Operating Systems (Four)

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

E-mail: johnson@dcs.glasgow.ac.uk
http://www.dcs.gla.ac.uk/~johnson
Telephone: 330 6053

February 1999.

Introduction

- Last time - network management protocols.
- This time fault tolerance.
- Transaction Processing.
- Process Control.

What's The Problem?

- We've covered some of this before.

- Distributed algorithms.

- Leadership elections if binder server fails.

- Byzantine generals if node sends 'bad' data.

What's The Problem?

- Two main scenarios.

- Transaction processing:

– can do roll-backs and locking;
– occasional failures/lengthy recovery time.

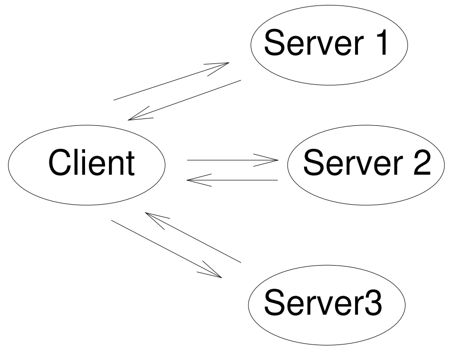
- Process control:

– can't roll-back a plane?
– recovery within very short time limit.

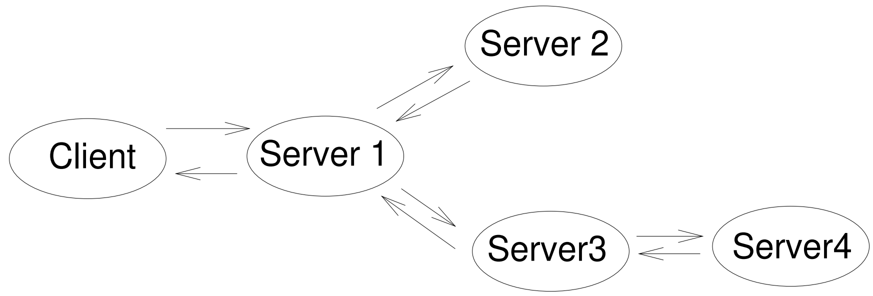
Transaction Recovery

- What are transactions

Simple Distributed Transaction



Nested Transaction



- Requests by client for operations on servers.

Transaction Recovery

- Transactions should be atomic:

– once committed no incomplete transactions;
– once committed no aborted transactions.

- Durability:

– save data items on permanent store.

- Failure atomicity:

– server can recover data even if it fails.

- So working data in volatile store

– but all committed data in a recovery file.

Transaction Recovery

- Usually implemented by a recovery manager:

– save data items for committed transactions;
– restores server's state after a failure;
– reorganises recovery file to improve recovery performance;
– reclaims store in the recovery file if necessary.

- Resilient to media failures, redundancy.

Transaction Recovery

- When transaction opened by client:

– server sets up and returns new transaction id;
– client sends transaction id to Commit or Abort.

Contents of recovery file:

Type of entry	Description of contents of entry
Data item	A value of a data item
Transaction status	Transaction identifier, transaction status (prepared, committed or aborted).
Intentions list	Transaction identifier and a sequence of intensions in the following form: <data item identifier> <position in recovery file of data value>

- Requests by client for operations on servers.

- Server records this in **intentions list**:

– names and values of all variables in transaction;
– when commit agreed this is used update values.

Logging Techniques

- Recovery file, history of all server transactions.

- Ordered by time of preparation, commit and abort.

- Snap shot of:

– the values of data items (at a checkpoint);
– history of transactions after the snap shot.

- Recovery manager append entry to log.

- Why does this have to be atomic?

Recovery With Logging

- Server resets data to default initial values.

- Recovery manager then:

– performs all committed transactions;
– finds data in intentions list of recovery file;
– what problems could arise?

- Server failure during recovery process?

- Must be idempotent: restor $\equiv$ restore(restore).

Shadowing

- Alternative way of organising recovery file:

– server's data items recorded in a versions store;
– a map shows current values in version store.

- When a transaction prepares to commit:

– place new versions of data values in version store;
– keep a record of the original values in case of abort.

- When a transaction commits:

– change the map to reflect new values in version store.

Recovery With Shadowing

- To restart:

– just use map to get values from the version store.

- Map always written to pre-determined place.

- Switch between maps must be atomic.

Comparison

- Shadow version faster recovery that logging:

– position of data accessed directly by map;
– log recovery involves searching through transaction.

- Logging faster to build than shadowing:

– logs built by series of simple appends on file;
– shadowing has two unrelated writes, location & value.

Process Control And Fault Tolerance

- In transaction based systems:

– you can recover despite a delay while state restored.

- In process control systems:

– you want to keep running even if failures occur.

- So systems must be fault tolerant.

Fault Tolerance

- Hierarchical masking:

– failure at one level hidden at higher levels;
– retry a request-reply protocol;
– if low level problem cannot be masked then exception;
– carries on up hierarchy until you reach the user.

- Exceptions translated up the hierarchy.

- User may not understand low level failure.

Fault Tolerance

- Group failure masking:

– provide a service by using a number of servers;
– if one fails the rest continue - Byzantine generals.

- t-tolerance - works if no more than t fail:

– 2t+1 required to mask t failures;
– t+1 must agree during a vote.

Close Synchronisation

- Replication:

– all servers start in same initial state;
– all servers receive same requests;
– all perform the corresponding actions.

- Must be deterministic - why?

- Close synchronisation:

– all members execute all requests immediately;
– client request to all servers by replicated procedure call;
– must be at most once semantics - why?

Relaxing Constraints

- All Servers dont need to perform all operations.
- Read only operations aren't repeated.
- Client only needs one reply, rest are the same.
- Or will they?
- This assumes a fail-stop architecture.

Loose Synchronisation

- One primary server responds to client.

- Collection of back-up servers:

– logs client requests;
– if primary server fails then backup takes over;
– executes all of the requests in the log.

- Comparison:

– loose synchronisation consumes less processing resources;
– only primary server responds to clients requests;
– but tight synchronisation provides faster response;
– no need to re-build server state from log.

Conclusion

- Transaction Processing:
  - failure atomicity;
  - gradual recovery possible;
  - logging and shadowing mechanisms.

- Process Control:
  - fault tolerance;
  - rapid response;
  - tight and loose synchronisation.

Introduction

- Security and resource protection

- Different threats.

- Components of security:
  - cryptology and authentication;
  - access control.

Case Study 1

- 1988, Donald Gene Burleson, Forth Worth, Texas.

- Leaves a virus on employers system:

– deletes 168,000 records on sales commission;
– affects 400 employees pay.

```
Charged under Texas State Law; felony to:
``intentionally and knowingly cause a computer to
malfunction or interrupt the operation of a computer
or alter, damage or destroy a computer program, stored,
maintained or produced by a computer without the consent
of the owner''.
```

- Seven years probation, plus civil damages.

Case Study 2

- Robert T. Morris Jr. - Internet Worm.

```
``It's now 3.45am on Wednesday 3 November 1988.
I'm tired so don't believe everything that follows...
Apparently there is a massive attack on UNIX systems
going on right now...  this may be a system wide
problem.  Symptom:  hundreds or thousands of jobs''
(Cliff Stoll, dockmaster.arpa)
```

- Why is this worse than case study 1?

Case Study 2

- Robert T. Morris Jr. - Internet Worm.

- Caused damage estimated at $10,000,000.

- Convicted, US Computer Fraud And Abuse Act, 1986:
  – didn't cover worms, found guilty of time stealing;
  – Computer Virus Eradication Act, 1990?

- Not a virus, didn't attatch to another program.

Case Study 2

- Attack based on remote parameter passing.

- Passed string longer than utility buffer.

```
P.Fites, P. Johnston, M. Kratx, The Computer Virus
Crisis, Van Nostrand Reinhold, New York, 1992.
```

- Excess executed remotely and allowed worm access.

- This carried second attack on password file.

Introduction

- Why is security an increasing problem?

- Technological factors:

– interconnection of global networks;
– distributed processing;
– remote procedure calls.

- Economic factors:

– companies exploit technology;
– distribute information to point of need;
– centralised logging and computation.

- Any others?

Categories of violation

- Anderson's categories.

- Unauthorised information release:

– unauthorised person accesses data;
– includes unauthorised use of a program.

- Unauthorised information modification:

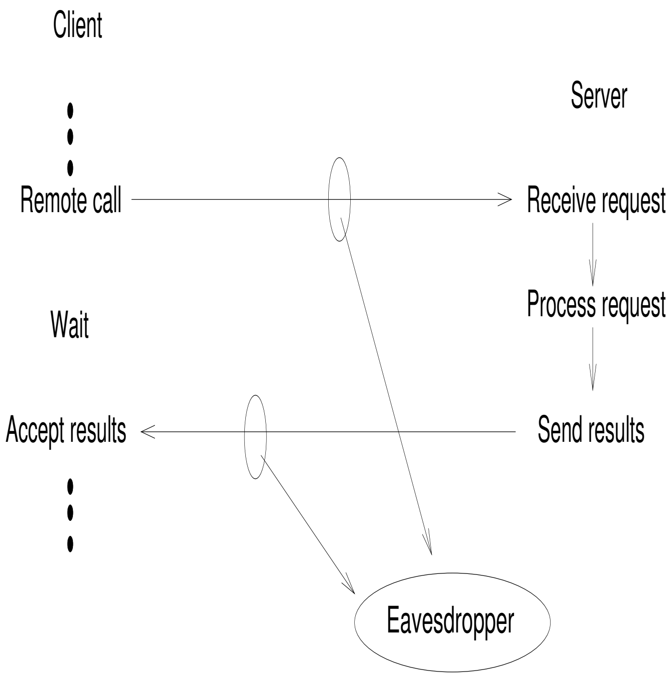
– changing marks, bank accounts, police records etc;
– blind writes are included - don't have to read file.

- Unauthorised denial of service:

– prevents real user from accessing system;
– crashing the system, alter privileges/scheduling.

Attack Methods

- Eavesdropping.

