G52OSC
OPERATING SYSTEMS AND CONCURRENCY

Summary, Exam format and Revision

Dr Jason Atkin

Evaluate: SET/SEM

- Please visit Bluecastle https://bluecastle.nottingham.ac.uk/Account/Login?ReturnUrl=%2f
- And choose “My Surveys”
  - Please find both the SET and SEM evaluations for G52OSC
- Please take care to understand which way round 1 and 5 are, i.e. which is good
  - Apparently 5 is BAD not good
  - Many people have got them the wrong way around this year

Exam questions

Exam structure

- **Complete three questions**
- **Including at least one on concurrency**
  - Choice of 2
- **And at least one on operating systems**
  - Choice of 3
- Either both concurrency and 1 of the ops
- Or one concurrency and 2 of the 3 ops
- All questions worth the same mark
  - 25 marks (I believe) for 25% of the module
  - 3 questions are worth 75% together

What to expect from concurrency questions

- Most G52CON exam questions will be good practice
  - Much less Java oriented now though
  - More likely to ask about pseudo-code than Java (or C)
  - Things like traces, critical sections, deadlock, etc still apply
- **May** ask about operating system support / concepts
  - Will not expect you to know API function names and parameters
  - MAY expect you to know what a common function does though
- I would expect you to know key concurrency concepts, elements (**e.g. semaphore, monitor**) and operating system support that you have seen
  - E.g. Locking (CreateMutex, CreateSemaphore, InitializeCriticalSection), and what locking means
  - Message passing (SendMessage vs PostMessage) and the difference between asynchronous and synchronous message passing

Example Exam-type Questions

We covered a lot of things…

- Creating threads
- Windows messages and message queues
  - Shared between windows, asynchronous vs synchronous
- **Lab 1: windows messages and threads**
- Creating processes
- Sharing memory between processes
- Interlocked (atomic) API
- **Lab 2: atomic operations and volatile data**
- Dekker and Peterson's algorithms
  - Memory barriers
- Critical section and mutex objects
- **Lab 3: critical sections, mutexes and atomic locks**
- Semaphores and consumer/producer
- **Lab 4: semaphores, (more) windows messages and sockets**
- Java, monitors and windows events

Windows messages

- There **may** be some link to what you have done with a specific operating system
  - I would expect to use things that you will know from labs and coursework
  - Especially when it relates to a generic concept, like this:
- Q: Explain the main features of the window handling system in Windows
  - Ensure that your answer explains how a single threaded windows program can handle multiple windows

Key Features

- Window classes (registered, by name)
- **Callback function**
  - **Associated with class**
  - **Handles messages**
- Create Window – create instance of window
  - Multiple windows associated with same thread
- **Message queue concept**
  - PostMessage (**async**) and SendMessage (**sync**)
- **Message Loop**
  - GetMessage – **pick up next message** (Peek?)
  - DispatchMessage – **call callback for target window**

A follow-up question

- Consider your answer to the previous question
- Explain how the SendMessage and PostMessage functions can be used for inter-process communication
- What are the limits on the data that can be sent and what the differences are between SendMessage and PostMessage?

- (Synchronous vs asynchronous comms)

Example – locking on Windows

- Q: What is/are the difference(s) between a mutex and CRITICAL_SECTION in windows?

- Q: Give an example of when you would use each of these in preference to the other

Example answer key features

- **Put your answer in full clear sentences**
- We will look for **understanding** of **key features**

- What is/are the difference(s) between a mutex and CRITICAL_SECTION in windows?
  - Critical section: lightweight, only within a process, fast if no contention
  - Mutex: works cross-process, may be slower
- Give an example of when you would use each of these in preference to the other
  - Depends upon cross process or intra-process

Example: semaphore vs mutex

- Q: What are the differences between a mutex and semaphore?

Answer: semaphore vs mutex

- What are the differences between a mutex and semaphore?

