CS7600
Peter Desnoyers
pjdl@cs.s.ned.edu
Room: 334

Office hours
Mon: 12:30 to 2 pm

- Material on Blackboard

- Check the syllabus on blackboard
- Midterm before Spring break
- Videos online on course website for reference
- Collaboration is allowed for programming assign
- Non-programming no collaboration allowed
- Grading: h/w 1 deadline by midnight
  15% taken off for every day delay
- Free extension
- For conferences please contact earlier.

- Main goal is to understand "how computer works?"
  e.g. the mechanism of pressing a key and display on monitor

- Focus on action rather than structure
- Sequence of things are is important
- How to do system Research
- Mathematical analysis and benchmarking
First half of Semester: threading & processes
- OS basics: pgm loading, context switching
  handled in asgn 1.

2. Synchronization:
   - Synchronizing threads using
     'symphony'.
   - The mathematical model of synchronizing

3. Virtual Memory

4. Virtualisation: h/w virtualisation like
   VM ware

Mid Term

# Second half

- I/O, block devices
- File systems
- Security
- Benchmarking (hopefully)

# Final on second half (major) & some
content of first.

- What is operating system?
  - Linux
  - DOS
  - minix
  - Windows
  - uCOS UCos
  - Symbian
  - mac OSx
  - Android
  - Hurd
  - Solaris
  - TinyOS
  - nachos
  - BSD
OS

1. interface with H/W
2. manages resources.

- Thing that goes between pgm & H/W
  pgm — [ ] — H/W

- sort of virtual mlc that talks to the API (not the device drivers) to open a function call.
- facilities to load pgm to design h/w with s/w to use I will be a hard mlc.
- manages resources of which part of memory to be used
- avoiding two different processes should not access the same memory location.

* Toy Computer
  - 16 bit processor with number of registers
  - 640 K of mem
  - 60 of 64K is for RAM
  - 4 K for various I/O devices
- Frame Buffer

- Display the characters on each mem location 24

So \((F7FE)\) may be

\[ \text{FOOD} + 24 \times 80 \]

Simple Program

\[
\begin{align*}
\text{begin:} & \quad \text{mov} \quad \text{FOOD}, \quad R3 \\
\text{begin:} & \quad \text{mov} \quad \& \text{Sty}, \quad R4 \\
& \quad \text{mov} \quad \text{count}, \quad R2 \\
\text{loop:} & \quad \text{mov} \quad \ast R1++, \quad R4 \\
& \quad \text{mov} \quad R4, \quad \ast R3++ \\
& \quad \text{sub} \quad R2, \quad 1 \\
& \quad \text{jmp} \quad \text{NZ, loop} \\
\text{done:} & \quad \text{jmp} \quad \text{done}
\end{align*}
\]

\[(\text{count}) = 11\]

\[\text{str = "Hello World!"}\]

* Enhancing the above program use a function \texttt{putchar}:

\[
\begin{align*}
\text{Put char:} & \quad \text{mov} \quad R5, \quad 0 \\
\text{Change begin:} & \quad \text{mov} \quad \text{FOOD}, \quad R3 \\
\text{loop:} & \quad \text{mov} \quad \ast R1++, \quad R4 \\
& \quad \text{Call putchar}
\end{align*}
\]
This program is entangled closer with the compiler and linker.

Library OS
+ abstracts HW complexity
- executable + OS linked

- block some part of mem with OS and use it for function

- By doing this we have separated
  Library & Pgm

- functions that don't vary between pgms

- when the diff locations are used in diff hw then it becomes closely related
indirection - jump table

- The location of the functions is agreed by all (DOS)

- INT XX 2

  like - keyboard handling
  - open files
  - write to screen.

- also used in early version of MAC

- Today we learned about just DOS

* Use of stack in function call

  \[ x = f(10) \]

  \( \downarrow \)
  \>

  save the argument \( 10 \) (pre-decrement the top of stack)
push the return address

- make space for local variables depends on the no. of local variables

```c
for (x) &
int a, b;

return s;
```

- The return values can be stored in some pre-defined place (register) to hold this value

like EAX

# Interrupts
ability to stop what it is doing in response to some external event and execute a piece of prog

- some specific location in the memory will be dedicated to every particular interrupt.

  e.g: 1) when a key is pressed:
  call all the associated interrupts

  2) A timer interrupt -
  e.g: the track of time consumed by a program to execute
and do some actions like sleep.

3. TinyOS: track of the temperature
   - To handle the interrupt, the same mechanism for function call can be used.
   - One should also save the current state of the program so that the program doesn't know that it was interrupted.

```c
// Program

(read char)

INT x13 table
```

[Diagram with keyboard input connected to an INT function]
Context Switching

Two separate pieces of code running simultaneously successfully.

- Linear sequence of code running in background.

- A function \( f() \) may wait for \( g() \) to complete its part.

- Two functions handle scheduling func back & forth.

\[
\begin{align*}
f() & \quad g() \\
yield f - g() & \quad yield - g \cdot f() \\
\end{align*}
\]
The stack & registers are at stack for this.

When yield is called:

1. \( f(c) \) \( \rightarrow \) \( A \) \( \leftarrow \) \( \text{SP} \) \( \uparrow \text{yield} \)
2. \( f(c) \) \( \rightarrow \) \( \text{PC} \) \( \leftarrow \) \( \text{SP} \) \( \downarrow \text{yield} \)
A timer interrupt is just like the above.