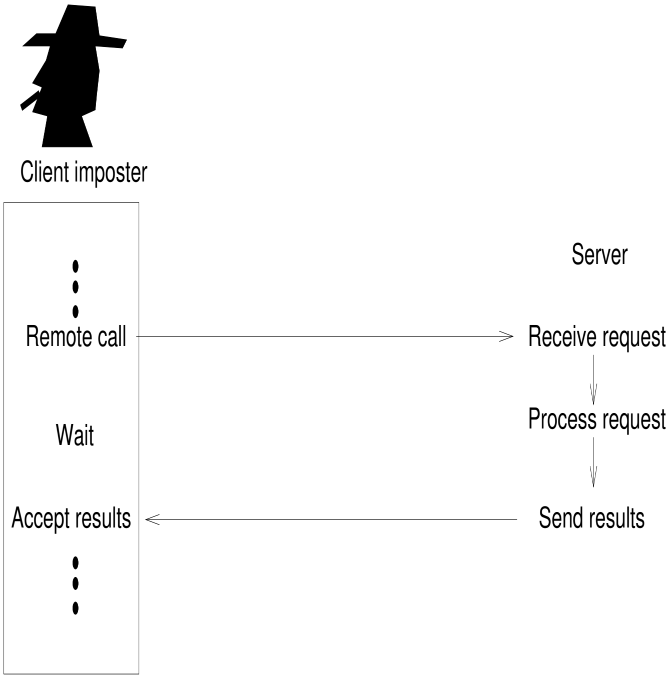


- Obtain copies without authority:

– reset address of workstation on Internet.

Attack Methods

- Masquerading.



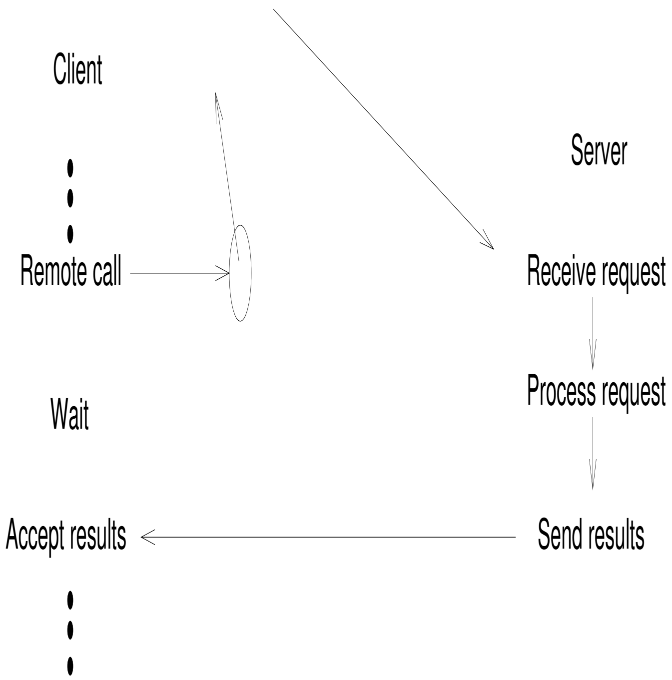
- Pretending to be someone else:

– password cracking, access privileges.

Attack Methods

- Message tampering:

– altering contents of messages before passing them on;
– easy in store & forward nets, hard in broadcast Ethernet.



- Replaying:

– store messages and play at later date;
– resource stealing even with encryption etc.

Infiltration

- Need access in order to attack.

- Most attacks are from authorised users.

- Illegitimate users:

– crack passwords;
– develop viruses via networks or physical store;
– worms, exploit facilities for running remote processes;
– Trojan horses, hides real function of system.

Design Principles

- What are the objectives?

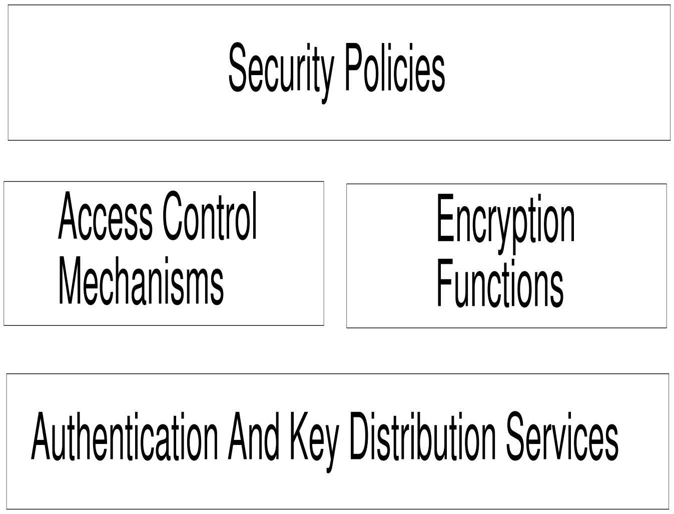
– *economy*, must still be able to use system;
– *complete mediation*, all accesses must be checked;
– *open design*, still workers if attackers know about it;
– *separation of privileges*, two keys work better than one;
– *least privileges*, give minimum privileges for task;
– *least common mechanism*, cut paths between users;
– *acceptability*, must be simple to use;
– *fail-safe defaults*, if in doubt deny access.

- Some of these contradict each other:

– a completely safe system would not be acceptable.

Security Implementation

- Cryptology and authentication.



From Coulouris et al, p.485

- Access control.

Conclusion

- Security and resource protection

- Different threats.

- Components of security:

– cryptology and authentication;
– access control.

Introduction

- Access abstraction: access matrix.

- Access control list.

- Capabilities.

- Comparisons.

Access Control

- Some of this covered in first part of course?.

- Resource protection is service specific:

– some measures taken at Kernel level;
– memory management, hardware support;
– others at higher level of service;
– detect arbitrary messages from network.

Access Matrices

- Instead of a list for each process.

- There is a matrix.

	Object1	Object2	Access Matrix
User 1	read,write	own, delete	none
User 2	execute	copy	none
User 3	own	read, write	none

Simplifications: users may be processes, processes may 'own' other processes. Negative priviledges need not be explicitly represented.

- Security state = (S,O,P), P is access matrix.

Implementing The Access Matrix

- Access matrices will be sparse.

- Many users cannot access many objects.

- lots of `none` values.

- Only represent capabilities or access rights.

Access Control List

- Can be thought of as column in matrix.

For each object in the system...

Subject	Access Rights
daw	read, write, execute
johnson	read
...	...
...	...
wf	read

- Maintain a list of client's privileges:

– search list each time they access an object;
– raise an exception if no permission.

Strengths And Weaknesses Of Control List

- Good points:

– revocation - remove subject from object's control list;
– review - use list to find all access privileges.

- Bad points:

– search - slow to search list on each operation;
– storage - lots of space - so list by groups of users?

- Who can alter the control list?

– owner process - has special access right;
– hierarchies - object owner grants it to process group.

UNIX And Access Control Lists

- Each resource has a list of (subjects, rights).

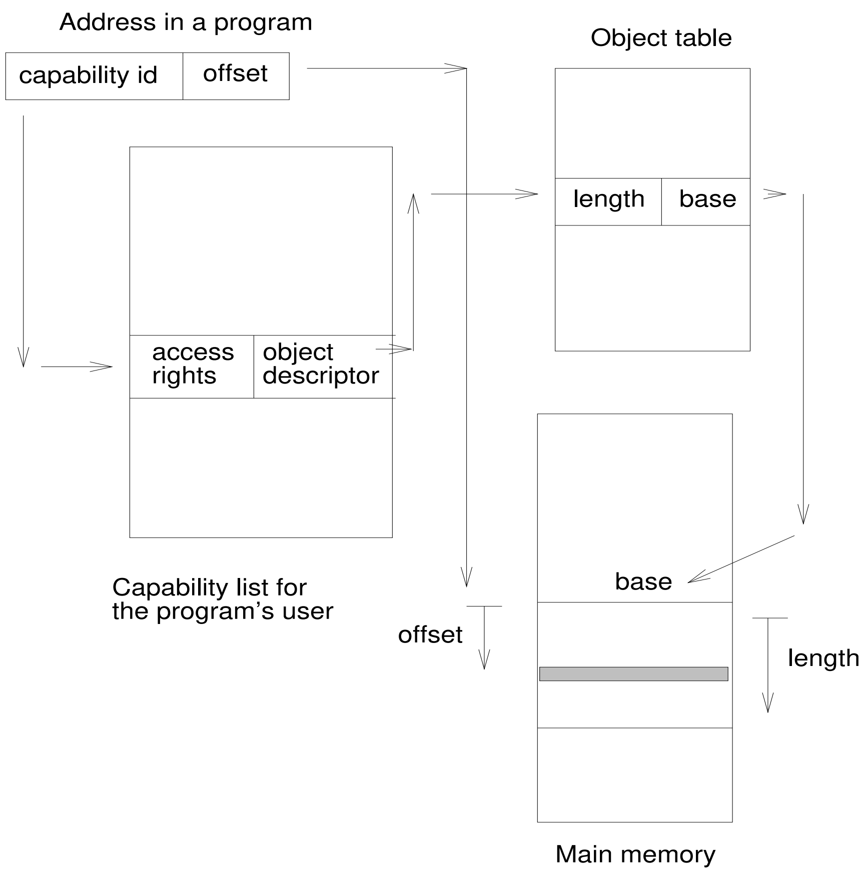
- These apply within that domain.

- In UNIX:

– the list is identified by user/group privileges;
– ie this process can write file if it belongs to user;
– the rights are the read/write/execute authorisation.

Capabilities

- Can be thought of as a row in the matrix.



- Access to resource granted if subject has key.

Capabilities

- User program issues request for word within 'object'.

- Address of request has capability ID of object.

- System uses ID to search list of user:

– checks there is a capability with the object desciptor;
– checks that the capability has access rights.

- Relocatability:

– only base field needs to be changed in object table;
– the capability list doesn't need to be changed.

- Sharing:

– programs have different descriptors for same object;
– different descriptors to same entry in object table.