- Semaphore has a count, whereas mutex is binary – locked or not
  - Could have a binary semaphore too though
- Mutex has an owner:
  - No problem with locking it when already locked
  - Operating system may be able to detect if thread dies when holding mutex
  - Any thread can do v() on semaphore without owning it

Mutual Exclusion Protocol

Q: How can you avoid interference between threads/processes

A: Identify critical sections

Define entry and exit protocols:

- Peterson's Algorithm
- Atomic operations (e.g. test and set spin-lock)
- Mutex
  - Binary lock with ownership
- Semaphore
  - Counting lock without ownership
- Monitor
  - Lock + condition variables (wait queue)
  - Atomic "unlock and wait" operation is needed
  - Event : operating system concept to wait on

Example pseudo-code question

Fundamental components

- **Data types:**
  - ***Locks:*** These are binary locks which are either locked or unlocked. There is no concept of ownership with a lock, so a thread which attempts to lock the same lock twice would hang.
  - ***ThreadQueues:*** These are first-in-first-out queues of threads (which may be in the same or different processes).
  - ***Basic data types***: You may create variables of any normal basic type, such as int, float, double, and may also use a string type if you wish.
  - ***Thread***: a type to store a reference to a thread if you need it
- **Operations: (assume that these are atomic)**
  - ***Lock(lockname):*** This will lock the object or block until it can do so if it is already locked.
  - ***Unlock(lockname):*** This will unlock the specified object if it is locked, regardless of which thread locked it.
  - ***Push(threadqueuename):*** this will add the current thread to the back of the specified thread queue and block until the thread is removed from the queue.
  - ***Pop(threadqueuename):*** this will remove whatever thread which was at the front of the thread queue from the queue and wake up the thread, so that it will continue its processing immediately.
  - ***PushAndUnlock(threadqueuename, lockname):*** this will add the current thread to the back of the specified thread queue, then unlock the specified lock, then finally it will block until the thread is removed from the queue.
  - ***Count(threadqueuename):*** return a count of the number of threads currently on the specified thread queue.

Questions

- Q: What functions can you perform on a mutex lock?

- Q: Write pseudo-code using the fundamental components and any conditionals/loops which you need, to create these functions for a mutex lock which **has ownership of the lock**

Fundamental components

- **Data types:**
  - ***Locks:*** These are binary locks which are either locked or unlocked. There is no concept of ownership with a lock, so a thread which attempts to lock the same lock twice would hang.
  - ***ThreadQueues:*** These are first-in-first-out queues of threads (which may be in the same or different processes).
  - ***Basic data types***: You may create variables of any normal basic type, such as int, float, double, and may also use a string type if you wish.
  - ***Thread***: a type to store a reference to a thread if you need it
- **Operations: (assume that these are atomic)**
  - ***Lock(lockname):*** This will lock the object or block until it can do so if it is already locked.
  - ***Unlock(lockname):*** This will unlock the specified object if it is locked, regardless of which thread locked it.
  - ***Push(threadqueuename):*** this will add the current thread to the back of the specified thread queue and block until the thread is removed from the queue.
  - ***Pop(threadqueuename):*** this will remove whatever thread which was at the front of the thread queue from the queue and wake up the thread, so that it will continue its processing immediately.
  - ***PushAndUnlock(threadqueuename, lockname):*** this will add the current thread to the back of the specified thread queue, then unlock the specified lock, then finally it will block until the thread is removed from the queue.
  - ***Count(threadqueuename):*** return a count of the number of threads currently on the specified thread queue.

Mutex

- Locking is no problem – we have something for that
- Ensuring that we don't deadlock if we already have it may be – using the existing lock will be a problem
- The fundamental component had no concept of ownership – we need to do that ourselves
- Principle: same as a monitor:
  - Use the lock component to enforce a critical section
  - Run the actual locking mechanism within the critical section
  - If it is already locked by someone else, leave critical section and wait on something until the other thread unlocks, then retry
- Lock: CriticalSectionLock – lock while in critical section
- Resource: OwnerThread – which thread currently owns it
- ThreadQueue: Tqueue - Wait on if lock currently locked

Pseudo-code

Lock: CriticalSectionLock
Resource: OwnerThread
ThreadQueue: TQueue

- Lock:
  1. Lock(CriticalSectionLock)
  2. If OwnerThread is None
  3. OwnerThread = current thread
  4. If OwnerThread is current thread
  5. Unlock(CriticalSectionLock) and then return (success)
  6. PushAndUnlock(TQueue,CriticalSectionLock)
  7. Go to step 1
- Unlock: (assuming we have it?)
  1. If OwnerThread not current thread then return (failed)
  2. Lock(CriticalSectionLock)
  3. OwnerThread = None
  4. Pop(TQueue) // In case something is waiting for this
  5. Unlock(CriticalSectionLock)

