Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨Daniel.Page@bristol.ac.uk⟩

January 21, 2016

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

COMS20001
Concurrent Computing

Notes:

- The concept of concurrent computing exists at *various* scales

Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are

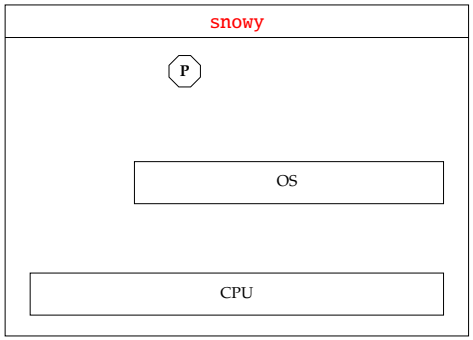
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).

- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are

  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

- The concept of concurrent computing exists at *various* scales



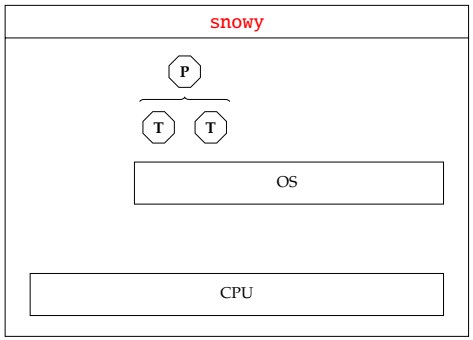
Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

COMS20001 intro. lecture: week #13/#19

- The concept of concurrent computing exists at *various* scales

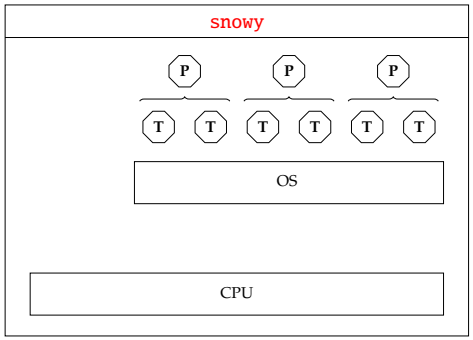


Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

- The concept of concurrent computing exists at *various* scales



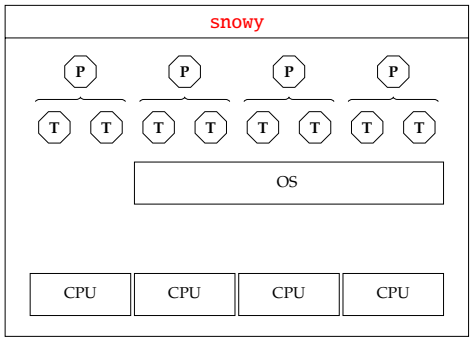
Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

COMS20001 intro. lecture: week #13/#19

- The concept of concurrent computing exists at *various* scales

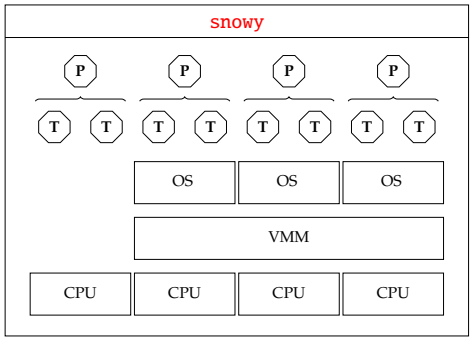


Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

- The concept of concurrent computing exists at *various* scales



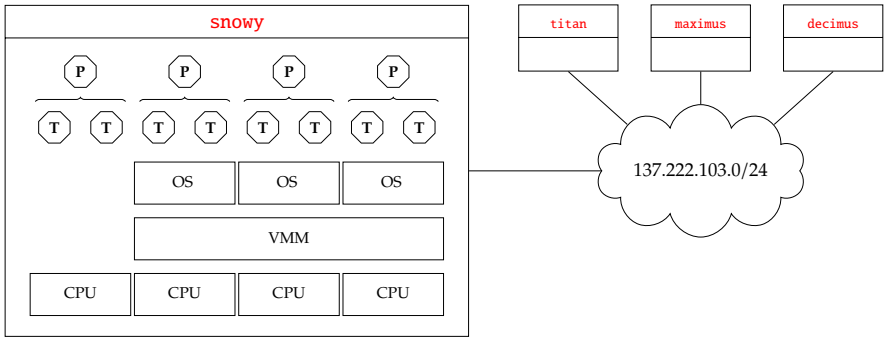
Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

