execution
- Consider having multiple program counters per process
- Multiple locations can execute at once
- Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB
- Example of multiple threads: In word processor, user enters data (one thread) and spell check is being ran (another thread)



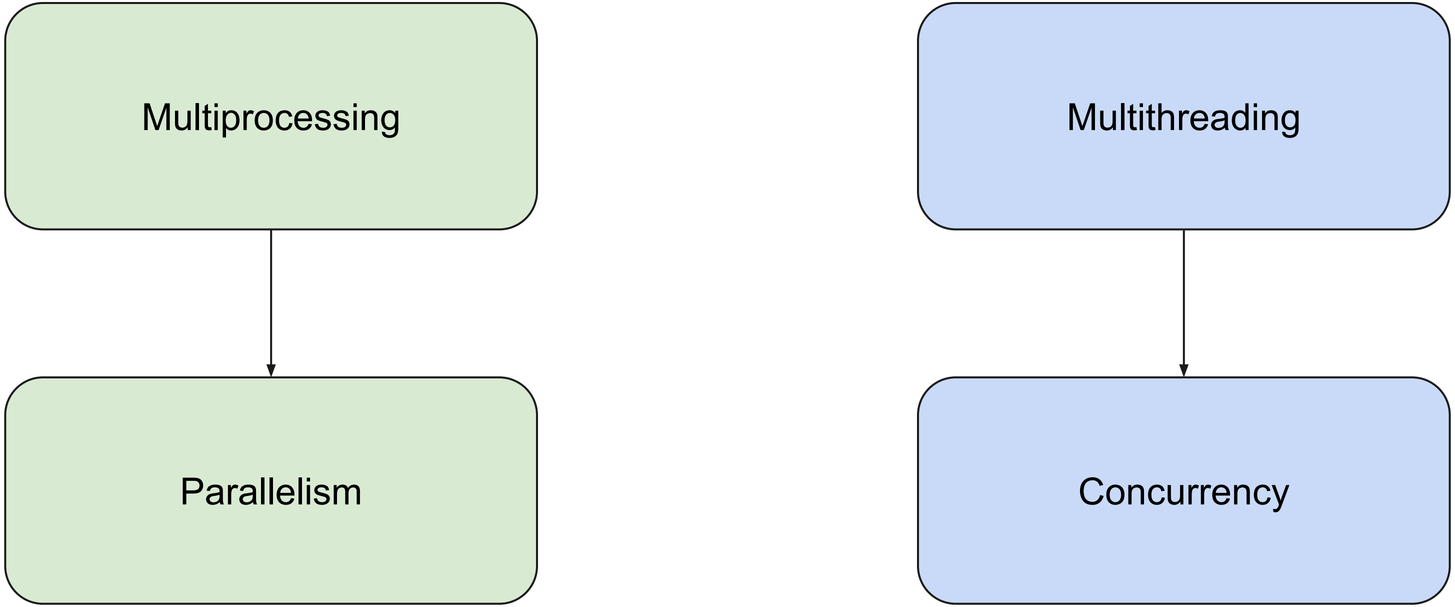
Multiprocessing

- A process is an independent instance
- Process creation is heavy-weight while thread creation is light-weight
- Processes do not share the same memory space, while threads do



single-threaded process

Only in Python



*Does not apply to languages other than Python

Multiprocessing in Python

Multiprocessing in Python

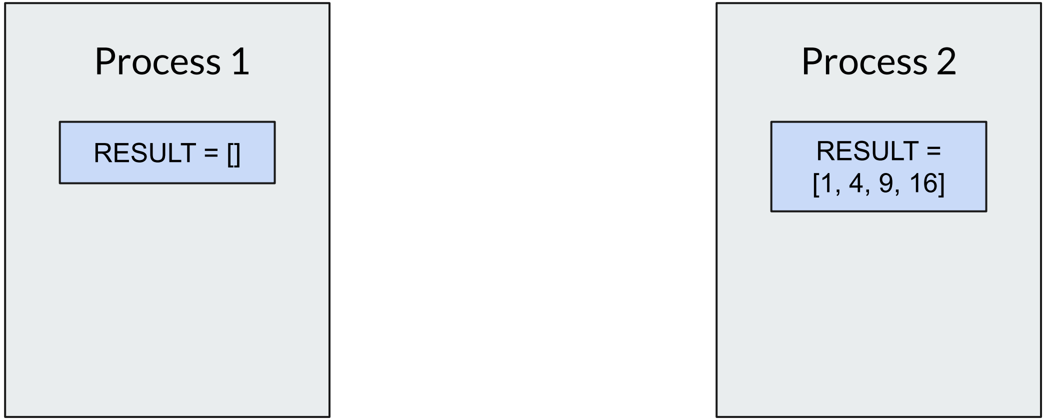
- You can create multiple processes and assign a function to each process.
- Slightly more elignet than fork in Xv6 (actually uses fork behind the scene)
- Let’s check “1-Intro_to_multithreading_in_Python.py”

