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

1

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

2

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.

3

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!

4

Part 5 – Analysis

- You need to experiment with threads
- Two options:
  - Experiment with part 4 webserver (probably w/pthreads)
  - 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

5

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

6

Project questions?

7

Scheduling review

- FIFO:
  - \+ simple
  - \- short jobs can get stuck behind long ones; poor I/O device utilization
- RR:
  - \+ better for short jobs
  - \- hard to select right time slice
  - \- poor turnaround time when jobs are the same length
- SJF:
  - \+ optimal (ave. waiting time, ave. time-to-completion)
  - \- hard to predict the future
  - \- unfair
- Multi-level feedback:
  - \+ approximate SJF
  - \- unfair to long running jobs

8

A simple scheduling problem

Thread	Arrival Time	Burst Time
A	0	10
B	1	5
C	3	2

- FIFO Turnaround time:
- FIFO Waiting Time:

9

A simple scheduling problem

Thread	Arrival Time	Burst Time
A	0	10
B	1	5
C	3	2

- FIFO Turnaround Time:
  - A: (10-0) = 10
  - B: (15-1) = 14
  - C: (17-3) = 14
  - (10+14+14)/3 = 12.66
- FIFO Waiting Time:
  - A: 0
  - B: (10-1) = 9
  - C: (15-3) = 12
  - (10+9+12)/3 = 10.33

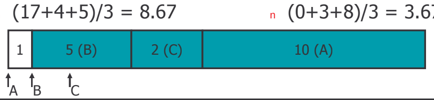
10

A simple scheduling problem

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

Thread	Arrival Time	Burst Time
A	0	10
B	1	5
C	3	2

- Ave Turnaround Time:
  - B: 5
  - C: 7-3 = 4
  - A: 1+5+2+10 = 18
  - (17+4+5)/3 = 8.67
- Ave Waiting Time:
  - B: 0
  - C: 5-2 = 3
  - A: 1+5+2 = 8
  - (0+3+8)/3 = 3.67



11

Priority Inversion

- 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?

12

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

Deadlock Questions

- Can there be a deadlock with only one process?
  - Yes, P(sema); P(sema);
- In a system w/Banker's algorithm, which of the following can be done safely?
  - **Add new resources**
  - Remove resources
  - Increase Max for one process
  - **Decrease Max for one process**
  - Increase the number of processes
  - **Decrease the number of processes**

Banker's Algorithm practice

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

- 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

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P1	1	0	0	0	1	7	5	0					0	7	5	0
P2	1	3	5	4	2	3	5	6					1	0	0	2
P3	0	6	3	2	0	6	5	2					0	0	2	0
P4	0	0	1	4	0	6	5	6					0	6	4	2

- 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

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P1	1	0	0	0	1	7	5	0					0	7	5	0
P2	1	3	5	4	2	3	5	6					1	0	0	2
P3	0	6	3	2	0	6	5	2					0	0	2	0
P4	0	0	1	4	0	6	5	6					0	6	4	2

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

19