Race Conditions

- What is a race condition?
- Race conditions in message passing and shared memory systems

BY ND

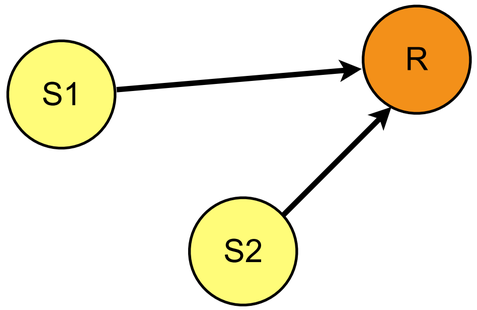
What is a Race Condition?

- A race condition can occur when the behaviour of a system depends on the relative timing of different actions – or when a shared value is modified without coordination
- Introduces non-deterministic behaviour and hard-to-debug problems
  - Difficult to predict exact timing of program behaviour
  - Difficult to predict effects of asynchronous, uncontrolled, modification to shared values

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Race Conditions: Message Races

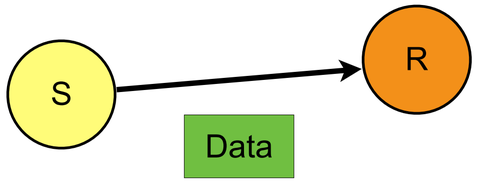
- In message passing systems, messages can be received from multiple senders
- Runtime ensures receiver processes messages sequentially, in the order they are received, but order of receipt can vary due to system and network load, external events, etc.
  - A **race condition** occurs when messages arrive in unpredictable order
  - A **deadlock** occurs when a cycle forms, with actors waiting for messages from each other
- Structure communication patterns to avoid



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Race Conditions: Data Races

- In shared memory systems, data is conceptually moved from sender to receiver
  - In practice, a **reference** to the data is often copied, for performance reasons, and the underlying data remains in place



- A **data race** condition occurs if the data is modified after it is sent, and the modification is visible to the receiver via the reference
  - Unpredictable if the receiver sees the old or new version, depending on timing of changes, scheduling, etc.
  - Two approaches to avoid data races: **immutable data** or **ownership tracking**

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Avoiding Data Races: Immutable Data

- Race conditions cannot occur if data is not modified
  - Ensure any objects to be sent between threads is immutable
  - Erlang ensures this in the language → *all* variables are immutable
  - Scala+Akka requires programmer discipline → potential race conditions if message data modified after message sent

CC BY ND

Avoiding Data Races: Ownership Transfer

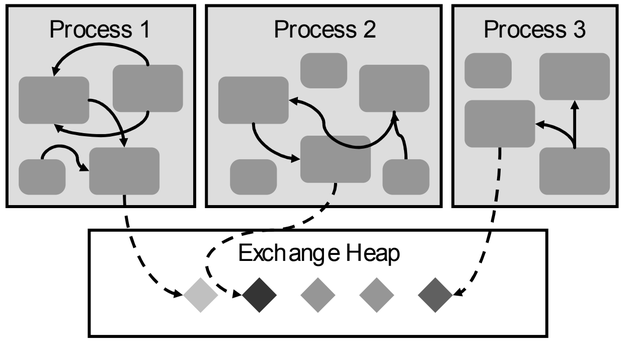
- Race conditions cannot occur if data is not shared
  - Ensure ownership of an object is transferred, so the sender cannot access the object after it has been sent
  - Natural fit for Rust:
    - Standard library provides a **channel** abstraction
    - The `send()` function takes ownership of the data to be sent
    - The `recv()` function returns ownership of the received data
    - The usual ownership rules in the type system ensure the data is not accessible once it has been sent
  - If sender can't access object once it's been sent, can't change it → race condition prevented

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Aside: Efficiency of Message Passing

- Assuming immutable message or linear types, message passing has efficient implementation
  - Copy message data in distributed systems
  - Pass pointer to data in shared memory systems
  - Neither case needs to consider shared access to message data

- Garbage collected systems often allocate messages from a shared *exchange heap*
  - Collected separately from per-process heaps
  - Expensive to collect, since data in exchange heap owned by multiple threads – need synchronisation
  - Per-process heaps can be collected independently and concurrently – ensures good performance



Source: G. Hunt *et al.*, Sealing OS processes to improve dependability and safety. In Proc. EuroSys 2007, Lisbon, Portugal. DOI 10.1145/1272996.1273032

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Summary

- Message passing as an alternative concurrency mechanism
- Increasingly popular
  - Erlang, Scala+Akka (or Java+Akka…)
  - Rust
  - Go, ZeroMQ, etc. – unchecked message passing

- Easy to reason about, simple programming model
  - Provided data is immutable, or ownership is tracked

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Summary

- Concurrency and memory models
- Transactions
- Message Passing

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow