Anthony Ventresque

anthony.ventresque@ucd.ie

Operating Systems

COMP30640

Process Management II
Threads

School of Computer Science, UCD

Scoil na Ríomheolaíochta, UCD

Outline

- Explain the challenges of process cooperation
- Understand the concept of a thread, its motivation, lifecycle, states and role in concurrency
- Discuss single vs. multithreading
- Explore concurrency accomplished by multiplexing CPU Time

Take home message:

Program -> Process -> Thread

Independent Processes

- ***Definition***: Independent processes are those that can neither affect nor be affected by the rest of the system
  - Two independent processes cannot share system state or data
  - Processes running on different non-networked computers

- Properties:
  - ***Deterministic behaviour***: only the input state determines the results → ***reproducible***
  - Can be stopped and restarted with no detrimental effects

Cooperative Processes

- ***Definition***: Cooperative processes are those that share something (not necessarily for a purpose)
  - Two processes are cooperative if the execution of one of them may affect the execution of the other
  - Processes that share a single file system
- Properties:
  - ***Nondeterministic behaviour*** (from the point of view of one of the processes) → it may be ***difficult to reproduce***
  - hence: testing & debugging may be troublesome
  - Subject to ***race conditions***: the outcome of the process may depend on the sequence or timing of events in other processes

Why Allow Processes to Cooperate?

- Resources and information sharing
  - One computer, many users
- System speed-up, by introducing concurrency into program execution
  - Overlap I/O with computations
- Convenience
  - Editing a file at the same time it is being printed
- Modularity, simplicity, divide & conquer
  - structured programming, object programming, etc.

Process and Collaboration

- Multiple concurrent cooperative activities necessarily happen within any OS
- Why not define each and every one of these concurrent activities as a different process?

- ***Processes are not ideal for cooperation:***
- Processes are not very efficient:
  - Creation of a new process is costly
  - All the process structures must be allocated upon creation
- Processes don't (directly) share memory
  - Each process runs in its own address space
  - Parallel and concurrent processes often want to manipulate same data
  - Most communications go through the OS: slow

Motivation for Threads

- Example: consider a file server process that occasionally has to block waiting for the disk to respond (I/O wait)
  - Assume that it keeps a cache of recently used files in its memory space, in order to speed up future operations
  - We might think of running a second concurrent file server to serve files while the first file server waits for disk I/O (principle of multiprogramming)
- However concurrency is not efficient using processes
  - They would have to run in the same address space to efficiently share a common cache (but they do not)
  - We could use shared memory, but slow system calls would be required
- ***Solution: threads***
  - More than one thread of control within a single process
  - i.e.: more than one active entity within a single process

Threads

- Modern OSs support both entities (process & thread): ***multi-threaded OS***
  - ***Process***: defines the address space and general process attributes
  - ***Thread***: defines a single sequential execution stream within a process
  - ***Multithreading***: a single program made up of a number of different concurrent activities

- Concurrency in some existing OS:
  - MS-DOS: one address space, one thread
  - Unix (originally): multiple address spaces, one thread per address space
  - Mach, Chorus, Solaris, NT: multiple address spaces, multiple threads per address space (multi-threading)

