SuperFastPython
making python developers awesome at concurrency (https://superfastpython.com/)

Python Concurrency Glossary

A glossary of terms that are helpful when working with Python concurrency.

A

Alien Thread A thread created by a C or C++ library called by a Python program. Python may create a dummy thread object for each alien thread, but offers limited interaction or control over alien threads. Alien threads are often daemonic and cannot be joined.

Amdahl's Law A formula proposed by Gene Amdahl for the theoretical speedup of a task composed of subtasks with a fixed time or effort by adding more parallel execution. Calculated as 1 / (1 – p), where p is the proportion of time that would benefit from a speedup.

Async *See* **Asynchronous**

AsyncIO *See* **Asynchronous IO**

Asynchronous The occurrence of events that happen independent of the main program flow, meaning that the main flow of the program is not blocking or waiting for the results. For example, an asynchronous task may be submitted by the program, which may not wait for the result. Asynchronous tasks may or may not be executed concurrently. Opposite of synchronous.

Asynchronous IO A programming pattern that allows IO tasks to be performed without blocking the thread of execution. Opposite of synchronous IO tasks that must wait for the task to complete before continuing with the thread of

execution. There are many ways to implement asynchronous IO, such as callbacks, polling, lightweight threads, and more. Python supports asynchronous IO via the asyncio module.

Atomic An instruction or operation that cannot be subdivided and either occurs or does not occur. In concurrency programming, a context switch cannot occur during an atomic operation, only before or after the operation. Sequences of instructions can be simulated to be atomic under a defined context using synchronization mechanisms like locks.

async/await A syntax for asynchronous IO programming that allows tasks to be defined as asynchronous, such as with an async keyword, and for other tasks to wait on the result from asynchronous tasks, such as via the await keyword. Python supports async/await directly via keywords in the asyncio module.

B

Backoff An algorithm in parallel programming to avoid multiple processes or threads competing for a resource at the same time and potentially overwhelming the target or wasting resources. The back off will have a schedule that will increase with each subsequent failure, e.g. an exponential number of seconds.

Barrier A synchronization primitive where threads and processes will reach and wait at the barrier until the barrier is triggered, after which the barrier is lowered and the threads or processes may continue their execution. The trigger for the barrier is typically a fixed and pre-defined number of threads or processes reaching the barrier. It is different from a latch, where processes or threads waiting on the latch are different from those that trigger it. Python provides barriers via the threading.Barrier and multiprocessing.Barrier classes.

Blocking A programming instruction that halts execution for some amount of time requiring the thread of execution to wait. It means that the instruction or function call does not return immediately and may be waiting on a

	resource (e.g. file or network connection) or another thread or process (e.g. via a synchronization primitive). Blocking a thread or process often triggers a context switch in the operating system.
Busy Waiting	*See* **Spinning**

C

CPU	*See* **Central Processing Unit**
CPU Core	A central processing unit within an integrated circuit chip. Modern microprocessor chips have more than one CPU core.
CPU-bound	A task where the time taken to complete is determined by the speed of the CPU. This is the opposite of an IO-bound task. For example, performing a numerical calculation is an example of a CPU-bound task. CPU-bound tasks are fast relative to the speed of IO-bound tasks.
Central Processing Unit	Computer circuitry that executes programming instructions, also called a core or CPU core. Historically, a microprocessor chip contained a single CPU, e.g. a single core. A computer with more than one CPU had more than one microprocessor chip inside. Modern microprocessors are one physical chip with more than one CPU core inside, e.g. multiple CPU core processes. These are referred to as multi core processors.
Channel	A programming pattern for communicating between threads or processes in a thread-safe manner. If communicating between processes, it typically involves serialization of messages transmitted through the channel and the use of a synchronization primitive for reading and/or writing from/to the channel. Python provides channels via the multiprocessing.Pipe class.
Child Process	A process that is created by another process called the parent process. Child processes can be created by spawning a new process or by forking the parent process, depending on the capabilities of the underlying operating system.
Concurrency	Parts of a program that are independent and can be performed out of