Capabilities

- This approach used for file security:

– Unique file identifiers may contain key;
– covered UFIDs in first part of course?

- Client presents UFID in order to access file:

– don't need to know the client's identity;
– if the client knows the UFID then it must be ok.

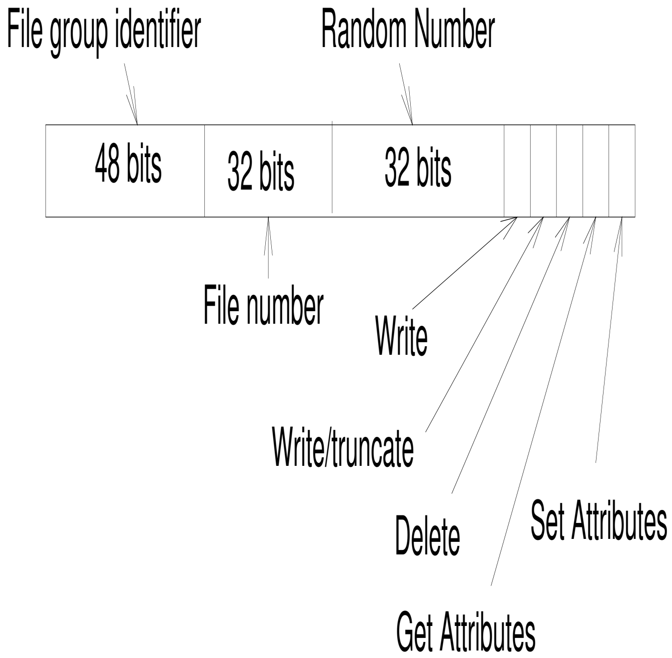
- So how do we generate UFIDs?:

– if the file is deleted we can't re-use the UFID;
– client might access new file by re-using old UFID;
– cannot include address in case file is moved;
– must be difficult to forge.

Capabilities

- Use an integer file identifier:

– increment each time so don't re-assign same UFID.



- UFID includes:

– a file group identifier to help find it;
– a random number so we can't guess a UFID;
– access bits and encrypt them (next lecture).

Strengths And Weaknesses Of Capabilities

- Good points:

– efficient - access is valid if key presented;
– simple - use the key to locate the object as well;
– flexible - put other information in key as well.

- Bad points:

– propagation - subject may pass key to other processes?;
– review - difficult to work out all clients with key;
- revocation - how do you prevent access once granted?
- garbage collection - lose key and object becomes garbage.

Conclusion

- Access abstraction: access matrix.
- Access control list.
- Capabilities.
- Comparisons.

Introduction

- Last time: control lists and capabilities.

- Hybrid techniques:
  - lock and key model;
  - take and grant model;
  - military model.

- Case studies: HYDRA and Andrew.

Recap

- Access matrix.

	Object1	Object2	Access Matrix
User 1	read,write	own, delete	none
User 2	execute	copy	none
User 3	own	read, write	none

Simplifications: users may be processes, processes may 'own' other processes. Negative priviledges need not be explicitly represented.

- Capabailities are rows in the matrix.

- Access lists are columns.

Hybrids - The Lock-Key Method

- Features of capabilities and access control.

- Subjects have capability list:

– can_use(object, key).

- Objects have access list of locks:

– can_access(lock, access_mode).

- Easy revocation by removal of key.

- Still inefficient as searches on both lists.

- Used in old IBM360 and ASAP file systems.

Hybrids - Take And Grant Model

- Directed graph models access control.

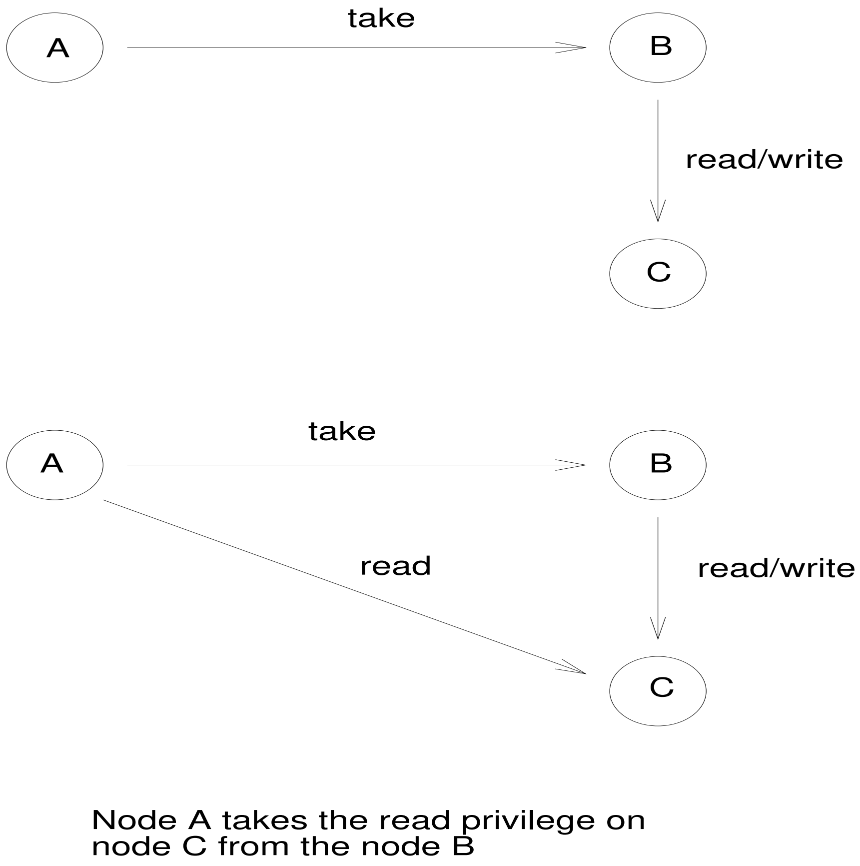
- Nodes are either objects or subjects.

- Edge if subject/object accesses other subject/object:

– edges labelled with access mode;
– read/write/execute/**take**/**grant**;
– take and grant control access propagation.

Hybrids - Take And Grant Model

- Taking access.

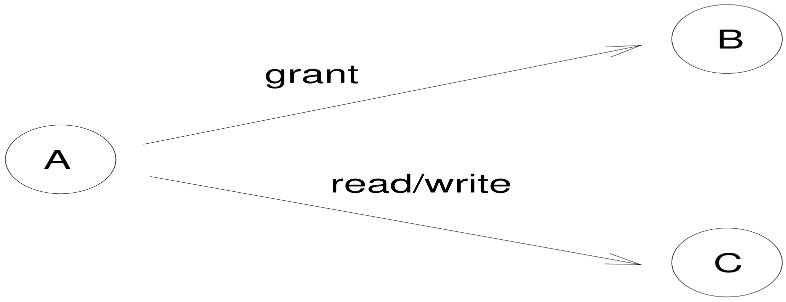


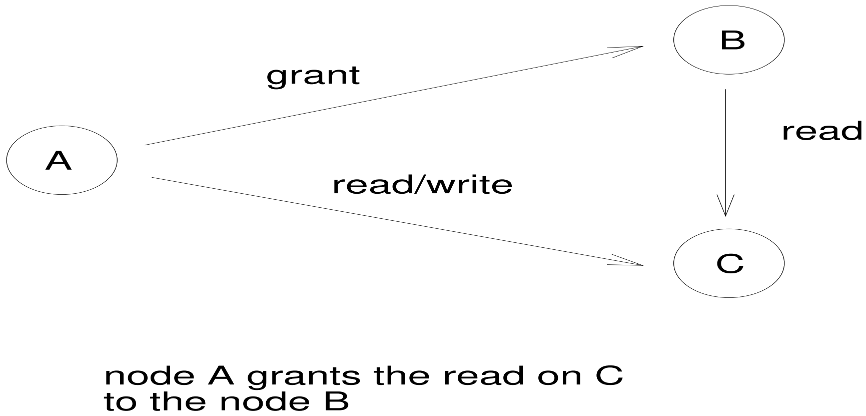
- If one node can take another then

– it can pass access rights to any other node.

Hybrids - Take And Grant Model

- Granting access.





- If one node can grant access to another
  – then it can pass any of it's right to second node.

Hybrids - Military Security Model

- Objects ranked as follows:

- $unclassified \leq confidential \leq secret \leq top\ secret$

- Compartments used to enforce need to know rule:

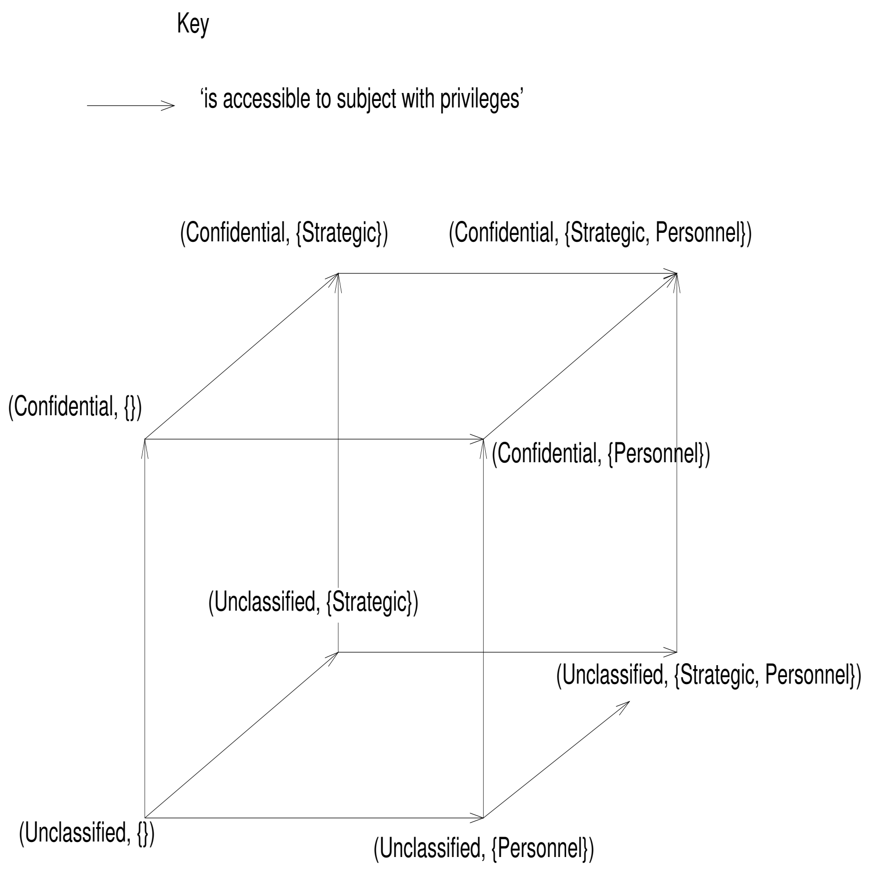
– for instance ‘personnel’ or ‘strategic’;

- Subjects granted access to objects in compartments:

– subjects with access to personnel compartment may not
– be able to access objects in the strategic compartment.

Hybrids - Military Security Model

- The whole thing forms a lattice.



- Must protect against insecure information flows.

Case Studies: HYDRA

- Carnegie Mellon's multiprocessor operating system.

- Uses capabilities to support range of models:

– could build military model, take-grant etc.

W. Wulf and E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, F. Pollack, 'HYDRA: The Kernel Of A Multiprocessor Operating System, Communications Of The ACM, June, 1974.

- Objects are abstractions of resources:

– unique type, name and representation;
– representation = data + capabilities;
– data accessed by objects that have rights;
– capabilities refer to access on other objects.

Case Studies: HYDRA

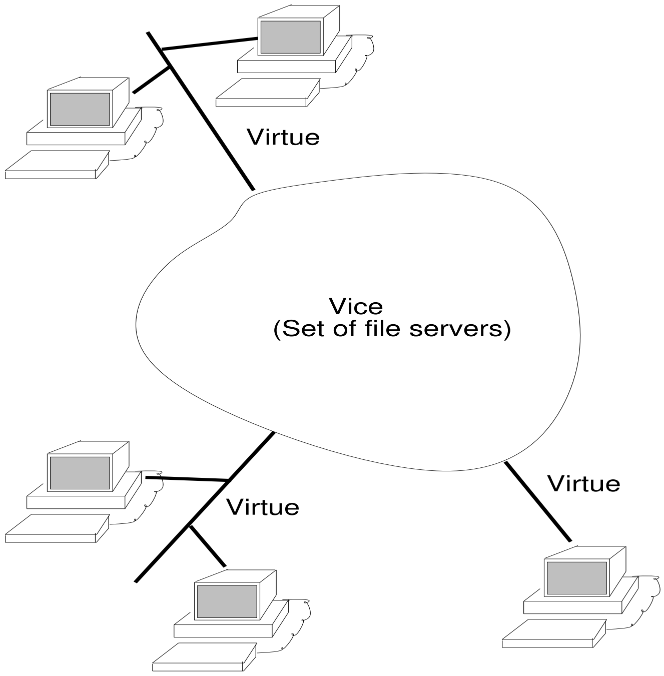
- Program presents capability for an object:
– kernel examines access rights in the capability;
– operation only allowed if rights are there.

- Kernel rights - standard read/write/execute/copy.

- Auxiliary rights - user defined, kernel provides
– 24 bit mask to encode access rights - like a key.

Case Studies: Andrew

- Again Carnegie Mellon but distributed.



- LAN - Ethernets and IBM token rings.

- Vice maintains shared files and protection.

Case Studies: Andrew

- Groups are collections of groups and users.

```
M. Satyanarayanan, Integrating Security In Large
Computer Systems, ACM Transactions On Computer
Systems, August, 1989.
```

- Users Current Protection Subdomain:

– inheritance of membership and of rights;
– transitive, reflexive closure of user's membership;
– U is a member of G and a member of all groups in G.

- Implemented using access control list mechanism:

– Universities - simple mechanism and complex protection;
– revocation of privileges is common in Universities!

Case Studies: Andrew

- Access list maps user or group to:

– two other lists, positive and negative rights;
– in each list there are sets of rights;
– these are represented by a 32 bit mask.

- Contradiction if both positive and negative rights:

– assume that denial overrides possession.

- Revocation easy by entering negative right:

– then you have to search distributed system and remove
– user from all groups that provide access to resource.

- Granting of access at lower level of granularity:

– grant this to all members of group G except user U.

Case Studies: AMOEBA

- How to protect server capabilities?
- Could be forged by ‘intruder’?
- Solution is to encrypt information.
- Valid clients must be able to issue requests.
- Public key cryptography, next time...

Conclusion

- Last time: control lists and capabilities.

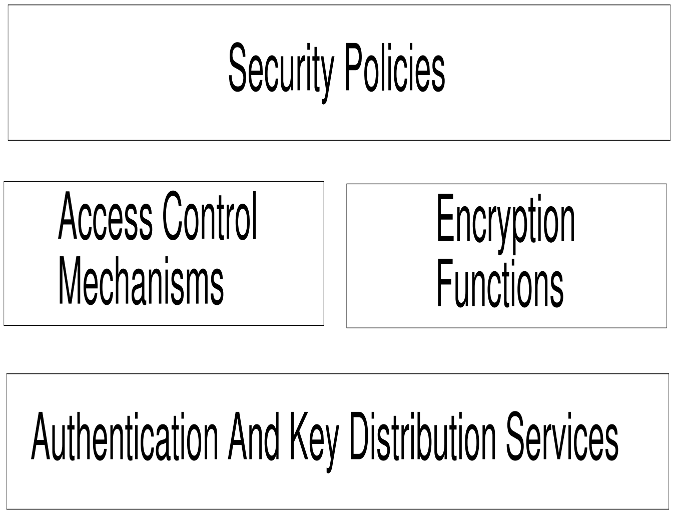
- Hybrid techniques:

– lock and key model;
– take and grant model;
– military model.

- Case studies: HYDRA and Andrew.

Introduction

- Previously access control.



From Coulouris et al, p.485

- This time: cryptology.

Cryptology

- Send information so that only the intended
– recipient can understand or use it.

- Plain text $\rightarrow$ cipher text.

- Shannon's confusion principle:
– must be no obvious relation between input and output.

- Shannon's diffusion principle:
– changes should be dispersed throughout the cipher.

Standards

- U.S. Government:

– Orange Book criteria from National Security Agency.

- Degrees of security.

- At level V2:

– a system guarantees that any object can be made
– secure at the discretion of its creator.

- Standards impose limits on security.

- Public policy to decrypt messages between citizens?

Architecture

- Server sends cipher to a client.

- If there are a number of clients:
  - different forms of encryption must be used;
  - otherwise clients might eavesdrop on communication.

- How do you communicate encryption algorithm?

- Assumes secure communication channel.

Problem

- Encryption rules must be changed if compromised:

– could have to generate huge number of rules.

- So most technique use a function and a key.

- Function is the encryption algorithm:

– add the key to the data in some way then
– apply the function so that the result
– cannot be guessed without knowledge of the key.

- If security compromised, change key not function.

Attacks

- Two forms of attack - on the key or the message.

- Given encrypted message, find plain text:

– this is the ultimate objective.

- Given encrypted message, find key:

– if you want to keep decoding message.

- Need a secure key distribution service.

Traditional Technique

- Caesar's cipher:

– shift letters N places in the alphabet;
– $encrypt\_position := (original\_position + N)\ mod\ 26$.

- Julius loves Cleopatra transforms into

– mxolxv oryhv fohrsdwud.

- Simple to encode, simple to decode.

- Simple to automate the cracking process.

Secret-Key Encryption

- What you'd expect - keep key secret.

- Computers simply increase the size of possible

– keys and transformation functions available.

- Encrypt by applying function to plain text and key;

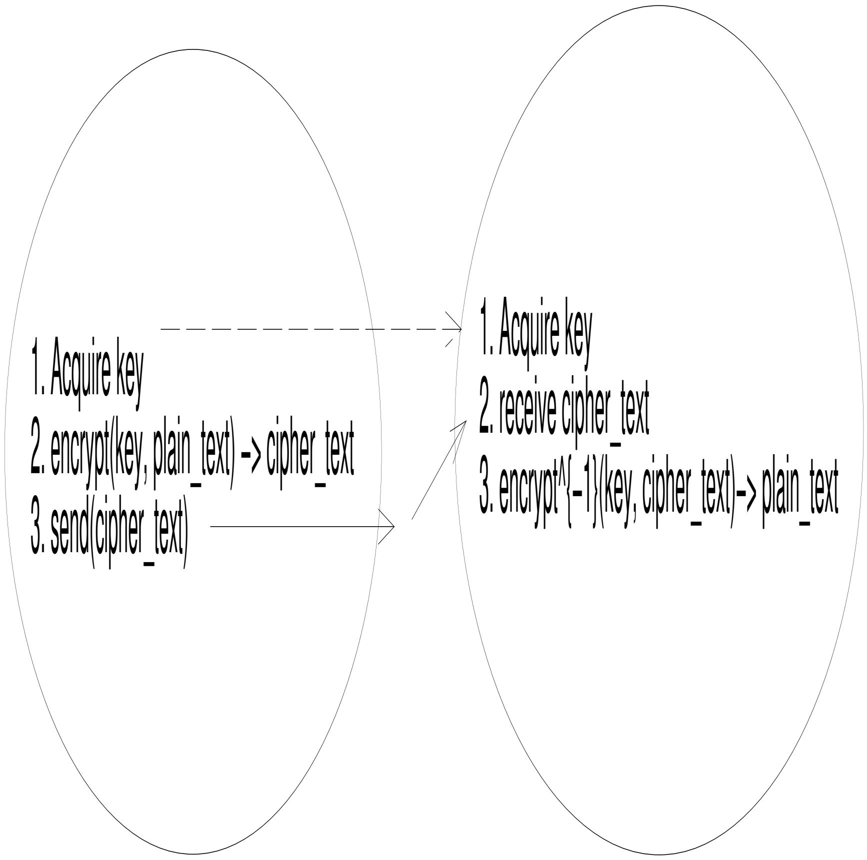
– decrypt using inverse function and extract key.

- Encryption and decryption functions are public

– because the keys are kept secret.

Secret-Key Encryption

- Both parties must acquire the key.



- Encryption function can be its own inverse:

– simple without loss of security through key.

Secret-Key Encryption

- Data Encryption Standard (DES)

- Developed by IBM - government standard:

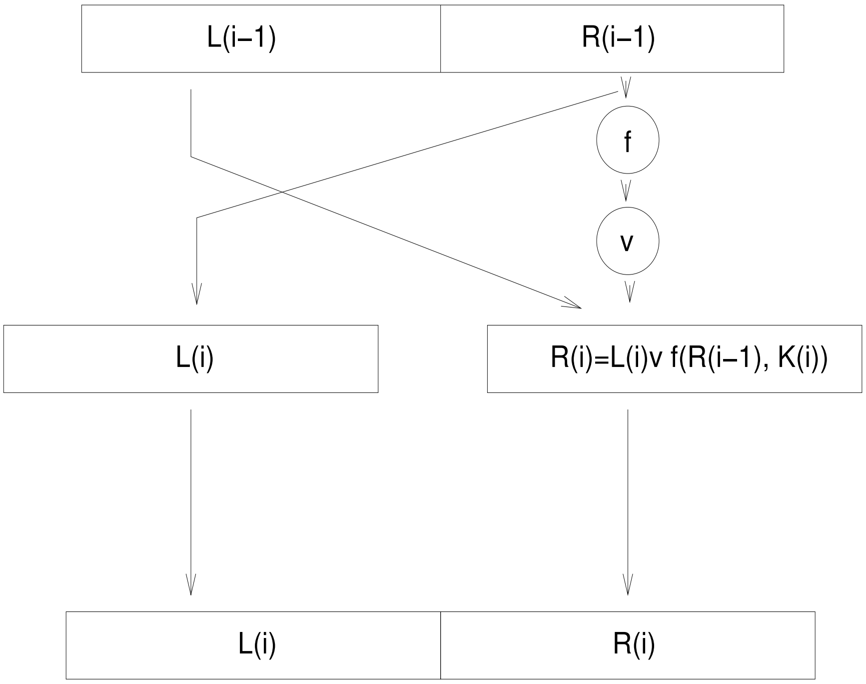
– 64 bit plain text + 56 bit key $\rightarrow$ 64 bit cipher;
– key expanded to 64 bits to allow for parity check;
– 16 rounds during which the bits are rotated;
– details of rotation dependent on the key;
– plus 3 key independent transpositions.

- No known successful attack but weakness:

– 56 bit key yields 2^{55} or 3×10^{16} trials.

Secret-Key Encryption

- L_i - left half of cipher after ith iteration.



v denotes exclusive OR

function f computers substitution

- K_i - 48 bits derived from key and iteration number.

Public-Key Encryption

- Problem: need to keep secret keys a secret.

- You need a secure communications channel.

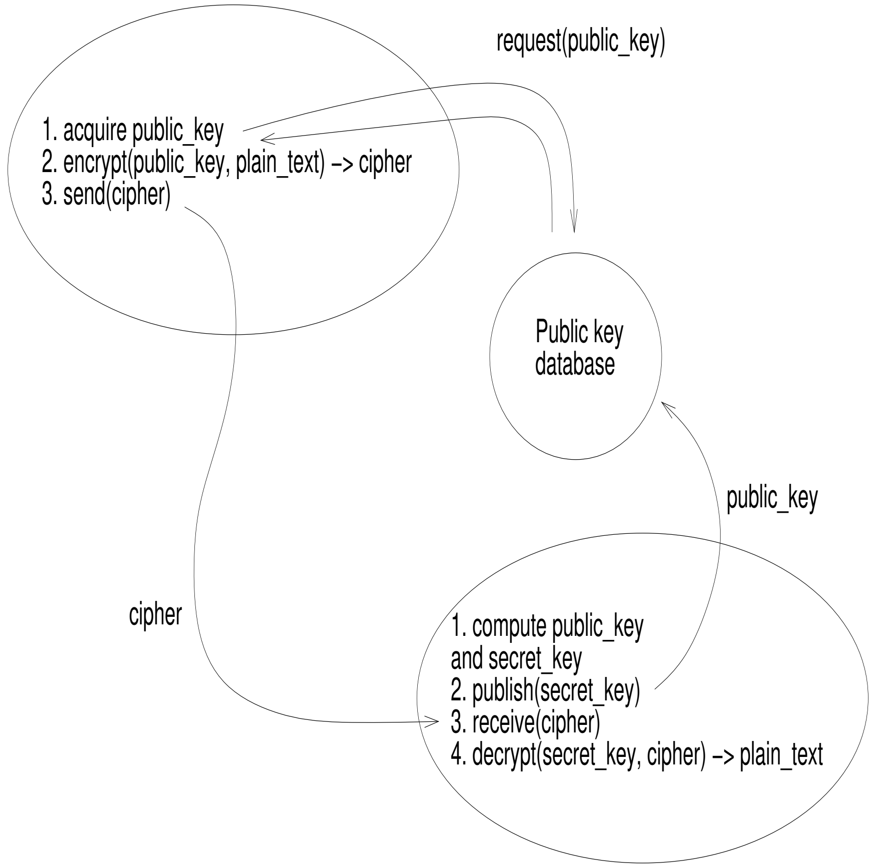
- Traditionally done using separate physical link.

- Not practical with most computer networks.

Public-Key Encryption

- Two keys, server sends out public one;

– keeps the decryption key a secret at local site;
– cannot guess decryption key from encryption key.



- Key sent to likely clients and public databases.

Public-Key Encryption

- Rivest-Shamir-Adleman Algorithm

- Encrypts integers:

– binary plaintext divided into blocks between 0 and (N-1);
– key is (e, n) where e is a positive integer;
– $block\_cipher = block\_number^e \ modulo \ N$.

- To decrypt $block\_cypher$:

– decryption key is (d, n) where d is a positive integer;
– $block\_number = block\_cipher^d \ modulo \ n$.

- How do we generate $(e, n), (d, n)$?

– pick any two large prime numbers $n = p \times q$;
– you are free to choose any large integer for d;
– then $e \times d = 1(modulo(p - 1) \times (q - 1))$.

Public-Key Encryption

- Signing messages.

- Could use secret key to encrypt a message.

- Use public key to decrypt the message.

- This enables sender to sign message
– nobody else has the secret key?

Public-Key Encryption

- Apple's System 7 pro:

– digital signatures using the RSA algorithm;
– public & private password issued on joining net;
– to 'sign' document drop it in DigiSign program;
– signature attatched to documents resource fork.

- IBM's object request brokers:

– use cryptology to protect objects;
– checks senders identity using public keys;
– Windows object dispatcher does this as well.

Public-Key Encryption

- So are there any problems?

- As key is public anyone could send encrypted message:
– some form of authentication needed for source of message.

- Vice versa - clients must know servers:
– imposter sends out their own key and intercept messages

Comparisons

- Security:

– no successful attacks known on either approach.

- Convenience:

– no secure channel required in public key encryption;
– but authentication required to check sender.

- Performance:

– secret key encryption is much faster;
– comparisons depend on optimisations (hardware etc).

- Hybrids:

– use public key encryption to pass secret key;
– then rely on secret key to speed transmission.

Conclusion

- Encryption.

- Secret Key techniques:

– Data Encryption Standard (DES)

- Public key techniques:

– Rivest-Shamir-Adleman Algorithm

Introduction

- Active redundancy.

- Passive redundnacy.

- Case study, Taragon/32:

– 'anti-leadership' elections;
– process replication;
– inter-process communication;
– timing and replica determinism.

- Next time: MARS case study.

Redundancy

- Physical resource redundancy:

– hardware devices replicated and ready to take over;
– (e.g., stand-by processors).

- Information redundancy:

– extending software to provide checks;
– (e.g., check digits to avoid corruption).

- Time redundancy:

– repeat operation to ensure success;
– (e.g., re-transmit a lost message).

- Active vs passive techniques.

Active Techniques

- Also known as hot redundancy;

– example MARS operating system (next time).

- All available resources are active:

– group systems into fault-tolerant units.

- Then we have to use Byzantine generals etc

– to arbitrate against split/erroneous decisions.

- All subsystems achieve same states simultaneously:

– replica determinism - if not then inconsistencies arise.

Passive Techniques

- Also known as cold or standby redundancy.

- Extra resources only activated after failure.

- Error must be detected before output

– passed to the user, could be disastrous.

- Must be *fail-silent*:

– a crash failure when the system stops;
– an omission failure when no output is produced.

- Error must be detected:

– watchdog signals help monitoring.

Passive Techniques

```
-- Author:  C.W. Johnson
-- Address:  Dept.  of Computer Science,
-- Univ.  of York, York, YO1 5DD, U.K.
-- E-mail:  johnson@minster.york.ac.uk
-- Last modified:  23/2/93

generic
   type ITEM is private;
package GENERIC_STACK is
   type STACK is private;

   procedure COPY(FROM: in STACK; TO: in out STACK);
   procedure CLEAR(A_STACK: in out STACK);
   procedure PUSH(I: ITEM; A_STACK: in out STACK);
   procedure POP(A_STACK: in out STACK);
   function IS_EQUAL(S1, S2:  STACK) return BOOLEAN;
   function DEPTH_OF(A_STACK: STACK) return NATURAL;
   function IS_EMPTY(A_STACK: STACK) return BOOLEAN;
   function TOP_OF(A_STACK: STACK) return ITEM;

   OVERFLOW, UNDERFLOW: exception;

private
   type NODE;
   type STACK is access NODE;
   type NODE is
      record
         AN_ITEM: ITEM;
         NEXT: STACK;
      end record;
end GENERIC_STACK;
```

- Exceptions within package specification.

Passive Techniques

- Raise exceptions in package body.

```
-- Author:  C.W. Johnson
-- Address:  Dept.  of Computer Science,
-- Univ.  of York, York, YO1 5DD, U.K.
-- E-mail:  johnson@minster.york.ac.uk
-- Last modified:  23/2/93
--
-- Filename:  stack.A
--

package body GENERIC_STACK is

   -- constructors

   procedure PUSH(I: ITEM; A_STACK: in out STACK) is
   begin
      A_STACK := new NODE'(I, A_STACK);
      exception
         when STORAGE_ERROR => raise OVERFLOW;
   end PUSH;

   -- etc etc
end GENERIC_STACK;
```

- Exception handler by `STORAGE_ERROR`.

- Application program doesn't require operating system...

Passive Techniques

- Rollback recovery:

– uses time redundancy;
– error detected then rolls back;
– to mile-post or checkpoint state.

- May require operating system support:

– ISIS and Gossip approach to replication of data;
– here we can backup processes.

- For instance, Taragon/32.

Case Study: Taragon/32

- Why Taragon/32?

– it's a commercial product;
– common features with (Rose and Clouds OS);
– Rose uses process backup and replicated execution;
– will deal with MARS next time.

- Taragon/32 - Nixdorf fault-tolerant UNIX:

– User/systems admin. pick processes to be backed up;
– Network of 2 to 16 machines on fast dual bus;
– Each machine has shared memory multiprocessor;
– Each multiprocessor has 3 processors.

- Why a dual bus?

Taragon/32 - Leadership Elections

- Two processors execute user processes

– the third is available as a passive backup.

- One process server per system

– need leadership elections to determine server?

- Many page servers per system:

– backs-up virtual memory for subset of primary processes;
– back-up virtual memory of backup processes;
– establishes check-points for backup processes.

Taragon/32 - Leadership Elections

- Fault-tolerance leads to a bizarre form of

– 'anti-leadership' protocol to aid fault detection;
– not to elect leader but to get rid of a member.

- The machines are organised in a ring

– each neighbour periodically reports to machine on left.

Taragon/32 - Leadership Elections

- If machine doesn't report...

- Machine on right tries:

– to access other machines;
– if not it assumes it is faulty and goes down

- If it can communicate then:

– it assumes neighbour down;
– sends a message to force neighbour to crash;
– (assumes neighbour is receiving messages...).

- Broadcasts message that it's neighbour has failed.

- Locates new neighbour to it's left.

Taragon/32 - Process Replication

- For each process to be backed up:

– one primary process on one machine;
– a second back-up process on a different machine.

- Process server decides where these reside:

– has to be backed up as well 8-)

- If process crashes then backup is primary process:

– in one scheme the backup is now backed up;
– in another scheme multiple backups are not created.

Taragon/32 - Inter Process Communication

- Requires duplicate communication:

– whenever sender sends message to receiver;
– message also sent to backup;
– three way atomic broadcast - all three or none.

- Backup notified whenever primary process sends

– out a message - adds one to `write_synch_count`;
– uses this so it doesn't repeat messages sent.

- When process fails, backup demand pages

the primary processes state and carries on.

Taragon/32 - Replica Determinism

- Hard to ensure replica determinism under UNIX.

- Signals are dealt with as they arrive:

– if a signal sent to primary process;
– how does the back-up know when exactly they occurred?

- Primary process executes a synch operation before

– each signal is handled - like task timing in MARS.

- This guarantees that any backup process will

– have the same state as the primary process before
– the signal was handled - if primary process goes down.

Conclusion

- Active redundancy.

- Passive redundancy.

- Case study, Taragon/32:
  – 'anti-leadership' elections;
  – process replication;
  – inter-process communication;
  – timing and replica determinism.

- Next time: MARS case study.

Introduction

- Previous lectures covered:

– transaction processing systems;
– control of replication;
– recovery through logging.

`P. Verissimo and H. Kopetz, Design Of Distributed Real-Time Systems. In S. Mullender (ed.), DIstributed Systems, Addison Wesley, 1994.`

- Here, MARS reliable, real-time system.

MARS - Why Bother

- MARS real-time operating system:

– provides timely services;
– provides fault-tolerance;
– provides programmers with convenient means
– of exploiting these features...

- Also provides contrast with other systems (MACH).

- Good way of reviewing the material so far.

MARS - Introduction

- MAintainable Real-time System.

- Maintainability:

– restore system service after change;
– permanent hardware faults;
– system design problems;
– modify software etc.

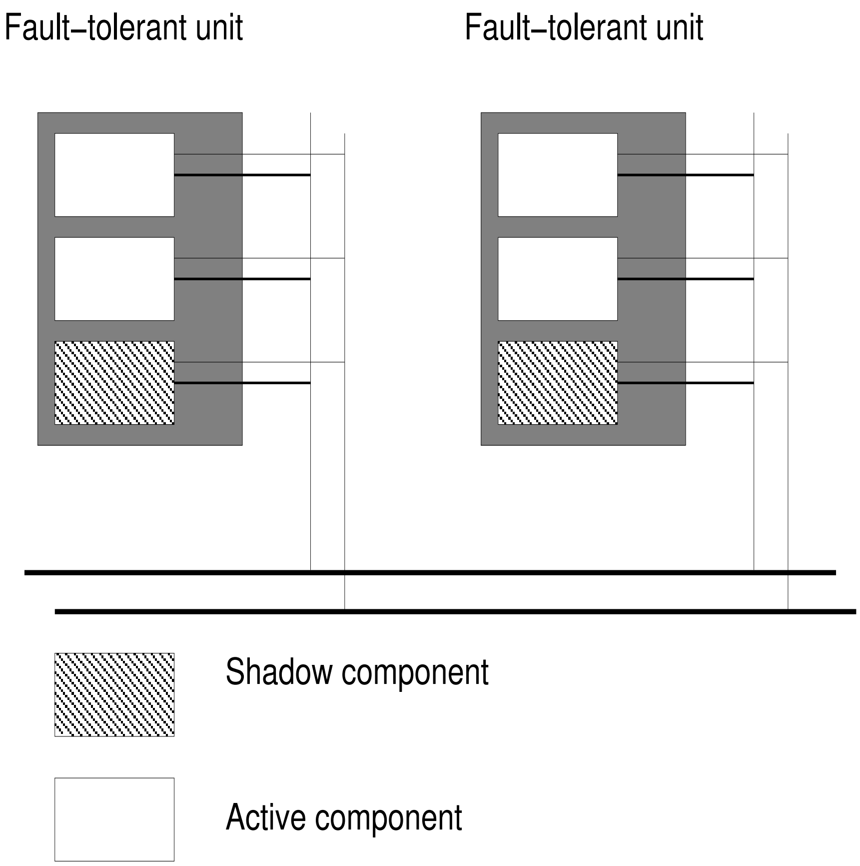
- Exploits 'encapsulated architectural units'.

- Faults:

– hardware problems;
– MTTF - Mean Time To Failure;
– if MTTF $\leq$ system life;
– redundancy provides solution.

MARS - Architecture

- Clusters include fault-tolerant units.



- Redundant communications channels.

MARS - Synchronisation

- Time triggered.

- For each control object:

– take a snap-shot of its state;
– done at regular intervals;
– specified at compile time.

- Intelligent sensors:

– monitor entity more frequently;
– than the interval compiled into system;
– the ‘perception granularity’;
– pass on an abstraction of these samples.

MARS - Time

- Contrast with MACH, Chorus etc.

```
H. Kopetz and V. Ochsenreiter, Clock
Synchronisation In Real-Time Systems,
IEEE Transaction On Computers, 1987,
933-940.
```

- Redundancy so we can't have a single clock:

– local, real-time clocks;
– establish fault-tolerant global time;
– clock synchronisation even if failures occur.

- Remember that the Berkley algorithm

– ignores time values that are way out from others...

MARS - Locking And Commits

- Must make sure state consistent in replicas.

- Must change data at same point in time in each
– of the replicas so can carry on if one fails.

- Independent local clocks so time will drift.

- So synchronise clocks before starting task
– don't allow reads on local clock during task.

MARS - Inter-Process(or) Communication

- How are values passed through system?

- Could store in buffer until read?

- MARS uses state messages:

– like variables - over-written by new values;
– minimises synchronisation between sender and receiver;
– updated at pre-arranged intervals;
– all communicating partners know about intervals.

MARS - Distributed Naming

- Systems built from autonomous clusters:

– different clusters have disjoint name spaces;
– can (will?) be heterogenous.

- System requires global synchronised time-service

– describe the problems with providing such a facility...
– time division multiple access protocol.

MARS - Byzantine Generals

- Replicated real-time communications channel:

– FTU is operational if one element survives;
– shadow updates it's internal state from active units;
– only active elements have write access to bus;
– if element fails then shadow takes the bus slot.

- Elements must be fail silent:

– correct results or none at all.

MARS - Scheduling

- Computation based around tasks:

– no communication until task is over;
– then broadcast state to other components in cluster;
– compare this with the lazy updates of GOSSIP.

- The execution time of a task is known:

– it is possible to predict the communications time as well;
– scheduling decisions made at compile time.

MARS - Mutual Exclusion

- MARS provides development tools for timing
– analysis on a restricted range of programming languages.

- Good comparison with MACH etc:
– most distributed OS use Ricart & Agrawala etc;
– to achieve mutual exclusion without semaphores.

- MARS doesn't need all this:
– mutual exclusion enforced at compile time;
– need to know timings on tasks;
– some dynamic scheduling needed (Emergency stop?);
– no dynamic scheduling just change scheduler mode.

Conclusion

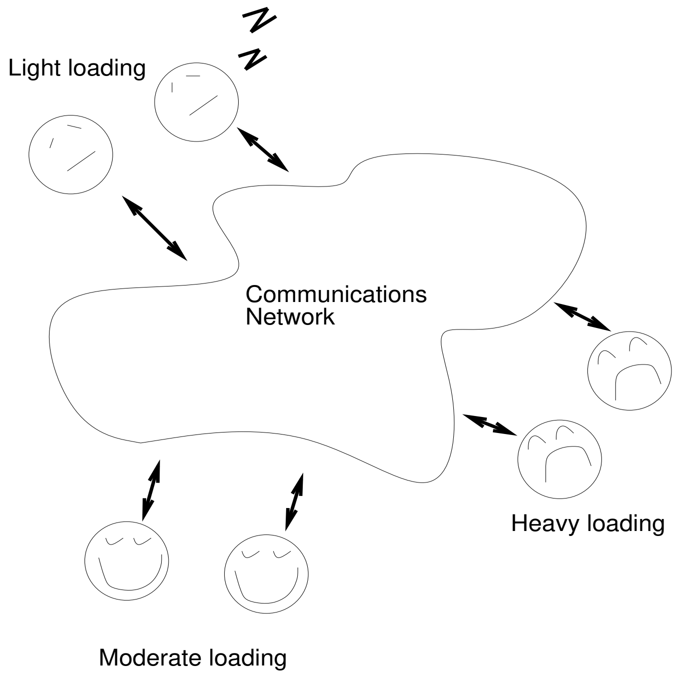
- MARS:
  - time;
  - synchronisation;
  - inter-process communication;
  - distributed naming;
  - replication management;
  - Byzantine generals problem;
  - mutual exclusion.

Introduction

- Last time - how to improve security.
- This time - how to improve performance.
- Reduce task response times.
- No unshared states: all active or all idle.
- Mainly experimental techniques, this is changing...

Load balancing

- Distributed systems without load distributing.



- Uneven loads throughout network.

Load balancing

- Loadings will vary:

- in heterogenous networks (processor speeds vary);
- in heterogenous applications (job demands vary).

- Even in homogenous systems:

– servers send jobs to clients;
– this leaves servers lightly loaded;
– they could share in some processing.

Measuring Performance

- CPU queue length:

– easy & cheap but misses jobs in transit;
– could increase CPU queue if node accepts task;
– but if task fails en route CPU count is wrong?
– decrement CPU queue after a timeout;
– still inaccurate for interactive systems.

- CPU Utilisation:

– requires background process to monitor CPU:
– imposes higher overhead in scheduling monitor etc.

Load Distribution Algorithms

- Static:

– do not vary strategy depending on current loading;
– hard wired into system, rely on a priori knowledge.

- Dynamic:

– do vary response depending on loading;
– alter response on basis of short term changes in loading.

- Adaptive:

– can vary response and strategy depending on loading;
– assess their own performance and try to improve on it.

- Preemptive vs Non-preemptive:

– transfer a partially executed task - overheads?

Load Distribution Algorithms

- Sender Initiated Algorithms.

- Overloaded node seeks help:

– if $CPU\_queue > Upper\_Limit$;
– (only tries to get rid of newly arrived tasks).

- A node will accept a task if:

– $CPU\_queue + 1 < Upper\_Limit$;
– cannot overload itself...

- So how do we pick the destination node?

– Random vs Threshold vs Shortest algorithms.

Load Distribution (Sender) Algorithms

- Random:

– if destination overloaded send it away again (nomadic);
– in worst case, system spends all time moving tasks;
– accept it (after n transfers) & send another away?
– better than no load-sharing at all.

- Threshold:

– poll a node at random to see what its loading is like;
– if job accepted it must be executed at that node;
– give up if N polls tried (net is heavily loaded);
– avoids useless job transfers under random policy.

- Shortest:

– poll N nodes with $CPU\_queue + 1 < Upper\_Limit$;
– send task to the nearest node in network;
– only marginal improvement over threshold.

Load Distribution (Sender) Algorithms

- Problem is instability.

- At high loads most nodes will be busy.

- Unlikely to find suitable receiver.

- Sender algorithms dont adjust to system state:
– costs of polling actually increase at high loadings.

Load Distribution (Receiver) Algorithms

- When task finishes request another if:

– $CPU\_queue < Lower\_Limit$.

- Node polls other nodes to request tasks:

– ensure $CPU\_queue - 1 > Lower\_Limit$;
– keep polling until some fixed number of tries;
– if this occurs then wait until another task completes;
– during this time power of node may be lost.

- Avoids instability:

- at high loadings, high prob. of finding more work;
- at low loadings, no instability as there are space cycles.

Load Distribution (Receiver) Algorithms

- Problem is preemption.

- Operating systems typically:

– use quantum based policies (round robbin);
– so each task partially executed on node.

- When receiver asks for a new task:

– sender passes partially executed state of task;
– messy and considerable overheads.

- Less of a problem in sender algorithm:

– sender just passes on newly arrived task.

- If system at high levels $\rightarrow$ preemption

– otherwise pick sender initiated algorithm;
– adaptive mixes...

Mixed (Adaptive) Algorithms

- When node polls it builds three lists:

– `Sender/overload` if $Queue < Upper\_limit$;
– `Receiver/idle` if $Queue < Lower\_limit$;
– `OK` if $Upper\_limit \geq Queue \geq Lower\_limit$.

- If node becomes sender, selects from receiver list.

- If node becomes a receiver, it polls sender list.

- At high loadings:

– sending nodes fail with lots of polls;
– busy nodes are removed from receivers list;
– so they aren't polled again until they ask to receive.

- At low loadings:

– lots of unsuccessful request for work;
– but this is useful as nodes added to receiver list.

The V System

- Slight variant on methods described above.

- Each node broadcasts state if state changes:

– CPU/memory use, processor type, co-processors etc.

- State information cached at nodes:

– large networks only cache N-best nodes.

- Only select newly arrived tasks for transfer:

– local node looks in cache for N lightest nodes;
– if local node is one of N then schedule it locally;
– else randomly pick one in N & poll to check cache data;

- Background process used to monitor CPU usage.

The V System

- Why pick one at random?

– otherwise, all nodes send work to lightest node.

- Why broadcast/publish state of nodes?

– in query methods overhead determined by no. of tasks;
– as tasks rise there will be more and more query overhead.

- In publishing approach:

– overhead determined by number of nodes, not workload;
– frequency of notification can be adjusted to cut overhead.

The V System

- If you stop task, copy it, restart it:

– lots of time wasted during transfer to remote machine.

- Copies bulk of task state to remote node:

– while task to is still running!
– send over entire virtual address space;
– only sends dirty pages once task actually stopped.

- Increases communications overhead and CPU use.

The Sprite System

- Runs on SPARCSTATION 1:

– kernel similar to BSD UNIX;
– task migration built on Sun-RPC;
– (see earlier!).

- When node falls below threshold:

– notifies central server that it can take more work;
– (NB election algorithms to pick central site????);
– overloaded nodes contact central server to send tasks.

- Users control load balancing:

– they select tasks to be executed remotely;
– as user 'owns' their own workstation, the user;
– can evict other foregin tasks to run their own;
– foreign tasks are returned to 'home' workstation.

The Sprite System

- Node can receive a task if:

– keyboard/mouse etc idle for more than 30 seconds;
– and no. of tasks < no. of processors on workstation

- Node can send tasks if:

– it needs to evict foreign tasks from a local site.

- Foreign process evicts another foreign process if:

– central site cannot find idle node for new process;
– and existing process' owner already has lots of tasks.

- Parallelised `make`: 5x faster, 12 workstations.

The Sprite System

- Built on a location independent file system.
- Modified pages sent to file server.
- File descriptors & page tables sent to remote node.
- Gradually address space stored on remote machine.
- Task moves over piece by piece, short stop time.

Conclusion

- Measuring requests vs utilisation.
- Sender vs receiver based algorithms
- Stability vs preemption.
- Adaption through mixtures.
- Case studies: V and Sprite.
- Next time - multimedia & efficiency.

Introduction

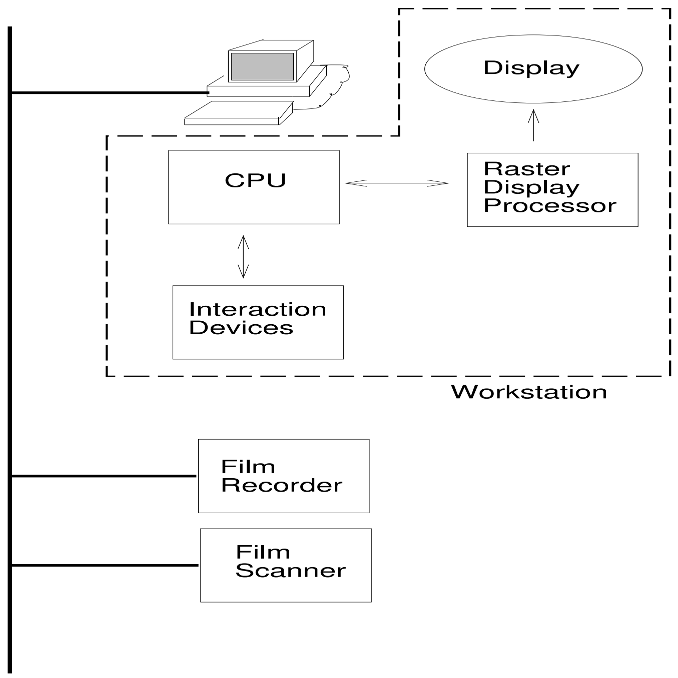
- Key issues in multimedia:

– efficiency of communications network;
– timing properties in synchronous environment;
– heterogenous devices (storage, capture, presentation);
– operating system dependencies;
– security and access control.

- Just about everything we've looked at...

Multimedia Workstations

- The traditional model.

