Operating Systems (Four)

Chris Johnson,
Computing Science Department,
University of Glasgow,
Glasgow,
G12 8QQ.

E-mail: johnson@dcs.glasgow.ac.uk
Telephone: +41 330 6053

January 1999.

Introduction To Part 2

- Gets more interesting as we go on...

- Distributed systems:

– mechanisms (RPC);
– general algorithms (clocks, decision processes);
– management (efficiency, security, reliability);
– interaction (multimedia and multi-user systems).

- Moving towards managament issues...

This Lecture

- Distributed systems:

– multiple autonomous processors;
– do not share primary memory;
– cooperate and communicate over network.

- Main advantages:

– improved performance through parallelism;
– increased reliability through redundancy;
– reduce communication, do processing at point of need;
– incremental growth by addition of nodes.

This Lecture

- Two approaches.
- Access distribution through OS calls.
- Access distribution through language constructs.
- In both cases, operating system required.

This Lecture

- How do we support distributed processing?

- Client-server architectures.

- Simple event passing protocols.

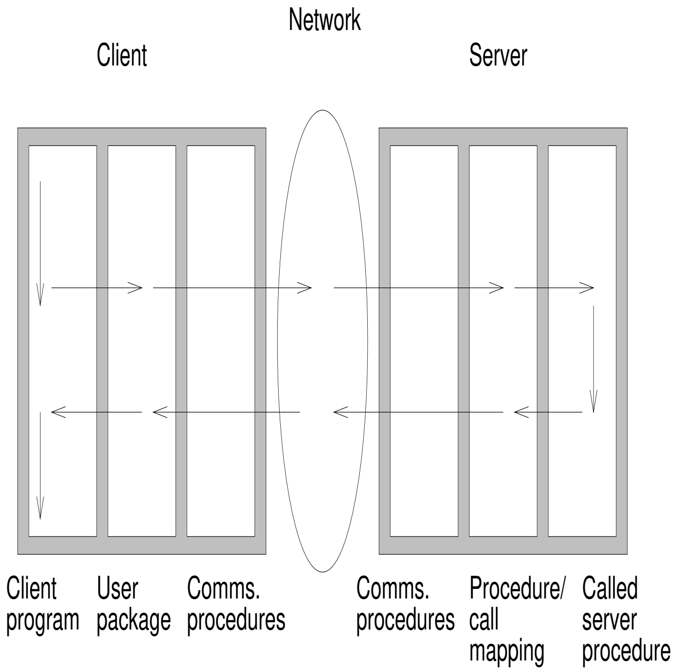
- Stream protocols:

– open flow between source and destination;
– users at terminals. general input/output.

- Or remote procedure call model...

RPC.

- Time flows from top to bottom.



- Simplified architecture.

What Is A RPC Mechanism?

- Similar to ordinary procedure calls.

- Different architecture:

– executed for client by server;
– server process can be on remote machine.

- Different local environments:

– interface defines service between processes;
– global environments different.

- Different call semantics:

– input parameters passed by value, why?
– call by reference difficult, why?

The Main Difference - Errors.

- Network delay/server saturation.

- Maybe call semantics:

– no fault tolerance;
– if request times out then no re-tries;
– server could be down before request arrived;
– request could be done but reply lost.

- At least once semantics:

– if time-out then retransmit message;
– if reply lost then request serviced > 2;
– problem if request not idempotant $req \neq req(req)$

- At most once semantics:

– if time-out then retransmit message;
– unique request identifier used;
– filter out multiple requests.

The Main Difference - Errors.

- Client failure before return,

– creates orphan requests;
– try crashing console on X window manager;
– server performs them, where does client re-start?
– higher level software needed for roll-backs.

- Server failure:

– client may not know if service performed;
– if server restarts - how does it know to filter?
– reliability, save state to file after transaction.

Visibility

- Differences visible in Client-Server interface.

```
/* pseudo code description of common interface */

define remote_procedure remoteproc where
  params (<args>)
  returns(a_type);
  exception(problem);
end remote_procedure;
```

- Increases learning curve.

- Can choose semantics: maybe, at least once...

- Can optimise call (local caches etc).

Transparency

- Make RPC look like normal procedures.

- Reduces learning overhead.

- Compiler detects remote calls:

– what to do with call by reference?

- Some errors difficult to report:

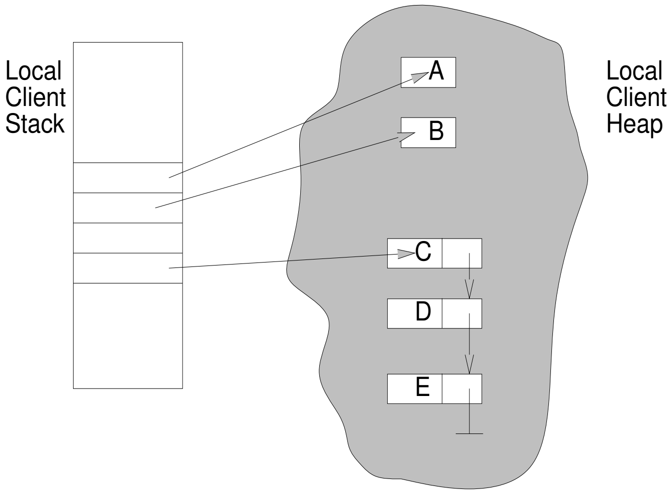
– `kernel low on heap in server'`?.

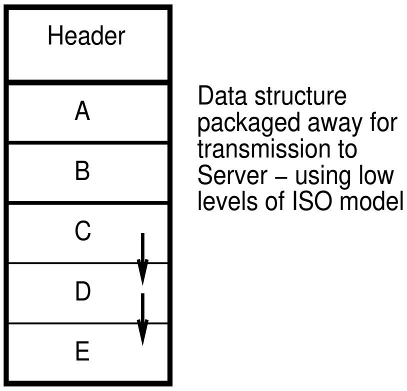
- Efficiency difficult if small part of

– large object needed over network.

Implementation (Client)

- Marshalling arguments.





- User defined data structures?

- Object oriented, methods as well?

Implementation (Client)

- **Stub** converts call to remote request.

- Type of arguments must conform to remote procedure.

- No. of arguments must conform to remote procedure.

- Marshal & pack arguments with remote procedure ID.

- NB heterogenous environments cause problems - what happens on 16/32 bit architectures? Ugh.......

Implementation (Server)

- Dispatcher, responds to request.

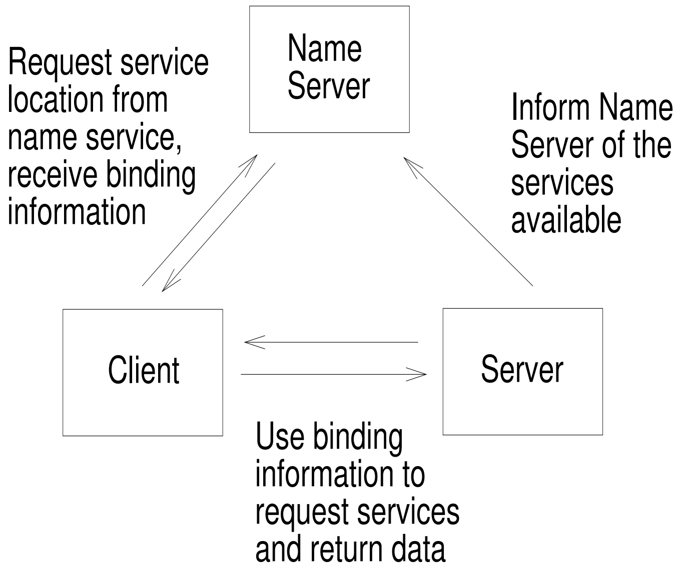
- Unpack request:

– select corresponding procedure;
– pass it the relevant arguments;
– report on any errors;
– marshal up return arguments.

- Interface compilers ensure consistency between client and server argument types etc.

Implementation (Binding Client to Server)

- Evaluated when client first run.



- All service depend on binder,

– how do we achieve fault tolerance?

Implementation (Binding Client to Server)

- In UNIX, typically a socket address:

– Internet address of machine plus port number.

- Mach, Chorus etc, location independent port number.

- Distributed systems use binder service:

– map service names to server ports;
– Domain name service, domain name to Internet addresses.

Sun RPC (Support)

- Sun develop network file systems:

– NFS is stateless;
– file server treats each request as new;
– no problem if client fails;
– if server fails, client retries (see earlier).

- Generalises to full RPC support.

- Now available on top of other NFS.

- Part of At& T's System V (4.0 and up).

Sun RPC (Opaque)

- XDR - external data representation.

- Same syntax as C for types.

```
/* taken from Colouris et al page 139 */
program FILEREADWRITE {
   version VERSION {
      void WRITE(writeargs) = 1;
      Data READ(readargs)=2;
   } = 2;
} = 9999;
```

- Program identifier and version number,

– helps link client's call to server service.

Sun RPC (Parameter Handling)

- Length of request and reply limited to 8k

– if used with User Datagram Protocol;
– simple but unreliable at transport level.

- Most Internet stuff uses the alternative

– Transport Control Protocol to avoid limitation.

- Only one input and one output parameter.

- Reduces consistency checking across interface.

- Would be a significant pain but...

- Can have user defined structures.

- Creates need for marshalling.

Sun RPC (Stubs)

- Client call passed to client stub.

- These in turn call `clnt_call`.

- Parameters:

– client handle;
– server procedure identifier;
– information on how to marshal arguments.

Sun RPC (Interface Compiler)

- Interface compiler called *rpcgen*.

- Generates the stubs for you.

- Will generate marshalling procedures.

- Chosen from pre-defined library:

– integers (all sizes) strings, chars, reals, enumerations.

- User can provide other marshalling procedures.

Sun RPC (Interface Compiler)

- Will generate header file with:

– definitions of common constants;
– definitions of common types;
– signatures of service procedures.

- You have to implement server procedures.

Sun RPC (Binding)

- Local binding service via portmapper.

- Procedure `clnt_create` returns handle.

```
clientHandle =
 clnt_create(serverName, FILEREADWRITE, VERSION, ‘‘udp");
```

- Usually returns socket descriptor.

```
data = read_2(&a, clientHandle);
```

```
/* provide address of local data structure to store results.*/
/* lower case used from interface definition, hence READ is read.*/
/* Version number appended to name, hence read becomes read_2.*/
```

- Calls look like normal C calls.

Sun RPC (Call semantics)

- `clnt_call` - at least oce call semantics.
- Time out between re-tries set when handle obtained.
- If no response after time-out, then repeat request.
- And again, and again until max period reached.
- Error value from client stub if still no reply.

Advanced Network Systems Architecture

- Second example RPC system.

- The system includes:

– an interface language;
– a compiler;
– C comms library for Server $\Leftrightarrow$ client;
– a binder (known as the ‘interface trader’).

- Exploit UNIX socket environment.

ANSA (Opaque)

- DPL - Distributed Programming Language.

