Processes

Steve Goddard
goddard@cse.unl.edu

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

Processes

◆ The basic agent of work, the basic building block

◆ Process characterization
  - Program code
  - Processor/Memory state
  - Execution state

◆ The state transition diagram

Ready

Waiting

Running
Process Actions

- Create and Delete
- Suspend and Resume
- Process synchronization
- Process communication

Physical v. Logical Concurrency

Why is logical concurrency useful?

- Structuring of computation
- Performance

```plaintext
process P
begin
Read(var)
end P
```

```plaintext
system call Read()
begin
StartIO(input device)
WaitIO(interrupt)
EndIO(input device)
Read
end
```

» Single process I/O
Physical v. Logical Concurrency
Performance considerations

- Multithreaded I/O

```plaintext
begin
StartRead()
compute
Read(var)
end P
```

```plaintext
begin
WaitForRequest()
System_Read(var)
WaitForRequestor()
end loop
end Read
```

```plaintext
begin
RequestIO(input device)
end StartRead
```

```plaintext
begin
SignalReader(input device)
end Read
```

Process Creation Paradigms

- COBEGIN/COEND
  ```plaintext
cobegin
S. ||
S. coend
```

- FORK/JOIN
  ```plaintext
begin
procedure foo()
begin
fork(foo)
join(foo)
end foo
end
```

- Explicit process creation
  ```plaintext
begin
process P
begin
P
end
end P
```
Process Scheduling
Implementing and managing state transitions

Why Schedule?
Scheduling goals

- Example: two processes execute concurrently
  ```
  process P1
  begin
  for i := 1 to 5 do
    <read a char>
    <process a char>
  end
  end
  
  process P2
  begin
  <execute for 1 sec>
  end
  ```

- Performance without scheduling

- Performance with scheduling

**Types of Schedulers**

- Long term schedulers
  - adjust the level of multiprogramming through admission control
- Medium term schedulers
  - adjust the level of multiprogramming by suspending processes
- Short term schedulers
  - determine which process should execute next

---

**Short Term Scheduling**

**When to schedule**

- When a process makes a transition...
  1. from *running* to *waiting*
  2. from *running* to *ready*
  3. from *waiting* to *ready*
     (3a. a process is created)
  4. from *running* to *terminated*
Short Term Scheduling
How to schedule — Implementing a context switch

context_switch(queue : system_queue)
var next : process_id
begin
DISABLE_INTS
insert_queue(queue, runningProcess)
next := remove_queue(readyQueue
dispatch(next)
ENABLE_INTS
end context_switch

implementing a context switch

Running
Ready
Waiting

device/condition queues

Implementing a Context Switch

Dispatching

Case 1: Preemption
Case 2: Yield

main()
switch()
deposit()
wait()
switch()
dispatch()

main()
foo()
timerInt()
switch()
dispatch()

main()
read()
startIO()
switch()
dispatch()
waitIO()

main()
bar()
timerInt()
switch()
dispatch()

“running”
“next”
“running”
“next”

“running”’s dispatch:
dispatch()
begin
save state of running
load state of next
end dispatch

“next”’s dispatch:
dispatch()
begin
load state of running
save state of next
end dispatch
Producer/Consumer Implementation

```plaintext
Producer

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

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

```plaintext
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

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
Process Coordination
Producer/Consumer systems

Example: Synchronous I/O

system module I/O

procedure WaitForRequest()
begin
Wait();
end

procedure WriteData()
begin
Signal();
Wait();
<copy results>
end

procedure Read()
begin
Signal();
Wait();
<copy results>
end
I/O