Real-time Operating Systems and Systems Programming

Interrupts, Signals

λ ARVUTITEADUSE INSTITUUT
DEPARTMENT OF COMPUTER SCIENCE

Interrupts (recap)

- A way of handling input/output
- Used for other things too
- Hardware must have support
- Interrupt tables with handling routines
- Can be generated in software

Use in security

- Protected mode relies on interrupts
- Memory pages for User and Privileged data
- Interrupts switch from user to superuser

Interrupt predictability

- Problems in debugging: difficult to predict
- Synchronous (arrive when you are ready)
  - Predictable
- Asynchronous (arrive from external sources)
  - Unpredictable
- Latter the reason for criticizing use in life-critical embedded applications.
- Still effective for less risky multi-tasking

Critical data

- Shared data may cause corruptions
- Possible solutions:
  - Disable interrupts
  - Serialize access

Example: Buffered IO

- Control taken from user
- Software buffers, error handling
- Data is waiting in buffer, program thinks it's sent
- When data sent/received – interrupt is generated

Trap & System calls

- Processors provide syscall *n* – trap instruction
- System calls encode arguments, execute syscall to run service *n*
- Then system call decoded & executed on kernel level
- Seem identical to normal functions to programmer
- man syscalls for complete list

Signals

- Interface to interrupts & other conditions on user level
- Sent for 2 reasons:
  - Kernel has detected an event such as divide-by-zero error, illegal memory access etc
  - A process uses kill() system call to send a signal. Can be sent to process itself (shortcut: raise()).

Life of a signal

- *Pending signal* – signal was sent, not received
  - At most one pending signal of any single type for a process, others discarded
- Blocked signals wait
- Ignored signals are discarded
- Other signals are delivered to the process (only once).

Received signal

- Each signal has predefined default action:
  - Process terminates
  - Process termiantes and dumps core
  - Process stops until restarted by SIGCONT signal
  - Process ignores the signal
- Process, if still there, continues where it was.
  - Unless a system call was interrupted, which sometimes is an error

Signal handling issues

- Pending signals are blocked while handling the same type of signal.
- Pending signals are not queued: if one SIGINT is already pending, other is discarded
- System calls are interrupted on some systems (read(), wait(), accept(), set errno to EINTR)
- For more portable signal handling sigaction() is defined in Posix-compilant systems

Blocking signals

- Processes can explicitly block and unblock signals with sigprocmask() function.
  - sigprocmask(), sigemptyset(), sigfullset(), sigaddset(), sigdelset(), sigismember() - returns 1 if member, 0 if not, -1 on error

Signal handling patterns

- Make the handler as small as possible, possibly changing only one global variable
- Block some of the signals during handling
- Handler which quits might do only cleanup, then re-assign default action to current signal, then raise() the same signal again.