10 Lecture: Scheduling

Outline:

Announcements
We talked a lot about LWP
So what: Scheduling
  Process States
  Policy vs. Mechanism
  Process types
  When to schedule
  Evaluation Criteria
  Non-preemptive sheduling: run-to-completion
  Preemptive sheduling

10.1 Announcements

- Coming attractions:

Event	**Subject**	**Due Date**			**Notes**
asgn3	dine	Wed	Feb 4	23:59	
lab03	problem set	Mon	Feb 9	23:59	
midterm	stuff	Wed	Feb 11		
lab04	scavenger hunt II	Wed	Feb 18	23:59	
asgn4	/dev/secret	Wed	Feb 25	23:59	
lab05	problem set	Mon	Mar 9	23:59	
asgn5	minget and minls	Wed	Mar 11	23:59	
asgn6	Yes, really	Fri	Mar 13	23:59	
final (sec01)		Fri	Mar 20	10:10	
final (sec03)		Fri	Mar 20	13:10	

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

- Don't just click the little 'X' to stop minix
- tryAsgn2
- Call stacks, Algol 60
- Wind up a stack.
- Do atomic swaps

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/`swap_rfiles()`)

- Be paranoid
- **Alignment** — it's a thing

10.3 So what: Scheduling

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

10.3.1 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 17.

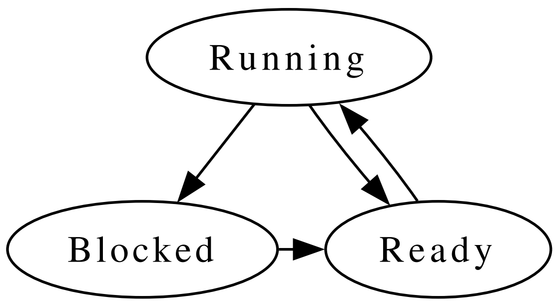


Figure 17: Possible states for a process

10.3.2 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 (particlarly after asgn2), but what is it we want to do anyway?

10.3.3 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.

10.3.4 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()ing)
2. When an IO interrupt occurs
3. When a timer interrupt occurs

10.3.5 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.

10.3.6 Non-preemptive sheduling: run-to-completion

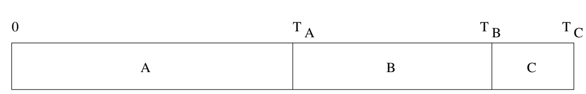
- Run to completion/blockage

Examples:

1. FCFS
2. Shortest Job First:
   - Provably optimal
   - Problem: Starvation and how do you know?

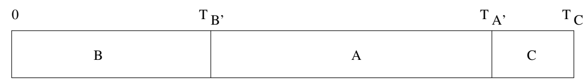
   Add an aging function?

   The proof:



$$T_{\text{turnaround}} = \frac{T_A + T_B + T_c}{3}$$

Now place the shorter B ahead of A:



$$
\begin{aligned}
T'_{\text{turnaround}} &= \frac{T'_B + T'_A + T_c}{3} \\
&= \frac{T'_B + T_B + T_c}{3}
\end{aligned}
$$

But we know that $T'_B < T_A$, so

$$T'_{\text{turnaround}} < T_{\text{turnaround}}$$

We have invented bubble sort.

10.3.7 Preemptive sheduling

Round Robin Every (runnable) process goes in turn.