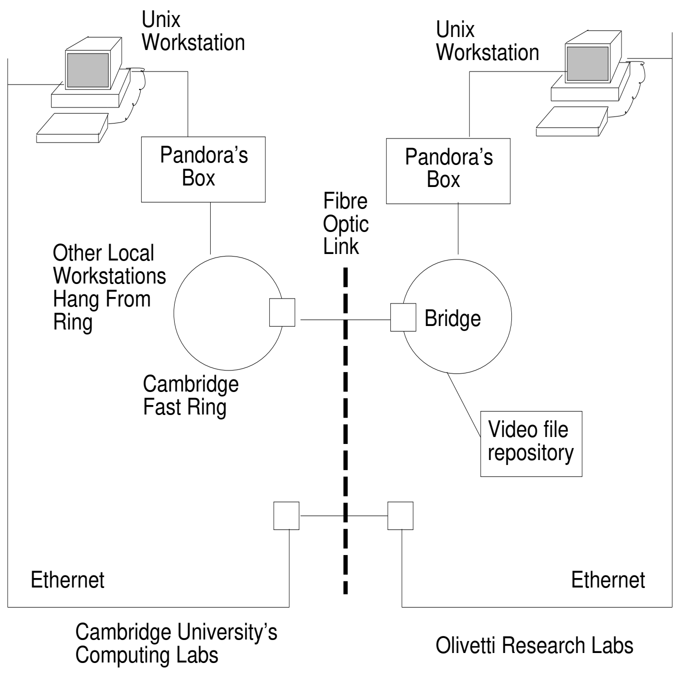


- Ethernet communcations.

Pandora's Box - Network Architecture

- Early 1990's attempt distribution...

`T. King, Pandora: An Experiment In Distributed Multimedia, Eurographics'92, Elsevier, 1992. C-23..C-34`

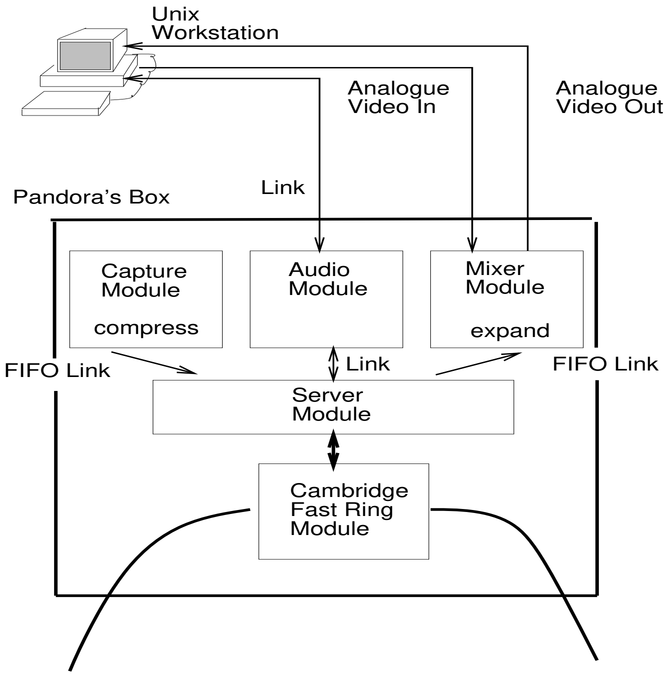


- Novel models (this one from Olivetti):

– standard workstation plus specialised box.

Pandora's Box - Internal Architecture

- Combination of networking and operating systems.



- Cambridge fast ring provides small packet size:

– ideal for continuous media such as video - fine grained;
– properties of asynchronous transfer mode (ATM).

Pandora's Box - Network

- Pandora used two networks:

– Ethernet for traditional traffic;
– Cambridge fast ring for local traffic.

- Initial trials with 3 giggabytes video store:

– only enough for about 6 hours of video and audio.

- Five modules inside the box:

– Capture, Mixer, Audio, Server and Network modules;
– each has transputer, memory & high performance ports.

Pandora's Box - X

- Presentation on top of X:

– same client server architecture.

- Two servers want to set up video channel:

– create video input and output on each machine;
– link one to the other over the Cambridge Fast Ring.

- Why is the X model a problem:

– X servers support any clients on network;
– any client can access local video camera;
– Olivetti and Silicon Graphics use lens caps.

- Story about my mother and the microphone...

Medusa

- Late 90's improve on Pandora.

```
S. Wray, T. Glauert and A. Hopper, Networked
Multimedia:  The Medusa Environment, IEEE Multimedia,
Winter 1994, 54-63.
```

- Medusa now relies on solely on ATM.

- Difficult to add to Pandora incrementally:

– it was a single box of transputers etc.

- 'Biggest' problem was lack of access control;

- see lectures on security issues in operating systems.

Medusa - Environment

- Runs under UNIX, ATMOS and WindowsNT.

- Build to run under ATM:

– assumes global address scheme exists;
– anything on networks can connect to anything else;
– connections assumed to be plentiful and cheap.

- Build using C++ object hierarchies.

- Programmed via Tcl/Tk on top of X

Medusa - Messages

- Typed messages:

– text;
– integers;
– capabilities (some security control);
– mark segments (to delimit messages);
– time segments (Global Positioning System);
– audio segments (byte string sample rate, encoding);
– video segments (byte string pixel format, height);
– command segments (integer command identifiers);
– reply segments (status code for commands etc);
– event segments.

Medusa - Connections

- Messages are passed over connections:

– not typed, one way data flows;
– synchronous and demand driven;
– the sink demands resource from supply;
– cannot demand more until resource supplied;
– reliable - messages arrive or connection broken.

- Simple for the application programmer:

– window requests camera input once screen refreshed;
– pipeline processors demand input when ready for more.

- Slight problem when running over network:

– machines demand shared resource at different rate;
– need buffers to preserve synchronisation over net.

Medusa - Making Connections

- In order for two modules to connect they:

– need respective names of the port;
– need respective capabilities.

- Medusa capabilities contain:

– network id of process on particular machine;
– large random number to prevent fraud (?).

- Receiving node uses call-back to create connection:

– sends port name and direction of requested connection;
– doesn't send capability, original site accepts/rejects.

Proxies

- If a resource is available for some time:

– capabilities will not restrict access;
– common problem with private key cryptology.

- Proxies handle communication with resource:

– application manager may create proxy for limited time;
– when proxy destroyed user loses access to that resource.

- Proxy protects resource:

– if access to resource changes, proxy enforces it.

The Future

- Technology mature but not fully established.

- Hardware still highly specialised:

– by 2000 camcorders & televisions provide ATM sockets?

- Olivetti have an ATM autostereoscope frame store:

– virtual 3D image without special glasses etc;
– image appears to be presented behind the screen.

- Now adding mobile resources to Medusa.

Introduction

- STM versus ATM.
- Statistical multiplexing.
- VCIs and VPIs.
- UNIs.
- The future...

Synchronous Transfer Mode

- STM used to transfer voice and data.

- A circuit switched mechanism:

– establish connection between both endpoints;
– hold connection open during communication;
– tear-down connection once it is finished.

- Why is this wasteful?

What is ATM?

- Relies upon statistical multiplexing.

- Support multiple sessions over same channel.

- Low probability that they all send at once?

- Sum of peak bandwidth requirement can exceed
  – aggregate average bandwidth of the network.

What is ATM?

- Negotiate your connection.

- What quality of service do you require?

– do we have to keep some bandwidth in reserve?

- How much will you pay?

– to discourage automatic request for high QOS.

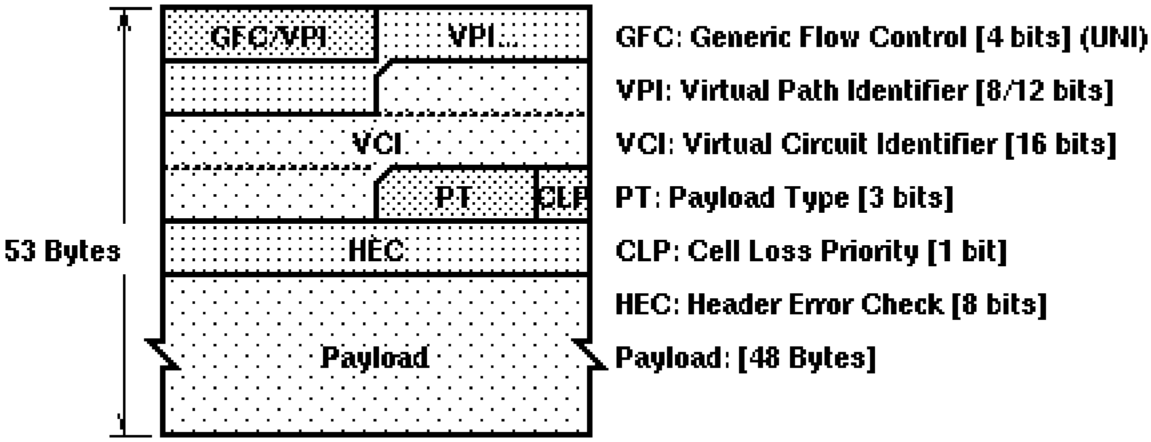
- User-network interfaces on ATM devices.

What is ATM?

- Fixed-length cell switching standard.
- Cells - multiple sources & multiple destinations.
- Asynchronously multiplexed on mult packet switches.
- Every circuit has Virtual Path Identifier (VPI).
- Each circuit has Virtual Circuit Identifier (VCI)

What is ATM?

- Small, fixed size, cells.



- Small to reduce cost of loss/delay.

What is ATM?

- GFC field for routing cell at endpoints:

– User Network Interface exploits it.

- VPI gives multiple circuits destined for same endpoint

– provides alternative routes in case of delay;
– switches en-route dont need to work out new path;
– just use the other entries in VPI.

- VPI = 8 bits for UNI and 12 for Network-Network.

What is ATM?

- ATM switches are responsible for:

– switching cells between ports;
– buffering cells;
– translating VPI/VCI's;
– guaranteeing Quality of Service;
– connection set-up;
– connection tear-down.

Routing & Flow Control

- TCP/IP uses end to end flow control.

- Can send messages if buffers flood.

- ATM just blasts it out.

- Network switches must take action...

- Re-routing, eventually turn down requests.

Causal Ordering

- If you establish an ATM 'connection':

– causal ordering is preserved.

- Cells may be congested or buffered in switches.

- But no store and forward on the network.

- Cells travel over same Virtual Channel.

Causal Ordering

- But connectionless services exist.
- These might use multiple virtual channels.
- Must reassemble order at the end.
- ATM does not guarantee reliability.
- No retransmission.

The Future

- ATM UNI in your television set.

- Or at least on top of it...