COMS20001 intro. lecture: week #13/#19

- The concept of concurrent computing exists at *various* scales

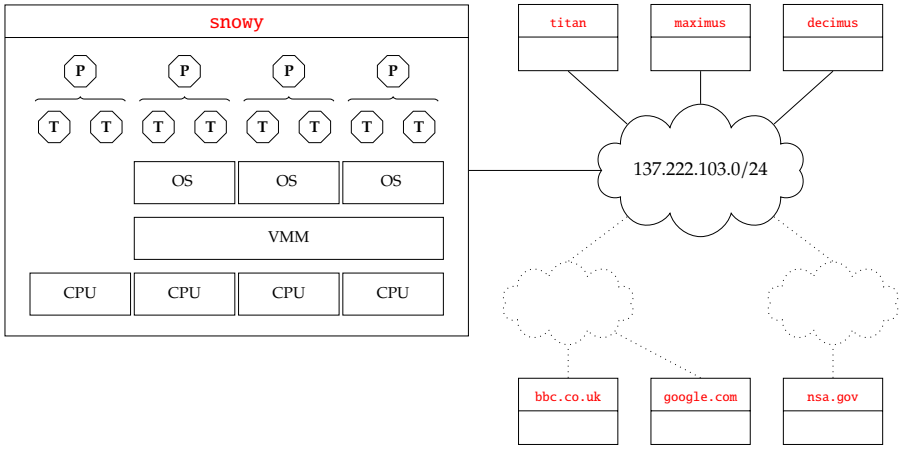


Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

- The concept of concurrent computing exists at *various* scales



Notes:

- You can think of communication (via whatever medium, e.g., a network) as a subtle but natural requirement for useful concurrency, and therefore a clearly important topic within the unit as a whole.
  For example, if components operate concurrently but never interact, there is no need for communication: they operate as separate systems. When they *do* need to interact, however, which is arguably the norm given how we actually utilise them, the question is *how*.
- From start to finish, the concurrent components illustrated here are
  1. many threads,
  2. many processes (or applications),
  3. many processors,
  4. many operating systems,
  5. many hosts, and
  6. many networks (cf. an *inter*network).
- Another way to look at the relationship with earlier parts of the unit is to consider them as the same, bar the issue of **scale**. Or, a natural next step if one considers what components can be (or *are* already) concurrent: phrased link this, overall we've covered concurrent system that are
  1. small-scale (or local) concurrency within one host, and now
  2. large-scale (or distributed) concurrency within multiple hosts

  but that, fundamentally, rely on similar overarching theory. This issue of scale really represents the central challenge: solutions which may be viable on a local system (e.g., shared memory), are often ineffective (or at least are forced to cater for different requirements, and so need to be designed differently somehow) on a distributed system.
- The issue of scale *will* get worse, so understanding of fundamentals will (arguably) increase in importance: next-generation paradigms such as the so-called **Internet of Things (IoT)** stress every aspect of current solutions (e.g., IPv4-based networks), so, as part of a familiar technology cycle, we need to cope somehow!

COMS20001 intro. lecture: week #13/#19

- Content: the latter half of COMS20001 concerns
  1. **operating systems**, and
  2. **computer networks**.
- Delivery: our approach to each topic is st.
  - we want to focus on principles to avoid dependency on specific techniques tied to specific (time-limited) technology, *but*
  - although one could view each topic abstractly, this conflicts with the practical, real-world nature of both topics, *so*
  - we'll use concrete examples and technical detail to illustrate the underlying principles, *and*
  - we'll take a bottom-up approach.