More on Threads

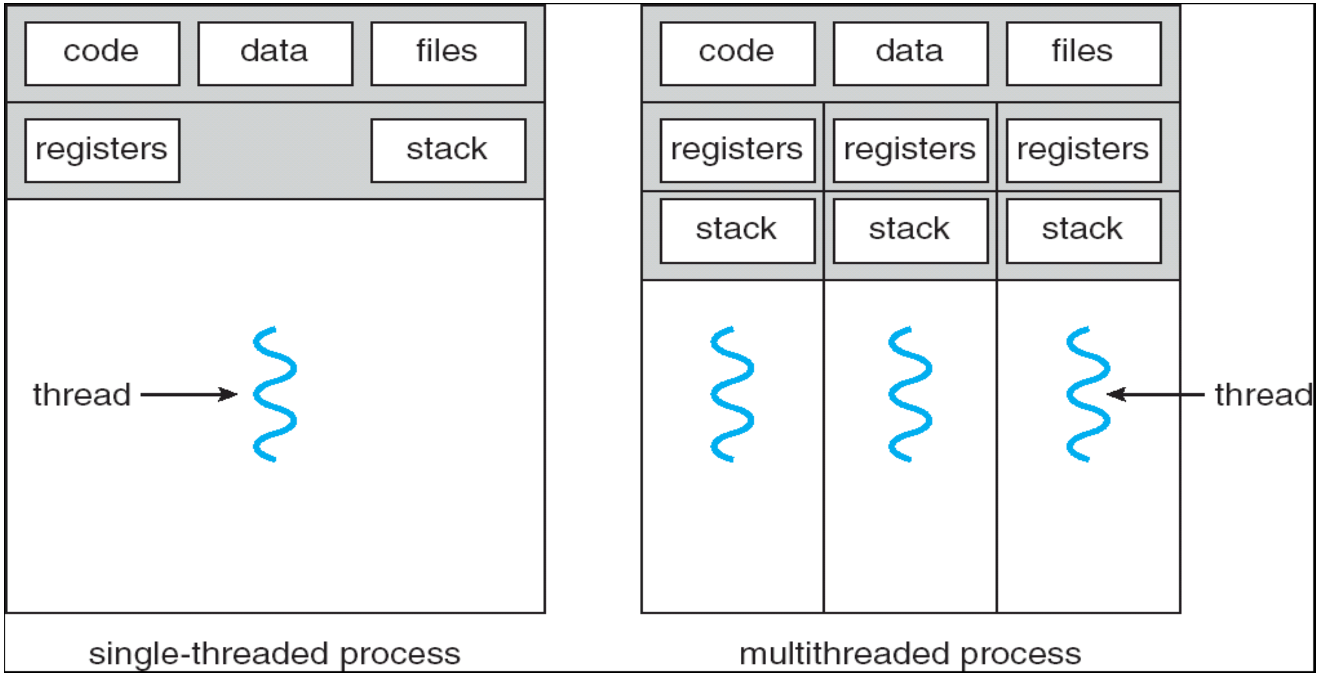
Threads’ features:

- ***Cheap to create*** (no need to allocate PCB, new address space); they can be created statically or dynamically (by a process or by another thread)
- Threads ***do not exist on their own***, they belong to processes: number of threads in a process ≥ 1
- Threads can ***communicate with each other efficiently*** through the process global variables or through common memory, using simple primitives
- Threads ***facilitate concurrency***, and therefore are useful even on single processor systems
- If a thread needs a service provided by the OS (system call) it ***acts on behalf of the process*** it belongs to

Examples of Multithreaded Programs

- Embedded systems
  - Elevators, Planes, Medical systems, Wristwatches
  - Single Program, concurrent operations
- Most modern OS kernels
  - Internally concurrent because have to deal with concurrent requests by multiple users
  - But no protection needed within kernel
- Database Servers
  - Access to shared data by many concurrent users
  - Also background utility processing must be done

Single and Multithreaded Processes

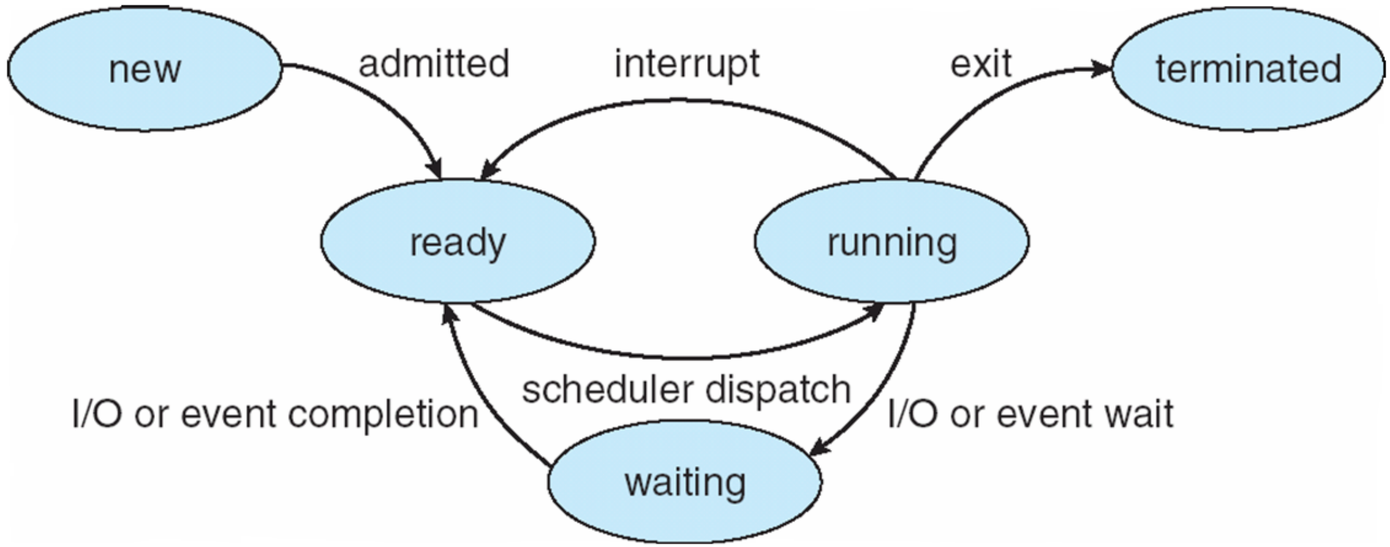


- Threads encapsulate concurrency: "Active" component
- Address spaces encapsulate protection: "Passive" part
  - Keeps buggy program from trashing the system
- Why have multiple threads per address space?

Thread State

- ***State shared*** by all threads in process/address space
  - Address space (code and data)
  - Content of memory (global variables, heap)
  - I/O state (file descriptors, open files, network connections, etc)
  - Privileges
  - Timers, Signals, Semaphores
  - Accounting information
- ***State "private"*** to each thread
  - Kept ***in TCB ≡ Thread Control Block***
  - CPU register set, in particular:
    - program counter (PC)
    - stack pointer (SP)
    - interrupt vectors
  - Stack (Parameters, temporary variables)
  - State
  - Child threads (threads can spawn new threads)

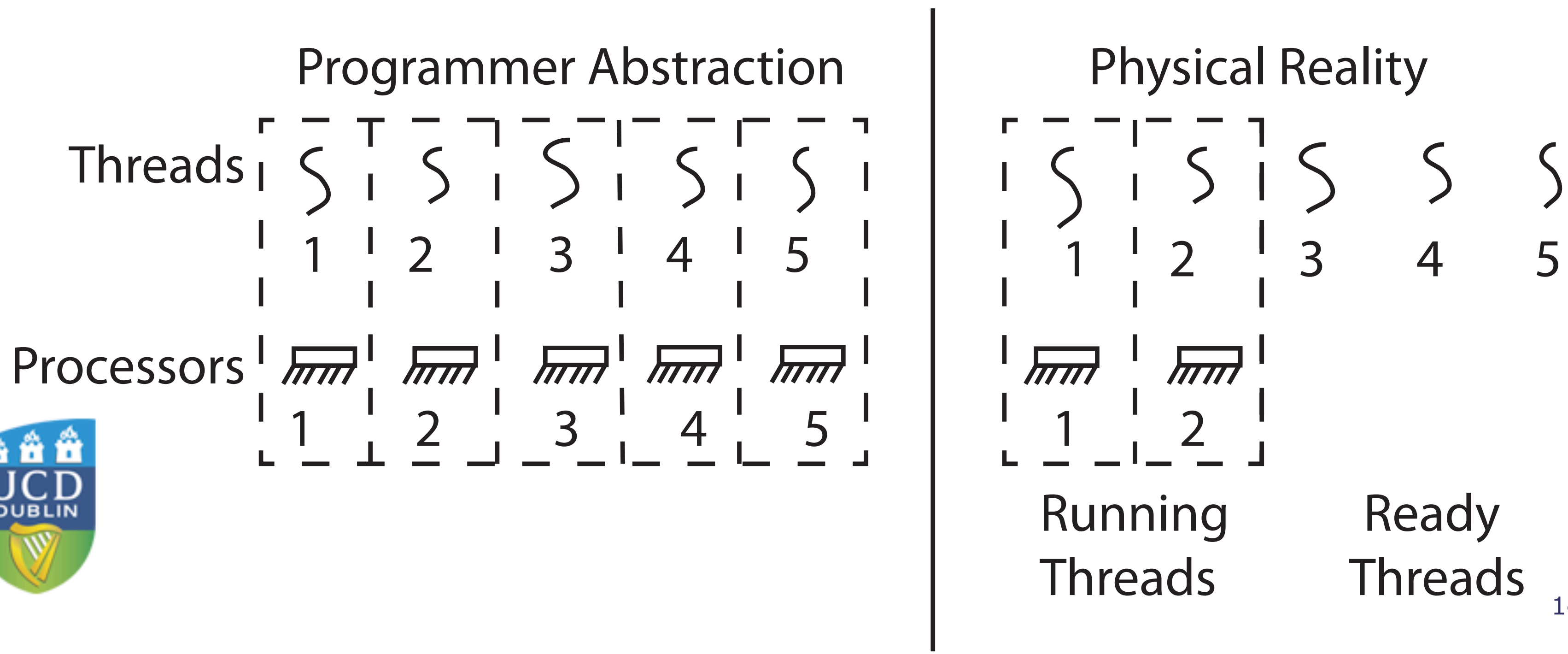
Lifecycle of a Thread (or Process)