Multiprocessing overhead

- Let’s time our multiprocessing code and compare it to serial code (synchronous).
- Let’s check “2-Timing_multithreading_in_Python.py”

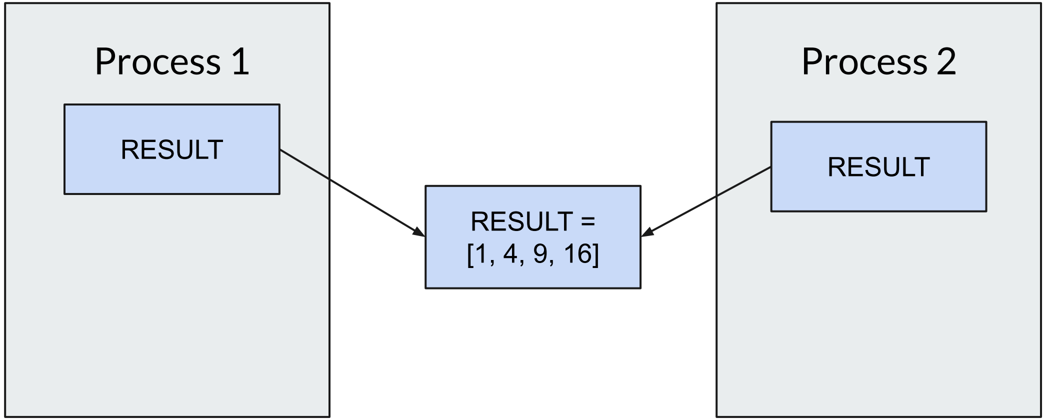
Sharing Data Between Processes

- Each processes maintains a separate memory space.
- Let’s check “3-Sharing_Data_between_processe.py”



Sharing Data Between Processes

- Then how can we share data between processes?
- Let’s check “4-Sharing_Data_between_processe.py”



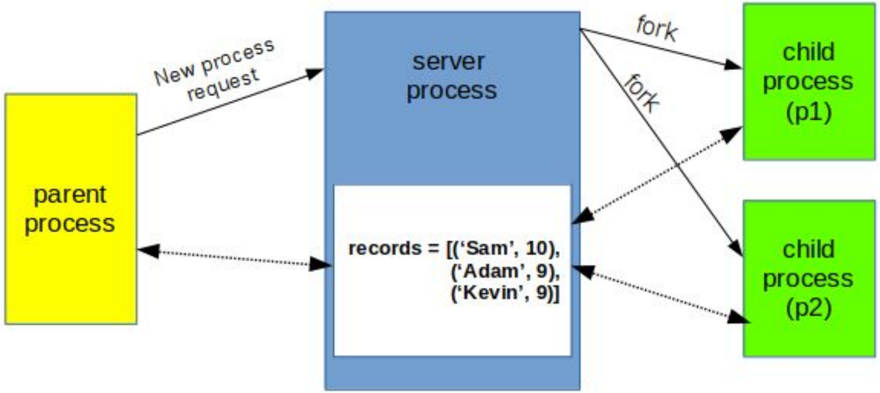
Sharing Data using Server Process

- The previous technique is not very flexible, what if I had a slightly more complicated structure?
- Check “5-Server_process_sharing_data.py”

Sharing Data using Server Process

- The previous technique is not very flexible, what if I had a slightly more complicated structure?
- A manager object returned by Manager() controls a server process which holds Python objects and allows other processes to manipulate them using proxies.
- A manager returned by Manager() will support types list, dict, Namespace, Lock, RLock, Semaphore, BoundedSemaphore, Condition, Event, Barrier, Queue, Value and Array
- Check “6-Server_process_sharing_data.py”

