Lecture 4 – Further Inter-Process Communication Techniques

Karl R. Wilcox
Karl@cs.rhul.ac.uk

Royal Holloway University of London

Administration

- **There WILL be a lecture this Friday**
  - **In MFLT, 11:00**
- **Lecture Notes**
  - **Are available at http://www.cs.rhul.ac.uk/~karl**
  - **Formats**
    - **Original Powerpoint 2000 slides**
    - **PDF files, A4 2 slides per page, colour**
    - **Let me know if you would like other formats**

Royal Holloway University of London

Objectives

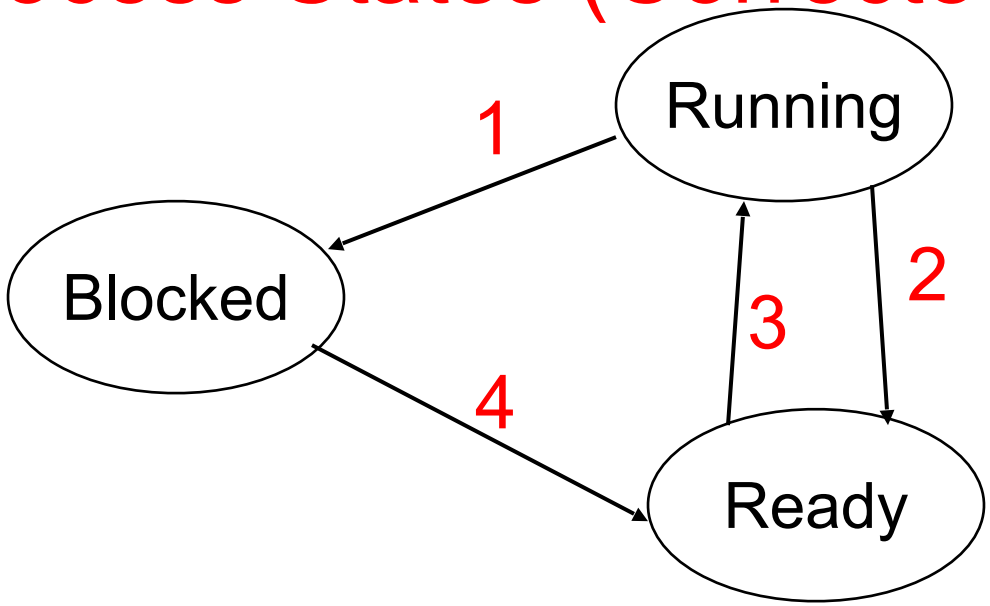
- In this class we will discuss:
  - The story so far…
  - Event Channels and Monitors
  - Other Classical IPC problems

Royal Holloway University of London

Review - 1

- **What are operating systems for?**
  - Top down view – virtual machine
  - Bottom up view – resource manager
- **Processes**
  - What is a process?
  - Process states and process pre-emption
- **Process scheduling**
  - Round Robin Scheduling
  - Priority Scheduling
  - Multiple Queues
  - Shortest Job First
  - Two-level Scheduling

Process States (Corrected)



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Royal Holloway University of London

Review - 2

- **Test and Set Lock instruction**
- **Peterson's algorithm**
  - **Problems with "busy waiting"**
- **The producer / consumer problem & its solutions**
  - **Sleep and Wakeup**
  - **Semaphores**

Event Counters

- Event counters (a new variable type) are a tool to solve the producer-consumer problem without mutual exclusion
- Using an event counter a process can wait until a certain event has happened
- An event counter E has three operations
  - Read (E)
    - Return the current value of E
  - Advance (E)
    - Atomically increment E by 1
  - Await (E, x)
    - Wait until E has a value of x or more
- An event counter can only be incremented not decremented

Royal Holloway University of London

P-C Solution Using Event Counters 1

```
#define N 100  // buffer size

typedef int event_counter;

event_counter in = 0;
event_counter out = 0;
```

Royal Holloway University of London

P-C Solution Using Event Counters 2

```
void producer () {
int item, sequence = 0;

while (TRUE) {
    produce_item ( &item );
    sequence = sequence + 1;
    await ( out, sequence - N );
    enter_item ( item );
    advance ( &in );
  }
}
```