Notes:

- Within both topics, there are is *huge* range of content that could be covered but will not be (or will be, but only at a very high level). On one hand, this is unfortunate: various such examples are interesting and/or useful. On the other hand, it's important to see
  - there is no time to cover everything, so solid understanding of a more limited set of topics is preferred to superficial treatment of *everything*, and
  - this compromise is reasonable in the sense that later units are often specifically intended to cover such content: sine examples include COMS30004, COMSM1500, COMSM0010 and COMSM2006.
- The goal of focusing on principles but also deliver technical detail may seem contradictory, and up to a point it is. Understanding the former is attractive since it allows the underlying principles to be used in other contexts (e.g., communication networks and on-chip interconnects are not *so* different in some respects), and to innovate in terms of improving the state-of-the-art. However, being able to *use* such principles to solve real problems (e.g., write high-performance network applications) is also important, and demands knowledge of a more technical nature. So, in reality, *both* goals are important and therefore we try to reach a compromise between them in most cases.

COMS20001 intro. lecture: week #13/#19 – operating systems

- Goal: understand
  1. hardware/software interface
     - interrupts, port- and memory-mapped I/O, DMA
     - ARMv7-A, Cortex-A8
  2. device management
     - block devices, character devices, network devices device drivers
  3. process management
     - threads, processes, context switches, cooperative scheduling pre-emptive scheduling
     - `fork`, `exec`, `exit`, etc.
  4. memory management
     - translation, protection, swapping, segmented memory, paged memory, demand paging
     - PMSA, VMSA
  5. file management (i.e., file systems)
     - files, directories, meta-data, contigious, linked and indexed block allocation
     - ext2, ext3

  while emphasising general concepts as applied in specific technologies (using a running example) ...
- ... *or*, in simple terms, understand how Linux works.

Notes:

- A non-exhaustive list of out-of-scope topics include
  1. anything from the first half of the unit (e.g., semaphors, deadlock),
  2. history,
  3. security,
  4. kernel-specific (sub-)topics in data structures and algorithms,
  5. IPC,
  6. kernels for real-time systems,
  7. kernels for multi- and many-core processors.

COMS20001 intro. lecture: week #13/#19 – computer networks

- Goal: understand
  1. physical layer
     - transmission, modulation, multiplexing, metrics
  2. link layer
     - addressing, framing, multiple access, switches
     - 802.3 (Ethernet), 802.11 (WiFi)
  3. internet layer
     - addressing, fragmentation, forwarding, routing
     - IP, ICMP, DHCP, ARP
  4. transport layer
     - connection management, ARQ, flow control
     - UDP, TCP,

  and then
  5. application layer
     - sockets API
     - NAT, DNS

  while emphasising general concepts as applied in specific technologies (using a running example) ...
- ... *or*, in simple terms, understand how the Internet works.

Notes:

- A non-exhaustive list of out-of-scope topics include
  1. anything from the first half of the unit (e.g., blocking vs. non-blocking communication),
  2. history,
  3. security,
  4. wireless communication (instead covering wired mediums alone),
  5. IPv6 (instead covering IPv4 alone),
  6. Software Defined Networks (SDN),
  7. techniques for error detection and correction,
  8. (advanced) techniques for routing in IPv4,
  9. (advanced) techniques for congestion control in TCP.

References

[1] A. Silberschatz, P.B. Galvin, and G. Gagne.
Operating System Concepts.
Wiley, 9th edition, 2014.

[2] W. Stallings.
Data and Computer Communications.
Pearson, 9th edition, 2010.

[3] A.S. Tanenbaum and H. Bos.
Modern Operating Systems.
Pearson, 4th edition, 2015.

Notes: