Goals

• Study some of the fundamental problems and solutions of OS

• Work on a small toy OS, PintOS
  • Extend functionality
  • Add functionality

• Project based course
Course Setup

• Lectures, notes and readings provide an exposition of the problem(s) and their solutions

• Projects rely on you to apply (as-is, modified, or your own) solution to PintOS

• Course staff treats students as professional developers
Course Evaluation

• 4 projects work as teams of 3

• You are professional developers
  • Clear and Clean and documented source code
  • Documentation for your solution
    • Design and reasons for your choice
    • Analysis (space, time)
    • Concurrency (starvation, liveness)
Hard Work

• There is a lot of work!

• We will expose some of the inner workings
  • what most classes up to know relied upon as “black box”

• Low lever (architecture, assembly)

• Interesting and hard problems
  • Concurrency, Garbage Collection etc.
Hard Work

• Assignments will take time and effort (2nd onwards)
  • understand PintOS code
  • some of the standard C libraries that you know are not available in PintOS (some close replacements exist)
  • work with some less that intuitive tools
  • “live on the command line”
Assumptions

• Medium level familiarity with C
  • structs, pointers, malloc, free etc.

• See last Fall’s class web site (http://www.ccs.neu.edu/home/skotthe/classes/cs5600/fall/2015/labs.html)
  • Labs 2 and 3, Assignments 1 and 2.

• Basic understanding of shell and OS
  • process, pids, pipes, files, directories etc.

• Basic understanding of linking loading and execution
  • object files, libs, linking, execution stack etc.
Logistics

• Class Web Site:
  • http://www.ccs.neu.edu/home/skotthe/classes/cs5600/fall/2016/index.html

• Piazza (no communication outside of Piazza)
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Compile
   gcc -O0 loop.c
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Compile
   gcc -O0 loop.c

gcc -E
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Compile
   gcc -O0 loop.c
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Compile
   gcc -O0 loop.c

gcc -E
gcc -S
gcc -o -l
a.out
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Compile
   gcc -O0 loop.c

Compile

loop.c
pre-process
assembly
link

gcc -E
gcc -S
gcc -o -l

a.out
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
gcc -O0 -E loop.c

int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++) {
        sum = sum + i;
    }
    return sum;
}
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
Run
Run

......

fork

process

shell

shell
Run

fork

shell

process

a.out

load

shell
Run

shell ➔
fork

process ➔
exec

load

shell ➔
terminate

a.out ➔
load
Run

fork

shell

process

exec

load

a.out

process

shell

terminate
int main(void) {
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}

1. Run
./a.out

What type of a file is a.out?
int main(void) {
  int sum = 0;
  int i;

  for (i = 0; i < 10; i++){
    sum = sum + i;
  }

  return sum;
}

1. Run 
./a.out

What type of a file is a.out?

therapon@vdebian:~/Lectures/01$ file a.out

a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=f725959a7360fc04d6cc7190ccc4a0428946c217, not stripped
ELF

• **Executable and Linkable Format**

• A way to organize the information in the file

• One header and data

• Data is made up of
  
  • Program header table, describes 0 or more segments
  
  • Section header table, describe 0 or more sections
  
  • Data referred to by entries in the program and section header tables
therapon@vdebian:~/Lectures/01$ readelf -h a.out

ELF Header:
Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: UNIX - System V
ABI Version: 0
Type: EXEC (Executable file)
Machine: Intel 80386
Version: 0x1
Entry point address: 0x80482d0
Start of program headers: 52 (bytes into file)
Start of section headers: 3616 (bytes into file)
Flags: 0x0
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 8
Size of section headers: 40 (bytes)
Number of section headers: 30
Section header string table index: 27
The “black boxes”

• How do we go from file to running program?

• How do we manage a running program?
  • stack, access to files, devices, memory, etc.

• How do we provide isolation and communication between programs?
  • inter-process communication, privileged access to devices

• How do we get the OS to run and who manages the OS while running?
It’s all 0s and 1s

• We will see
  • how a machine boots up
  • how processes are created and managed
  • how code is loaded, executed and managed (memory management)
  • access to hardware features (shared, privileged etc)
  • filesystem