COMP 8551 Advanced Games Programming Techniqu

Borna Noureddin, Ph.D.

British Columbia Institute of Technology

Realtime Issues and Multithreading II

Review

- Overview of multithreading
- Basic defi
- Multithreading chall
- Race conditions
- Mutexes

Overview

- Semaphores

- Critical secti

- Deadlocks

Semaphore

- Protected variable or abstract data type
- Synchronization method of controlling access by m common resource
- Binary semaphore (flag): or locked/unlocked variable
- Counting semaphore: multiple access to shared resource

Semaphore

- Restaurant analogy:
  - Tables = “resources”
  - People = “threa
  - Host = “sem
  - Host keeps track of unocc (utables) and who is to be seated next. Very focused and cannot be interrupted when performing duties.
  - Initially, utables = # of tables

Semaphore

- Restaurant analogy (cont'd):
  - When someone arrives, seated and utables updated as long as utables > 0
  - First come, ith reservations may be sea of others (priority)
  - If utables < 1, people wait in a queue for their table
  - When people leave, utables = utables + 1

Semaphore vs. Mutex

- Mutex = semaphore with only two values
- Mutex for single chair: only one person at a time can be
- Semaphore ple chairs, or multiple tables in a res each table/chair can only be occupied by one group/person, but there are multiple tables/chairs
- Mutex more efficient than binary semaphore

Critical sections

General use of term (Wikipedia):

In concurrent programming a **critical**
section is a ccesses a shared res re or device) that must not be concu ccessed by more than one thread of execution. A critical section will usually terminate in fixed time, and a thread, task or process will have to wait a fixed time to enter it (aka bounded waiting).

Critical sections

- Kernel-level (vs. application-level):
  - Processes/threads cannot migrate to other processors
  - No pre-em es or interrupts
- Windows object:
  - More lightweight than mutex/semaphore/event
  - Can only be used within single process
  - See http://msdn.microsoft.com/en-us/library/ms682530(VS.85).aspx

Deadlocks

- One or more threads wait for resources that can never become available

- Classic case: . equire two shared resources, and u ng mutexes to lock them in opposite order

Deadlocks

Thread A locks resource X

Thread B locks resource Y

Thread A attempts to lock resource Y

Thread B attempts to lock resource X

Both resources already locked (by the other thread): both threads wait indefinitely!

Deadlocks

- Necessary conditions:
  1. Mutual exclusion: a resource that cannot be shared by more than one process
  2. Hold an ses already request new resources
  3. No preemption condition: only a process holding a resource may release it
  4. Circular wait condition: two or more processes form a circular chain where each process waits for a resource that the next process in the chain holds

Deadlocks

Kansas legislature:

When two ach other

at a crossi e to a full

stop and neither shal again
until the other has gone.

Deadlock avoidance

- Check for availability before granting resource:
  - Will system enter an unsafe state?
  - System mu mber and type of all re
  - E.g., Banker's algorithm
  - Normally, impossible to know in advance what every process will request

Deadlock avoidance

- Symmetry-breaking techniques:
  - Wait/Die and Wound/Wait
  - Process a time stamp

	W	ound/Wait
O needs a resource held by Y	O waits	Y dies
Y needs a resource held by O	Y dies	Y waits

Deadlock prevention

- Remove mutual exclusion condition
  - Non-blocking synchronization algorithms
  - No exclus rce
  - Impossible
  - Not foolproof even w ling

Deadlock prevention

- Remove "hold and wait" conditions
  - Each process/thread must request all resources   t startup)
  - Very difficult a
  - Alternative: release   quest (all-or-none algorithms – not always practical)

Deadlock prevention

- Use timeouts
  - Only allowed to have resource for limited tim
  - Difficult t
- Avoid circular wait cond
  - E.g., disable interrupts during critical sections
  - E.g., use a hierarchy to determine a partial ordering of resources

Deadlock detection

- OS or resource scheduler can detect deadlocks
- Roll back or res threads/pr
- Not always possible, an uaranteed
- Generally, impossible to know if waiting for “unlikely” or “impossible” set of circumstances

Additional Reading

http://en.wikipedia.org/wiki/Semaphore_(programming)

http://en.wikipedia.org/wiki/Critical_section

http://msdn.micros 30(VS.85).aspx

http://www.drdobbs.com/high-perform /225400066

Review

- Semaphores

- Critical secti

- Deadlocks

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro