Parallel and Distributed Algorithms

Description: This course will cover topics in interprocess communication, synchronization, concurrent programming, parallel processors, distributed networks, and local networks.

1) Architecture of high-performance computers
   a) FlynnÆs Taxonomy (SIMD, MIMD, etc.)
   b) Pipelining and vector processing
   c) Dataflow architectures
   d) Memory hierarchy
   e) Clustering
2) Performance measurement
   a) MIPS, MFLOPS
   b) Granularity
   c) Speedup and AmdahlÆs Law
3) Parallel Programming models
   a) Data parallel
   b) Shared memory (pthreads)
   c) Message passing (MPI/PVM)
   d) "SPMD" model
4) Parallelism in programs and compilation for parallelism
   a) Dependence analysis and load balancing
   b) Architectural constraints – memory hierarchy, computation vs. communication, bank conflicts on vector machines.

1) Architecture of high-performance computers
   a) FlynnÆs Taxonomy (SIMD, MIMD, etc.)
      i) SIMD: Single-instruction, Multiple Data: multi-processor machines will execute the same instruction on data that is different. Examples of this type of processing would include problems that can be divided, such as the Mandelbrot set where there is no-interdependency of data.
      ii) MIMD: Multiple-instruction, Multiple-Data:
      iii) SISD:
      iv) MISD:
      v) SPMD: Single Program, Multiple-Data
   b) Pipelining and vector processing
   c) Dataflow architectures
   d) Memory hierarchy
   e) Clustering
2) Performance measurement
   a) MIPS, MFLOPS
   b) Granularity
   c) Speedup and AmdahlÆs Law
      i) Speedup factor
         (1) $S(n) = \frac{t_s}{t_s / n} = n$
         (2) S (n) > n
      ii) Amdahl's Law
         (1) $S(n) = \frac{n}{1 + (n-1)f}$

(2)

3) Parallel Programming models
   a) Data parallel
      i) Data Parallel means that the same operation is performed on multiple data simultaneously.
      ii)
   b) Shared memory (pthreads)
   c) Message passing (MPI/PVM)
   d) "SPMD" model
4) Parallelism in programs and compilation for parallelism
   a) Dependence analysis and load balancing
   b) Architectural constraints – memory hierarchy, computation vs. communication, bank conflicts on vector machines.

Describe the **Bubble Sort**

The bubble sort algorithm attempts to sort a list by comparing consecutive values and moving a value up or down in the list according to its order. The motion of a value in the list resembles a bubble, in that values that are being sorted appear to move upward (depending on the order, i.e. ascension/descension).

Is the Bubble Sort Parallelizable?

What sorting algorithms are "*sortable*" and why?

Advanced Operating Systems

Description: Topics will include areas such as methods of interprocess communication, reliability, maintainability, security, and large-scale design considerations.

1. *Protection of system memory-how can this be done [protected instructions, interrupts, user vs. supervisor mode].*
2. *Virtual memory-what is it, how does it work [associative storage, page table], what are its advantages and disadvantages.*
3. *Synchronization techniques-locking and semaphores in detail, other techniques such as time stamps, monitors, rendezvous should have some awareness that these options exist.*
4. *Synchronization problems-readers/writers, dining philosophers should be understood to give examples of applying techniques and possible problems.*
5. *Deadlock-what is it, how can it be prevented, avoided, detected, recovered.*
6. *Multiprocessor systems-problems such as synchronization-message passing vs. shared memory.*
7. *UNIX internals-what is an inode, what is stored in an inode, what implications does this have to the user-how is a directory managed-fork and exec to run a child process.*
8. *Remote procedure calls-what are they, how can they be used, how can they be implemented.*
9. *Security-problems and solutions.*

1. **Protection of system memory-how can this be done [protected instructions, interrupts, user vs. supervisor mode].**
    - **a.** Protection of system resources that is done in software by attributing its permissions to flags. These flags can be anything from a semaphore – allowing only one access to a data structure at a time -, to a permission flag (User vs. Kernel access).
    - **b.** In order to implement a semaphore, we must have an atomic relationship to set the semaphore. In the Linux kernel, there are two methods, down() and up(), to do this. The down operation uses the atomic "lock" and tries to employ a decrement, whilst the up operation uses the lock and tries to commit an increment

