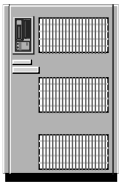
The Nucleus of a Multiprogramming System



Luke Levesque
COP 5611

Outline

- Introduction
- Nucleus Hardware
- Nucleus
- Message passing
- Processes
- Process Hierarchy
- Summary

Introduction

- Per Brinch Hansen of A/S Regnecentralen describes a new "OS" in development known as the Nucleus.
- Designers of system wanted flexibility.
- Wanted to break free of rigid OS structure.
  - More control.
  - Run multiple OSes or change OS easily.
- Created system 'nucleus' that can be extended with an OS.
  - Provides base functions.
- Runs on RC 4000 computer.

The RC4000

- RC 4000 is a 24bit system originally designed for real-time control (chemical plants, etc.) in ~1967.
- Has at least 16k-32k words of RAM.
- Supports a clock, TTYs, paper tape in/out, printer, magnetic tape, and a drum or disk.
- No real virtual memory since made for real-time use.
  - Processes can be swapped.
- Each word in RAM has a protection bit that must be set, so process creation/deletion and memory allocation are costly compared to other systems.

RC 4000 Pictures

RC 4000 Pictures

System Nucleus

- Multiprogramming and process communication handled by nucleus.
  - Not considered it's own process.
  - Provides only basic services.
- Semaphores are not considered reliable enough.
  - Bad programs could cause deadlocks.
- Instead, processes send messages to each other.
  - Nucleus provides the buffering and delivery services throughout system.
  - Each process has it's own queue (like MPI).

Message Operations

- send message (recv, msg, buf)
  - Copies message into available buffer and puts it in queue of receiver. Process then continues and may check status by looking at buffer.
- wait message (sender, msg, buf)
  - Causes process to sleep until a message arrives. The arguments are then filled in and the buffer is ready to be filled with an answer.
- send answer (result, answer, buf)
  - Copies answer to a message into a buffer (allocated from wait message) and puts it in the queue of a original sender.
- wait answer (result, answer, buf)
  - Causes process to sleep until an answer is present in the buffer referenced. The buffer is copied into answer and then freed. Result indicates if answer was a dummy or not.

Messages

- The primitives listed force a process to answer messages in a FCFS manner.
- Once a message has been received from a process(es), it can use the buffer to service requests from them in any order.
  - Did not know about process previously.
  - Can delay sending an answer (service in any order).
- When a process is terminated, it's messages remain in queues. Answers to them go to null and the buffer freed.
- When a process with messages in it's queue terminates, dummy answers are sent out automatically to those waiting.
- The system checks buffers when sending to avoid processes interfering with each other's messages.

Messages

- Finite pool of message buffers introduces resource problem.
  - Nucleus limits how many messages a process may send.
  - Answers always use existing buffer to conserve resources.

Message Example

- The consumer/producer problem described in class could be implemented without any semaphores using three processes:
  - Producer process messages buffer process with items.
  - Buffer process adds items to queue and removes them to pass on to consumer process.
  - Consumer process messages buffer to get new items.

Message Example

- Producer
  - LOOP
    - create item in temporary memory
    - send message() // Sends item to buffer
    - wait answer()
    - If error in answer, retry or exit
  - END LOOP
- Buffer
  - LOOP
    - wait message() // Wait for request from producer or consumer
    - if message from producer
      - Put item in queue
      - send answer() // Indicates success / failure of queue insertion
    - else
      - Get item from queue
      - send answer() // sends next item to consumer
  - END LOOP

Message Example

- Consumer
  - LOOP
    - send message() // To buffer, make request
    - wait answer() // Has item in message buffer
    - Process item
  - END LOOP
- Very simplified version!
  - Real version would have more error handling, better support for empty/full queue, ability to end processes, etc.

Processes

- Two major types of processes:
- Internal Process
  - The execution of a program in memory.
  - Has a unique name for reference by other processes.
  - What we typically think of for a process.
- External Process
  - Very similar to device drivers.
  - Interfaces with outside world.
  - Disks, terminals, real time clock, etc.
    - A 'document' is accessed within these (files, registers, etc).
    - Document is the external process, basically.
  - Also has a unique name.
  - Created on request by an internal process.

