Simon Fraser University
School of Engineering Science
ENSC351: Real Time and Embedded Systems
Fall 2011

Examples of Classic IPC and Synchronization Problems solve with Semaphores

September 27th, 2011

1) Synchronization without ordering:

```
//Example 1 Initialization:
ResourceType *CS_resource; //Critical Section resource
semaphore mutex =1;
Create_thread(thread_0, 0);
Create_thread(thread_1, 0);
```

```
thread_0 ()
{
        while (TRUE)
        {
                <compute_section>;
                P(mutex)
                   access(CS_resource);
                V(mutex)
        }
}
```

```
thread_1 ()
{
        while(TRUE)
        {
                <compute_section>;
                P(mutex)
                   access(CS_resource);
                V(mutex)
        }
}
```

2) Synchronization with ordering (thread0 then thread1):

```
//Example 2 Initialization:
ResourceType *CS_resource; //Critical Section resource
semaphore mutex0 =1;
semaphore mutex1 =0;
Create_thread(thread_0, 0);
Create_thread(thread_1, 0);

thread_0 ()
{
        while (TRUE)
        {
                <compute_section>;
                P(mutex0)
                   access(CS_resource);
                V(mutex1)
        }
}
```

```
thread_1 ()
{
        while(TRUE)
        {
                <compute_section>;
                P(mutex1)
                   access(CS_resource);
                V(mutex0)
        }
}
```

3) Producer/Consumer problem:

```
//Example 3 Initialization:
int N;
semaphore mutex =1;
semaphore full = 0;
semaphore empty = N;
buf_type buffer[N];
Create_thread(producer, 0);
Create_thread(consumer, 0);
```

```
producer()
{
        buf_type *next, here;
        while (TRUE)
        {
                Produce_item(next);
                //Claim an empty buffer
                P(empty);
                  //Manipulate buffer pool
                  P(mutex);
                    here = obtain(empty);
                  V(mutex);
                  Copybuffer(next, here);
                  //Manipulate buffer pool
                  P(mutex);
                    release(here, fullPool);
                  V(mutex);
                //Signal a new full buffer
                V(full);
        }
}
```

```
consumer()
{
        buf_type *next, here;
        while(TRUE)
        {
                //Claim a full buffer
                P(full);
                  //Manipulate buffer pool
                  P(mutex);
                    here = obtain(full);
                  V(mutex);
                  copybuffer(here, next);
                  //Manipulate buffer pool
                  P(mutex);
                    release(here, emptyPool);
                  V(mutex);
                //Signal another empty buffer
                V(empty);
                consume_item(next);
        }
}
```

4) Readers/Writers problem - Solution 1:

```
//Example 4 Initialization:
resourceType *resource;
int readCount = 0;
semaphore mutex = 1;
semaphore writeBlock = 1;
Create_thread(reader, 0);   //Could be many
Create_thread(writer, 0);   //Could be many
```

```
writer()
{
        while(TRUE)
        {
                <other computing>;
                P(writeBlock);
                  //Critical Section
                  access(resource);
                V(writeBlock);
        }
}
```

```
reader()
{
        while(TRUE)
        {
                <other computing>;
                P(mutex);
                  readCount++;
                  if(readCount ==1)
                        P(writeBlock);
                V(mutex);
                //Critical Section
                access(resource);
                P(mutex);
                  readCount--;
                  if(readCount==0)
                        V(writeBlock);
                V(mutex);
        }
}
```

What's the problem with this solution?

5) Readers/Writers problem – Solution 2:

```
//Example 5 Initialization:
resourceType *resource;
int readCount = 0;
int writeCount = 0;
semaphore mutex1 = 1;
semaphore mutex2 = 1;
semaphore writeBlock = 1;
semaphore readBlock = 1;
semaphore writePending = 1;
Create_thread(reader, 0);   //Could be many
Create_thread(writer, 0);   //Could be many
```

```
reader()
{
        while(TRUE)
        {
                <other computing>;
                P(writePending);
                  P(readBlock);
                    P(mutex1);
                      readCount++;
                    if (readCount==1)
                       P(writeBlock);
                    V(mutex1);
                  V(readBlock);
                V(writePending)
                  access(resource);
                P(mutex1);
                  readCount--;
                  if(readCount==0)
                        V(writeBlock);
                V(mutex1);
        }
}
```

```
writer()
{
        while(TRUE)
        {
                <other computing>;
                P(mutex2);
                  writeCount++;
                  if(writeCount==1)
                        P(readBlock);
                V(mutex2);
                P(writeBlock);
                  access(resource);
                V(writeBlock);
                P(mutex2);
                  writeCount--;
                  if(writeCount==0)
                        V(readBlock);
                V(mutex2);
        }
}
```

What's the problem with this solution?

6) The Sleeping Barber problem:

```
//Example 6 Initialization:
int NumberofFreeSeats = N;
semaphore Customers = 0;
semaphore Barber = 0;
semaphore accessSeats = 1;
Create_thread(barber, 0);
Create_thread(customer, 0);   //Would be many

barber()
{
        while(TRUE)
        {
                //Get a customer/goto sleep
                P(Customers);
                P(accessSeats);
                  //Free up a waiting seat
                  NumberOfFreeSeats++;
                  //Ready to cut hair
                  V(Barber);
                V(accessSeats);
                <Barber cuts hair>;
        }
}

customer()
{
        while(TRUE)
        {
                P(accessSeats);
                  //Any free seats?
                  if(NumberofSeats > 0)
                  {
                        NumberofSeats--;
                        V(Customers);
                        V(accessSeats);
                        //Customer waits
                        P(Barber);
                  }
                  else //No free seats  =(
                  {
                        V(accessSeats);
                        //No hair cut =(
                  }
        }
}
```