- Can embed variety of programming languages in DPL.

- Easy to re-use/distribute existing code.

- DPL provides facilities to handle exceptions in C etc.

- Signatures can include lists of parameters.

- Handles basic types plus enumerations,

– arrays, sequences and records.

ANSA (Server)

- Common definitions shared by client and server.

```
fileops :  INTERFACE =
begin
   read :  OPERATION (<args_in>) RETURNS (<args_out>)
   write:  OPERATION (<args_in>) RETURNS (<args_out>)
end.
```

- IDL, ! prefix indicate preprocessor required introduce run time system calls to marshal arguments.

```
![fileops_exportRef] := traderRef
 $EXPORT(``fileops'', ``/ANSA/services'',\``NAME`green''', NTHREADS)
```

- Server exports it to trader to offer to clients.

- Returns `fileops_exportRef` as link to client call.

ANSA (Client)

- Set up binding via the trader.

```
!  [fileops_importRef] := traderRef
 $IMPORT(``fileops'', ``/ANSA/services'',\``NAME`green''')
```

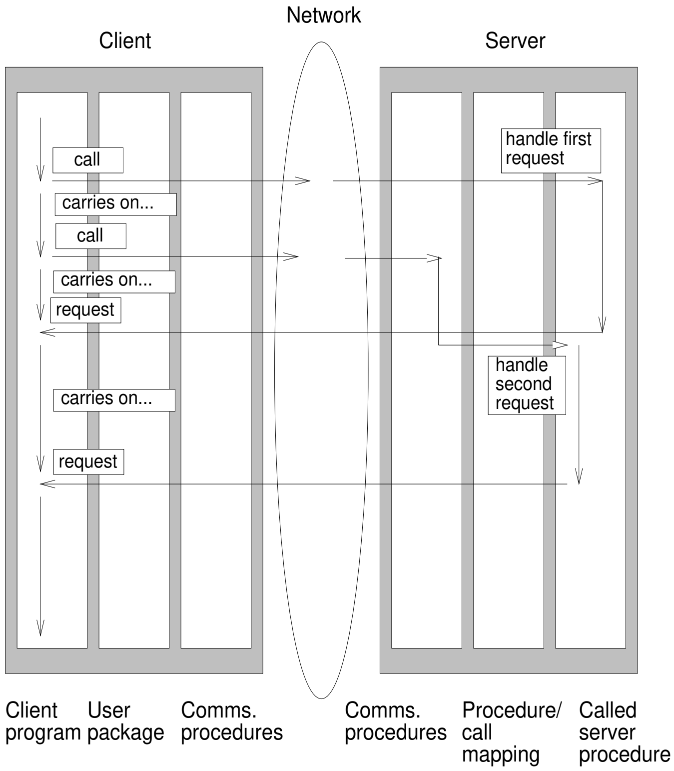
- First has to get `fileops_importRef` from the trader.

```
!  [result] := fileops_importRef$read(<args_out>)
```

- Then issues call to server.

ANSA (Call Semantics)

- Typically uses 'at most, once call' but...



- Distributed window systems such as X-11

ANSA (Call Semantics)

- Asynchronous versions of RPC:

– client issues request for server;
– picks up result from server later.

```
voucher v;
!  [v] := fileops_importRef$read(<args_out>)
!  [result] := fileops_importRef$REDEEM(v)
```

- Vouchers available for asynchronous calls.

Conclusion

- RPC comparison SunRPC to ANSA.

- Possible exam question.

- You must consider:

– interface definition language;
– concurrency in a service;
– binder;
– call semantics;
– limitations on arguments;
– transparent vs opaque;
– marshalling.

- See page 149 in Colouris et al.

Introduction

- So what's the problem?

- External/internal synchronisation

- Cristian and Berkley Algorithms.

- Network Time Protocol.

So what's the problem?

- Accountancy:

– find out when an event happened on a distributed system.

- Coordination:

– two systems cooperate at some point in future.

- Recovery:

– one system rebuilds another from particular point in time;
– transaction locking to prevent duplication of service.

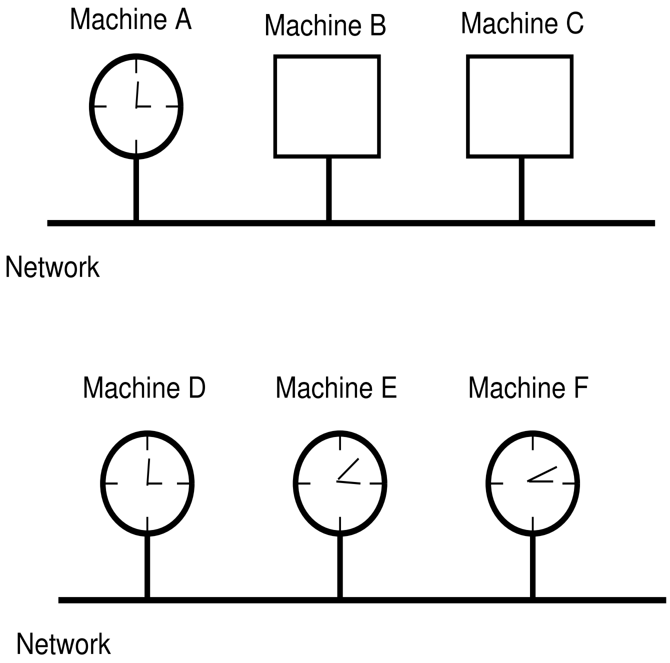
- Criticality:

– one system issues task for completion in set time.

So what's the problem?

- External synchronisation:

– one authority provides time service?



- Problems:

– network unpredictable for communicating time with clients;
– authorities clock drifts away from other time sources

External sources of time

- Radio transmissions:

– radio waves at 3×10^8 meters per second;
– accurate to 0.1-10 milliseconds, depends on location.

- Geostationary satellites:

– accurate to about 0.1 milliseconds (GEOS system);
– accurate to 1 millisecond (GPS system).

- But receivers are expensive.

- 10 milliseconds:

– 100,000 instructions at 10 million instr/sec;
– not accurate enough.

So what's the problem?

- Internal synchronisation.

- Systems have their own clocks and try to ensure that they agree about time (consistency).

- Problems:

– network unpredictable for synch. with local clocks;
– local clocks will drift away from other time sources

Internal Sources Of Time

- Clocks are typically:

– based on oscillating crystals;
– subject to physical differences in crystal;
– affected by temperature of environment.

- Accumulator used to record ticks.

- Resolution period between updates in accumulator.

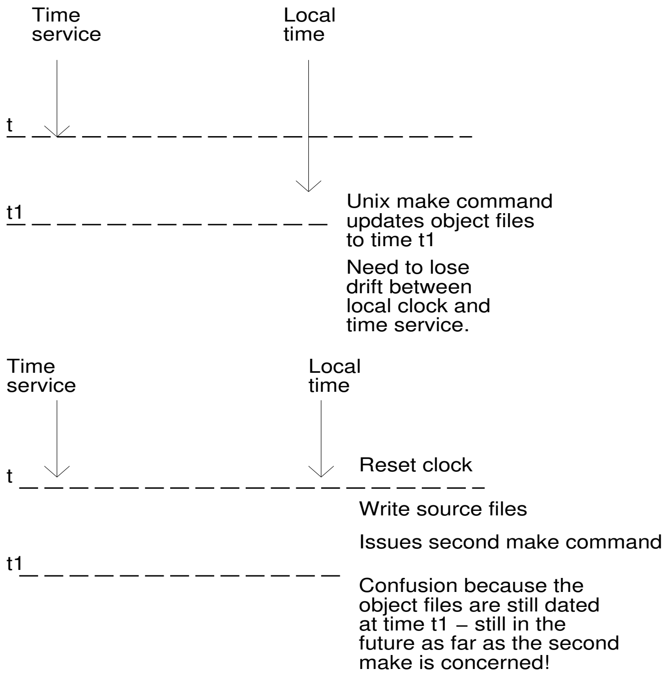
- Drift rate is difference with 'perfect' clock.

- 1 second/11.6 days in quartz crystal.

Synchronising Internal Clocks

- If the local clock is too slow:

– add extra ticks to the accumulator.



- if the local clock is too fast:

– can't just subtract ticks from 'accumulator'.

Synchronising Internal Clocks

- Slow down local clock, usually in software:

– to get rate of change, assume a and b are some constants.

$$
\begin{aligned}
&software\_clock(t) = hardware\_clock(t) + compensation(t) \\
&compensation(t) = a(hardware\_clock(t)) + b
\end{aligned}
$$

- Could be $T_{skew} > T_{real}$ or $T_{real} > T_{skew}$.

$$
\begin{aligned}
&a = (T_{real} - T_{skew})/N \\
&b = T_{skew} - (1 + a)(hardware\_clock(t))
\end{aligned}
$$

- After N ticks of the clocks.

- See page 291 of Coulouris et al.

Calculating Local Drift

- Receivers are expensive - so fit one to central server.

- But transmission time depends on network.

- $T_{trans} = min + x$ where $x \geq 0$.

- min is the ideal smallest transmission time:

– no network delay;
– no competing processes.

- Derive min by empirical means/analysis.

- x derived as mean of empirical values.

Cristian's Algorithm

- Can we do better than this?

- We know transmission time:

– period between request and message from server;
– this may be subject to small local drift on client;
– estimate 1-10 milliseconds for service on LAN;
– drift rate liable to be 10^{-5} seconds.

current_time = time_returned + trip_time/2

- First approximation.

Cristian's Algorithm

- This assumes round trip takes same time.

- Might be quicker on the way back - or slower.

- Must take some min on the way back.

accuracy_range = +-(trip_time/2 - min)

- Issue several requests.

- Most accurate has smallest transmission time.

- Very small probability; both messages approach min.

Berkley Algorithm

- Berkley UNIX uses variation.

- Time server samples clocks of clients:

– observes trip times as in Cristian's algorithm;
– calculates mean and includes it's own time in this..

- Doesn't send time stamp back:

– introduces further inaccuracies/delays in transmission.

- Instead, sends back + or - drift.

- Lots of problems and refinements on this:

– what if one clock very, very slow?
– affects the average so fault-tolerant ranges used.

Network Time Protocol

- Standard for clock synchronisation over Internet.

- Primary servers attached to radio/satellite.

- Secondary servers feed from primary servers.

- Forms a hierarchy:
  – inaccuracies increase further away from primary servers.

- Procedure call mode is Cristian's algorithm.

- Symmetric mode is like handshaking with times.

For instance...

- Departmental set-up.

We run two set of three interlinked timeservers in the department - the first set synchronise with off-site machines and with each other. The second set synchronise with the first set and with each other. The rest of the machines which use NTP stuff to set their time twice a day by asking the second set of three machines what the time is - I think they take the first answer they get.
I can't remember how many machines we actually have in our primary set - it was three originally but it may have changed over time. They chime against different external machines. Hawaii gets external time from Strathclyde for example. Douglas (support).

