Lecture 3 – Race Conditions and the Producer / Consumer Problem

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

Administration

- There is no lecture this Friday
- Next lecture, same time next week

Objectives

- In this class we will discuss:
  - Solutions for race conditions
    - TSL
    - Peterson’s Algorithm
  - The Producer / Consumer Problem
  - Sleep and Wakeup

Solutions For Race Conditions

- First solutions for race conditions
- TSL test and set lock
  - Hardware help required
- Petersons solution
- Though these solutions are better than e.g. strict alteration they both require busy waiting
- This can lead to the priority inversion problem

Test, Set and Lock

- This solution needs special hardware help
- Instruction TEST AND SET LOCK (TSL)
- Shared variable *flag*
- A TSL instruction reads into a register and sets a variable while no other process can access this variable during this command
- The operations read and store are indivisible
  - “atomic”

TSL in Assembler

```
enter region
   tsl register flag
   cmp register
   jnz enter region
   ret

leave region
   mov flag
   ret
```

Peterson’s Algorithm

- This is a software solution to ensure mutual exclusion for a critical region
- Shared variable *turn*
- Each process can be interested or not interested to enter the critical region
- To enter the critical region each process
  - announces its turn
  - announces its interest
  - waits until it is really allowed to enter

Implementation of Peterson

```
#define FALSE 0
#define TRUE 1
#define N 2 // number of proc
int turn;    // whose turn
int interested[N];
void enter_reg(int proc){
   int other;
   other = 1 - proc;
   interested[proc] = TRUE;
   turn = proc;
   while (turn == proc && interested[other] == TRUE)
              {}
}
void leave_reg(int proc){
   interested[proc] = FALSE;
}
```

The Producer / Consumer Problem

- The producer / consumer problem is also called the **bounded buffer** problem
- Two processes share a common, fixed size buffer
- One of them producer puts information into the buffer
- The other one consumer takes it out
- Possible problems
  - Producer wants to enter a new item but the buffer is full
  - Consumer wants to get an item from the buffer but the buffer is empty

Inter-Process Communications

- Two inter-process communication primitives that block instead of wasting CPU time -
  - Sleep and Wakeup
- SLEEP is a system call that causes the caller to block
  - The caller is suspended until another process wakes it up
- The WAKEUP call has one parameter
  - the process to be awakened

Producer implementation using Sleep & Wakeup

```
#define N 100    //  buffer size
int cnt = 0;     //  items in buffer

void producer() {
int item;
while (TRUE) {
  produce_item(&item);
  if (cnt == N) sleep();
  enter_item(&item);
  cnt++;
  if (cnt == 1)  wakeup(consumer);}
}
```

Consumer implementation using Sleep & Wakeup

```
void consumer {
int item;
while (TRUE) {
  if (cnt == 0)   sleep();
  remove_item(&item);
  cnt--;
  if (cnt = N - 1) wakeup(producer);
  consume_item(&item) }
}
```

Royal Holloway University of London

Semaphores

- A semaphore is a new variable type
- A semaphore counts the number of wakeups saved for future use
- It has two operations DOWN and UP which are atomic actions
  - Therefore need hardware or Operating System support
- DOWN
  - If > 0, decrement semaphore & continue
  - else sleep
- UP
  - Increment semaphore
  - Wakeup one process sleeping on semaphore

Producer / Consumer solution using Semaphores

```
define N 100  //  buffer size

typedef int semaphore;

semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

int cnt = 0; //  number of items in buffer
```

Producer Implementation using Semaphores

```
void producer () {
int item;
while (TRUE) {
 produce_item (&item);
 down (&empty);
 down (&mutex);
 enter_item (item);
 up (&mutex);
 up (&full); }
}
```

Consumer Implementation using Semaphores

```
void consumer () {
int item;
while (TRUE) {
  down (&full);
  down (&mutex);
  remove_item (&item);
  up (&mutex);
  up (&empty);
  consume_item(item) }
}
```