2. **Virtual memory-what is it, how does it work [associative storage, page table], what are its advantages and disadvantages.**
    - Virtual memory is the separation of user logical memory from physical memory.
    - It allows us to have programs in memory larger than the actual physical memory of the system.
    - We also do not have to have the entire program in memory, as some parts are rarely executed (error handling areas, for example).
    - Virtual memory refers to the use of "Soft memory". This is usually employed as a local operation on the hard disk.
3. **Synchronization techniques locking and semaphores in detail, other techniques such as time stamps, monitors, and rendezvous should have some awareness that these options exist.**
    a)
4. **Synchronization problems-readers/writers, dining philosophers should be understood to give examples of applying techniques and possible problems.**
    - a. A reader/writer problem indicates the problem of a reader reading a file, whilst the writer is making changes. Do we worry about the reader not getting the changes
    - b. **Dining Philosopher problem**: A certain number of philosophers spend their lives alternating between thinking and eating. They are seated around a circular table. There is a fork placed between each pair of neighboring philosophers. Each philosopher has access to the forks at her left and right. In order to eat, a philosopher must be in possession of both forks. A philosopher may only pick up one fork at a time. Each philosopher attempts to pick up the left fork first and then the right fork. When done eating, a philosopher puts both forks back down on the table and begins thinking. Since the philosophers are sharing forks, it is not possible for all of them to be eating at the same time. This problem is a problem of handling deadlock and starvation.
        - i. **Deadlock** can occur when each of the philosophers has a fork, but cannot eat since it requires that each has two forks to eat.
        - ii. **Starvation** occurs because the philosophers can never get both forks (resources) to complete eating, so it can move back to hungry.
    - c.
5. **Deadlock-what is it, how can it be prevented, avoided, detected, recovered.**
    - a. Deadlock is the occurrence of two resources waiting on each other to release resources.
    - b. It can be prevented by ensuring that each process requests the processes it needs beforehand and using a critical region to hold these resources.
        - i. Synchronization may be an issue.
        - ii. If so, we should use an atomic operation to access the lock.
    - c. **Livelock**: When a message is passed around the network and never reaches its destination.
    - d.
6. **Multiprocessor systems-problems such as synchronization-message passing vs. shared memory.**
    - **a. Cache Coherency:** Each cache on multi-processor systems, that hold the same data structures should have the same values.
    - **b. Cache Snooping:**
    - **c.**
7. **UNIX internals-what is an inode, what is stored in an inode, what implications does this have to the user-how is a directory managed-fork and exec to run a child process.**

a. **Inode**: inodes are data structures that contain information about files in Unix file systems. Each file has an inode and is identified by an inode number (i-number) in the file system where it resides. inodes provide important information on files such as user and group ownership, access mode (read, write, execute permissions) and type.
   - inodes are created when a file system is created. There are a set number of inodes, which indicates the maximum number of files the system can hold.
   - A file's inode number can be found using the *ls -i* command, while the *ls -l* command will retrieve inode information

b.

8. Remote procedure calls-what are they, how can they be used, how can they be implemented.

a. Remote Procedure calls are function calls that are executed on a remote machine (server) but called by a local machine (client).

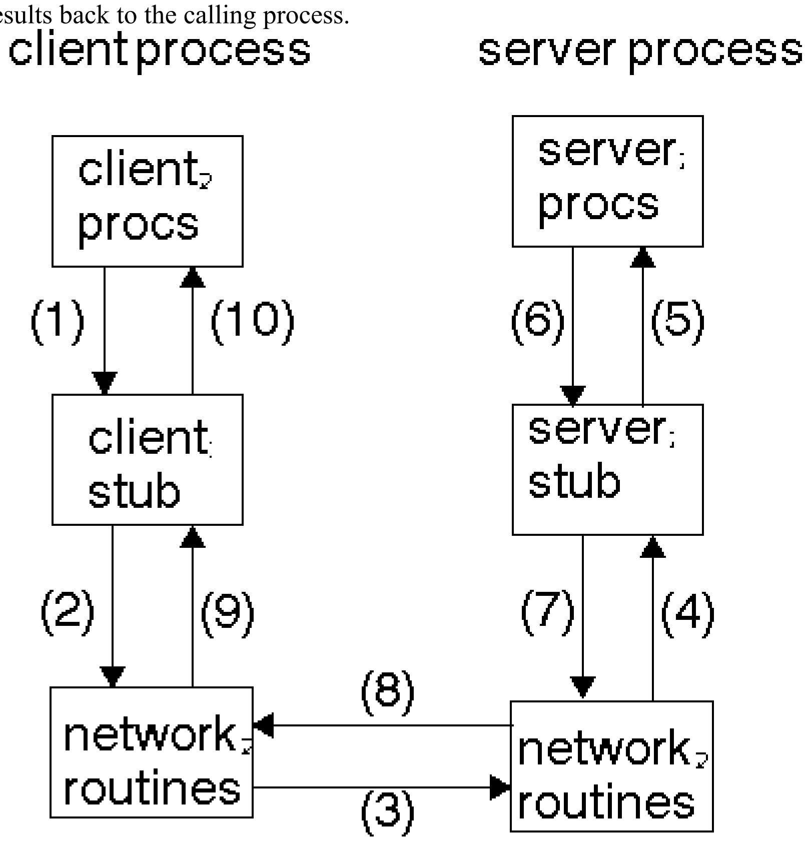
b. They can be used to instantiate objects or executables on remote machines.