- Draw this up as a diagram and explain.

Conclusion

- So what's the problem?
- External/internal synchronisation
- Cristian and Berkley Algorithms.
- Network Time Protocol.

Introduction

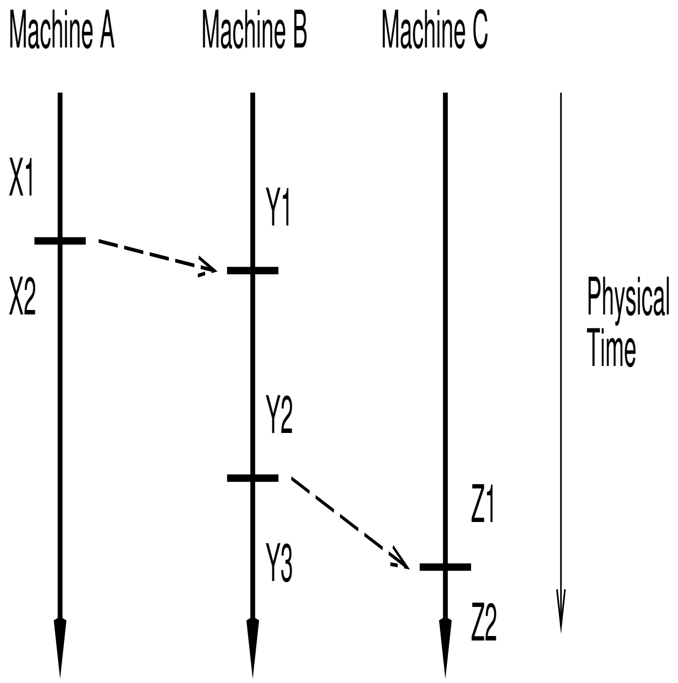
- Logical clocks.
- Ladkin's work on causation.
- Vector time.
- Gossip architecture.

What's The Problem

- Cannot perfectly synchronise clocks over network.
- Synchronisation error over 30 milliseconds using NTP.
- Can't use physical time to order events.
- Big problem for transaction based systems.
- Especially on replicated databases etc.
- Add-delete, delete-add - not commutative.

Logical Clocks

- What do we know about ordering between events?



- X1 must occur before Y1...

Causality

- Communication events impose a partial ordering.

- Message must be sent before it is received.

- We can deduce:

– events in region X1 occur before events in Y2 and Y3;
– events in region Y2 take place before Z2;
– transitivity as well so X1 before Z2.

- Cannot tell without further communication about relative ordering of events in regions X1, Y1 and Z1.

Lamport's Work

- Lamport formalises these intuitions.

L. Lamport, Concurrent Reading And Writing Of Clocks, ACM Transactions On Computer Systems, 8(4), 1990.

- Causal links provide interleaving.

- If no causal links they are concurrent ($a||b$).

Lamport's Work

- Developed idea of **logical clocks**.

- Aim: consistent ordering of events - not
to achieve consistent value of time at every node.

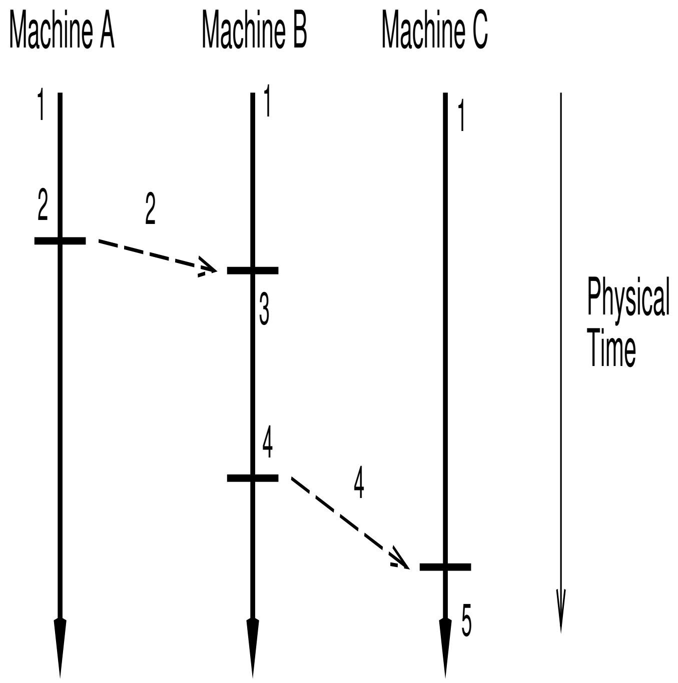
- Contrast this with Cristian and Berkley algorithms.

- Monotonically increasing logical counter.

- In motor racing:
– lap counter is like the logical clock;
– the lap timer is a physical clock.

Logical Clocks

- Transmit logical clock with message.



- Update local logical clock.

Logical Clocks

- The basic algorithm.

```
1:      Each node updates logical clock after each event:
        logical_clock := logical_clock + 1;

2:      When message sent it piggybacks value of its logical clock:
        message(text, logical_clock); -- then does 1

3:      On receiving message the node works out maximum:
        sent_clock := clock_in(message(text, logical_clock));
        clock := Max(logical_clock, sent_clock); -- then does 1
```

- Page 299 Coulouris et al for more detail.

- If a causes b then clock at a $<$ clock at b.

- Because receipt of message updates clock at b.

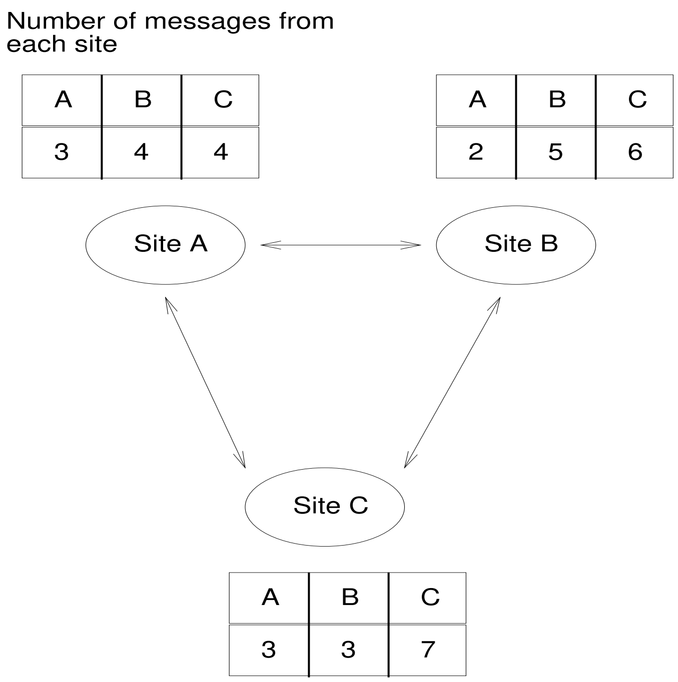
Problem

- Causal orderings are not explicitly represented.
- Add to clock if no relationship between messages.
- For example, items posted to Internet news.
- Might arrive at site in arbitrary order.
- Need to ensure up to date versions on each node.
- Then reconstruct thread.

Update Events

- Series of updates lead to current state of news.

- These are held as vectors of event counts.



- Each node will only have part of the picture.

Time Vectors.

- Imagine node A receives message from C.
- A will receive C's vector (3,3,7).
- A's vector is (3,4,4).
- A needs 3 more from C.
- Can tell that B needs one more - lazy?

Gossip Architecture

- Generalised use of time vectors.

- Front end requests operations on replicated data.

- For non-commutative operations must know vector.

- Replication managers maintain a log that contains:

```
Log_Item := <TimeStamp, OperationID, VectorSentWithOp>;
```

- Time_Stamp is the Vector_Sent_With_Op with the replication manager's clock incremented by one.

- Time_Stamp sent back to merge with front end vector.

Gossip Architecture

- Problem if request arrive and:
local time < Vector_Sent_With_Op.

- Wait for other messages to create correct context.

- Periodically, other managers send updates.

- Can be used to repair problems.

- Senders note destinations to avoid repetition.

Conclusion

- Logical clocks.

- Ladkin's work on causation.

- Vector time.

- Gossip architecture.

Introduction

- Atomicity

- Mutual Exclusion.

- Leadership elections (next time).

- Byzantine Generals (next time).

What's the problem?

- Next two lectures about consensus algorithms.

- For instance, distributed replicated data.

- If one changes, all must change - make it atomic.

- Otherwise data will be inconsistent.

Atomicity

- Global transaction coordinator:

– starts local execution of transaction.

- Manages execution of task:

– breaks transaction into smaller sub-transactions;
– distributes sub-tasks throughout the nodes.

- Approves results:

– if everything ok then commit transaction at all sites;
– if problems then abort transaction at all sites.

Atomicity

- Two-Phase Commit (TPC).

- Phase 1:

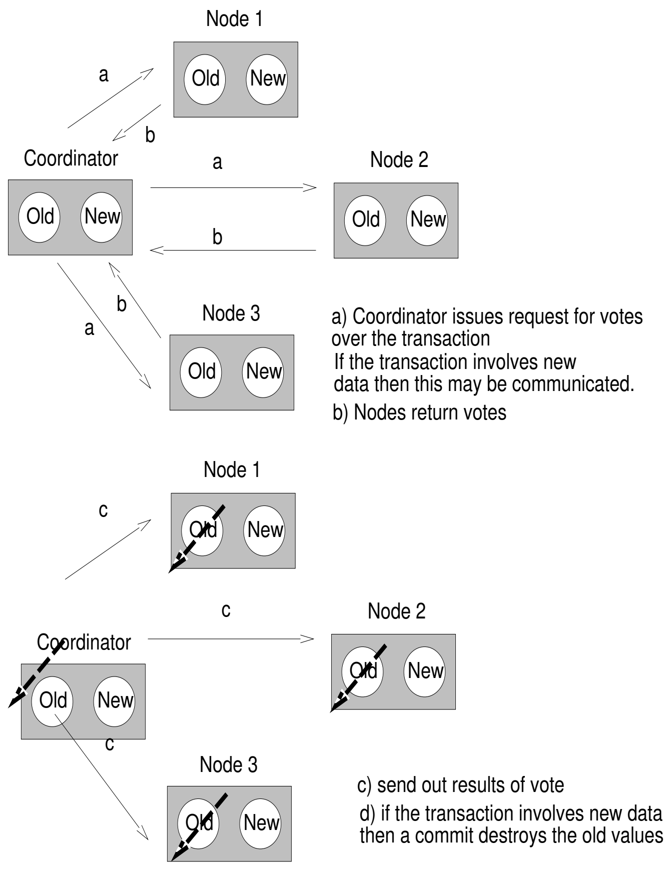
– coordinator logs prepare(transaction);
– send preparation message to all nodes;
– local sites log accept or reject transaction;
– local sites send accept/reject to coordinator.