- As a thread executes, it changes state:
  - ***New***: The thread is being created
  - ***Ready***: The thread is waiting to run
  - ***Running***: Instructions are being executed
  - ***Waiting***: Thread waiting for some event to occur
  - ***Terminated***: The thread has finished execution
- "Active" threads are represented by their TCBs
  - TCBs organized into queues based on their state

Thread Abstraction

- Infinite number of processors
- Threads execute with variable speed
  - Programs must be designed to work with any schedule

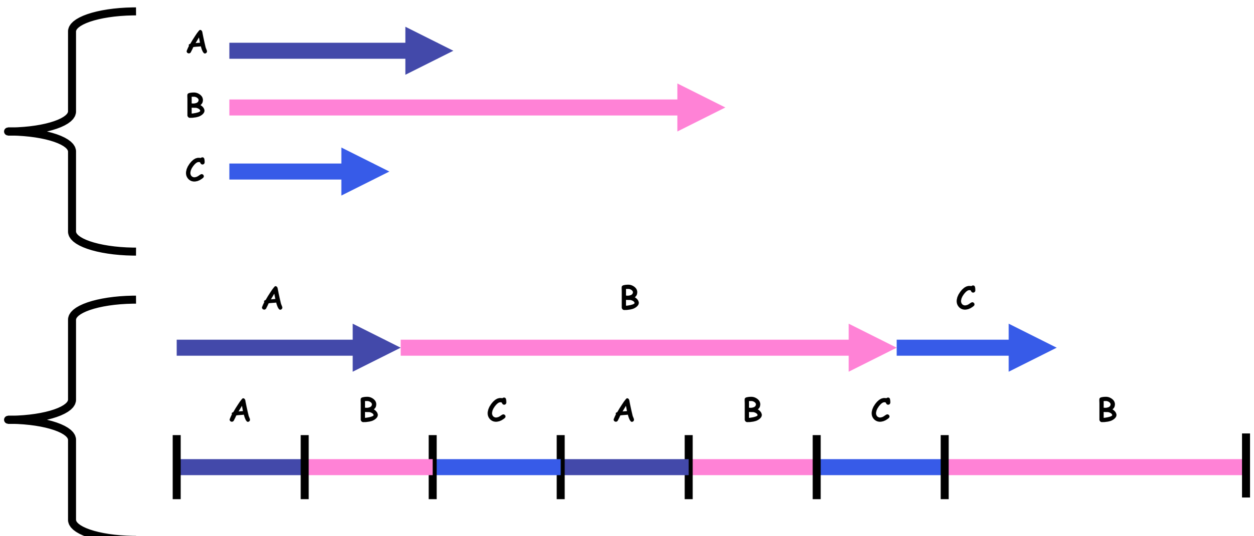


Multiprocessing vs. Multiprogramming

- Definitions:
  - Multiprocessing ≡ Multiple CPUs
  - Multiprogramming ≡ Multiple Jobs or Processes
  - Multithreading ≡ Multiple threads per Process
- What does it mean to run two threads "concurrently"?
  - Scheduler is free to run threads in any order and interleaving: FIFO, Random, …
  - Dispatcher can choose to run each thread to completion or time-slice in big chunks or small chunks

Multiprocessing

Multiprogramming



Programmer vs. Processor View

Programmer's View	Possible Execution #1	Possible Execution #2	Possible Execution #3
.	.	.	.
.	.	.	.
.	.	.	.
x = x + 1;	x = x + 1;	x = x + 1	x = x + 1
y = y + x;	y = y + x;	....	y = y + x
z = x +5y;	z = x + 5y;	thread is suspended	....
.	.	other thread(s) run	thread is suspended
.	.	thread is resumed	other thread(s) run
.	.	....	thread is resumed
		y = y + x	....
		z = x + 5y	z = x + 5y

Conclusion

- It is difficult for processes to cooperate
- ***Thread***: defines a single sequential execution stream within a process
- Multithreading: a single program made up of a number of different concurrent activities
- Threads encapsulate concurrency: the “Active” component of a process