CS 736: Advanced Operating Systems

Andrea Arpaci-Dusseau

Lecture 2: Background – OS Structure

Papers for Today: THE, Nucleus, and UNIX

Questions for Today:
- What are the good abstractions chosen by THE and Nucleus?

To do:
- Read papers for Wed, Sept 11 lecture: Disks and FFS
- Form reading group and email by Friday

Overview of Course Reading

Quick overview for background
- OS Structure: THE, Nucleus, UNIX
- File and storage systems: Disks and FFS
- Memory Management: VAX VMS

OS Structure
- Mach, Nooks, Disco, Hydra, Exokernel

File and storage systems:
- LFS, Journaling, File not a file, IRON, ZFS, Optimistic Consistency, RAID, AutoRAID

Memory management
- Multics, Superpages, VMware ESX

Concurrency, Scheduling, Resource Management
- Mesa, Monitors, Eraser, Scheduler Activations, Resource Containers, Lottery

Parallel and Distributed Systems
- RPC, NFS, AFS, MapReduce, GFS, CFS, Scaling

Current SOSP Topics

OS Structure

How "should" an OS be organized?

What OS structure(s) lead to the best…

- performance?
- reliability?
- flexibility/tuning for different workloads?
- scalability?
- energy savings?
- maintainability?

What interfaces, mechanisms and policies should be provided?

Example Systems

- Today: THE, Nucleus, and UNIX
- Later: Mach, Nooks, Disco, HYDRA, Exokernel

THE

Why called THE?

- Technische Hogeschool Eindhoven
- Dutch for Eindhoven University of Technology in the Netherlands

When?

- 1968

Who?

- Dijkstra; what else do you know about him?
  - Shortest-path algorithm
  - Semaphores
  - Goto statement considered harmful

Major contributions?

- processes, semaphores, hierarchy

Primary Goal of System?

Smoothly process a continuous flow of user programs

- Be all things for all people with multi-programming

Why is multi-programming good?

- Reduces turn-around time for short programs
- Economic use of peripheral devices
- Automatic control of backing store to be combined with economic use of central processor
- Economic feasibility to use for flexibility and not capacity or processing power

Research Principle?

“wishing to contribute to the art of system design”

How?

- Ambitious as possible, keep routine work to minimum
- Select sound machine, keep specifics out of design
- Learn from previous experience

Basic Structure of THE

Automatic backing store
- "Storage Allocation"

Society of processes with synchronization
- "Processor Allocation"

System hierarchy
- Strictly followed

How is memory addressed?

Need to differentiate between "data" and where it is stored

Terminology?
- stored - memory units: physical "pages" (frames) in core memory or drum/disk
- data - information units: "segments"-- just fitting in a page

Virtual address space (not introduced in THE) – Great Abstraction
- Much larger than physical memory space
- Segment has "seg id"
- Use "seg id" (in "page table" stored in core) to map from virtual to physical ("seg variable" designates location)
- Advantage of page table?
  - Flexible placement of pages anywhere (do not need consecutive pages)
- Con?
  - Extra lookup-cost per access (in core memory here)

How is computation organized?

Society of Processes

- Everything in system is a process – each user program, each input and output peripheral

Definition of sequential process:

- Only the time succession of various states has a logical meaning, not the actual speed w/ which the sequential process is performed

Natural for user applications, bigger deal within OS

- Past: real-time interrupts and irreproducible errors
- Great abstraction

How to regulate interactions across processes?

Synchronization provided with semaphores

- Appendix because reviewer asked for more details!

Definition of semaphores:

- Special integer variables accessible with P() and V()
- P(sem); test or "proberen"
  sem.value--;
  if (sem.value >= 0) continue; else wait on sem.list
- V(sem); increment or "verhogen"
  sem.value++;
  if (sem.value <= 0) wake waiting process on sem.list
- P() and V() are "indivisible actions"

Two purposes

- Mutual Exclusion
- Scheduling (Private Semaphores)

How to use semaphores for mutual exclusion?

Semaphore variable shared between 2 processes

Example Use:

- P(sem);
- ...Critical section...
- V(sem);

How to initialize sem?

- Value of 1

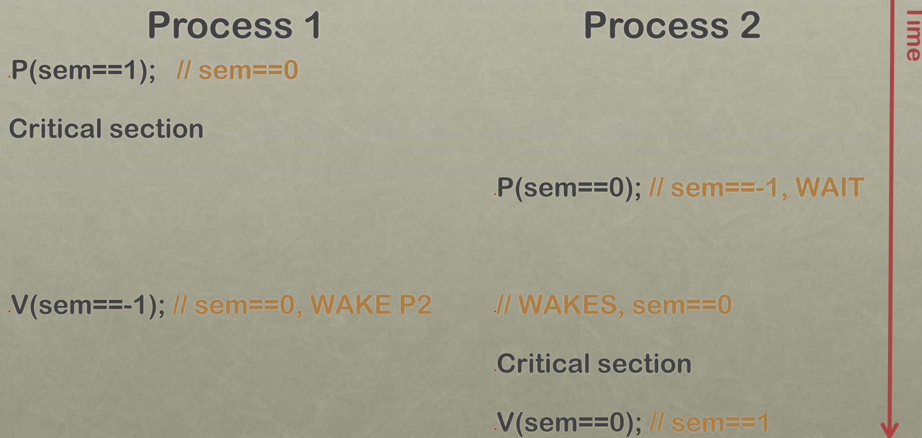
P(sem)
sem.value--;
if (sem.value >= 0) continue; else wait

