Edsger W. Dijkstra
The Structure of the "THE" Multiprogramming System

1) A model for multiprogramming is 
   > to prove delaying process does not hurt. 
   > provable system.

2) (1) Hierarchical abstractions
   - semaphores are synchronizing primitives & for mutual exclusion

3) Hierarchy
   - Level 0 - abstract the processor with a process scheduler
   - Level 1 - segment controller, abstracts secondary storage into “segments”
   - Level 2 - message interpreter, support conversation b/t user and their
     high-level processes
   - Level 3 - buffering I/O
   - Level 4 - user programs
   - Level 5 - operator (not implemented by THE)

4) Virtual address space - separates data from where it is stored. Can map virtual address to physical address.
   Advantage is that virtual address space can be much bigger than physical address space. Also don't need contiguous memory, meaning more efficient memory usage.
   Disadvantage is takes extra space (page table) and lookups take extra time.
Semaphore for mutual exclusion

Process 1

\[ P(\text{mutex}) \]
Critical Section
\[ V(\text{mutex}) \]

Semaphore for scheduling

Consumer
\[ P(\text{mutex}) \]
If element in queue
\[ V(\text{private}) \]
\[ V(\text{mutex}) \]
\[ P(\text{private}) \]

Producer
\[ P(\text{mutex}) \]
Add element to queue
\[ V(\text{mutex}) \]
\[ V(\text{private}) \]
Motivation:
Create an extensible OS, which lets the applications decide
their own policies and even behave as OS themselves.
They wanted to do this because current OS don’t allow
enough flexibility to applications.

2. High level point
The nucleus is used to implement the simulation of processes,
comm. b/w them, creation, deletion & control of the processes.

3. Interprocess communication:
Get away with semaphore driven synchronization for better
efficiency & security. Use message passing through buffers.
This technique is an asynchronous way of communication &
scales well. Basic message calls:
send, wait, send message & send, wait, get response

4. Process Hierarchy
The memory space originally belongs to the initiating process, which
can allocate a subset of its memory to a process it creates. This
results in a parent-child protection scheme for memory access. As an
extensible generic process management scheme, this also allows arbitrary OS’s

5. 2 processes: internal, external

- treat internal and external processes uniformly to
create extensible system. The difference between
them is merely a matter of processing capability.
Review:

1. Introduced process names.
2. Document is also a process (external)
1) Wanted to make a general multi-user operating system that was simple, easy to use. (Authors thought filesystem was the most important role of an OS), so they focused mostly on that. Also wanted an easy to use shell and command execution.

2) Conceptual contributions - I/O equivalent to file system access - fork/exec/pipe/wait/exit style process management

   Practical contributions - the Unix system, including the file system & shell particularly that a multiprocessing system can work on relatively inexpensive hardware.

3) File system with directories, inodes, links, special files, etc. /dev
   -> linking of files -> protect objects -> setuid
   -> for general purposes, not for any predeence objectives

4) Process - execution of an image (code, registers, open files, current directory, etc.)
   - Exists in memory
   - Created by fork() in parent process; copy of memory, shared files, etc.
   - IPC with pipes
   - Execute new program with exec()()

Adv.
fork() + exec() clean & easy

P.T.O.
5) TRAPS → How they work?

**Scenario 1**
- PDP-11 H/W detects
- System terminates the process
- Writes image on file "core".

**Scenario 2**
- Unwanted output by a program
- Halted by user using "delete"
- System program calls execution
  - If (quit)
    - T
    - Produce img file

**Scenario 3**
- How generated faults
- Either
  - Ignored
  - Catch, interrupt, handle
1) The motivation of this paper is that researchers use disk models to improve overall I/O performance in disks by running simulations on these models. However, the problem is that little aside has been done to create accurate disk drive models. This severely limits the useful information gathered, because simulations are run on models that do not reflect reality. Therefore, the author aims to build an extremely accurate disk model.

2. The authors discuss the physical characteristics of disk drives and explain how they affect I/O behavior. They present an accurate model of the disk drive. They created a simulator based on their physical understanding, which can accurately predict and emulate disk I/O behavior.

3. The primary technique was to model specific physical elements of the disk's operation. The simulator tried a model with linear seek times, which was not that accurate, and then a model with seek times that accurately reflected the acceleration, max speed movement, deceleration, and head settling time of a seek, resulting in much more accurate modeling.