- Phase 2:

– if messages missing then timeout and repeat request;
– if some sites refuse then coordinator adds abort to log;
– if some sites refuse then coordinator broadcasts abort;
– if all sites accept then coordinator commit transaction;
– if all sites accept then coordinator adds commit to log;
– if all sites accept then coordinator broadcasts commit.

Atomicity

- Two-Phase Commit.



- Assemble votes and propagate decision.

Failures

- Why maintain the logs?
- If node fails then consult local log on restart.
- If log contains a commit then do transaction.
- If log contains abort then undo transaction.
- If log contains accept then:

– must find out if all other processes accepted it;
– communicate with coordinator.

- Coordinator fails → leadership election (next time).

Mutual Exclusion

- This works for uniform changes on shared data.

- But what if data or resource cannot be shared?

- Safety:

– prevent two process from entering critical region.

- Liveness:

– eventually process granted non-sharable resource.

- Distributed systems raise problems:

– communications delays over network;
– ordering, requests in ‘happened-before’ order?

What's The Problem

- In single case, semaphores like railway signal.

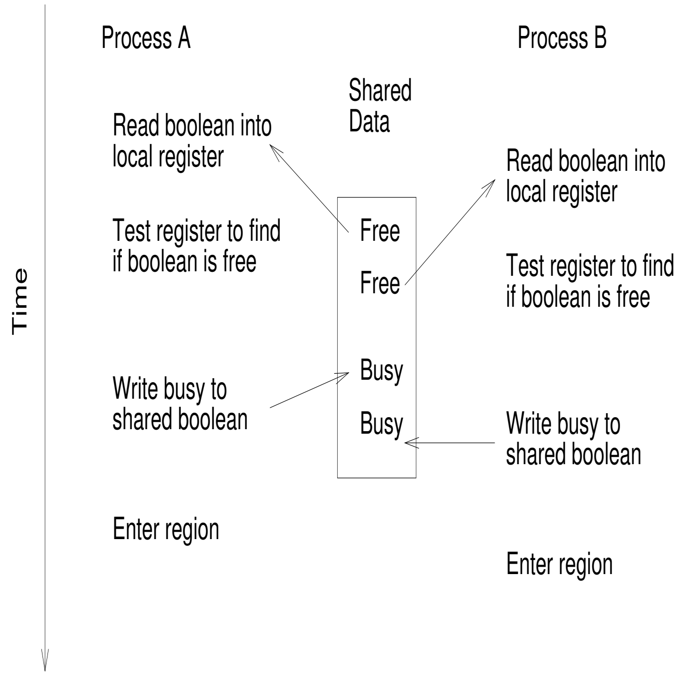
```
semaphore:= 1; -- why is this ??
Process P1
   -- does something
   wait(semaphore);
   -- enters critical section if non-negative
   signal(semaphore); -- leave critical region
   -- does some more
end Process P1;


Process P2
   -- does something
   wait(semaphore);
   -- enters critical section if non-negative
   signal(semaphore); -- leave critical section
   -- does some more
end Process P2;
```

- Why must `wait` and `signal` be atomic?

- Why wont this work on distributed systems?

What's The Problem

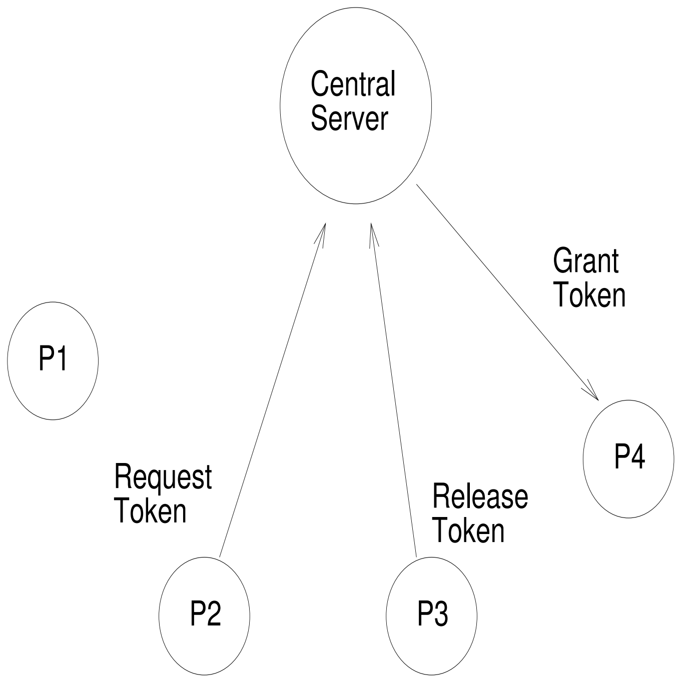
- Two request fail to guarantee exclusion.



- Hardware test & set instructions - atomic.

Central Servers

- Pass a token around in order of receipt.



- What problems are there with this?

Central Servers

- MACH facilities:

– port used as a synchronisation variable;
– thread requesting resource issues recieve call on port;
– when resource available, receive message from that port.

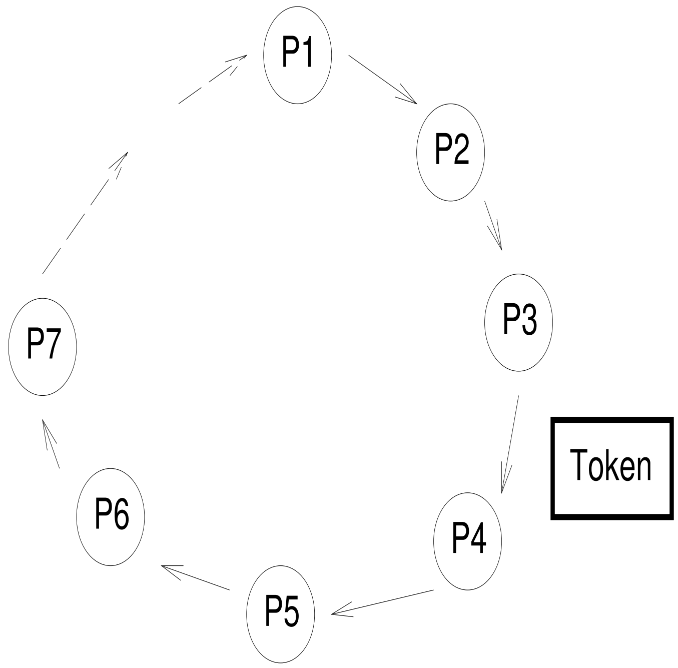
- Only works among threads of same task:

– only one task has receive rights on a port.

- Have to write a daemon to do it betweem tasks.

Ring Algorithms

- Pass a request round the circle.



- Does this guarantee 'happened before' order?

Ricart & Agrawala's Algorithm.

- Broadcast request to enter critical section.

```
state := RELEASED; -- on initialisation

state := WANTED; -- to obtain token
multicast_request(timestamp, process_id);
T := timestamp;
wait_until(no_of_replies = (n-1));
state := HELD;

enter_critical_section;

state := RELEASED;
reply to any requests;

-- receipt of multicast_request(timestamp, process_id);
if (state = HELD) or (STATE = WANTED and (local_time < timestamp)) then
   queue(multicast_request(timestamp, process_id));
else
   reply(request_approved(process_id, local_id));
   enf_if;
```

- Enter when all others have replied.

Trade-offs

- The bottom line is 2(n-1) messages:

– n-1 to broadcast the request;
– n-1 replies to approve request.

- More expensive than the central server.

- More robust than the central server.

- Look again - is it more robust?

- What if a process fails or blocks?

Conclusion

- Atomicity

- Mutual Exclusion.

- Leadership elections (next time).

- Byzantine Generals (next time).

Introduction

- Consensus algorithms.
- Atomicity (last time).
- Mutual exclusion (last time).
- Leadership elections.
- Byzantine Generals.

Recap

- Must protect critical regions.

- Distributed form of mutual exclusion.

- Ricart and Agrawala's algorithm:

– broadcast request and wait for others to agree.

- Ring algorithms:

– pass request for access around ring.

Leadership Elections

- All of these can break-down:

– token can be lost in transmission;
– ring may lose a link;
– processes may block or crash in critical section.

- Central servers can be used:

– to replace lost new tokens;
– re-route rings and re-start passing;
– restart after process failure.

- If server fails elect a new one:

– must be unique server chosen;
– even if concurrent requests for election.

The Bully Algorithm

- Process requests, server doesn't reply by timeout:

– process tries to elect itself as server;
– sends request to all processes with higher priority.

- If no replies by timeout:

– assumes others failed;
– restarts new copy of coordinator;
– broadcasts this to lower priority processes.

- If a higher process does reply:

– the first process waits;
– gives the higher priority process time T to be elected;
– if no one declares itself then first process restarts election.

The Bully Algorithm

- If a failed process recovers:

– it might have a high process number so...
– tries to get itself elected.

- Process with highest priority forces others

– to let it become the coordinator.

- The highest priority process bullies others.

The Ring Algorithm

- Links are unidirectional, pass token to right.

- Active list of priorities of all active processes.

- If process detects coordinator failure:

– create's own active list, initially empty;
– passes message elect(me) to right;
– adds itself to its own active list.

The Ring Algorithm

- 3 things happen if process gets message to elect another.

- If this is the first message seen:

– creates new active list with itself on;
– adds number of process making request to list;
– send elect(me) & elect(other) round ring.

- Elsif message isn't to elect itself:

– add number of requesting process to active list;
– forward message to next process on right.

- Else message is to elect itself:

– its list now has all numbers of active processes;
– largest number in list is now the coordinator.

The Ring Algorithm

- How does a recovering process find coordinator?
- Why not start an election?
- Sends a request round ring.
- Coordinator sends a reply.
- Then start an election?

Failure Models In Distributed Algorithms

- Why is this a problem?

- US Space shuttle:

– built using off the shelf processors;
– what if it goes wrong?;
– so four identical processors computing same algorithm;
– need to coordinate response of unreliable processors.

- April 8-10 1981 delayed, coordination software fails.

Failure Models In Distributed Algorithms

- Initially-dead process (benign):

– process doesn't execute a single step of local algorithm.

- Crash model (benign):

– local process ok up to a point then doesn't do any more.

- Byzantine model (malicious):

– process begins to execute arbitrary steps - sends garbage.

The Byzantine Generals Problem

- Several divisions of Byzantine army, each division has its own general.

- Surround enemy camp in the dark.

- Divisions must decide whether or not to attack at dawn.

- All generals must agree, split would lead to defeat.

- Only communication via messengers running.

The Byzantine Generals Problem

- So what can go wrong?

- Messengers can get caught:

– like unreliable communications problems.

- False generals and traitors:

faulty processors broadcast confusing messages.

The Byzantine Generals Problem

- Faulty communications.
- Use a timeout - if messenger doesn't return in time T.
- If no reply then re-transmit.
- When should it give up?
- When site or process failure is signaled.

The Byzantine Generals Problem

- Faulty communications (continued).

- What if responding process want to know if it's acknowledgement got through (i.e., agreement)?