V(sem);
sem.value++
if (sem.value <= 0) wake waiting

Show correctness for mutual exclusion?

P(sem): sem--; if (sem < 0) wait;
V(sem): sem++; if (sem <= 0) wake;

How to initialize sem?
Value of 1



How to use semaphores for scheduling?

Private semaphores: no other process performs P()

Process 1: Consumer

- P(mutex);
- If (ele on q) V(private);
- V(mutex);
- P(private);
- Remove ele from q;

Process 2: Producer

- P(mutex);
- Put ele on queue
- V(private);
- V(mutex);

How to initialize private?
Value of 0

How to use semaphores for scheduling?

P(sem): sem--; if (sem < 0) wait
V(sem): sem++; if (sem <= 0) wake;

How to initialize private?
Value of 0

Process 1: Consumer

- P(mutex);
- If (ele on q) // TRUE
  V(private); // private==1
- V(mutex);
- P(private); // private==0
- Remove ele from q;

Process 2: Producer

Scenario: Producer ran in past and placed element on queue

How to use semaphores for scheduling?

P(sem): sem--; if (sem < 0) wait
V(sem): sem++; if (sem <= 0) wake;

How to initialize private?
Value of 0

Process 1: Consumer

P(mutex);

If (ele on q) // FALSE
 V(private);

V(mutex);

P(private); // private==-1; WAIT

//WAKES!

Remove ele from q;

Process 2: Producer

P(mutex);

Put ele on queue
V(private==-1); // private==0, WAKE

V(mutex);

Time

System Hierarchy

Virtualize each layer so next layer can utilize

Level	Component	Resulting Abstraction
0	clock interrupt	Virtualized CPUs
1	segment controller	Virtualized Memory
2	msg interpreter	Virtualized Console
3	buffering of I/O	Virtualized I/O Devices
4	apps	
5	operator	

Pros?
- Ease of construction (layer i can use all abstractions < i)
- Simplifies testing (ensure layer i is correct, then test i+1)
- Simplifies reasoning (with ordering, no cycles and no deadlock)

Cons?
- Can add overhead
- Cannot use layers above (e.g., segment controller and I/O buffering

THE Summary

Choosing the right abstractions helps

- Virtual memory
- Processes and Semaphores
  - Can prove that processes interact correctly
  - Useful internally to system as well as for users
- Hierarchy
  - Aids testing and proofs of correctness as well

Nucleus

Who? Per Brinch Hansen

When? 1970

Goal of Nucleus?

- Provide basic primitives that allow for extension

Assumptions?

- Try to make none about workload or usage (installation)

Major contributions of Nucleus?

- Replace process with another (interposition, extensibility
- Using messages for communication and synchronization

System Nucleus

Components of system nucleus?

- Processes (simulation, creation, control, and removal)
- Communication among processes

What did Nucleus add to notion of process?

- Unique process name (pid)

Why is pid needed?

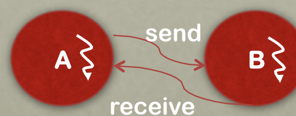
- Enables communication between processes

Two types of processes

- Internal process (program in execution)
- External process (I/O)

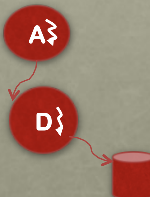
Processes

Internal process:
Execution of one (or more) interruptable programs in a storage area



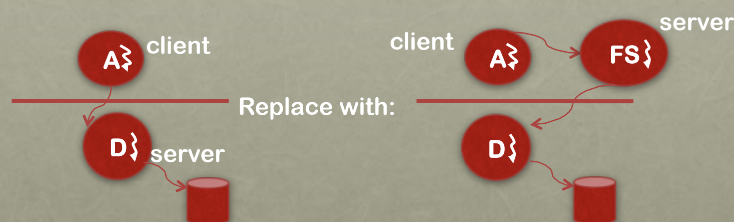
External process:
Executing code that interprets messages from internal processes and initiates I/O with storage devices

- "device driver"



Important Property of Internal vs External?

- Must be treated uniformly to create extensible system
- "We consider it an important aspect of the system that internal and external processes are handled uniformly as independent, self-contained processes"
- What does that enable one to do?



