Exam Notes

Nicholas Land

Tuesday 23rd February, 2016

Processes

Process & Thread Synchronization

Background

- Parallelism can provide a distinct way of conceptualizing problems
- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure that the orderly exectution of cooperating processes
- Suppose that we wanted to provide a solution to the consumer-producer problem that fills **all** the buffers
  - We can do so by having an integer *count* that keeps track of the number of full buffers
  - Initially the **count** is set to 0
  - It is incremented by the producer after it produces a new buffer
  - It is decremented by the consumer after it consumers a buffer

Race Condition Race conditions can occur when two operations on shared variables are not **atomic**

Definitions

- **Synchronization** using atomic operations to ensure cooperation between threads
- **Critical Section** piece of code that only one thread can execute at once. Only one thread at a time will get into this section of code
  - **Mutual Exclusion** ensuring that only one thread does a particular thing at a time
  - **Progress** selecting a thread to enter cannot postpone indefinitely
  - **Bounded Waiting** before entering the critical section

Important idea: all synchronization involves waiting

Peterson's Solution

- A solution for two processes
- Assum that the LOAD and STORE instructions are atomic
  - **Atomic** == cannot be interrupted

- The two processes share two variables
  - int turn
  - Boolean flag[2]
- The variable turn indicates whos turn it is to enter the critical section
- the flag array is used to indicate if a process is ready to enter the critical section
  - flag[i] = true implies that P_1 is ready

Alogrithm for Process P_1

while *true* **do**
  flag[i] = TRUE;
  turn = j;
  while *flag[j] && turn == j* **do**
    ⋮
    CRITICAL SECTION;
    flag[i] = FALSE;
    REMAINDER SECTION;
  end
end

Algorithm 1: Peterson's Solution

Semaphore

- Synchronization tool that does not require busy waiting
  - integer variable
  - Two standard operations
    - S.wait() → P()
    - S.signal() → V()