CSCE 451/851
Operating Systems Principles

Process Synchronization

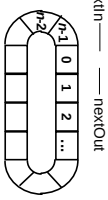
Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Producer/Consumer Implementation

```
process Producer
var c : char
begin
loop
  <produce a character "c">
  while nextIn+1 mod n = nextOut do
    NOOP
  end while
  buf[nextIn] := c
  nextIn := nextIn+1 mod n
end loop
end Producer
```

```
process Consumer
var c : char
begin
loop
  while nextIn = nextOut do
    NOOP
  end while
  c := buf[nextOut]
  nextOut := nextOut+1 mod n
  <consume a character "c">
end loop
end Consumer
```

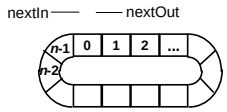



```
globals
  buf : array [0..n-1] of char;
  nextIn, nextOut : 0..n-1 := 0
```

Producer/Consumer Implementation with a shared counter

```
process Producer
  var c : char
begin
  loop
    <produce a character "c">
    while count = n do
      NOOP
    end while
    buf[nextIn] := c
    nextIn := nextIn+1 mod n
    count  := count + 1
  end loop
end Producer
```

```
process Consumer
  var c : char
begin
  loop
    while count = 0 do
      NOOP
    end while
    c := buf[nextOut]
    nextOut := nextOut+1 mod n
    count   := count - 1
    <consume a character "c">
  end loop
end Consumer
```




```
globals
   buf   : array [0..n-1] of char;
   nextIn,nextOut : 0..n-1 := 0
   count : integer := 0
```

3

The Critical Section Problem

- One implementation of the shared counter

```
process Producer
begin
   :
 <count := count + 1>

  MOV R1, @count
  ADD R1, 1
  MOV @count, R1
   :
end Producer
```

```
process Consumer
begin
   :
 <count := count - 1>

  MOV R2, @count
  SUB R2, 1
  MOV @count, R2
   :
end Consumer
```

4

Algorithms for Mutual Exclusion

- General algorithm structure

```
process P1
begin
  loop
     :
    Entry_Protocol
    <critical section>
    Exit_Protocol
     :
  end loop
end P1
```

- Correctness conditions
  - » Does it guarantee mutual exclusion?
  - » Is it expedient?
  - » Does it provide bounded waiting?

5

2-Process Mutual Exclusion
Algorithm 1

- Turn taking/strict alternation

```
global var turn : int := 2

process P1                    process P2
begin                         begin
  loop                          loop
    while turn = 2 do             while turn = 1 do
      NOOP                          NOOP
    end while                     end while
    <critical section>            <critical section>
    turn := 2                     turn := 1
  end loop                      end loop
end P1                        end P2
```

6

2-Process Mutual Exclusion
Algorithm 2

- Use status flags

```
global var inCS : array[1..2] of boolean := (FALSE,FALSE)

  process P1                    process P2
  begin                         begin
    loop                          loop
      while inCS[2] do              while inCS[1] do
        NOOP                          NOOP
      end while                     end while
      inCS[1] := TRUE               inCS[2] := TRUE
      <critical section>            <critical section>
      inCS[1] := FALSE              inCS[2] := FALSE
    end loop                      end loop
  end P1                        end P2
```

7

2-Process Mutual Exclusion
Algorithm 2a

- Move the setting of the status flags

```
global var inCS : array[1..2] of boolean := (FALSE,FALSE)

  process P1                    process P2
  begin                         begin
    loop                          loop
      inCS[1] := TRUE               inCS[2] := TRUE
      while inCS[2] do              while inCS[1] do
        NOOP                          NOOP
      end while                     end while
      <critical section>            <critical section>
      inCS[1] := FALSE              inCS[2] := FALSE
    end loop                      end loop
  end P1                        end P2
```

8

2-Process Mutual Exclusion
Algorithm 3

- New and improved use of status flags

```
global var inCS : array[1..2] of boolean := (FALSE,FALSE)

process P1                    process P2
begin                         begin
  loop                          loop
    inCS[1] := TRUE               inCS[2] := TRUE
    if NOT inCS[2] then           if NOT inCS[1] then
      <critical section>            <critical section>
    end if                        end if
    inCS[1] := FALSE              inCS[2] := FALSE
  end loop                      end loop
end P1                        end P2
```

9

2-Process Mutual Exclusion
Algorithm 4 (Peterson's algorithm)

- Careful combination of alternation and status flags

```
global var inCS : array[1..2] of boolean := (FALSE,FALSE)
           turn : integer := 2

process P1                          process P2
begin                               begin
  loop                                loop
    inCS[1] := TRUE                     inCS[2] := TRUE
    turn    := 1                        turn    := 2
    while turn = 1 AND inCS[2] do       while turn = 2 AND inCS[1] do
      NOOP                                NOOP
    end while                           end while
    <critical section>                  <critical section>
    inCS[1] := FALSE                    inCS[2] := FALSE
  end loop                            end loop
end P1                              end P2
```

10