10 Lecture: Scheduling

Outline:
Announcements
We talked a lot about LWP
From Last Time: Further generalization: message passing
Robust Programming Clinic
Classic IPC Problems
So what: Scheduling
Process States
Policy vs. Mechanism
Process types
When to schedule
Evaluation Criteria
Non-preemptive scheduling: run-to-completion
Preemptive scheduling

10.1 Announcements

• Coming attractions:

<table>
<thead>
<tr>
<th>Event</th>
<th>Subject</th>
<th>Due Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>asgn5</td>
<td>minget and minls</td>
<td>Wed Jun 5</td>
<td>23:59</td>
</tr>
<tr>
<td>asgn6</td>
<td>Yes, really</td>
<td>Fri Jun 7</td>
<td>23:59</td>
</tr>
<tr>
<td>final</td>
<td>stuff</td>
<td>Sat Jun 8</td>
<td>10:10 (in 03-201)</td>
</tr>
</tbody>
</table>

Use your own discretion with respect to timing/due dates.

• tryAsgn2

• Reminder about SPC Meeting

• Old exams on the web site (warning...)

10.2 We talked a lot about LWP

List of stuff to talk about wrt asgn02:

• Draw stack

• pointer arithmetic in size of pointee

• Allocate enough stack

• Do atomic swaps (w/\texttt{swap\_rfiles()})

• Be paranoid
10.2.1 From Last Time: Further generalization: message passing

*If you need me, just whistle.*

Monitors require language support, a tricky thing in the “real world”. Message passing gets much of the same safety using operating system support.

Processes communicate via two primitive functions:

- `send(dst, message)`
- `receive(src, message)`

Where do they get sent? (How do we address them?) Options:

- individual processes
- mailboxes (possibly multiple consumers)

In either case, there may or may not be buffers. If there is a buffer, the OS has to manage this buffer (a meta-producer-consumer problem :). If not, a sender is stopped until there is a receiver: rendezvous.

Issues with message passing:

- messages can be lost (acknowledgement)
- messages can be duplicated (sequence numbers)
- messages can be forged (authentication)
- performance:
  - Copying overhead (copy, copy, copy)
  - Transmission latency (You may as well do it yourself)

But it can elegantly solve some problems, as with the producer-consumer solution given in Figure 18. Message passing producer-consumer: send empty messages and send back full ones.

10.3 Classic IPC Problems

- Producer/Consumer (been there, done that?)
  
  Competing access for limited resources.
  
  Issues:
  
  - deadlock
  - starvation

- Readers/Writers: e.g. a database.
  
  Issues:
  
  - Many readers can work simultaneously, but only one writer.
  - Starvation: If we let all the readers by (no problem), the writer may starve.
  - perhaps an aging scheme?
Figure 18: A message-passing solution to the producer-consumer problem.

10.4 So what: Scheduling

We talked a lot about running along until we block, then pick who gets to run, but how is this done?

10.4.1 Process States

Remember at any given time, any process can be in one of three possible states: Running, Ready, or Blocked. Possible transitions between these states are shown in Figure 19.

10.4.2 Policy vs. Mechanism

Policy How we want things to behave. (graduate students should finish within 7 years.)
mechanism How we’re going to make it happen. (cut off funding. (vs. throwing them out.))

We’re pretty clear on how it happens (particularly after asgn2), but what is it we want to do anyway?

10.4.3 Process types
Processes are usually roughly categorized into one of two different types

IO Bound characterized by short bursts of computation before blocking on IO (or a semaphore)
  • Might want to give priority because they can get done and go back to sleep. (hide IO latency)
  • Also, more likely to be interactive.

Compute Bound characterized by long bursts of computation before blocking on IO (or a semaphore)

These are dynamic. A process may move back and forth.

10.4.4 When to schedule
Scheduling is mandatory in two cases:
  1. When a process exits
  2. When a process blocks

It might be desirable under a few other conditions:
  1. When a new process is created
     (Consider the situation of parent and child after fork(jing))
  2. When an IO interrupt occurs
  3. When a timer interrupt occurs

10.4.5 Evaluation Criteria
What makes a good algorithm?

Fairness Make sure each process gets its fair share

Efficiency/Utilization keep the CPU busy 100 percent of the time

Response Time minimize response time for interactive users

Turnaround minimize turnaround time for batch users

Throughput maximize the number of jobs processed per time.

Faster, better, cheaper, choose two.
10.4.6 Non-preemptive scheduling: run-to-completion

- Run to completion/blockage

Examples:

1. FCFS

2. Shortest Job First:
   - Provably optimal
   - Problem: Starvation

Add an aging function?

10.4.7 Preemptive scheduling

**Round Robin**  Every (runnable) process goes in turn.