



CS 3530 OS

System Simulation with Threads

CS 3530 Dr. Ken Hoganson, Copyright © 2009

A simple model of a computer system.

Contains two threads:

- Processor Thread
- Memory Thread
- The threads communicate through shared memory, which represents the bus:
 - Address bus
 - Data bus
- The processor has a cache that can hold 5 pages (each “page” will be contain a single data item – a simple character
- The memory stores 10 characters. The initial contents of memory will be the letters A – J.

- The processor thread will generate a request for data (an “address” – a random number between 1 and 10)
- The processor will check the cache for the requested data (will be a simple linear search through the five storage locations in the cache to see if the needed page is in the cache)
- If the cache has the data, then the processor will record a tally of cache hits

- If the cache does not have the data, then the processor will record a tally of cache misses, and will then place the address on the bus (in the shared memory location)
- The processor will print the data (letter A-J) on the screen
- The processor will iterate a fixed number of times (10 to begin with), then terminate and terminate the program

- The memory thread will continually check the address bus (shared memory) for an address.
- If an address is found, then the memory will use that address to find the data in the memory (characters A-J), and will place the data on the data bus
- Sound like a critical section? Review Critical Sections as needed. Review synchronized shared memory java example.

- The bus is simply shared memory
- Check java examples for how to do
- Contains two data values –
 - an integer (address)
 - A character (A-J) (data)
- Access to the bus must be synchronized to prevent the two threads from racing.

- The cache is private memory to the processor thread.
- It is fully associative (data can go anywhere)
- 5 storage locations, each with an address
- Address is the memory page that the cache duplicates

- What about cache replacement algorithm? – just use a random number from 1-5 to chose a cache page to replace with a new page on page faults.

- Data Assigned: Monday March 30, 2009
- Monday April 6, at Midnight
- Part of today, and April 6 used for lab work time.

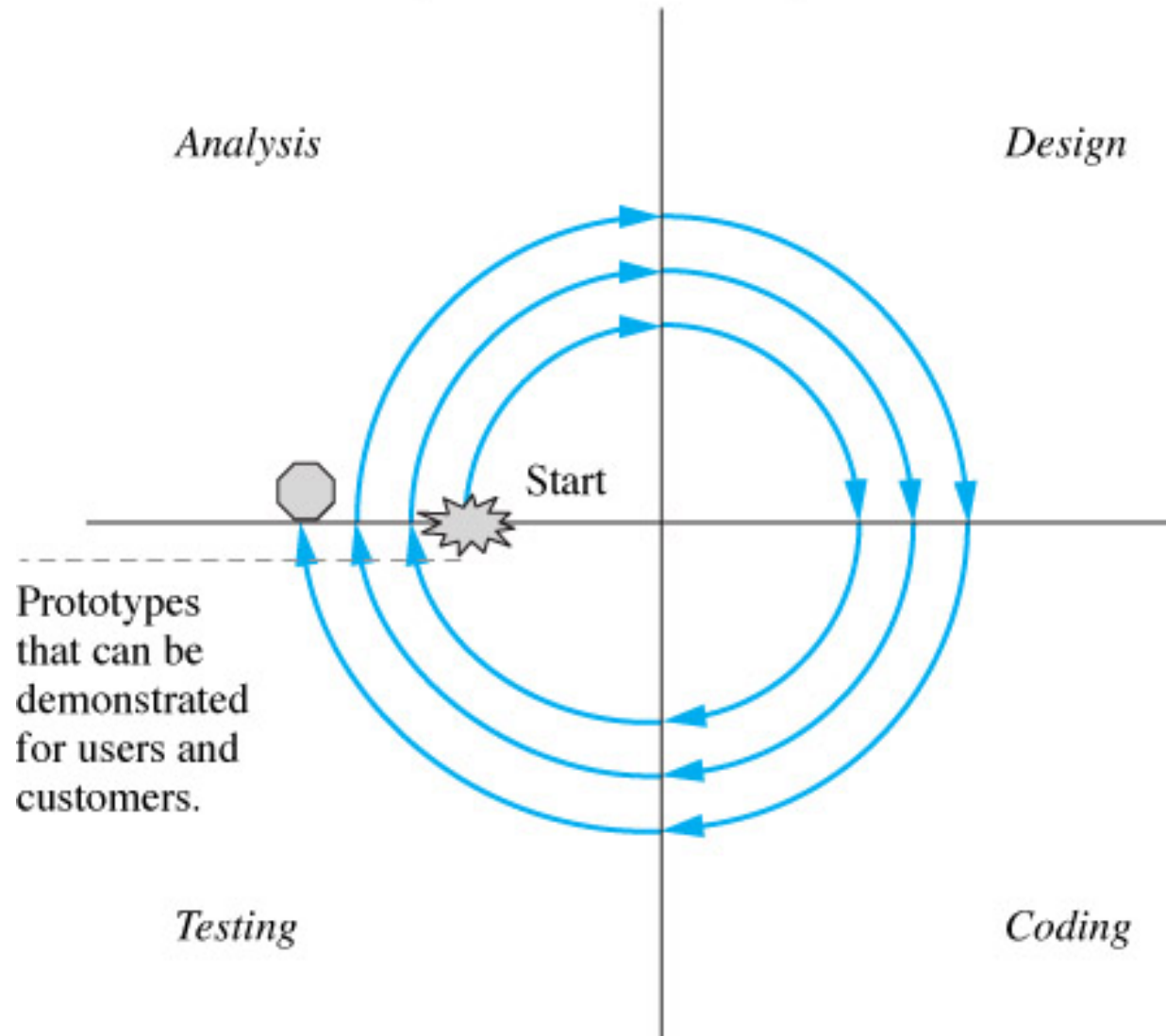
- Turn in by email by Midnight on April 6:
 - documented java code