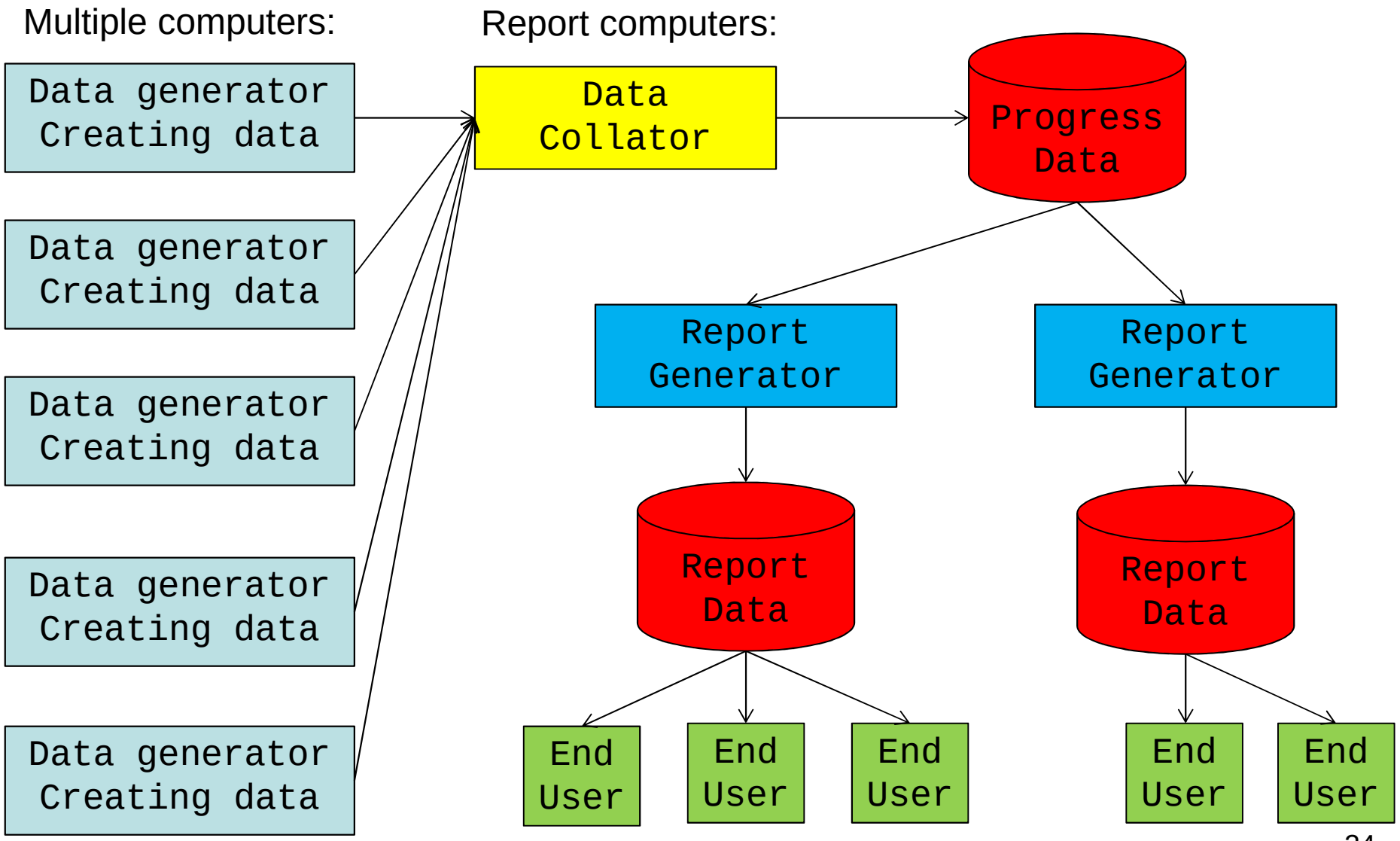
Why?