	order. For example, tasks may be decomposed into many sub-tasks that can be performed in any order and achieve the same final result. Concurrent tasks may or may not be executed in parallel.
Concurrency Failure Case	A bug in implementation or behavior of a concurrent program. Examples include race condition, deadlock, livelock, starvation, corruption, lock convoy, and thundering herd.
Concurrent Programming	An engineering discipline for writing programs that contain independent subtasks that may be completed out of order and may or may not be executed in parallel. It is perhaps more abstract (higher-order) than parallel programming.
Condition Variables	A synchronization primitive where threads or processes may wait on a variable until a condition is true, after which they are notified and may continue their execution. Condition variables often support to both notify a single waiter or all waiters once the condition is met, e.g. notify and notify all respectively. Python provides condition variables via the threading.Condition and multiprocessing.Condition classes.
Context switch	A programming pattern that allows more than one process or thread of execution to run on a CPU, e.g. changes the “context” for the CPU that executes instructions. Stores the state of a thread of execution so that it can be resumed later and allows another thread of execution to run. Performed normally by modern operating systems. Having a vast number of concurrent programs, processes, or threads may result in a lot of context switching, which may not be efficient.
Cooperative Multitasking	A type of multitasking in an operating system where programs explicitly yield control to other programs. Typically used in embedded systems.
Core	*See* **CPU Core**
Coroutine	A programming pattern that generalizes routines (e.g. subroutines, functions, or blocks of code) to allow them to be suspended and resumed. The capability for coroutines is often provided at the programming language level with specialized keywords. Coroutines are a type of lightweight thread, like green threads and fibers, that may but often do not run in parallel. Coroutines use cooperative multitasking,

	requiring threads of execution to explicitly yield control. Python provides coroutines via the yield keyword, via generators, and via the async and await keywords all within a single system thread.
Critical Block	*See* **Critical Section**
Critical Section	Code that operates on state or resources that are shared between processes or threads. Critical sections should not be executed by more than one process or thread at a time. This can be achieved using synchronization primitives like locks. If access to the shared resource is not synchronized, it can lead to concurrency failures, such as a race condition.

D

Daemon	A type of process or thread that runs as a background process rather than interacting with the user, e.g. daemon thread or daemon process. A thread or process that is a daemon may be referred to as daemonic. In Python (like many languages), the presence of a running daemon thread will not prevent the main process from exiting when the main thread exits, unlike non-daemon threads.
Deadlock	A concurrency failure case when a thread or process is waiting on another thread or process, such for a result or for a lock that will never become available. This can happen for many reasons, such as a lock that was acquired but not released or a thread or process waiting on itself.
Dummy Thread	An object created to represent threads created by a Python program in C and C++ libraries, so-called alien threads. A dummy thread object may be available when enumerating all threads in a process, such as via the threading.enumerate() function.

E

Embarrassingly Parallel	A task that can be decomposed into independent (or nearly independent) subtasks such that executing subtasks concurrently is trivial. Examples

include a large number of algorithms such as the Monte Carlo method, ray tracing, fractals, brute force search, and many more.

F

Fiber A type of lightweight thread, like green threads and coroutines, that may but often do not run in parallel. Support for fibers is often provided by the operating system, as opposed to the programming language for coroutines or a code library in the case of green threads. Fibers use cooperative multitasking, requiring threads of execution to explicitly yield control.

Fork A method for creating a new process provided by the underlying operating system. A process can be forked to create a child process. In Python, forking a process is supported on Linux and MacOS but not Windows.

Future A programming pattern in concurrent programming that provides a placeholder or proxy for a result to an asynchronous task. Also called a promise or a delayed result. For example, a future object may handle any inter-process or inter-thread coordination and synchronization involved, allowing the caller to check on the status of the task or wait for the task to complete.

G

GIL *See* **Global Interpreter Lock**

Global Interpreter Lock A programming pattern in the reference Python interpreter (CPython) that uses synchronization to ensure that only one thread can execute instructions at a time within a Python process. This means that although we may have multiple threads, only one thread can execute at a time.

Green Thread A user thread or virtual thread of execution provided by a runtime environment such as a library or a virtual machine, as opposed to a

system thread. Python uses system threads and not green threads, although some libraries do provide green threads.

H

Hyperthreading Technology in modern integrated circuits used in CPU cores that allows for the parallel execution of instruction in some circumstances. This allows each physical CPU core to be viewed and used by the operating system as two logical CPU cores.

I

IO I/O or Input/Output. Refers to tasks that either read data in from a resource or write data to a resource. Resources may be streams like stdin and stdout, network connections, files, devices, and more. IO operations are slow relative to executing instructions in the CPU.

IO-bound A task where the time taken to complete is determined by the time spent reading from input or writing to output, not the speed of the CPU. This is the opposite of a CPU-bound task. For example, IO-bound tasks include writing to file and reading from a network connection. IO operations are slow relative to the speed of CPUs and a thread performing an IO-bound task can block the CPU until the task is complete.

IPC *See* **Inter-Process Communication**

Inter-Process Communication A programming pattern for communicating between processes. Typically uses message passing and requires the serialization of data transmitted between processes. Examples include messages queues, pipes, and channels, although it is also common to use signals, files, and sockets. Python provides inter-process communication via the multiprocessing.Queue and multiprocessing.Pipe classes.

J

Join A concurrency operation where one thread or process will wait on another thread or process to complete execution, after which execution will continue. A join in a blocking call for the joiner. The joinee is unaffected. Python provides the ability to join a thread or a process object via the join() function.

K

Kill A process by which a thread or process is forcibly terminated. This may be a feature provided by the programming language or the killing of a thread or process externally to the program, such as the “kill” command on POSIX systems. Python does not provide the ability to kill threads or processes.

L

Latch A synchronization primitive that will notify waiting threads or processes once the latch condition has been triggered. Typically, a latch is triggered once a predefined number of other processes or threads have reached the latch, called a count-down latch. Waiting and triggering threads on the latch are separate, unlike a barrier where they are the same. Python does not provide a latch class.

Livelock A concurrency failure condition where threads or processes are unable to progress as they are blocked by synchronization primitives. Unlike a deadlock in which the threads or processes do not progress, in a livelock, the threads and processes may progress through a cycle of locks and triggers, yet the overall tasks do not progress.

Lock A synchronization primitive that limits access to a resource or block of code. A process or thread must acquire the lock before accessing the resource, then release the lock once finished. Forces access to the resource to be mutually exclusive among processes or threads; as such, it is often referred to simply as a mutex. Python supports locks by the

threading.Lock class for threads and the multiprocessing.Lock class for processes.

Lock Convoy A concurrency failure case where multiple threads or processes repeatedly compete for the same lock. A cost is incurred by threads or processes being notified that the lock is available, obtaining time to execute via a context switch, and waiting again for the lock, ultimately wasting the context switch.

Lock-Free A programming pattern where concurrency is achieved without locks (called lock-free) and/or without waiting (called wait-free), together referred to as non-blocking. May require the use of specialized data structures and/or specialized atomic language primitives such as read-modify-write and compare-and-swap.

Lock-Less *See* **Lock-Free**

Logical CPU Core A virtual CPU core as seen by the operating system, as opposed to a physical CPU core. Enabled by hyperthreading where each physical CPU can be seen and used by the operating system as two logical CPU cores.

M

Main Thread The thread of execution created and run by a Python process. This thread has the name “Main Thread”. Once the main thread exits, the process will exit if there are no other non-daemon threads running.

Multicore A microprocessor chip that has more than one CPU core. Modern CPUs are multicore and may have 2, 4, 8, or even many more physical CPU cores.

Multiprocessing A program that uses more than one process, e.g. child processes. Python supports multiprocessing via the Process class and the ProcessPoolExeuctor class.

Multitasking A capability of modern operating systems that allows the concurrent execution of multiple program processes. Achieved using context

switching. There are two main types of multitasking: cooperative and preemptive.

Multithreaded A program that uses more than one thread of execution. Python supports multithreading via the Thread class and the ThreadPoolExecutor class.

Mutex *See* **Lock**

Mutual Exclusion Lock *See* **Lock**

N

Non-blocking An instruction, or more typically a function, call that does not block the execution of the thread and returns immediately. Suggests it is not waiting on another resource (e.g. a file or network connection) or another thread (e.g. via a synchronization primitive).

Notify A concurrency programming pattern where threads or processes that are blocked waiting on a resource are alerted that the resource is available. Typically, a single waiter can be alerted via notify, whereas all waiters can be altered by notify all. Python provides the notify() and notify_all() functionality via the threading.Condition and multiprocessing.Condition conditional variable objects.

P

PID *See* **Process Identifier**

Parallel Execution of program instructions at the same time. For example, four tasks can be executed at the same time with multiple CPU cores. Parallelism is a way of implementing task concurrency and task asynchrony.

Parallel Programming An engineering discipline for writing programs that contain code that is expected to execute simultaneously. Perhaps a subset of concurrent programming and distributed programming.

Physical CPU Core	An actual CPU core within the microprocessor, as opposed to a logical CPU core.
Pipe	*See* **Channel**
Pipeline	A programming pattern where a sequence of processing steps are executed in a sequence. Raw data or triggering events may be fed into one end of the pipeline that may then pass from step to step within the pipeline, resulting in a final outcome. More or fewer tasks may be created in response to data at each step of the pipeline. In concurrency programming, a thread pool may be used to execute tasks at each step in the process.
Polling	A concurrency programming pattern where a resource is periodically checked. For example, an algorithm may poll for the arrival of new data on an IO resource like a network connection. A watchdog process may use polling to check the status of a resource and a spinlock may use polling to check if a lock can be acquired.
Preemptive Multitasking	A type of multitasking used in modern operating systems (e.g. Linux, Windows and MacOS) where the operating system kernel will decide when to context switch between programs, attempting to preempt the process that needs to execute next. Context switches can be triggered when a process blocks or waits for a resource, such as an IO-bound task.
Priority	An ordinal relationship between threads or processes in which threads and processes with higher priority (a lower number) are offered more opportunity for execution by the operating system than threads and processes with lower priority (larger number). Python currently does not provide the ability to specify the priority of threads or processes.
Priority Inversion	A concurrency failure condition where lower priority threads or processes are given priority over higher priority threads or processes, inverting the expected assigned priority. Can happen if a low priority thread or process monopolizes (directly or indirectly via third thread or process) a resource that is dependent upon both higher and lower priority threads or processes.
Process	An instance of a computer program that will have at least one (and

perhaps more) threads of execution called the main thread. A process will have a unique process identifier (pid) allocated by the operating system. Running a Python program is a process that is an instance of the Python interpreter execution instructions in a Python script file. A program process may spawn or fork additional sub-process called child processes. Python provides access to processes via the Process class.

Process Identifier Unique identifier for a process allocated by the underlying operating system. Python processes access to the pid via the get_pid() function and to the parent process identifier via the get_ppid() function.

Process Pool A thread pool where workers are implemented using processes instead of threads. Python provides a process pool via the ProcessPoolExecutor class.

Promise *See* **Future**

R

RLock *See* **Reentrant Lock**

Race Condition A concurrency failure case when two or more threads are able to access a critical section at the same time without synchronization. This can result in the non-deterministic changes to program state as multiple processes and threads may “race” to and then through the critical section. This can lead to unexpected behavior and corruption of data. For example, many threads or processes updating the same variable at the same time.

Reentrant Lock A synchronization primitive that grants a lock to a thread or process without blocking if it has already acquired the lock previously. This is unlike a (non-reentrant) Lock that requires each thread or process to acquire the lock again, even if it has already acquired the lock. Useful if the same lock is used to protect multiple critical sections that may be executed in any order. Python provides a reentrant lock via the threading.RLock and multiprocessing.RLock classes.

S

Semaphore A synchronization primitive for controlling access to a resource for a fixed number of processes or threads. Unlike a lock where access to a critical section is mutually exclusive for a single process or thread, a semaphore allows multiple concurrent threads or processes to access a critical section. A semaphore configured with access set to one is equivalent to a mutex lock. Python supports semaphores via the threading.Semaphore class for threads and multiprocessing.Semaphore class for processes.

Sleep An instruction that will pause or suspend a process or thread of execution for a specified time. Often, sleep will trigger a context switch by the operating system, much like any blocking call. Python supports sleep via the sleep() function.

Spawn A method for creating a new process provided by the underlying operating system. A child process can be spawned by a parent process. In Python, spawning processes are supported on all major operating systems.

