Project 2 Overview

- Semaphores
- Semaphore Service in Minix
- Pizza Synchronization Problem

Semaphore

- What is a semaphore?
  - "A semaphore is a data structure that is useful for solving a variety of synchronization problems" Downey, *The Little Book of Semaphores*
- Types of synchronization problems
  - Serialization: A must occur before B
  - Mutual Exclusion: A and B cannot happen concurrently

More on Semaphores

- Like an integer but…
  - Can be initialized to any value and then restricted to two main operations operations
  - Incremented -- V(), up()
  - Decremented – P(), down()
- Why P and V?
  - Initials of Dutch words verhogen (increase) and the portmanteau prolaag [probeer te verlagen] (try to reduce)

Adding Semaphores to Minix

- Implemented as a service
  - Needs to call sef_startup on initialization
  - Calls sef_receive_status to retrieve messages
  - The service requires the appropriate permissions (/etc/system.conf) to send/receive messages from other processes
- User-level interface needs to construct messages to pass to the service
  - Use minix_rs_lookup to find the dynamic service endpoint

Semaphore Interface

- int sem_init(int value)
  - Initializes new semaphore to value and returns the lowest available semaphore number >= 0
- int sem_up(int sem_num)
  - If no one is waiting, increases the semaphore value
  - otherwise “wakes up” a the oldest waiting process

Semaphore Interface cont.

- int sem_down(int sem_num)
  - Decreases the semaphore value
  - If the semaphore value <= 0 “sleep” the requesting process
- int sem_release(int sem_num)
  - If no one is waiting, free the semaphore so that it can be re-used
  - Otherwise return EINUSE error (need to define in errno.h)

Error Handling

- If an invalid value is passed anywhere EINVAL should be returned to the user
- If any functions return an error, the errno corresponding to that error should be returned
  - Such as: malloc, minix_rs_lookup

Semaphore (Pizza) Challenge

- 6 grads, 2 ugrads, 2 tables with pizza
- Only 1 student can eat at a table at a time
- Student can only enter room if table is available
- Grads have priority
  - 1 ugrad is eating and 1 grad comes in, the ugrad must least
  - 1 grad is eating, no ugrad can enter

Semaphore Challenge cont.

- Write two programs
  - grad.c – manage the 6 graduate students
  - ugrad.c – manage the 2 undergraduate students
- Use semaphores to correctly manage the consumption of pizza
- Solution cannot cause starvation
- Explain your solution in pizza.txt

Semaphore Challenge Questions

- How can you dynamically share semaphore numbers between processes?
- How do you determine how often the students want to eat, and for how long they eat?

Suggested Implementation Order

- Create skeleton server sema
- Load and unload sema server via
  - service up /usr/sbin/sema
  - service down sema
- Complete sema service
- Complete semaphore “pizza” challenge
- Ensure patch builds everything by running “make world” and your service starts up after a reboot

Implementation Notes

- Copy sched service in servers/ to sema
- Update etc/systems.conf to add sched service (see man systems.conf if needed)
- Add constants to include/minix/com.h
- Add appropriate rc file to start semaphore server
- Under /usr/src
  - make includes (updates include files)
  - make libraries (builds and updates libraries)
  - make etcforce (updates etc files)
  - make –C servers install (builds all servers)

Resources

- The Little Book of Semaphores
  - http://greenteapress.com/semaphores/downey08semaphores.pdf
- Driver programming in Minix
  - http://wiki.minix3.org/en/DevelopersGuide/DriverProgramming
  - Similar setup process