4. Another major component they modeled was the caching behavior of typical disks. They were able to show that a model that does not provide a cache simulation is wildly inaccurate since a large percentage of I/O is handled at the cache. After implementation, their model was much more inline with a real disk.

5) Track skewing: offset block data on each track so that data can be read continuously with fewer seek back on the disk. When head or track switches occur.
1) the problem was that the original Unix FS was incredibly slow, getting a blazing 2% of max disk transfer.
   - there were 3 problems:
     1. Small block sizes - more transfers, w/ fixed cost per transfer
     2. Poor freelist organization - consecutive blocks not close together?
     3. No locality - inodes far from data blocks

2) Contribution:
   - Better locality is proposed - using cylinder group.
     - larger block size
     - fragments for small files

3) - Organize data which is accessed together in close spatial locality. Like inodes and data blocks.
   - Use 4 KB blocks to get better bandwidth, but also support smaller fragments to reduce wastage.
   - Place file data in rotationally optimal blocks to increase sequential I/O performance.

4) Layout policy: Files in a folder are put in the same cylinder group if they fit and are smaller than 48 KB. Advantage: Less seek time when accessing files in same directory, seek time is less of overall cost for large files. More files in folder fit together.
   - Disadvantage: Large files are split across disk at 1 MB intervals, which increases read time.
5) if the CPU doesn't have I/O channel, the data has to be put in a "rotationally optimal" location, so the disk head will be on the right position after the CPU interrupt.
1) Goal was to support a variety of physical mem sizes and demands of time-shared systems, realtime and batch processes.

2) Predictable performance can be given by maximizing giving more space in memory (using techniques like swapping and paging our page tables) and reducing I/O (using clustering).

3) Their implementation of the swapper was a solution to the problem of the high cost associated with paging in pages after a process is able to execute again. Without such the swapper keeping a process resident set to load in pages on a process wanting each page would have been thrashed on. √

4) VAX also implemented a pager service designed to efficiently manage paging in and out. In particular, when a page is removed from a process' resident set, the pager puts it on a free list (if it is unmodified) or on a modified list, which serve duelively as caches for pages that might be read in again later and as sources of free pages for when new pages are read into memory. They could also be used to batch I/O of writing modified pages back out to disk. √

5) VAX's virtual addresses were as follows:

<table>
<thead>
<tr>
<th>Translation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN 21 Offset 9</td>
</tr>
</tbody>
</table>

To convert translate an address into a physical memory location, VAX would:

1. Look at the first 2 bits, and go to the corresponding segments base register.
2. Jump to that segment's page table, and translate VPN into PPN.
3. Jump to that physical page, and apply the offset.
UNIX's virtual memory system relied on limited functionality for portability. Mach aimed to create an easily-portable VM system that provided more functionality than UNIX and minimized hardware-dependent code.

2. The high level contribution of this paper is the abstraction they provide between the machine dependent/independent code. Before this work virtual memory management was almost always machine dependent. However this work demonstrated/implemented the construction of a memory management system that was independent of hardware. This allowed the system to run on any hardware configuration, something commonplace now.

3. Shadow objects allow multiple processes to use the same memory for the same objects. Instead of allocating memory twice, the second process contains a shadow object which points to the original memory. This uses less memory for shared objects. A disadvantage is that when memory is written to a shadow object, many new smaller memory objects may be created and this can be complicated to manage.

4. Inter-procedure Communication (IPC) was an important part of their design. Each task had "parts" which other tasks were able to use to send/receive messages. Messages could be passed back and forth through the copy-on-write mechanism, which significantly reduced the IPC overhead, relative to Nucleus.

Mmap acts as a cache of virtual/physical mapping, but kernel mappings need always be kept. Other mappings can be reconstructed by the process independent part.
i) jumble

ii) C write B → B'

iii) Look

C

P

P

C

B

B
1) They wanted to enable the running of commodity OS's on ccNUMA machines. At the time, new scalable computers with 10s or 100s of processors were being released, but the complexity of implementing software for these machines was too great, and thus software was lagging behind hardware.

2) Disc allowed multiple OS's to run mostly unmodified on one system. It manages resources between all OS's without requiring OS to worry about non-uniform memory. commodity OS's can be used.