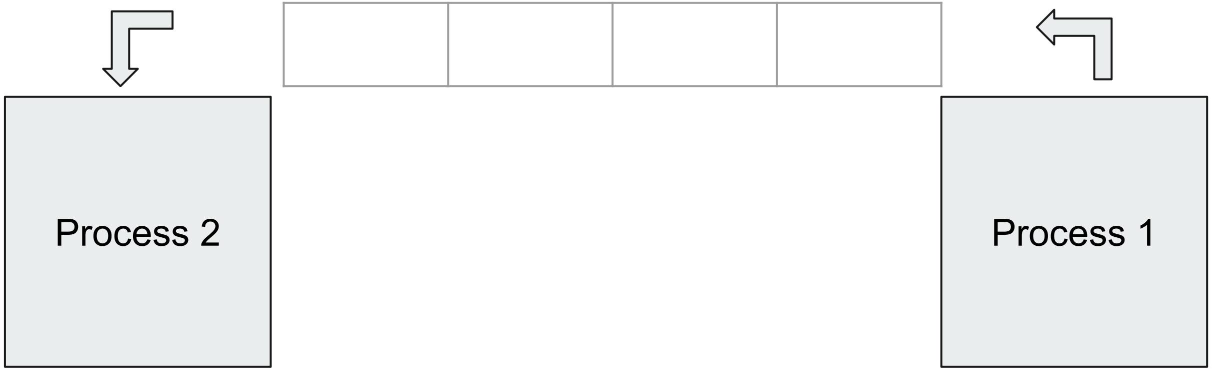
Sharing Data using Server Process



Interprocess Communication

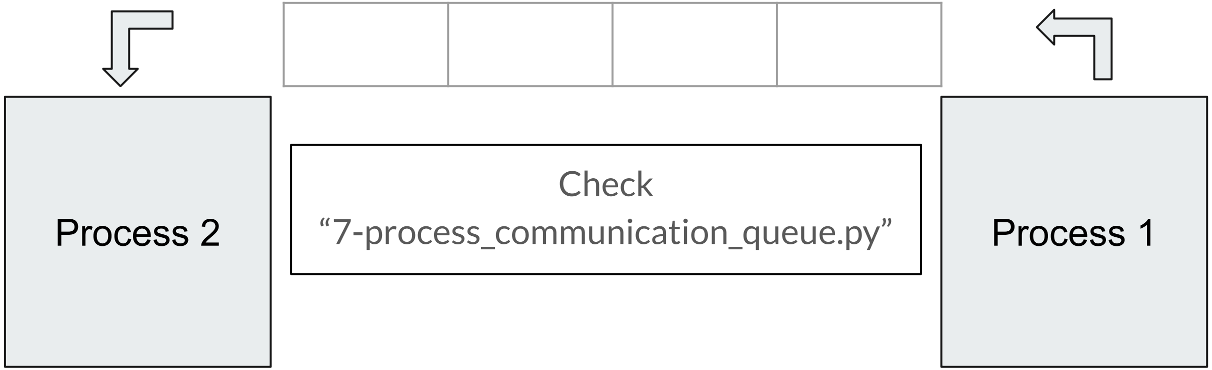
Queue

- What if the shared data is a stream or very big? We can't pass it all at once.
- We can use a Queue to pass data between processes.



Queue

- What if the shared data is a stream or very big? We can't pass it all at once.
- We can use a Queue to pass data between processes.



Check
"7-process_communication_queue.py"