- In general, this can't be done with faults.

- The processes can never be sure that both of their messages have got through (Silberschatz p.599).

The Byzantine Generals Problem

- Faulty processes.

- Communication medium reliable, content can be odd:

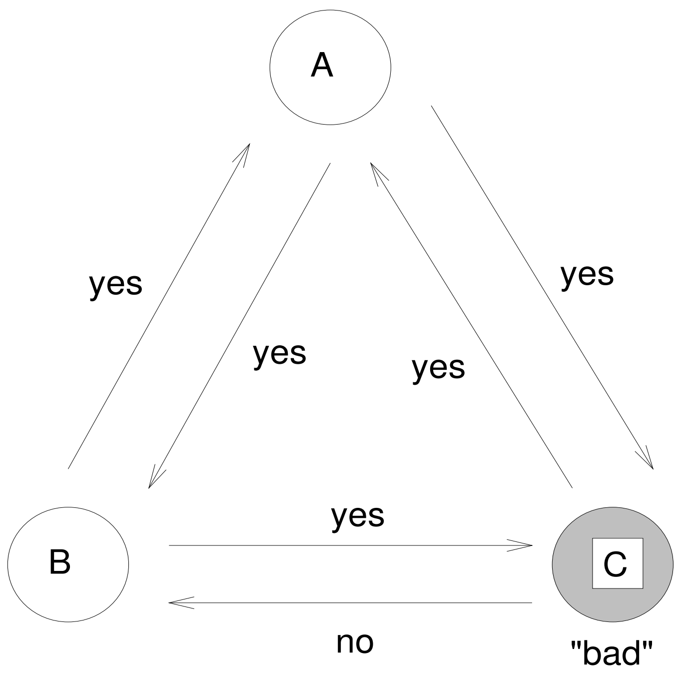
– N processes, no more than m faulty;
– each process p_i has local value V_i;
– non-faulty P_i builds vector $X_i = A_{i,1}, A_{i,2}...$;
– vector such that if P_j is nonfaulty then $A_i, j = V_j$;
– if P_i and P_j are nonfaulty then $X_i = X_j$.

- In other words all of the good general will produce

– the same message within some bounded time.

The Byzantine Generals Problem

- Authentication By Receiver.



- A gets {y,y,n}, B gets {y,y,y}, majority is yes.

- Total of 2N+1 servers can cope with N bad ones.

The Byzantine Generals Problem

- Nightmare scenarios...
- Unreliable communications with Byzantine generals.
- Also what if bad General impersonates a good one?
- Bad C sends messages with A's id attached.
- Include digital signatures, not easily replicated.

Conclusion

- Consensus algorithms.
- Atomicity.
- Mutual Exclusion.
- Leadership elections.
- Byzantine Generals.

Introduction

- Garbage Collection:
  - conventional approaches;
  - distributed approaches

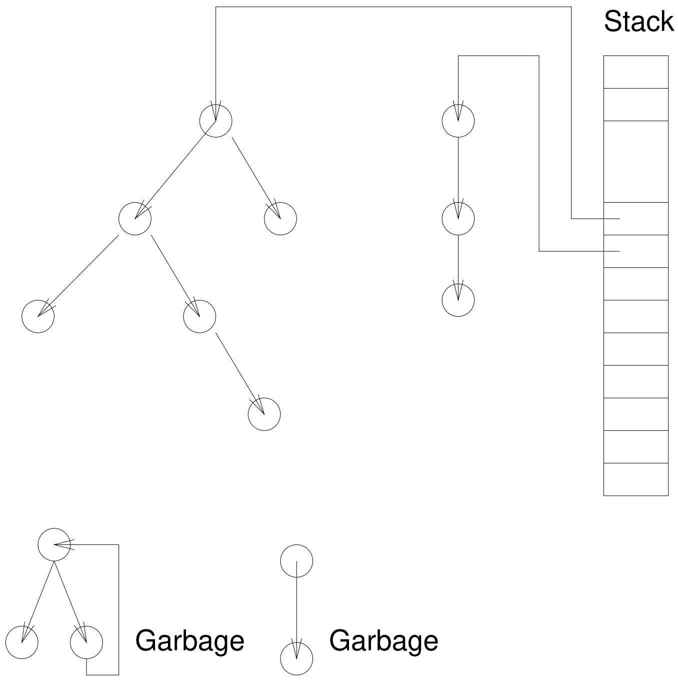
- Direct Identification.

- Indirect Identification.

- Hybrid Approaches.

What's The Problem

- On a single processor machine.



- Hanging references, unconnected subgraph.

What's The Problem

- Originally:

– programmers performed garbage collection;
– free (C), UNCHECKED_DEALLOCATION (Ada);
– error prone, cumbersome, boring;
– Smalltalk, Prolog, Modula etc. use runtime system.

- Time problems:

– Smalltalk-80, 70% time spent garbage collecting;
– Lisp, 40% of time spent garbage collecting;
– currently Smalltalk-80 wastes 5% in garbage collection.

- So what problems remain?

What's The Problem

- In distributed systems:

– no homogenous address space;
– high communication overhead vs. local processing.

- Two main problems:

– identification - what is garbage?
– reclamation - how do we reallocate it?

Identification

- Begin by looking at undistributed techniques...

- Direct method:

– identify cells that become disconnected.

- Indirect method:

– identify cells that are connected, rest is waste.

- Identification technique affects reclamation.

Direct Identification

- Recursive freeing:

– each cell has counter of references to it;
– after mutator operate on it, count can change;
– if it falls to 0 then it is waste and can be reclaimed;
– but must recursively decrement and free its referents.

- Advantages:

– garbage reclaimed as soon as it is identified.

- Disadvantages:

– memory overhead of reference count;
– what size of counts do we represent - use ceiling;
– circular references have infinite counts?

Indirect Identification

- Mark and sweep:

– postpone collection until free store exhausted;
– traverse computation graph marking all accessible cells;
– use a single markbit, saves space over reference count;
– sweep entire memory store and free unmarked cells.

- Advantages:

– can deal with circular references.

- Disadvantages:

– non-incremental so unpredictable reclamation;
– poor for interactive environments;
– poor for real-time environments.

Distributed Garbage Collection

- Each node has access to local heap.

- Cell can migrate between nodes.

- Root typically on node that started computation.

- Remote access:

– reference local copy of export record;
– entry record in export record references remote cell.

- Entry record could be a remote cell as well...

Keeping Track Of The Heap

- Aim is to increase locality of reference.

- Reduces communication.

- Eases garbage collection.

- One approach: partition memory into clusters.

Keeping Track Of The Heap

- Clusters:

– are logical partition of cells;
– these are locality sets for computation;
– not physical partitions or single node's memory.

- Each cluster has:

– import table for external cells referenced by cluster;
– export table of cells in cluster that can be referenced.

- Public cells:

– given public identifiers (PIDs) as opposed to (LIDs);
– known outside cluster, only referenced within cluster.

Keeping Track Of The Heap

- So what is garbage?

- Nonreachable public cells:

– can be referenced from other clusters;
– but not really part of the cluster.

- If subgraphs only referenced from export table:

– remove from cluster and place in archive cluster.

- What do we do with cells in the archive?

- If cell references element in archive then extract

– subgraph and place it in that cluster.

- Cells migrate, cluster $\rightarrow$ archive $\rightarrow$ cluster.

Keeping Track Of The Heap

- So what is garbage?

- Private items not reached from any public cell.

- Archival cells after some time-out.

Direct identification: Reference Counting

- Direct method (recap):
– identify cells that become disconnected.

- Hudder and Keller extend reference counting:
– check reference count when changes made to a cell/pointer;
– spawn a remote asynchronous task to do this;
– actions must be atomic in distributed environment.

- But must preserve order of adding/removing references.

- Why? How?

Direct identification: Reference Counting

- References could be on different machines.

- Benjamin's algorithm:

– known as Generational Reference Counting;
– each cell initially has 0 generation reference;
– ith copy of cell is give $ith + 1$ generation;
– each cell has ledger with references from each generation;
– cell is garbage if no ledger references from any generation.

- GRC overheads:

– one acknowledge message per interprocess reference;
– extra space for ledger reference.

Indirect identification: Mark and Sweep

- Indirect method (recap):

– identify cells that are connected, rest is waste.

- Variants of Dijkstra's mark and sweep:

– wait until memory exhausted;
– sweep memory marking unused cells;
– single root for distributed graph;
– remote tasks for marking can be concurrent;
– check if other process referencing cell;
– if cell is locked reschedule task.

- Disadvantages:

– high message overhead;
– unpredictable space required for communication;
– falls apart if more unwilling/able to cooperate.

Hybrids

- Local collectors.
- May be independent.
- Need not be homogenous.
- One could be mark and sweep.
- Another could use reference counting.

Hybrids

- Bennetts hybrid local/global mark and sweep:

– each node holds a Remote Call Table.

- Global mark and local sweep:

– each node marks local cells currently referenced;
– each node sends out message to increase external
- count of all referenced remote cells;
– local sweeping now removes nodes with 0 count.

- This will not sweep inter-node cycles.

- Distributed mark and sweep as well:

– start with local cells and mark as used;
– send messages to mark remote cells as used;
– send messages to build up links of remote cells etc.

- Internode cycles not part of chain so get swept.

Hybrids

- Lang's algorithm:

– groups are collections of nodes that will cooperate;
– use concurrent mark and sweep collector;
– each group assigns unique identifier to each GC cycle;
– allows overlapping groups to concurrently sweep areas.

- Tolerate failure:

– if node doesn't cooperate then group reforms without it;
– cells referenced by excluded nodes cannot be removed;
– this represents safe (if conservative) fault tolerance;
– acknowledgements and timeouts to detect node failure.

Conclusion

- Garbage Collection:
– conventional approaches;
– distributed approaches

- Direct Identification.

- Indirect Identification.

Introduction

- Configuration management.
- Devices (processors and file servers).
- Networks (next time).
- Users (next time).

What's The Problem?

- Configuration management.

- How do operating systems cope with change?

- Causes of change:

– work profile alters - heavier processing;
– the user profile may change - more of them;
– software may change - how easily can we update;
– hardware may change - how easily can we upgrade.

What's The Problem?

- Configuration management involves:

– identifying software/hardware to provide correct service;
– specifying components eg 5 workstations;
– installing desired component (remotely or manually);
– allocating software to hardware platform;
– binding of external interfaces to internal structures.

- Typically:

– small scale have static installation configuration;
– large scale have dynamic, run-time reconfiguration.

Devices - Processors

- What are the configuration problems?

– introducing changes to hardware?
– introducing changes to software?

- System configuration models:

– workstation approach;
– processor pool approach.

- What impact do these models have?

Devices - Processors

- The Workstation model.

- Workstations dedicated for different purposes:

– public machines (occupied, in use);
– private machines (occupied or in use);
– diskless workstations accessing fileservers;
– full workstations managing local and remote fileservices.

Devices - Processors

- Diskless workstations popular in Universities:

– cost - cheap to have a few large central servers;
– effectiveness - centralised compaction more efficient;
– maintenance - software upgrades handled centrally;
– backups - easier to administer from centralised file space;
– environmental reasons - diskless nodes have smaller fans;
– accessibility -users access central filespace from any node

- Good for configuration management of software:

– no changes to local versions;
– all software accessed on central disk.

- Not so good for configuration of hardware:

– if file servers go down the network goes down;
– if local processor dies you have to find free workstation.

Devices - Processors

- Manufacturers provide local disks as standard:

– Macs (here); PC's (York); Silicon Graphics and Suns etc.

- How do we manage this?

– paging and temporary files stored locally;
– paging, temporary files and system binaries;
– paging, temporary files, system binaries & file caching;
– complete local file systems.

- Not so good for configuration management of software:

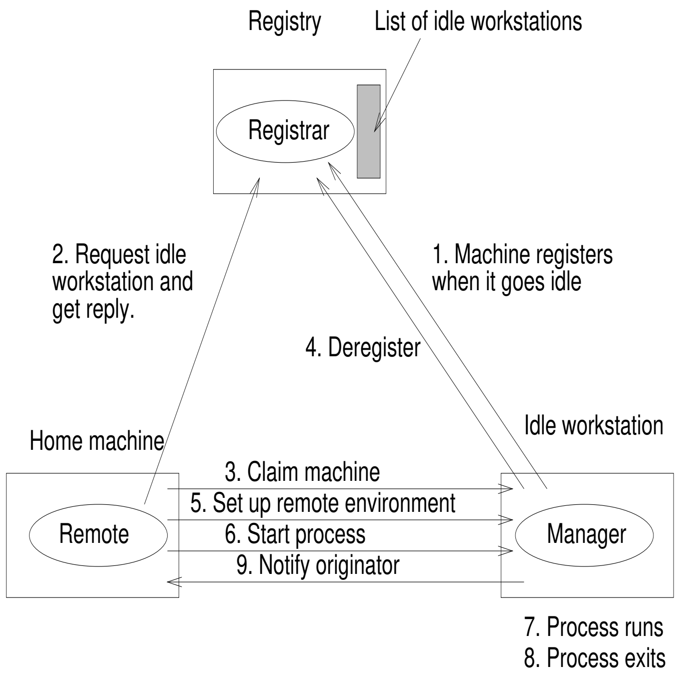
– autonomy & multiple versions of software on local disks;
– but users take responsibility for some administration;
– check own quotas, request/access software updates etc.

- Some flexibility for hardware configuration:

– greater autonomy may improve reliability (eg file server);
– but reliability through redundancy - idle processors.

Devices - Processors

- Workstations are idle for long periods.



- Support `rsh`, `rlogin` etc.

Devices - Processors

- Processors are getting cheaper.

- What will happen in the future:

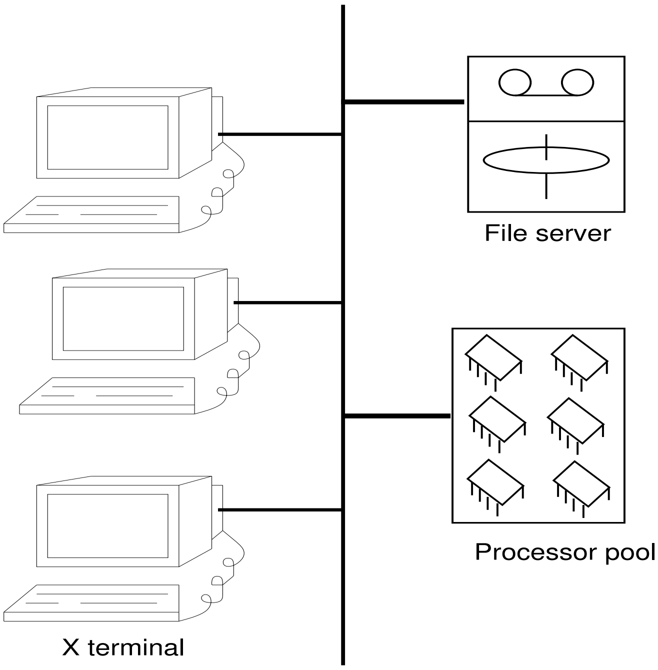
– multiprocessor workstations?
– not very efficient;
– communications in reallocation.

- What will happen in the future:

– multiprocessor pools?
– economies of scale;
– centralised management;
– centralised maintenance;
– centralised power supply;
– increased reliability by rack technology

Devices - Processors

- Processor Pool Model.



- Graphics facilities via X terminals...

Devices - Processor Pools

- Flexible hardware configuration:

– if processing demand grows buy more processors;
– if more users arrive buy more terminals;
– problem if LAN breaks, file server dies or pool burns.

- Flexible software configuration:

– file server eases maintenance;
– all the arguments against distributed systems;
– but OS must manage processor allocation.

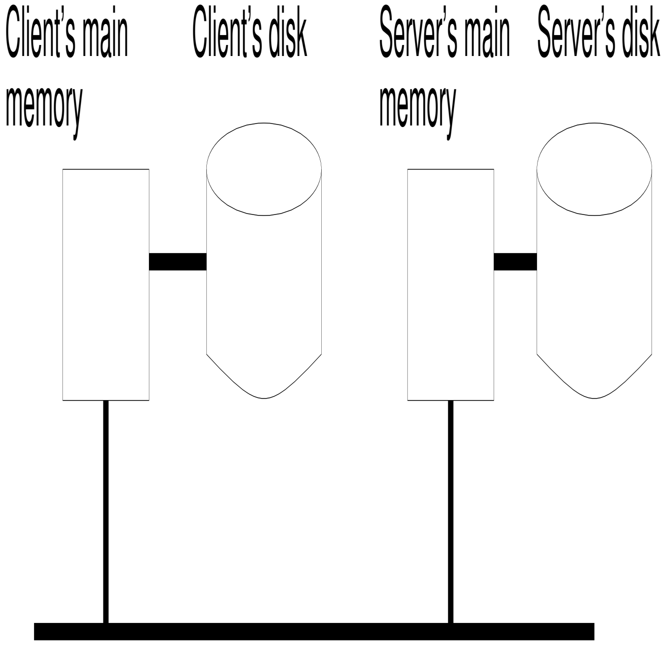
- Trade-off:

- flexible architecture easily changed, more processors...
- not robust as it centralises computing resources...

- Hybrid approaches increasingly popular.

File Servers

- Four places to store files.



- Affects efficiency through caching.

File Servers

- Changes to the hardware:

– more clients increases burden on server memory;
– more clients increases need for local memory.

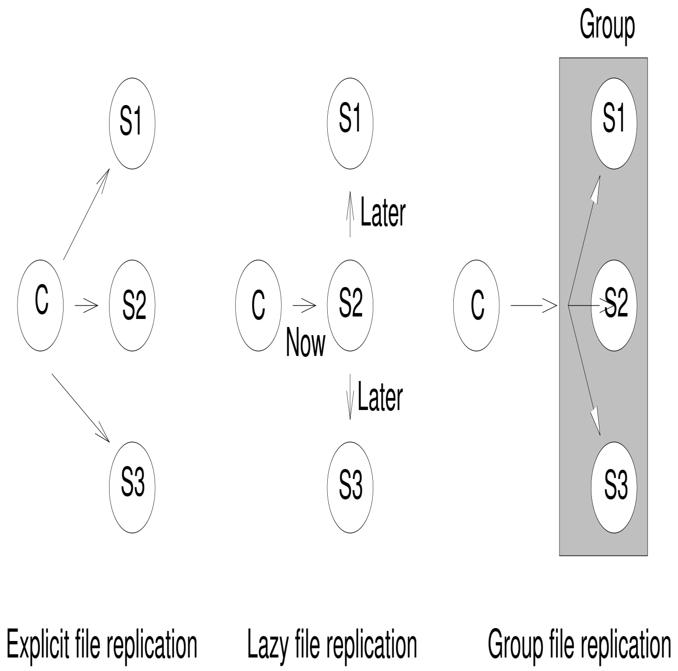
- Changes to software:

– greater traffic needs more Server memory for requests;
– greater traffic can saturate network resources.

- Caching eases burden of configuration changes.

File Servers

- Configuration at a high level.



- Affects consistency through replication.

File Servers

- Changes to hardware:

– explicit architecture affects client operating systems;
– lazy replication, S2 hides details from client;
– group replication, may hide details.

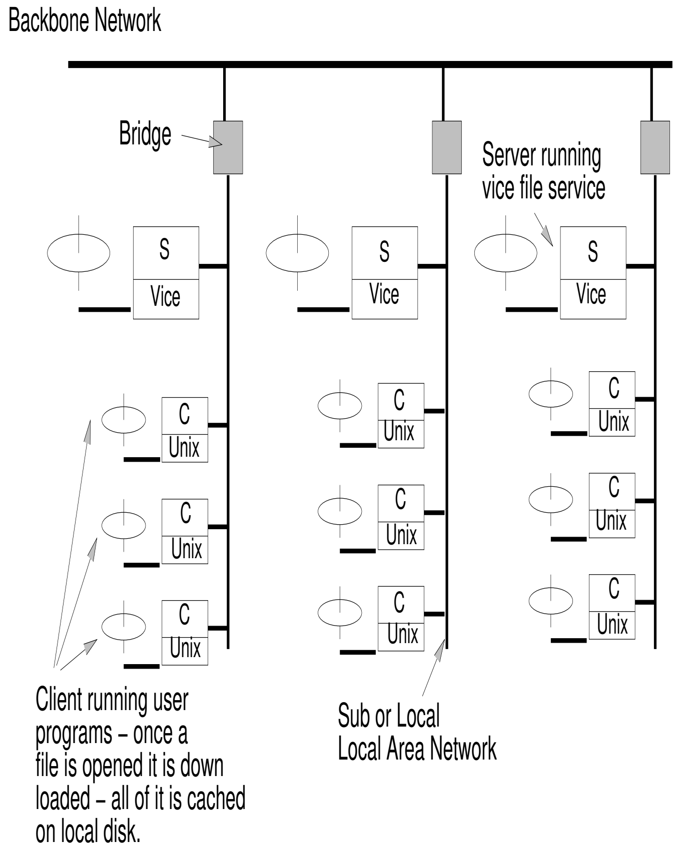
- Lazy configuration at S1 or S3 can cause problems - has client information passed through S2?

- Changes to software:

– explicit architecture creates overheads;
– lazy replication, what if S1 or S3 fails?:
– group replication, immediate update.

Putting It Together- Files & Processors

- Andrew File System at CMU



- Attempt to localise traffic over system.

Putting It Together- Files & Processors

- Software configuration:

– distributed amongst sub-LAN servers;
– centralised amongst sub-LAN clusters.

- Hardware configuration:

– addition of clients easy;
– addition of sub-LANs easy;
– up to 5,000-10,000 workstations!

