Reminders

- Homework 4 due next Friday
  - Virtual Memory
- Rest of project 2 due next Friday
  - Code due Thursday midnight
  - Writeup in Friday’s lecture
- I have some old homework/projects
  - Pick them up at the end

Today:
- Project 2 parts 4, 5
- Scheduling/deadlock stuff

Project 2 – web server

- web/sioux.c – singlethreaded web server
  - Read in command line args, run the web server loop
- web/sioux_run.c – the webserver loop
  - Open a socket to listen for connections (listen)
  - Wait for a connection (accept)
  - Handle it
    - Parse the HTTP request
    - Find and read the requested file (www root is ./docs)
    - Send the file back
    - Close the connection
- web/web_queue.c – an empty file for your use

What you need to do

- Make the web server multithreaded
  - Create a thread pool
    - A bunch of threads waiting for work
    - Number of threads = command-line arg
  - Wait for a connection
  - Find an available thread to handle connection
    - Current request waits if all threads busy
  - Once a thread grabs onto connection, it uses the same processing code as before.

Hints

- Each connection is identified by a socket returned by accept
  - Which is just an int
  - Simple connection management
- Threads should sleep while waiting for a new connection
  - Condition variables are perfect for this
- Don’t forget to protect any global variables
  - Use part 2 mutexes, CVs
- Mostly modify sioux_run.c and/or your own files
  - Stick to the sthread.h interface!

Part 5 – Analysis

- You need to experiment with threads
  - Two options:
    - Experiment with part 4 webserver (probably w/threads)
    - Experiment with something else that’s multithreaded
  - Play around with parameters, come to some conclusions, write a report
    - Examples for webserver:
      - number of threads in thread pool
      - number of clients
      - File size
      - How you distribute the work to the thread pool
    - Examples for matrix-multiply:
      - Compare performance of user threads vs kernel threads

Part 5 tools

- More accurate timer
  - /cse451/projects/timer.tar.gz
  - reads Pentium’s cycle counter
  - To find time, divide by processor speed
    - examine /proc/cpuinfo
- Web benchmark
  - /cse451/projects/webclient
  - Takes in # of clients, # of requests/client, URLs to request
  - Use time command for command line timing
Project questions?

A simple scheduling problem

<table>
<thead>
<tr>
<th>Thread</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- FIFO Turnaround time: 10
- FIFO Waiting Time: 10

Priority Inversion

A simple scheduling problem

<table>
<thead>
<tr>
<th>Thread</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- FIFO Turnaround Time: 10
- FIFO Waiting Time: 10

- B: (15-1) = 14
- C: (17-3) = 14
- (10+14+14)/3 = 12.66
- (10+9+12)/3 = 10.33

What about SJF with 1 unit delay? (just like HW)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- Ave Turnaround Time: 10
- Ave Waiting Time: 10

- B: 5
- C: 7-3 = 4
- A: 1+5+2+10 = 18
- 17+4+5)/3 = 8.67

- 10 (A)
- 5 (B)
- 2 (C)

Priority Inversion

A simple scheduling problem

- Have three processes
  - P1: Highest priority; P2: Medium; P3: Lowest
- Have this code:
  - P(mutex);
  - critical section;
  - V(mutex);
- P3 acquires mutex; preempted
- P1 tries to acquire mutex; blocks
- P2 enters the system at medium priority; runs
- P3 never gets to run; P1 never gets to run!!

- This happened on Mars Pathfinder in 1997?
- Solutions?
Deadlock review

- Deadlock solutions
  - Prevention
  - Kill one of necessary conditions
  - Avoidance
    - Banker’s algorithm (Dijkstra)
    - Detection & Recovery
    - The Ostrich Algorithm
      - “Put your head in the sand”
      - If each PC deadlocks once per 100 years, the one reboot may be less painful that the restrictions needed to prevent it.
      - Not a good strategy for a nuclear reactor!
  - Livelock
    - Processes run but make no progress

Deadlock

- Given two threads, what sequence of calls causes the following to deadlock?
  ```
  /* transfer x dollars from a to b */
  void transfer(account *a, account *b, int x)
  { P(a->sema);
    P(b->sema);
    a->balance += x;
    b->balance -= x;
    V(b->sema);
    V(a->sema);
  }
  ```

Deadlock Questions

- Can there be a deadlock with only one process?
  - In a system w/Banker’s algorithm, which of the following can always be done safely?
    - Add new resources
    - Remove resources
    - Increase Max resources for one process
    - Decrease Max resources for one process
    - Increase the number of processes
    - Decrease the number of processes

Banker’s Algorithm practice

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P0</td>
<td>0 0 1 2</td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>P1</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>P2</td>
<td>1 3 5 4</td>
<td>2 3 5 6</td>
</tr>
<tr>
<td>P3</td>
<td>0 6 3 2</td>
<td>0 6 5 2</td>
</tr>
<tr>
<td>P4</td>
<td>0 0 1 4</td>
<td>0 0 2 5</td>
</tr>
</tbody>
</table>

- Is the system in a safe state?
- If a request from P1 arrives for (0,4,2,0), can the request be satisfied immediately?

Banker’s Algorithm practice

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P0</td>
<td>1 0 0 1 2 0 3 1 2 0 5 2 0</td>
<td>1 0 0 1 2 0 3 1 2 0 5 2 0</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>1 0 0 0 1 7 5 0</td>
<td>1 0 0 0 1 7 5 0</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>1 3 5 4 2 3 5 6</td>
<td>1 3 5 4 2 3 5 6</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>0 6 3 2 0 6 5 2</td>
<td>0 6 3 2 0 6 5 2</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>0 0 1 4 0 6 5 6</td>
<td>0 0 1 4 0 6 5 6</td>
<td></td>
</tr>
</tbody>
</table>

- 1st step: figure out Need[] vector
- Is the system in a safe state?
- If a request from P1 arrives for (0,4,2,0), can the request be satisfied immediately?
Banker’s Algorithm practice

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1 1 2</td>
<td>A 1 1 2</td>
<td>A 1 1 2</td>
<td>A 1 1 2</td>
</tr>
<tr>
<td>B 1 1 0</td>
<td>B 1 1 0</td>
<td>B 1 1 0</td>
<td>B 1 1 0</td>
</tr>
<tr>
<td>C 0 0 0</td>
<td>C 0 0 0</td>
<td>C 0 0 0</td>
<td>C 0 0 0</td>
</tr>
<tr>
<td>D 0 0 0</td>
<td>D 0 0 0</td>
<td>D 0 0 0</td>
<td>D 0 0 0</td>
</tr>
</tbody>
</table>

1. Is the system in a safe state?
   - Yes: <P0, P3, P2, P4, P1>

2. If a request from P1 arrives for (0,4,2,0), can the request be satisfied immediately?
   - Yes: e.g. <P0, P2, P1, P3, P4>