- Interposition

Communication

- Semaphores are sufficient logically, but suffer in some environments; why?
  - Requires shared memory for semaphores and data
  - Process may call P(), enter critical section, then die
    - No other process can fix problem and enter critical section!
- Alternative Proposal: Message Buffering
  - Each process has message queue;
    nucleus manages pool of message buffers
- Basic Interface (send_msg, wait_msg, send_answer, wait_answer)
  - send (receiver process name, data, &buffer)
    - buffer is filled in by system with identify of message buffer
    - sender can continue immediately!
  - wait (&sending process name, &data, &buffer)

Example Protocol

Internal client wants to read from disk

send(disk, “READ from file A, offset B”, &buffer);

wait_answer(&result,&answer, buf_id)

// receives READ DATA

External device driver

while (1) {

wait(&sender, &msg, &buffer);

// receives READ request
//...initiates disk operation...

send_answer(result, data answer, buf_id)

}

Why is this better than “read(local_buffer)”?

Example Protocol: Questions

Internal client wants to read from disk

send(disk, “READ from file A, offset B”, &buffer);

wait_answer(&result,&answer, buf_id)

// receives READ DATA

External device driver

while (1) {

wait(&sender, &msg, &buffer);

// receives READ request
//...initiates disk operation...

send_answer(result, data answer, buf_id)

}

Why is the buf_id known for responses? Why is this useful? (Two reasons)

Example Protocol: Questions

Internal client wants to read from disk

send(disk, “READ from file A, offset B”, &buffer);

wait_answer(&result,&answer, buf_id)

// receives READ DATA

External device driver

while (1) {

wait(&sender, &msg, &buffer);

// receives READ request
//…initiates disk operation…

send_answer(result, data answer, buf_id)

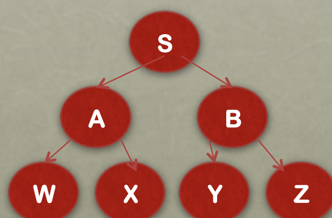
}

Why is it useful to have a separate “result” from “answer”?

Process Control

Any process can create a child process in its space and control its scheduling/execution

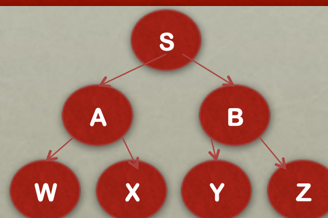
Any process can be an operating system (e.g., A and B)



B can swap Y and Z:
stop(Y);
output(Y);
input(C);
start(C);

Drawbacks to using this approach?
- Many context switches required across/down tree
- IPC can be expensive

Separate Policy and Mechanism?



Nucleus claims "no built-in strategy"

Can any scheduling policy be built?

What is their built-in strategy?

- Round-robin across all active processes

Some scheduling policies not compatible with RR base

- Example?
- What if Process W has real-time requirements and must be run every 100ms for 10ms?

Evaluation

What should a good evaluation of their goals contain?

1) Are they able to implement many different policies?

- See more attempts at this later in course

2) What are the costs of the IPC mechanism?

- IPC performance is fundamental to extensible systems
- Look at ratio between instruction and IPC time
- Instruction time?
  - 4us
- Round-trip communication time?
  - 2ms (2000 us)
- Ratio? How much slower is communication?
  - 500x slower! (Current systems worse ratio…)

Nucleus Lessons

Fundamental contributions

- Named processes
- Duality of internal and external processes
- Using messages for communication and synchronization

Key Results

- Replace process with another (interposition, extensibility)
- Ability to handle rogue, dead processes
- Ability to control child process from parent

Extensible OSes will be explored again and again...

UNIX

Who? Dennis Ritchie and Ken Thompson

When? Early 1970s

Goals?

- Cost effective, simplicity, elegance, ease of use

Major Contributions

- Widely used system, self-supporting
- C language
- File system with directory tree, inodes, links
- Shell (fork, exec, job control, pipes, redirection)

You should understand this as background material!

- Discuss internals of FFS file system next

UNIX Shells

How are Unix shells implemented?

```
while (1) {
    Char *cmd = getcmd();
    Int retval = fork();
    If (retval == 0) {
        // This is the child process
        // Setup the child's process environment here
        // E.g., where is standard I/O, how to handle signals?
        exec(cmd);
        // exec does not return if it succeeds
        printf("ERROR: Could not execute %s\n", cmd);
        exit(1);
    } else {
        // This is the parent process; Wait for child to finish
        int pid = retval;
        wait(pid);
    }
}
```

Conclusions

Lessons:

- Virtual memory and processes are elegant abstractions
- Need synchronization mechanisms between processes
  - Messages work well for modularity and relability

Next: Wednesday, Sept 11: Disks and FFS

Question: In what ways did the performance characteristics of the disks of the time influence the design of FFS?