- Security is a major headache - see later...

- All of this depends on good network configuration.

Conclusion

- Configuration management.
- Devices (processors and file servers).
- Networks.
- Users.

Introduction

- Configuration management.

- Devices (last time).

- Networks.

- Users.

What's The Problem?

- Configuration management.

- How do operating systems cope with change?

- Causes of change:

– work profile alters - heavier processing;
– the user profile may change - more of them;
– software may change - how easily can we update;
– hardware may change - how easily can we upgrade.

Networks

- Characteristics of configuration management
- operating systems depend upon network communication.

- Gather information about current configuration:
– what services/devices are current available to the system.

- How can we use this data to alter configuration:
– is the current network set-up optimal for services/demands?

- Produce reports based on the data obtained about
– network and any subsequent changes to configuration.

Networks

- Some typical tasks...

- Add another interface to a device on network:

– how many interfaces are there already - an it cope?
– what are the addresses of the interfaces?

- Errors propagating onto the network:

– check which interface is causing them;
– reconfigure network to disable that service;
– then check software set up on the device to find fault.

Networks

- Problem is that with 5,000 devices how do you do it?

- Automated support - Auto-discovery, Auto-mapping.

- 'Pings' elements on the network.

- Increases overheads and takes up bandwidth:

– alter check frequency-configure the configuration tool...

Networks

- Scale problem - 5,000 devices, say 1% need address changes or parameter modifications in a week, full time job!

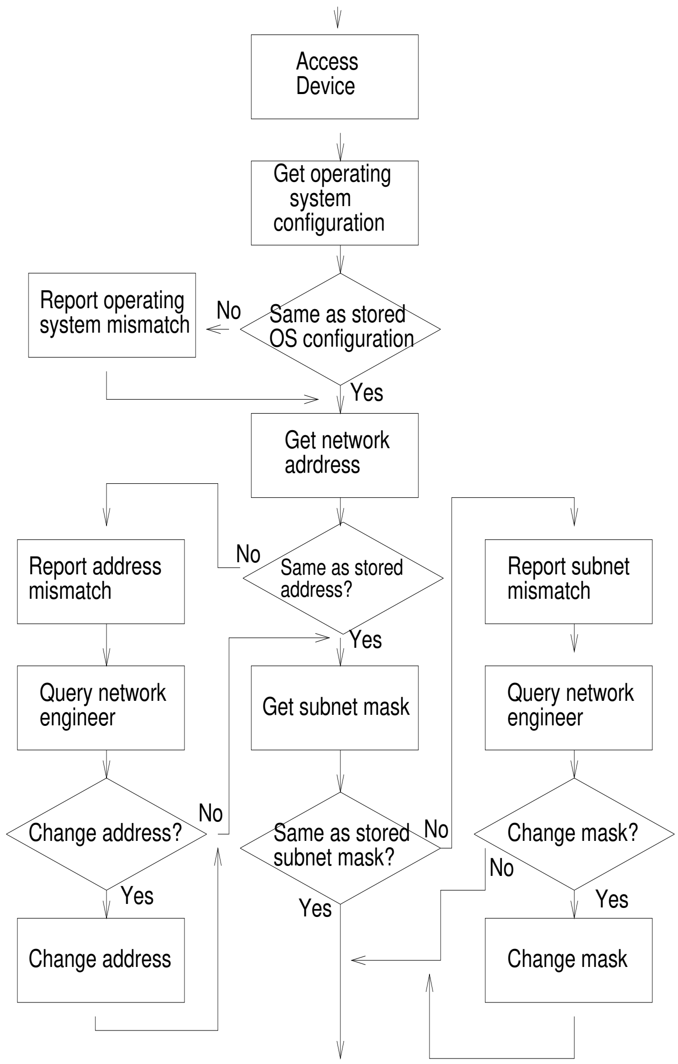
```
all bridges running version A of autoroute software
SELECT * FROM devices WHERE type = bridge and Software = autoroute A
```

Device Id.	Vendor	Serial #	Type	Software	Location
ENGRBR1	RouteIT	10921	bridge	autoroute A	ENGR
ADMINBR	RouteIT	10923	bridge	autoroute A	Admin
MKTGBR23	Bridger	A734-f	bridge	autoroute A	Bldg 10

- Basic information engineering techniques, but also interest in CASE based approached (DEC, BT etc).

Networks

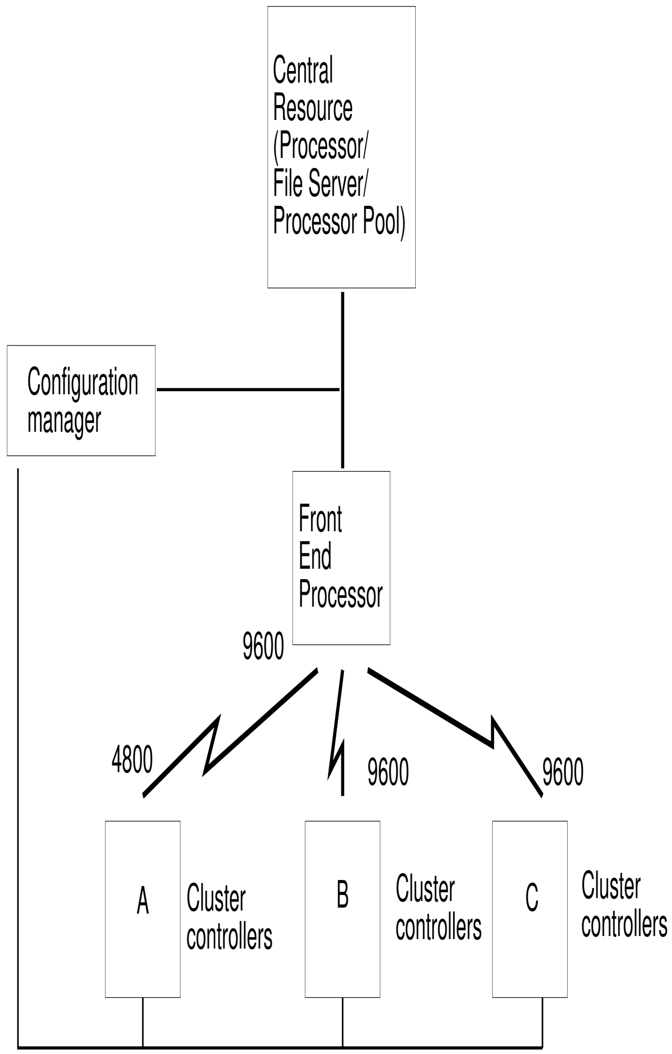
- Range of tasks for network configuration.



- Tool support essential...

Networks

- Case study - baud rate mismatch.



- Delays affect distributed algorithms...

- Complex problem, could be other causes?

Users

- What are the configuration problems:

– general issue of accountancy, quotas etc;
– gathering data, interpreting data.

- Three stages:

– gathering data about utilisation of resources;
– setting usage quotas from metrics;
– billing users for access and use.

Users

- Typical scenario:

– series of PC file servers;
– one is responsible for running payrole system;
– user downloads a copy of Doom onto the server;
– leaves no space for the payrole data;
– system manager arrives to find payrole failed;
– what do you do?

- Do you bill user for the time they used?

- Do you bill user for cost of failed system?

- Do you sack system manager for allowing problem?

Users

- Billing based on connection fee& monthly charge.

- Advantages:

– easy to implement and monitor, transparent;
– network resources not wasted on accountancy functions.

- Disadvantages:

– encourages intensive/unstructured use of resources;
– depends on good predictions of projected unit costs.

Users

- Billing based on per unit fee.

- What unit do you charge for:

– total number of transactions;
– total number of packets transferred;
– total number of bytes transferred.

- Key issue: bytes and packets in which direction?

Users

- Accounting per transaction.

- Advantages:

– simple to implement - count logins?;
– minimum of overhead and administration.

- Disadvantages:

– too coarse a measure, sustained logins;
– 5Mb ftp same as single e-mail message.

Users

- Accounting per packet.

- Advantages:

– cumulative in proportion to network usage;
– charge for net usage not processor time?

- Disadvantages:

– hard to implement, charge for packets sent or received?
– who pays if fault causes spurious or lost packets?
– doesnt reflect network usage - depends on size of packet.

Users

- Accounting per byte.

- Advantages:

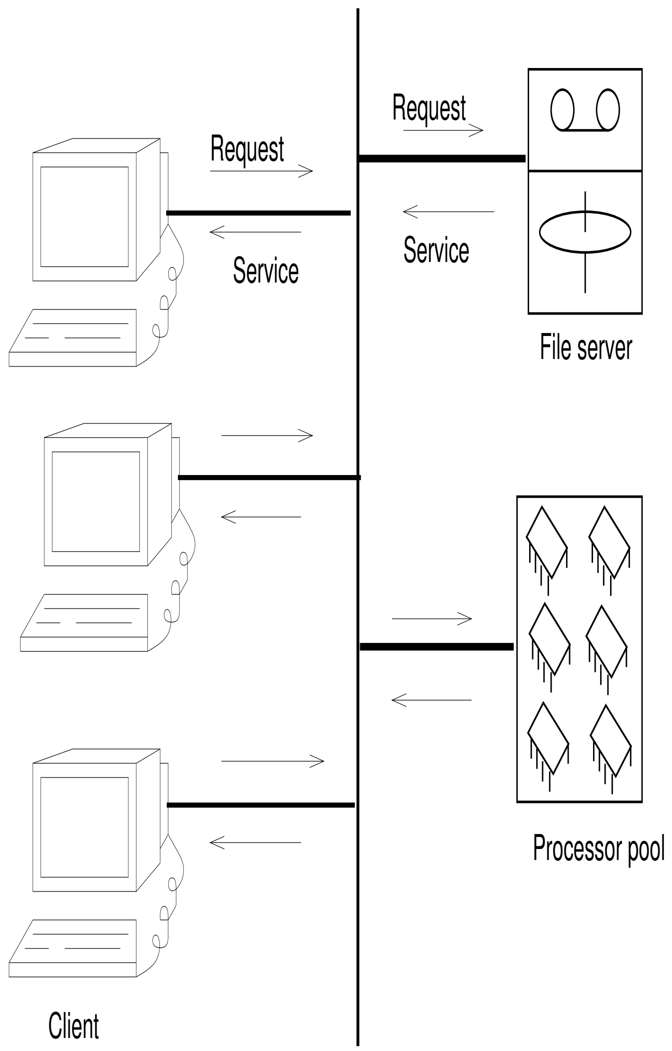
– cumulative in proportion to network usage;
– charge for net time not processor time.

- Disadvantages:

– significant administration overhead;
– do you charge for bytes sent or received?

Users

- Redundant to charge for both directions.



- Accounting per byte or packet.

Users

- Lots of other issues.
- What does the charge cover?
- Internet access relatively simple.
- EPSRC Cray T3D charge for time and technician.