Spin *See* **Spinning**

Spinlock A synchronization primitive where threads or processes waiting for the lock will spin while waiting — that is, execute a loop that re-checks whether the lock can be acquired each iteration. This can be more computationally expensive than a lock in which threads and processes block while waiting to acquire the lock. Can be useful if the wait time is expected to be short or the test for the lock is unreliable, e.g. subject to a race.

Spinning A concurrency primitive where threads or processes waiting for a resource from another thread or process will execute a loop re-checking the resource each iteration. This can be computationally expensive compared to blocking while waiting e.g. not executing instructions.

Starvation A concurrency failure case where processes and threads are unable to acquire access to a critical section via its protecting synchronization primitives (e.g. lock) because a controlling process or thread will not

release control. A deadlock can lead to starvation for other processes and threads.

Sync *See* **Synchronization**

Synchronization A programming pattern where two or more processes or threads are unable to execute the same instruction or block of code, called a critical section. Synchronization is often managed by primitives such as locks. For example, if a block of code is synchronized using a lock, then a process or thread must acquire the lock before executing the block, and if the lock has already been acquired, then it must wait. A failure to synchronize at a critical section can lead to concurrency failures, such as a race condition. Misuse of a synchronization primitive can lead to a concurrency failure, such as a deadlock.

Synchronization Primitives Commonly used programming patterns to synchronize processes or threads. Examples include lock, reentrant lock, semaphore, barrier, and latch.

Synchronous The occurrence of events or execution of instructions that are dependent on each other and must happen in a prescribed sequence. Opposite of asynchronous.

System Thread A native or kernel thread of execution provided and managed (scheduled) by the operating system, as opposed to a green thread. Python uses system threads.

T

Task A discrete unit of work, such as a function call. Typically meaningful within the context of the program or problem domain. In concurrency programming, a task is a unit of work that may be executed in another thread or process and return a result when completed.

Thread A sequence of instructions in a computer program that are being executed. A thread belongs to a process. For example, every process has

	a main thread, and may have additional threads. Python provides access to threads via the Thread class.
Thread Group	A programming pattern that allows threads to be grouped together. Python does not support thread groups.
Thread Pool	A programming pattern that provides a collection of reusable threads for executing submitted tasks concurrently. Allows the program to focus on submitting tasks and using results rather than coordinating and synchronizing threads of execution. Python provides thread pools via the ThreadPoolExecutor class.
Thread Safety	Concurrent code that ensures that critical sections are appropriately synchronized in order to avoid failure cases, such as deadlocks and race conditions. Code that does not appropriately protect critical sections is not thread safe.
Thread-Local Storage	A way of storing variables that are only available to the context of a thread of execution. A feature provided by modern operating systems. Python provides thread-local storage via the threading.Local class.
Thundering Herd	A concurrency failure case in which a large number of processes or threads waiting on an event are triggered, but only one waiter can handle the event. May result in significantly wasted resources in waking up and context-switching many processes or threads only to have them wait again. Can be solved by adding a back-off.
Timeout	A concurrency programming pattern where an algorithm will wait a fixed time for an event before giving up. The algorithm may spin or block while waiting. A timeout is often used on function calls that may block, such as reading from an IO resource (like a network connection) or waiting to acquire a lock. Python provides timeouts on most blocking calls.
Timer	A concurrency programming pattern where an algorithm will not perform an action until a pre-specified amount of time has elapsed. Python provides a timer for executing a task in another thread via the threading.Timer class.

V

Volatile A programming pattern that indicates that variables may change between accesses, such as in concurrent programs by separate threads. Python does not support volatile variables.

W

Wait A concurrency operation where a thread or process will block until a condition is met. For example, a thread or process may wait on a synchronization primitive, such as wait to acquire a lock, wait on a condition variable, or wait on a barrier. The function may be named wait() or may be less obvious, such as acquire(), result(), join(), or similar.

Wait-Free *See* **Lock-Free**

Watchdog A concurrency programming pattern where a thread or process executes a task whose responsibility it is to monitor a resource and take action if a problem is identified. A watchdog is often executed in the background, e.g. daemonic. May be implemented using a timed polling of the status of the resource that is being watched.

I hope that you find this glossary helpful!

Is something confusing?
Do you want me to define another term?
Let me know (https://superfastpython.com/contact/).