Clock and timer

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Previous lectures

- Device management
- Interrupt vectors
- Scheduling during interrupts
- Device interface; device switch table
- Tty device

Tty device history

A tty device gets its name from the very old abbreviation of teletypewriter and was originally associated only with the physical or virtual terminal connection to a Unix machine. Over time, the name also came to mean any serial port style device, as terminal connections could also be created over such a connection.

Some examples of physical tty devices are serial ports, USB-to-serial-port converters, and some types of modems that need special processing to work properly (such as the traditional WinModem style devices). tty virtual devices support virtual consoles that are used to log into a computer, from either the keyboard, over a network connection, or through a xterm session.

https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch18.html

Recall: Tty driver complexity

- Hardware: onboard FIFO
- Software features:
  - Modes (raw, cooked, and cbreak)
  - CRLF mapping
  - Input character echo
  - Visualization of echoed control characters (e.g., ^A)
  - Editing, including backspace over control characters

Reading from terminal in Linux without echoing

```
#include <unistd.h>

int main() {
  char c;
  while (read(STDIN_FILENO, &c, 1) == 1 && c != 'q');
  return 0;
}
```

https://viewsourcecode.org/snaptoken/kilo/02.enteringRawMode.html

Reading from terminal in Linux

```
#include <termios.h>
#include <unistd.h>

void enableRawMode() {
  struct termios raw;
  tcgetattr(STDIN_FILENO, &raw);
  raw.c_lflag &= ~(ECHO);
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

int main() {
  enableRawMode();

  char c;
  while (read(STDIN_FILENO, &c, 1) == 1 && c != 'q');
  return 0;
}
```

https://viewsourcecode.org/snaptoken/kilo/02.enteringRawMode.html

Recall: tty functions

Upper-Half	Lower-Half
ttyinit	ttyhandler (interrupt handler)
ttyopen	ttyhandle_in (input interrupt)
ttyclose	ttyhandle_out (output interrupt)
ttyread	
ttywrite	
ttyputc	
ttygetc	
ttycontrol	

Xinu code

Clock and timer management

V2: Clock support

- Time stamp counter
  - Counts the number of CPU cycles since reset
  - V2: RDTSC instruction in x86 (*not RTSC*)
- Real-time clock
  - Generate periodic interrupts
  - Called programmable if rate can be controlled by the OS
  - Intel 8253/8254, or High Precision Event Timer (HPET)

Two principles types of timed events in Xinu

- Preemption event:
    - Implements timeslicing by switching the processor among ready processes
    - Guarantees a given process cannot run forever
    - Scheduled during context switch
- Sleep event
    - Scheduled by a process
    - Delays the calling process a specified time

Managing real-time clock events

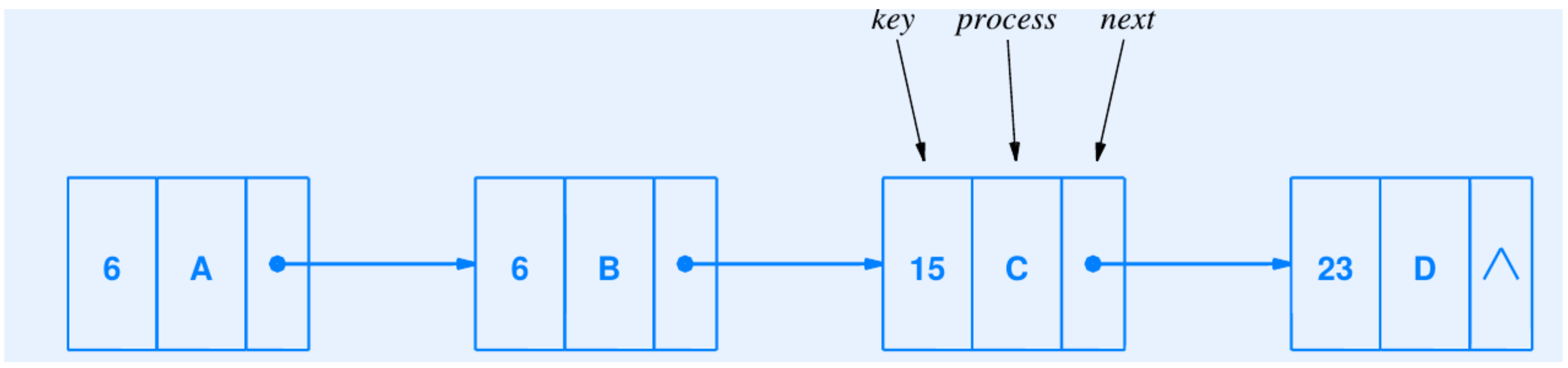
- Must be efficient
    - Clock interrupts occur frequently and continuously
        - 1.190 Mhz in Xinu, 1ms interrupt rate
    - Set of timed events examined on each clock interrupt
    - Should avoid searching a list
- Mechanism
    - Keep timed events on a linked list
    - One item appears on the list of each outstanding event
    - Called event queue

Delta list

- Data structure used for timed events
- Items on the list are ordered by time of occurrence
- The key in an item stores the difference (delta) between the time for the event and time for the previous event
- The key in first event stores the delta from now