3) Introduced new layer between hardware and software. New layer supports virtual CPUs, virtual physical memory, virtual I/O devices, memory management, virtual network interface.

4) Disco took advantage of existing virtual memory hardware to virtualize guests’ "physical" memory. This allows guests to be moved from processor to processor, share memory, and share disk reads. However, it causes TLB misses to cause traps to Disco; which then has to emulate hardware for guest kernels, increasing the cost of TLB misses.

5) In Disco, system calls to a guest OS look like this:

```
user app
    trap
        Disc
    return
        Disc
```

Edouard Bugnion, Scott Devine, Mendel Rosenblum.
Disco: Running Commodity Operating Systems on Scalable Multiprocessors
Proceedings of The Sixteenth Symposium on Operating Systems Principles (October 1997)
1) The prevalence of operating system extensions—code executing in fully privileged mode yet potentially written by relatively inexperienced programmers with little testing—leads to many system crashes.

2) Introduced mechanisms to isolate driver code from the kernel by using wrappers and XPC.

3) Basically, Noeks introduces indirection to implement isolation, interposition, object tracking, and recovery.

4) Noeks Isolation Manager (NIM) is inserted between the extension and OS kernel. Provides lightweight kernel protection domain by limiting write access of extension to kernel address space. Isolates kernel resources used by extensions & cleans during recovery upon extension crash/misbehavior.

"Well written!"
5) Isolating kernel from extension failures provides 99% recovery from system crashes. ✓
   → high reliability
   - Disadv.
     - large overhead introduced. ✓
       e.g. Web server: 60% ↓ in throughput.

point: was not good at recovering the modules, just good at recovering the system.

RecoveryManager - releases resources used by extension
  cleans the system state
W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack
HYDRA: The Kernel of a Multiprocessor Operating System
Communications of the ACM 17(6), June 1974, pp. 337-344.

1) Want to support multiprocessor environment and provide
   sufficient primitives for protection of resources (not security!).
   Also wanted to avoid a hierarchical system structure, which can
   be limiting.

2) Provided a way to separate mechanisms from policies.
   - Abstraction of resources as objects

3) i) Caller dependent capabilities & caller independent capabilities
   ii) Type dependent access rights (for custom obj's) & kernel access
       rights for common operations (like copying) for all objects.
   iii) Kernel can provide protection without "understanding" the rights bits.
   iv) Path-names to avoid explosion in no. of capabilities passed between
       procedure calls.

4) Capabilities can't be forged. Ensured by HGA or a support
   tag. Every word of memory is tagged. User process:
   - Only kernel can modify capabilities → put in segmented
     procedures
   - Only kernel can modify data (code or static data)

5) Flexibility to obtain protection from both protection
   of object or both data & capabilities.
Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.  
*Exokernel: An Operating System Architecture for Application-level Resource Management*  

1. The motivation of exokernel is that the lower the level of a primitive, the more efficiently it can be implemented and more flexibility to implementors of higher-level abstractions. Higher-level abstractions hurt application performance, thus information I limit its functionality and hence the need for a library OS.

2. Securely expose hardware → privileged instruction, access resources

   2.1. Expose allocatable → request physical resources

   2.2. Expose physical names

   2.3. Expose revocations

   *Multiplex resources, don’t abstract.*

3. i) Secure bindings → to securely provide applications the low-level resources of the systems.

   • Can be implemented in hardware or software

   • Downloading code into kernel to extend the capabilities of kernel.

   • Packet filtering by downloading code which by dynamically generated.

4. Visible resource revocation - e.g., CPU is explicitly removed from libos. Libos saves the processor state.

   Abort protocol - If a libos misbehaves with a resource, is disobedient then Exokuel takes that resource away from libos forcibly.
ASHs - Application Specific safe handler. An implementation of downloading code into kernel. They do message processing for packets arriving over network. It runs on packet receptions and it can also initiate a message to be sent. This avoids the application to be scheduled to respond send a response to the received new packet.
MOTIVATION:
Not enough analysis done in the past about home workloads.
To understand the I/O behavior and how users perceive system
delay and app. performance.

CONTRIBUTION:
In depth case study and analysis of the I/O behaviour of modern home-user applications. The analysis has several interesting revelations: (file is not a file, seq access i not seq, writes are forced, multiple threads perform I/O, frameworks influence I/O etc). + renaming is popular + auxiliary files dominate

SOLUTION/TECHNIQUE: advan./discov.