c. They can be implemented using socket calls, MPI.

Stubs

When the calling process calls a procedure, the action performed by that procedure will not be the actual code as written, but code that begins network communication. It has to connect to the remote machine, send all the parameters down to it, wait for replies, do the right thing to the stack and return. This is the client side *stub*.

The server side stub has to wait for messages asking for a procedure to run. It has to read the parameters, and present them in a suitable form to execute the procedure locally. After execution,it has to send the results back to the calling process.



1. The client calls the local stub procedure. The stub packages up the parameters into a network message. This is called *marshalling*.
2. Networking functions in the O/S kernel are called by the stub to send the message.

3. The kernel sends the message(s) to the remote system. This may be connection-oriented or connectionless.
4. A server stub unmarshals the arguments from the network message.
5. The server stub executes a local procedure call.
6. The procedure completes, returning execution to the server stub.
7. The server stub marshals the return values into a network message.
8. The return messages are sent back.
9. The client stub reads the messages using the network functions.
10. The message is unmarshalled. and the return values are set on the stack for the local process.

An RPC system consists of the following pieces

- A set of data types that can be sent as parameters and received as results
- A representation of these data types on the wire
- A message format for sending procedure name and parameters and getting procedure reply
- A means of representing the procedures and their signatures (parameters and result types) in a form independent of implementation - this will be used to generate two implementations, one on the client side, the other on the server. This acts as a specification of a remote procedure call
- A mechanism for automatically generating a client stub
- A mechanism for automatically generating a server stub
- A way of linking the server-side implementation of the procedure to the server stub so that the stub can call it

9. **Security-problems and solutions.**
    a. Security problems could include accessing files, executables or resources that shouldn't be accessed by a non-root user.

Other topics covered: (put in by ME- the author)

10. **Network RAM, What is it? How is it implemented? Why do we want to use it? Concerns during implementation?**
    a. Network RAM is the idea that we can use multicomputers on a network to increase the amount of in-memory RAM available to us.
    b. Typically, this is done in a NOW (Network of Workstations) environment.
    c. COW (Cluster of Workstations) :
    d. We should be concerned about the
        i. Scalability: Will the system scale to more nodes? What about less?
            1. A scalable system should not have to worry about the attributes of the new node (to be added)-memory size, operating system, architecture, etc.
        ii. Fault-tolerant: If a node goes down, the entire system continues to operate "normally".
11. **Parallel Computing: Idea behind it**
    a. Parallel computing is the idea that we can use multiple processors or multiple machines to accomplish a problem or set of problems faster.
    b.
12. **Shared Memory: What is it, how is it implemented? Why do we want to use it?**
13. **Distributed Shared Memory:** What is it? How is it implemented?

Questions

- What are Semaphores?

A semaphore is simply a counter associated with a data structure. It is checked by all kernel threads before we try to access the data structure. Each semaphore may be viewed as an object composed of :

- An integer variable
- A list of waiting processes
- Two atomic methods: Down () and Up()

- How are processes created?

  The

- Describe the Memory subsystem of Linux.

The Buddy System

External Fragmentation: The allocation and deallocation of pages of memory to processes and system resources causes memory to become "fragmented" such that, even though enough memory exists, there is no ability to allocate sequential blocks.

Internal Fragmentation:
Memory can be internally fragmented in the Buddy System by delving out partial blocks of an allocated block.

Slabs

Color-coding slabs

- What is a **lightweight process**?

A lightweight process is

- What is the difference between a NOW and a COW?

  NOW: Network of Workstations
  COW: Cluster of Workstations