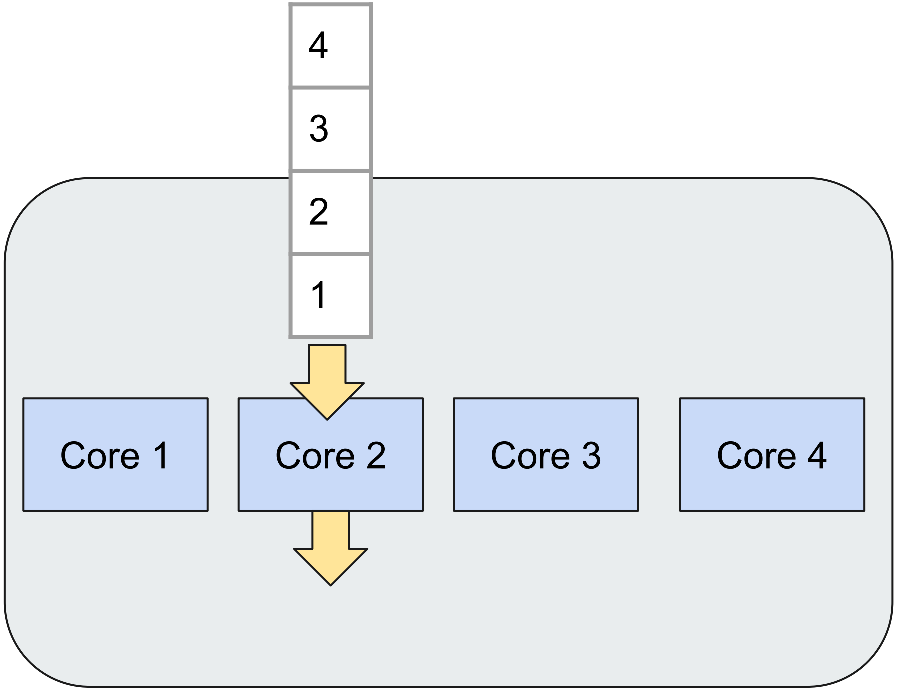
Pipes

- Queues are synchronized (at most one process can access the pipe at a time)
- Pipes are not synchronized, therefore they are faster, yet more prone to error
- Pipes can have only two processes, Queues can have many processes
- Let’s check “8-process_communication_pipe.py”

Pooling

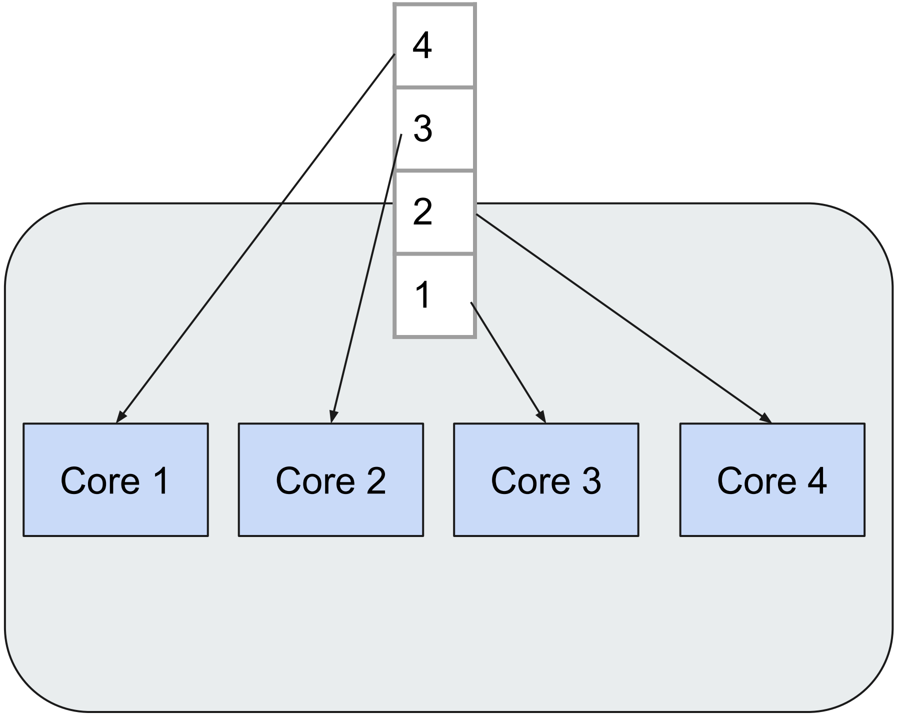
Pooling

- Super easy way to create a collection of processes to handle many requests/data.



Pooling

- Super easy way to create a collection of processes to handle many requests/data.



Pooling

- Super easy way to create a collection of processes to handle many requests/data.
- Let’s check “9-process_pooling.py” and “10-process_pooling.py”

References

- http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html
- https://github.com/nikhilkumarsingh/Parallel-Programming-in-Python/tree/master/06.%20Sharing%20data%20using%20Server%20Process