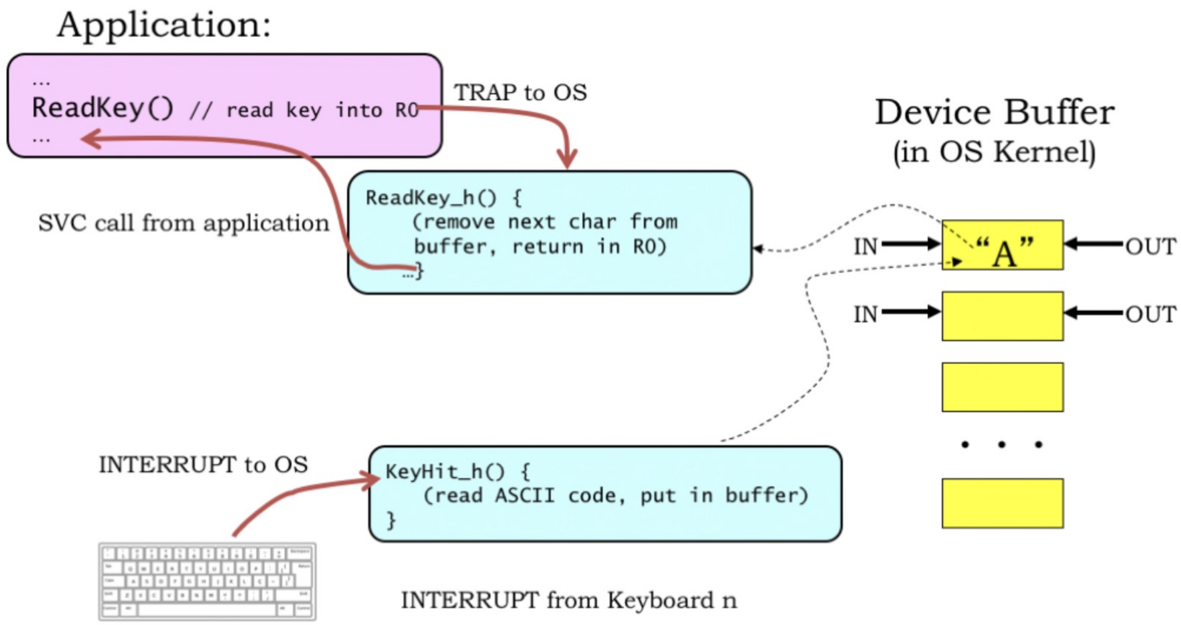
Ch 18

# Asynchronous I/O Handling




# Sophisticated Scheduling

To improve efficiency further, we can avoid scheduling processes in prolonged I/O wait:

- Processes can be in ACTIVE or WAITING ("sleeping") states;
- Scheduler cycles among ACTIVE PROCESSES only;
- Active process moves to WAITING status when it tries to read a character and buffer is empty;
- Waiting processes each contain a code (eg, in PCB) designating what they are waiting for (eg, keyboard N);
- Device interrupts (eg, on keyboard N) move any processes waiting on that device to ACTIVE state.

# The Need for "Real Time"

Side-effects of CPU virtualization
+ abstraction of machine resources
(memory, I/O, registers, etc. )
+ multiple "processes" executing concurrently
+ better CPU utilization
- Processing throughput is more variable

Our approach to dealing with the asynchronous world
- I/O - separate "event handling" from "event processing"

Difficult to meet "hard deadlines"
- control applications, e.g., ESC on cars
- playing videos/MP3s

Real-time as an alternative to time-sliced
or fixed-priority preemptive scheduling

Strong Priority Ordering

- allow handlers for lower priority interrupts to be preempted by higher priority requests

# Summary

Device interface – two parts:

- Device side: handle interrupts from device (transparent to apps)
- Application side: handle interrupts (SVCs) from application

Scheduler interaction:

- "Sleeping" (*inactive) processes waiting for device I/O
- Handler coding issues, looping thru User mode

Real Time constraints, scheduling, guarantees

- Complex, hard scheduling problems – a black art!
- Weak (non-preemptive) vs Strong (preemptive) priorities help…
- Common real-world interrupt systems:
  - Fixed number (eg, 8 or 16) of strong priority levels
  - Each strong priority level can support many devices, arranged in a weak priority chain