Users

- Can access files without privileges:

– Internet Protocol networks (ftp) access;
– Appletalk shared folders;
– Novell fileservers unsecured directories.

- Why does these systems typically bill on receipt?

- What happens when you send the user the bill?

- What happens when someone sends you a binary?

Conclusion

- Configuration management.

- Devices (processors and file servers).

- Networks.

- Users.

Introduction

- Last time - configuration management.
- This time - how do we do it?
- Network Management Protocols.
- Standards for management tools.
- Management Information Base
- Common Management Information Services.

What's The Problem?

- Can only manage system if we know what's there...

- Two DECnet routers support access via rlogin:

– two different manufacturers, two different protocols;
– one via graphical tool, other by command line;
– can we build common configuration tools?

Ping

- TCP/Internet Protocol provides:

– Internet Control Message Protocol (ICMP);
– Echo and Echo Reply tell you what's connected.

- PING - (Packet InterNet Groper):

– uses ICMP echoes to test and time response;
– can also tell percentage of lost ICMP messages.

- Not just TCP/IP also:

– Appletalk; Novell/IPX; Xerox NXS.

- What are the limitations?

Ping

- Drawbacks of Ping:

– unreliable delivery;
– need for polling;
– limited infomration.

- Echo failure doesn't mean lack of connectivity,

– could be lack of buffer space on recipient.

- Basic problem:

– lack of information for distributed system management.

- International standards:

– Simple Network Managemet Protototocol (SNMP);
– ISO Common Management Information Services (CMIS).

Standards

- Queries to be supported:

– name of device;
– version of software in device;
– number of interfaces in the device;
– number of packets/second on interface to device.

- Settable device parameters:

– name of device;
– address of a network interface;
– operational status of network interface;
– operational status of the device.

Management Information Base

- But how do we record network information?

– need identifiers for objects and companies.

- Common access structures:

– three trees: ccitt, iso, joint.

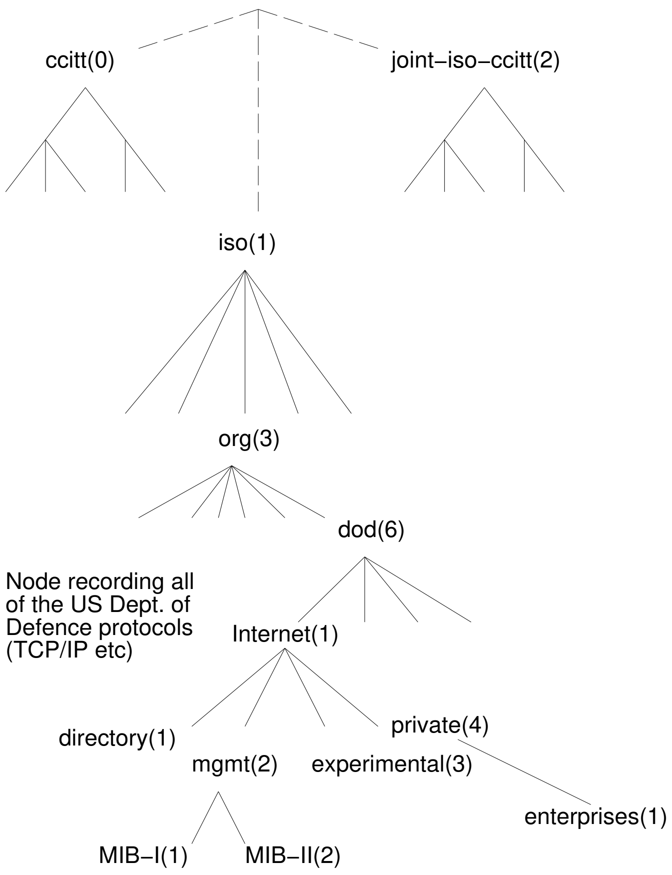
- Networks administered by different organisations:

– third node under iso is org node;
– sixth node under org is the dod node;
– first node under dod is Internet.

- Structure for network information.

Management Information Base

- Location of internet.



- MIB reference 1.3.6.1

Beneath Internet

- Directory (1) - reserved for future use.

- Mgmt (2) - see later.

- Experimental (3) - new protocols/information bases.

- Private (4) - company sepecific devices

– company given 1.3.6.1.4.22 identifier;
– new bridge might be given 1.3.6.1.4.22.1.

Internet Mgmt(2)

- Specific information about network devices.

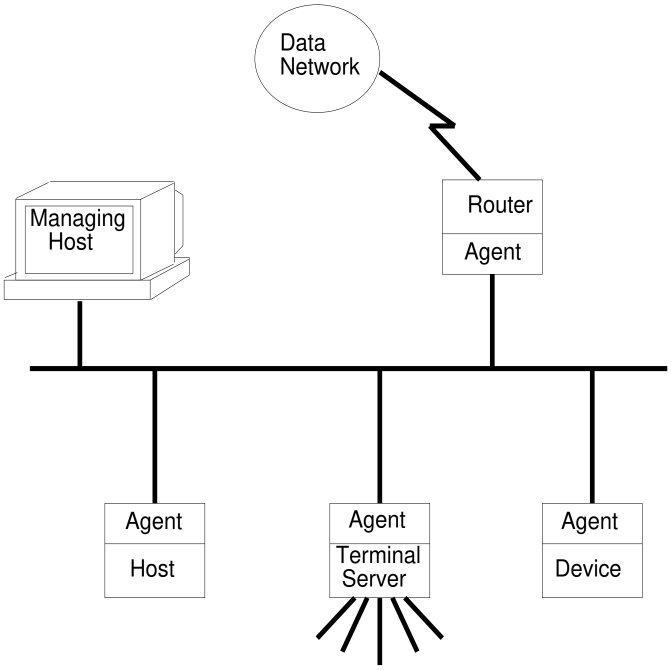
System (1)	Network Device Operating System
Interfaces (2)	Network Interface Specific
Address translation (3)	Address Mappings
IP (4)	Internet Protocol Specific
TCP (6)	Transmission Protocol specific
UDP (7)	User Datagram Protocol Specific
EGP (8)	Exterior Gateway Protocol Specific
CMOT (9)	Common Management Information
Transmission (10)	Services on TCP transmission protocol
SNMP (11)	Simple Network Management Protocol

- Catagories of specific network objects.

- Address translation(3) -IP addresses to Ethernet.

SNMP (11)

- At last a real system in the reference scheme!

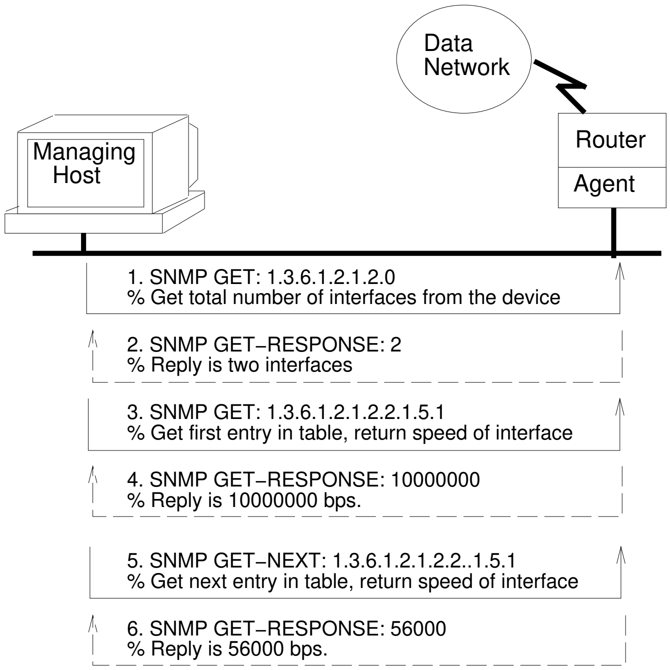


- Simple Network Management Protocol.
- Agents - software to answer SNMP queries.

An SNMP Dialogue

- Station sends request to device via agent.

`{ iso org(3) dod(6) internet(1) mgmt(2) mib(1) interfaces(2) ifNumber(1)}`



- SNMP to find speed of all interfaces on device.

SNMP Traps

- SNMP traps - unsolicitied messages from agent.

- Asynchronus changes to operating environment:

– coldstart of the system (changed configuration);
– warmstart of the system (no change to configuration);
– link down with network;
– link up with network;
– failure of authentication;
– exterior gateway protocol neighbour loss;
– enterprise specific event.

- Community strings:

– authentication with each station message;
– ASCII text so relatively easy to crack;
– changes being made to this area.

OSI CMIS

- Common Management Information Services.

- ISO reference model - 7 layers for all comms:

– CMIS does the same for distributed systems management.

- Provides following services:

– management association;
– management notification;
– management operation.

- Devices communicating at same ISO level

– are known as `peer open systems`.

CMIS Management Association

- Controls association between open peers.

- Control linking and unlinking of connection.

- Not the physical connections but registers
  – device connection with management server.

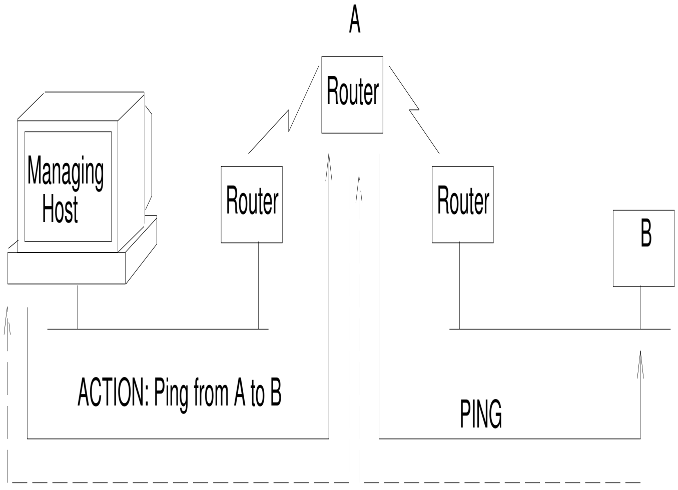
- At highest level of 7 layers.

CMIS Notification Services

- Similar to SNMP trap messages.
- Tell one CMIS service user of event at another.
- Recipient can in turn report event to server.
- Unlike SNMP the messages aren't defined.
- Specific to devices that generate the notification.

CMIS Operating Services

• One open system asks another to do something – for example, PING to test connectivity.



Managing host sends an ACTION request to machine A to send a ping to machine B, machine A sends back the result of the request, giving host information about network connectivity.

• Create, delete, get, set operations on objects.

CMIS Management Associations

- Object oriented architecture:

– define class of ethernet bridges;
– if you add another to the network then
– management tool adds instance to its records.

- Connect two peer services for management:

– event, just report events;
– event/monitor, report events and get information;
– monitor/control, get, set, create, delete, action, NOT event;
– full manager/agent, everything in CMIS.

Conclusion

- Network Management Protocols.
- Standards for management tools.
- Management Information Base
- Common Management Information Services.