Delta list example

- Assume events for processes A through D will occur 6, 12, 27, and 50 ticks from now
- The delta keys are 6, 6, 15, and 23



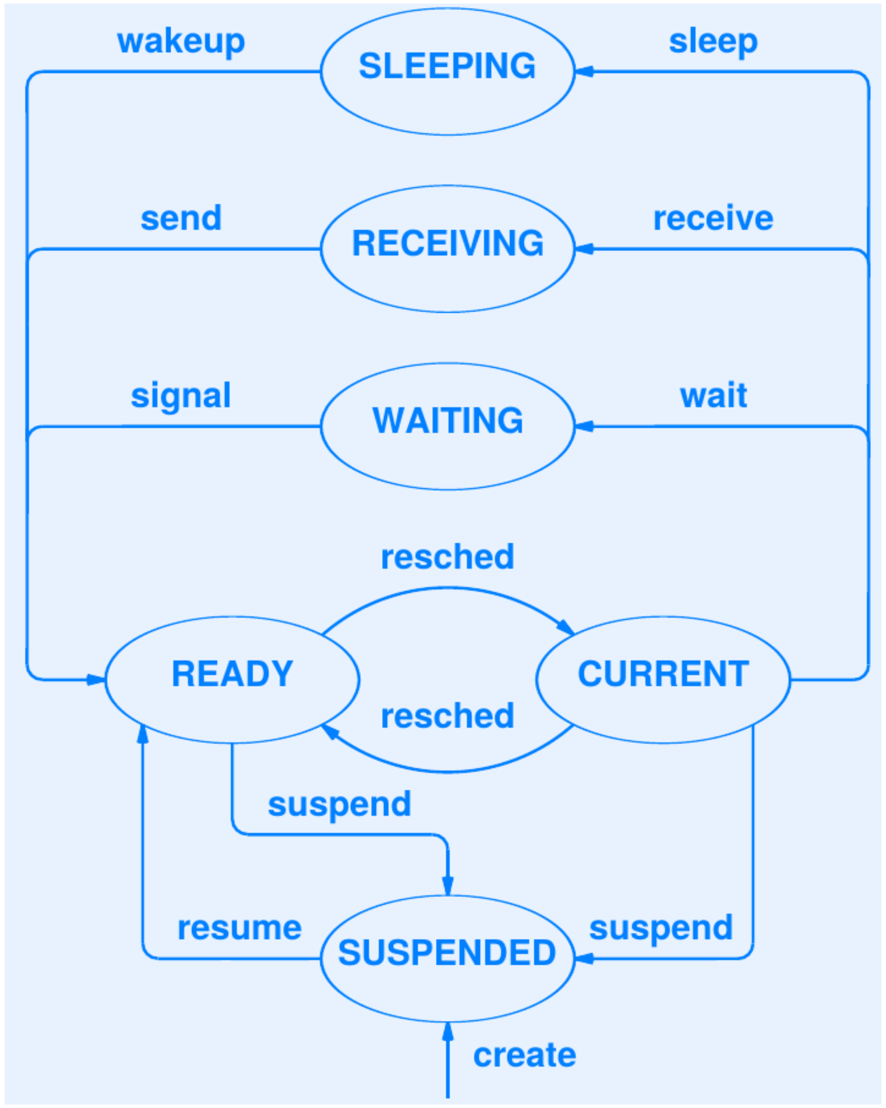
Keys on the Xinu Sleep Queue

- Processes on **sleepq** are ordered by time at which they will awaken

- Each key tells the number of clock tics that the process must delay beyond the preceding one on the list

- Relationship must be maintained whenever an item is inserted or deleted

New process state



Xinu sleep functions

- Current process calls **sleepms** or **sleep** to request delay
- **sleep()**
  - Delay the calling process n seconds
- **sleepms()**
  - Underlying function that performs an actual sleep operation
  - Inserts current process to a sleep queue **sleepq** using **insertd()**
  - Turns the current process into PR_SLEEP state.
  - Calls **resched** to allow other processes to execute

Initializing the clock in Xinu

- Initialize the preemption counter
- Initialize a clock hardware (Intel 8254) and a clock interrupt vector
    - See `clkinit()` and `set_evec()`

Clock interrupt handler

- Decrements preemption counter
  - Calls **resched** if counter reaches 0
- Decrements the count of the first element of **sleepq**
  - Calls **wakeup**() if counter reaches 0

Operation timeout

- Many operating systems components need a timeout feature
- Especially useful in building communication protocols
- Xinu provides a single facility for timeout
  - Using timed message reception

Timed Message Reception

- Implemented in **recvtime**()
  - Waits a specified time to receive a message and return
  - **prstate** becomes **PR_RECTIM**
  - Argument specifies maximum delay
  - A call to **recvtime**() returns
    - If a message arrives before the specified timeout
    - After the specified timeout if no message has arrived
  - Value **TIMEOUT** is returned if time expires

Xinu code

- recvtime()
- sleepms() and sleep()
- wakeup()
- clkinit() and set_evec()

Summary

- Device management
- Clock and timer management
  - Timestamp counter and real time counter
  - Timed events: preemption and sleep
  - Operation timeout