The results have a strong ramifications for the design of next generation local and cloud-based storage systems. Constructing the ibench task suite two-fold: 1. representative of tasks performed by home users?

2. SOL/Tech:
A) The authors study four categories of the behavior of the ibench suite,

1) Nature of files
2) Access patterns
3) Transactional properties
4) Multi-threading behavior

5) iTunes attempted to block efforts to trace it, but the authors
found a way to circumvent this by using gdb to force a
system call into a NO-OP.
1. The motivation for LFS is to match the performance (speed) of the disk with that of the CPU.

2. The use of log to club small random writes into large sequential writes to achieve better utilization of disk bandwidth.

3. They combined interleaving and compaction into a hybrid approach where they grouped blocks into segments and threaded the segments. They periodically ran a cleaner to find underutilized segments and write their data at the front of the log.

4. Identifying live blocks using segment summary block which contains for every database file inode and offset.

   Some more optimizations like maintaining versions in island map to avoid costly identification of live blocks (segmentation of files).

   ii) Using 2 timestamps for each of two CR regions and writing to alternate CRs

   Adv - No inconsistent CR
   D Adv - The CR commit interval is 80s which affects freshness of data

   write cost = \frac{2}{1-u} \cdot \text{cost/seg} \\

   As cost per byte storage approaches, the approach of having more underutilized blocks increases cost per byte.

   A cost benefit segment cleaning policy is proposed.

   The cost benefit policy involves bimodal distribution of segments,

   \text{benefit} = \frac{(1-u) x \text{age}}{\text{cost}} \left( 1 + u \right)
To use cost-benefit policy, a data structure which records live bytes and recent-modified time called segment usage table, is used.

Segment Usage Table
Motivation:
Present disk failure model for commodity system is faulty. It assumes that disk either works or fails.
In reality, disks exhibit complex disk failure patterns.
Present paper does have interesting contributions.

2. The paper presents a more advanced failure model for disks that provides partial failure support and develops a framework to determine how a file system handles failures. Analyzing commodity FSes shows that their handling of failures is buggy and inconsistent.
Finally, a more robust file system, "ext3," is demonstrated to show that error-correcting techniques such as checksumming provide substantial reliability improvements at a low cost.

3. To determine how the FS handled disk failures, the authors defined the IORN Taxonomy for various levels of detection and tabulated the behavior for various workloads.
   Adv: The tabulation (complex picture) gives a very detailed picture of the behavior.

4. Build a prototype version of an IORN filesystem (ext3) that is robust to various kinds of partial disk failures. Evaluation of ext3.
   (ext3 + checksumming, metadata replication, parity)

5. Works: Transactional checksums for ordering journal data. This improves the performance of journal data I/O.
   (checksum the transaction blocks & write the checksum with the transaction commit)
1. Motivation:
   To understand modern file systems' (ZFS as a representative FS) reliability mechanisms for dealing with disk & memory corruptions. ✓ ✗

2. ZFS while does checksum verification for on-disk blocks, it doesn't use these same checksums in memory to protect against in-memory corruption. ✓

3. Solution/technique: ZFS uses checksums for all blocks on disk. This checksum is stored in the parent block for each block; by doing this rather than storing it alongside the checksummed block, it becomes possible to detect errors such as misplaced writes. Maintaining and checking these checksums does cause some overhead, and updating a checksum necessitates propagating new checksums up the file system, through the diamond of
   This is guaranteed through copy-on-write semantics. ✓

4. ZFS in addition to checksums, use replication to maintain reliability
   - metadata = 2 ditto blocks
   - global metadata = 3 ditto blocks
   - provide
   - ZFS uses ties to simulate software-based RAID
   - system by having a total of storage pools among pool
   - pool + 1 it consists of virtual devices (physical and logical)
   - RAID Z (similar to RAID5)
   - ditto ➔ variable strips.
5. How data block works?

```
<table>
<thead>
<tr>
<th>vdev1</th>
<th>offset 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdev2</td>
<td>offset 2</td>
</tr>
<tr>
<td>vdev3</td>
<td>offset 3</td>
</tr>
</tbody>
</table>
```

Checksum

Diagram:
- DVAP
- DVA1
- DVA2
- DVA3
- Offset
- Block
- ""
Motivation - Crashes are rare but possible in FS. Pessimistic approaches anticipate crashes and do the careful logging to the journal before writing to disk. These ordering points make the FS perform slow. Main motive is to avoid ordering as much as possible and also decouple ordering from durability.

