The RC 4000 Nucleus and Unix

Robert Grimm
New York University

The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?

The RC 4000 Nucleus

Main Goal and Overall Structure

- Allow for many different “operating systems”
  - I.e., resource management policies
- Separate system into minimal nucleus and hierarchically nested policy implementations
  - Nucleus provides support for concurrent execution of programs and their interaction
  - Policies take care of scheduling and swapping

Process as Main Abstraction

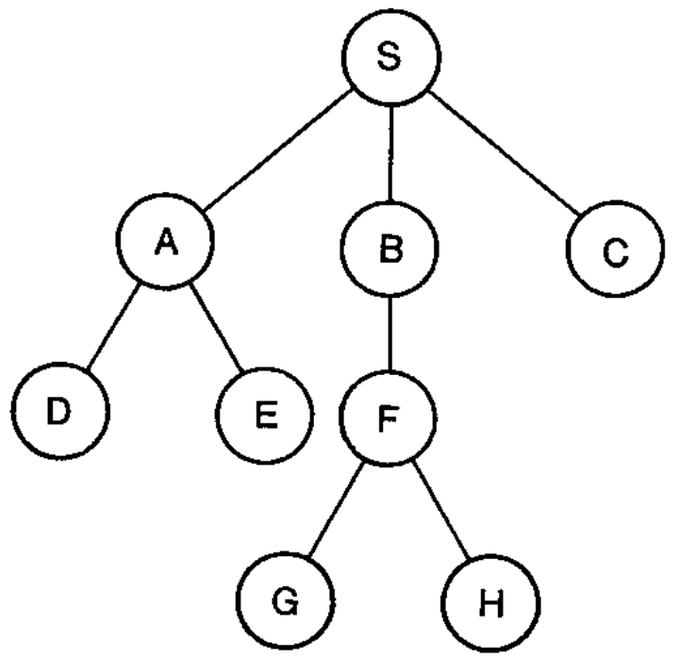
- Two kinds of processes
  - Internal: Execution of a program
  - External: Input/output from/to peripheral device
- Common interface
  - Unique name per process
  - Message passing for communication

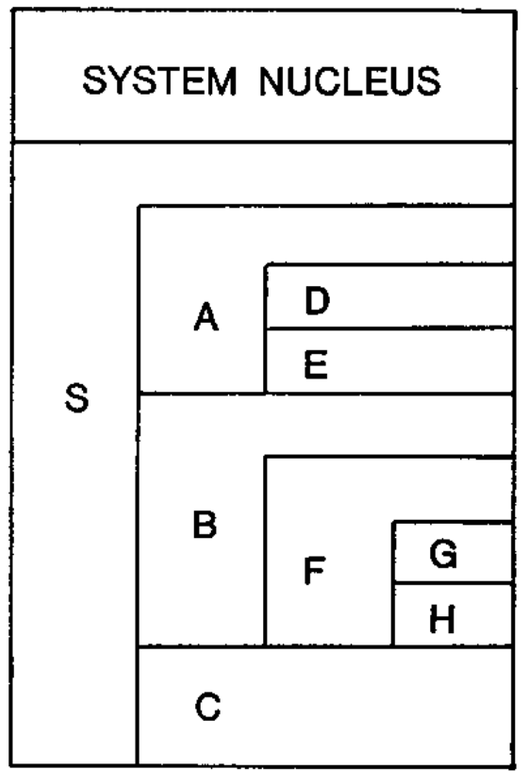
Operations

- Communication
  - Send/wait message & send/wait answer
    - Send is asynchronous (!)
  - Backed by buffers and queues in nucleus
- Creation
  - Internal: Assign name to memory region
  - External: Assign name to device
- Start & stop process
- Destruction

Process Hierarchy

- Parent provides memory, controls execution





- Nucleus
  - Schedules active processes round-robin
  - Supports communication between arbitrary processes

Unix

File System Is Central

- Ordinary files
  - Some arbitrary byte string
- Directories
  - Map names to files (“linking”)
  - Start with one root directory
  - Include themselves (“.”) and parents (“..”)
- Special files
  - To represent devices

File System Is Still Central

- Removable file systems
  - "Mounted" at any point in the tree by overlaying regular file
- Protection
  - Provided through read/write/execute permissions
    - For owner and all other users
  - Overridden through set-uid bit and super-user
- Interface
  - open, read, write, seek, close
    - Implicit cursor, no locking

File System Implementation

- Each directory entry maps name to file's i-number
  - Index into i-list identifying i-node, which contains metadata and (indirect) pointers to data
- Each application has table of open files (i-numbers)
- Mount table maps i-numbers to devices

Processes

- Created through fork()
  - Same core image and open files for parent and child
- Execute programs through execute()
- Communicate through pipe()
- Wait for children through wait()
- And terminate themselves through exit()

Shell Features

- Standard I/O
  - May be redirected
- Filters
- Multi-tasking
- Command files

Shell Implementation

- Standard I/O
  - Open input/output devices before creation
  - Fork, then execute command, which inherits I/O files
- Filters
  - Use pipes for standard I/O files
- Multi-tasking
  - Do not wait for child
- Command files
  - Redirect input *into* shell

So, What Made It Happen?

- Design for interactive use
  - When everyone else focuses on batch processing
- Keep it simple
  - Or, find “salvation through suffering”
- Make the system self-hosting
  - Or, eat your own dog food
- Really: Be smart and have fun
  - No deadlines, no interference from managers/marketers

Nucleus/Unix Smackdown

RC 4000 Nucleus vs. Unix

- RC 4000 Nucleus wins
  - IPC (Inter-Process Communication)
    - Between any processes, asynchronous
- Unix wins
  - Naming
    - One hierarchy with ability to mount new trees
  - File system
    - i-number and i-node organization
  - Shell
    - simple tools that can be easily composed