A bigger problem for you to consider

A problem to solve

- Assume a large company
- Multiple factories in many countries, producing items and reporting upon progress
- Central system receives the live reports and collates the information, adding it to a single reporting information database
- Multiple different report generator programs then analyse this data and produce live reports of the relevant information
  - E.g. summary tables, regularly updated
- Various end users can then use the reports to see current (almost live) progress

A problem to solve…

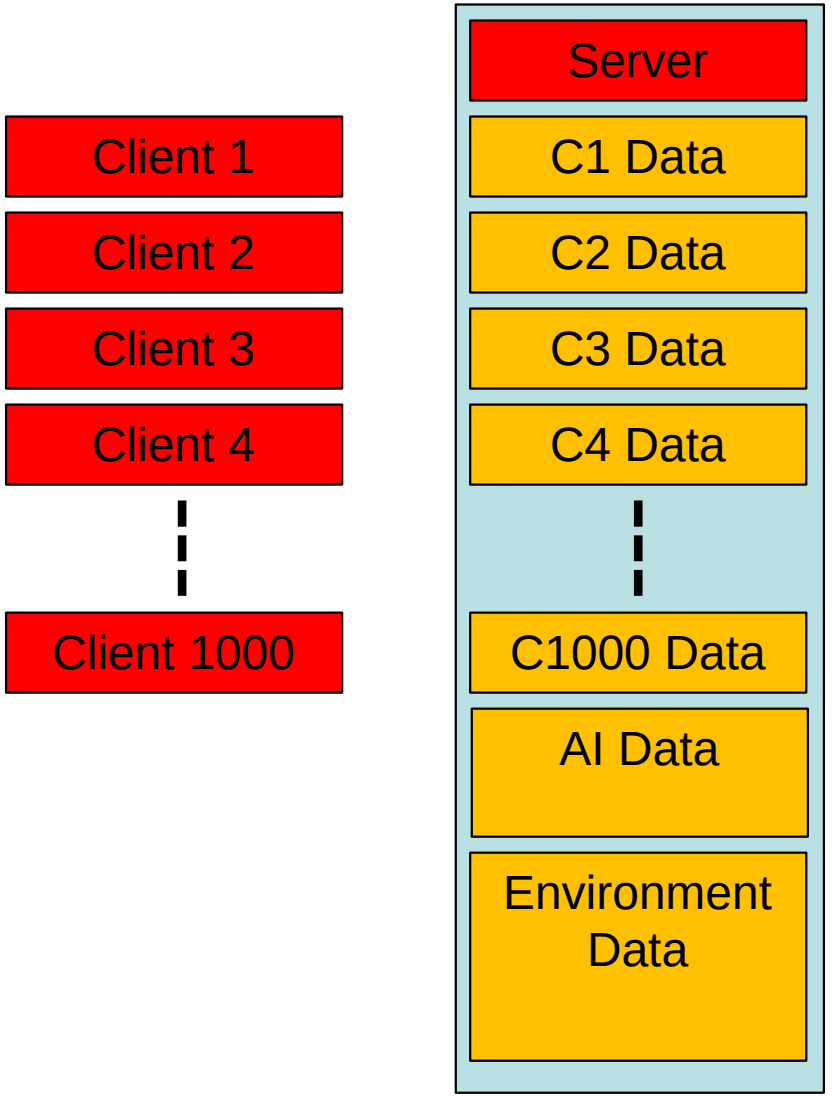


Sample problems (patterns?)

- Mutual exclusion and atomic operations
  - Ornamental garden
- Dining Philosophers
  - Avoiding deadlock by ordering locks
- Producer-Consumer
  - Shared memory buffer
  - Wait if you cannot act (e.g. no space or item)
- Readers-writers
  - Multiple simultaneous readers
  - One writer
  - Potential prioritisation issues

Multi-client client-server

- Consider a client-server game where each client sends information about what it is doing/wants to do and this information is passed on to all other clients
- How would you protect the data from interference?
  - Think of both safety and efficiency



More examples after Easter

- We can go through more example questions after Easter (two Thursday lectures)
- Based upon the previous exam questions for G52CON and others I come up with
- If you have specific questions (from previous G52CON papers) which you would like me to go through then please let me know in advance
- If you have not completed the Evaluate, please do so:
  - https://bluecastle.nottingham.ac.uk/Account/Login?ReturnUrl=%2f
- Have a good break, but finish the coursework!