- Build and test in phases
- More fun, builds confidence, less frustrating
- Small phases are easier to debug
- Often called the Spiral method or spiral design method

- A simple way to design software.
- Particularly effective for students new to programming, OR
- When learning a new programming language
- Also, when building an application in
 - new area,
 - or with a new technology,
 - or new techniques.
- Build and test in parts, that slowly evolve
- Build a prototype, and then evolve and grow the prototype.

- Build a prototype, and then evolve and grow the prototype.
- Prototype Demonstrations: and excellent way to communicate with
 - clients,
 - customers,
 - focus groups,
 - managers.
- Prototype demonstrations can **validate** the:
 - objectives,
 - goals,
 - and user interface

Spiral or Evolutionary Model



1. Processor and Memory Thread; both say hello
2. Add shared memory; confirm it works by having the processor place an item on the bus, and the memory prints it.
3. Memory has internal array of 10 items, initialized; have memory print the contents of the array
4. Processor has internal cache (5 pages) initialized to -1. (use -1 to indicate invalid – no data cached in that location.
5. Processor generates addresses randomly (from 1-10 or 0-9). Implement this and have the processor print out a series of random numbers.
6. Processor generates random addresses and places them on the address bus. Memory places the desired data on the data bus. The processor reads in the data and prints it out. [skipping the cache at this point]

7. Processor generates random addresses and places them on the address bus. Memory places the desired data on the data bus. The processor reads in the data and prints it out. [skipping the cache at this point]
8. Next, when the processor gets data from memory, it stores it in a location in its cache, and prints it out.
9. Next, have the processor randomly decide which page in the cache to replace (once all pages are full).
10. Next, have the processor check its cache for an address, prior to placing the address on the bus. If the page is found, do not go to memory, but print the found data. Make sure this is working correctly.
11. Record the number of cache hits and misses, and have the processor print this out at the end of your run. Processor thread should also compute and print the hit rate.

12. Have the processor cycle through 10 times, and 100 times. Do you have better hit rate after 100 times than for 10 times? Why?

End

Of

Today's

Lecture.



This slide intentionally left blank