Monitors

- A monitor is a programming language construct which obliges the programmer to declare shared data and resources explicitly and enforces mutual exclusion
- A monitor consists of
  - The data comprising a shared object
  - A set of procedures to access the object
  - Initialisation of the object (creation of object)
- The compiler (not the programmer) makes sure that mutual exclusion is guaranteed
- To ensure synchronisation the monitor provides objects called *conditions*
- Conditions have two operations
  - CWAIT (condition)
  - CSIGNAL (condition)

CWAIT and CSIGNAL

- CWAIT (condition): suspend execution of the calling process
- CSIGNAL (condition): resume execution of some process suspended after a CWAIT on the same condition
  - If there are several such processes, choose one
  - If there is no such process do nothing
- Signals to conditions are not saved (like in semaphores), they get lost
- A condition does not have a value, it can be thought of as a queue
- CWAIT adds a process to the queue
- CSIGNAL makes a process be removed from the queue if there is any

Royal Holloway University of London

The Dining Philosophers Problem

- Five philosophers are seated around a table
- Each philosopher has a plate of spaghetti
- A philosopher needs two forks to eat it
- Between each plate is a fork
- A philosophers life consists of alternate periods of eating and thinking
- When a philosopher gets hungry he tries to acquire his left and right fork, one at a time in either order
- if successful he eats than puts down the forks and goes on thinking
- Can you write a program for each philosopher such that each does what he is supposed to do and never gets stuck?

Royal Holloway University of London

Why Do We Care?

- **Because this is a simple example of DEADLOCK**
  - **If each philosopher takes the fork to his left then none of them can continue**
- **This is a good model of processes competing for *exclusive* access to limited resources**
- **Solutions to this simple problem may be solutions to real world operating systems deadlock problems**

Dining Philosophers Solution - 1

```
#define N 5
#define LEFT (i-1)%N
#define RIGHT (i+1)%N
#define THINK 0
#define HUNGRY 1
#define EAT 2
typedef int semaphore;
int state[N];
semaphore mutex = 1;
semaphore s[N];
```

```
void philosopher (int i)
{
   while (TRUE)
   {
       think();
       take_forks(i);
       eat();
       put_forks(i);
   }
}
```

Dining Philosophers Solution - 2

```
void take_forks (int i)
{
   down (&mutex);
   state[i] = HUNGRY;
   test(i);
   up(&mutex);
   down(&s[i]);
}
```

```
void test(int i)
{
   if (state[i]==HUNGRY
    && state[LEFT]!=EAT
    && state[RIGHT!=EAT)
       {
          state[i] = EAT;
          up(&s[i]);
       }
}
```

Dining Philosophers Solution - 3

```
void put_forks(int i)
{
   down(&mutex);
   state[i]=THINK;
   test(LEFT);
   test(RIGHT);
   up(&mutex);
}
```

- **Implementation**
  - **All variables are shared**
  - **mutex ensures mutual exclusion**
  - **5 processes**
    - **philosopher(1)**
    - **philosopher(2)**
    - **etc…**

Royal Holloway University of London

The Readers and Writers Problem

- **Another exclusive access problem**
  - **Models a database system**
  - **Any number of simultaneous readers**
  - **One writer with exclusive access**
- **Can be solved with a semaphore for database access and a semaphore protected counter for processes reading**
- **Real databases normally use much finer grained locking strategies**

Royal Holloway University of London

The Sleeping Barber Problem

- **Models a "service" process with a queue of "clients"**
  - **Clients are serviced in order**
  - **There is a fixed size queue for clients**
  - **If there is no room in the queue clients do not get service**
  - **If there are no clients, the service should sleep**

- **Again, can be solved with semaphores and counters**
  - **Note: our implementation of semaphores does not allow the value to be read, so we need a separate counter for clients waiting**

Royal Holloway University of London

Summary

- **Processes are fundamental to operating systems**
- **Processes are largely independent**
  - **Have their own "state"**
  - **From programmer's view have their own virtual machine**
- **But sometimes processes interact**
  - **"invisibly" as the operating system manages access to limited resources**
  - **"visibly" as processes need to communicate with each other**
    - **Producer – Consumer**
    - **Clients and Servers**
    - **Daemon Processes (see lectures on I/O)**

Royal Holloway University of London

Next Lecture

- **On Friday, MFLT**
- **We will talk about memory management and virtual memory**
- **Lecture Notes: http://www.cs.rhul.ac.uk/~karl**