External Processes

- Nucleus considers element within device (document) the external process.
  - Nucleus contains code that works as a device driver.
- Internal processes use messages to communicate with external processes.
  - Actually communicating with Nucleus device drivers as well.
  - An internal process could be made as a 'go between' (or a complete replacement) with an external process if complex access methods or scheduling is required.
    - Done by giving the internal process the same name.
  - Internal processes would instead message the 'go between' internal process.

External Processes

- Internal processes can reserve or release external processes to insure exclusivity to documents.
  - Files, terminals, etc.
- TTYs only external process that can initiate a message (all others send answers only).
- Examples:
  - Individual files can be made external processes.
  - Messages to the clock process can synchronize (delay) processes.
  - Wait for an answer from a tape unit to know when a tape is mounted.
    - Could kick off a tape process.
  - TTYs are documents.

Internal Processes

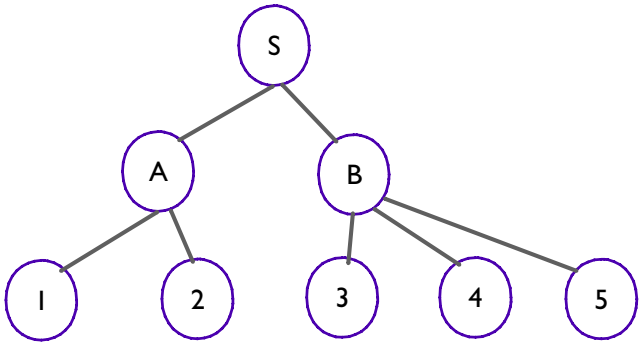
- Created on request by other internal processes.
- Procedure: Create, load, start, remove (when done).
- Process can be stopped (suspended). Messages or answers received are queued up.
- Internal processes are arranged in a hierarchy.

Process Hierarchy

- All internal processes are in a tree-like hierarchy.
  - Parents can create, load, start and terminate children.
    - Nucleus provides only basic services for process control (does NOT include loading).
  - Parents control resource allocation of children.
    - Parents can swap child processes in and out.
      - stop(A); output(A); input(B); start(B);
    - Parents own all resources of children.

Process Hierarchy

- S is a basic OS started by the nucleus.
- A and B can be 'real' operating systems.
- 1,2 and 3,4,5 are children of OSes A and B.



Process Hierarchy

- TTYs message S to have it start OSes (A and B).
  - A and B are really just programs.
  - The tree can extend as far as needed and can have more parents ('sub-operating systems'), etc.
- S (nucleus) has no strategy for resource allocation and scheduling.
  - Operating systems will fill in these gaps.
  - Parents allocate and reclaim resources of children as needed.
  - S has round robin scheduling for all ACTIVE processes in the tree.
    - Parents control children's CPU time by starting and stopping them.
  - All processes can send messages and answers to each other anywhere in the tree.

Process Hierarchy Rules

- Process can allocate a subset of resources to children.
- Process can only start, stop, and remove child processes.
  - Removal of process returns resources to parent.
- S owns all resources.

Development of new OSes

- The nucleus is helpful in creating new operating systems because
  - New OSes are created and ran just like any other program. Multiple OSes can be running at once.
  - OSes can be written in a high level language.
  - OSes can be replaced dynamically. Useful for testing, upgrades, etc.
  - Standard user programs can be ran under different OSes unmodified if there is an agreement on communications between the parent and children.

Summary and Questions

- Nucleus, using a process hierarchy, allows for multiple operating systems to be ran simultaneously via multiprogramming.
  - Parents own resources of children.
- Internal processes are standard programs, while external processes represent physical devices and files.
- P() and V() are replaced by message passing.
- Designed to increase flexibility of a computer system.
  - Core functions in Nucleus (Kernel), while everything else in processes.

References

- Per Brinch Hansen, **The Nucleus of a Multiprogramming System**. Communications of the ACM 13(4), April 1970, Pp. 238-w241, 250.
- P. Brinch Hansen, **The RC 4000 real-time control system at Pulawy**, BIT 7, 4 (1967), 279-288.
- RC4000 Pictures: http://www.prg.dtu.dk