High Level Point: Probability of inconsistency because of crash is high for workloads with a high number of random write I/Os, for probabilistic crash inconsistency. Maintain consistency to some extent as pessimistic, but nearly same performance with probabilistic consistency.

Solution:
1. Add checksums to avoid write ordering.
2. Use asynchronous durability notification to delay checkpointing until durable commitment has happened.
3. Ordered Journaling. (transactions)

Advantage 1. Disk itself determines when to flush, don't need to listen to higher level FS.

Disadvantage. Recovery takes a long time (not much longer than normal journaling).
   - greater code/memory overhead + stale data

Essential operations of osync:
When O/p FS calls on osync, it writes, in order, the data blocks, journal meta-data, and journal commit block. Checksums are added to the commit block so these can become durable in any order. Then, after receiving notification (or a simulation of this) that the blocks are durable, will it 'checkpoint' and write the meta-data to disk, which occurs in background.
D. Ongaro, S. Rumble, R. Stutsman, J. Ousterhout and M. Rosenblum
Fast Crash Recovery in RAMCloud

1. For DRAM store everything, need full performance of DRAM
   Fast recovery on crash < desirable + available

2) Techniques
   Single-recovery master

3) Disadv.
   - Very hard to achieve such short switching times
   + Hard to optimize to make it work
   Faster NICs. But badly worded

4) Used DRAM to support low latency
   + Volatile so backup efforts made to avoid data loss
   - Can potentially be used for massive real-time computations
5.) Recovery Process

1) When DRAM crashes, controller asks all nodes what pieces of the missing data they have ✓

2) Assigns a subset of recovery masters, tells them which segments to incorporate into their own memory ✓

→ Recovery is spread out among many disks ✓

Done, serve requests from restored data ✓ ✓

Servers keep "wills" indicating how data should be grouped ✓
CAN WRITES BE PARALLEL IN RAID 1?

1) * Single disk becomes bottleneck (because seek latency & bandwidth are not improving at a good rate compared to processor speed).
   - Thus, how can we use multiple array of inexpensive disks & collectively use them to get greater bandwidth for I/O?

2) a) categorize model & categorize different techniques
   b) take advantage of parallelism
      - 12 times higher bandwidth than state-of-art device at that time.
      - Good reliability & evolution of each case.

3) Low reliability can be dealt with using replication. The technique is simple: buy another disk, and copy the first disk over entirely. An expansion of this is data redundancy, used in all levels.
   - The advantage is it can improve reads and reliability, while the downside is it requires extra disk space (whether replicating or checksumming, etc.)

4) Parallel access improves aggregate bandwidth when?
   - Data is replicated or when partitioned (RAID-0). RAID-0 lacks redundancy, however RAID-1 can benefit from parallelism.
   - Write performance of RAID-2 is usually no worse than a single disk assuming no bus contention.

5) From RAID-0 to RAID-5 the models progress by improving disk utilization (although the performance for small writes decreases as we go through the levels) and reliability through parity disks. RAID-5,
improves upon RAID 3 & 4 by distributing the parity.
John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan
The HP AutoRAID Hierarchical Storage System

1. X? Motivation: A two-level storage hierarchy implemented inside a single disk-array controller. RAID 0 & RAID 5 by abstracting from the system.
   - HP AutoRAID: fully redundant storage system

2. Hig Contributions:
   - A two RAID-A storage hierarchy is developed.
   - Using which distinguishes between write-active (RAID 0) and write inactive (RAID 5).

3. Active freg. accessed data is kept in RAID 0 (mirroring)
   - Inactive less
   - Transfer of data between the diff. RAID levels according to the need of the workloads.
   - Data stored as: RB segments, PEXEs, PEGs, LUNs.

4. Advantages:
   - Automatically manages RAID groups.
   - Good job at balancing performance vs disk utilization
   - Scalable: possible to add new disks on the fly.
   - Upgrade: Replace disks without disrupting operation.
Disadvantages:
- Has variable response times. Not suitable for some workloads.