9/28

- E. W. Dijkstra, The Structure of the 'THE'-Multiprogramming System, Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 341-346.

(Additional historical background on semaphores in Wikipedia.)

Q: Dijkstra explicitly states their goals for the THE operating system. How do these goals compare to, say, Microsoft's goals for the Windows operating system? Why do we no longer build operating systems with the same goals as THE?

- P. B. Hansen, The Nucleus of a Multiprogramming System, Communications of the ACM, Vol. 13, No. 4, April 1970, pp. 238-241, 250.

Optional related paper on a deployment experience of RC 4000:
P. B. Hansen, The RC 4000 Real-Time Control System at Pulway, BIT 7, pp. 279-288, 1967.

Q: How does synchronization in the RC 4000 system compare with synchronization in the THE system?

sephormer / signal send wait

① why paper — they are trying to build the system and they write down the experience.

② What is the main point of the paper?

OS structural using a central abstraction (sequential process) organized in a hierarchy (layers)

trying to unified —

So we can debug

Benefits? ① verify soundness ② Prove correctness

Assume the lower layer to be working.

Why? Handle complexity for simple system maybe we don't need this complexity

think about complexity, goal?

Trade off? Efficiency, overhead if we only have one Application?

③ What layers do we have?

0) Process control, scheduling
- abstract away the processor

1) Memory management, paging

- separation of data (segment) from storage location (core, drum) and address (re-location)

2) console, message interpreter.
e.g. tty, virtual consoles

3) Peripherals, buffering, stream (virtual device)

4) User level programs

? 5) operator → 手動的

Advantages ① modularity, abstraction
② debugging ③ verification

DisAds? efficient
circularity — what is schedule (queues) page out?
Design issue → it will use physical, never page out. (linux) clx → pager
window pager's memory

Against? NT Graphics moved from user subsystem → Kernel.

Relative performance — it is essential that you have a good intuition for the relative performance of hardware.

① abs #s
② change over time
③ relative change to other h/w.

e.g. hard drive 5x
CPU ~ 75,000,000 x

Synchronization P↓ V↑

① mutexal sync
② waiting queue (init to some certain number)

Summary

① Layered OS

② General abstraction: sequential concurrent processes

③ Semaphores for synchronization

④ Correctness (proof + testing)

Nucleus

Interesting
- Howden (faculty at UCSD)?
- Grisnold (his tutler)

Motivation
- Existing OS structure inhibit "freedom of design"
- Need a new structure to support this goal

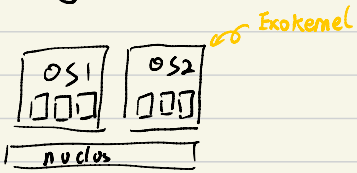
How?
- Small nucleus (kernel, today microkernel) for support multiple simultaneous operating system implementations

Reverse Trend
Nucleus tried to make things general and flexible

Opposite
make things more specific

Structure
- two layers?
  Nucleus, user-level

- three part of the nucleus
  scheduling, communication, primitives for manipulating process



Communication
Synchronization

Ads — less likely to deadlock.

NisAds — copy overhead. limited buffer. Can overl

Resource Allocation — ①

②

Schedule — Round Robin
Not a good Design.

implement size — 6200 24-bit words = 150 Kbytes

Summary
① How to be flexible?
② structure oms: nucleus on which OSes can be layered.
③ syn and comm via message passing.
④ small # of clean abstractions used thru sys.

Small stuff.