CSE 451: Operating Systems

Tom Bergan, TA

Why operating systems?

- OSes provide a fundamental service
  - resource *sharing* (cpus, disks, network, etc...)
  - resource *abstraction*
- More than just windows/linux
  - Java VM
  - web browsers

    ...
- Techniques are widely applicable
  - data structures, caching, **concurrency**, ...

What is this section for?

- Projects
- Questions!
  - please bring questions!
- Some extensions of the lectures / textbook
- Other resources:
  - discussion board (see course webpage)
  - office hours

Today

- ~~Introduction~~
- Vote on office hours
- C review
- Project 1 tips

Office Hours?

(room TBD)

- Monday 11:30 - 12:30 (right after class)
- Monday 1:00 - 2:00
- Monday 2:00 - 3:00
- Tuesday 2:00 - 3:00
- Wednesday 11:30 - 12:30 (right after class)

Why learn C?

- Because the Windows kernel is written in C . . .
  . . . and our projects use Windows

- OSes can be written in any language, e.g.:
  - LISP (see the LISP machines)
  - C# (see Microsoft Research's *Singularity* OS)

- Why use C for OSes?
  - historical reasons (other languages weren't fast enough)
  - ➡ precise control over memory layout

    C's biggest strength and weakness

C vs Java: constructs

Java

```
import java.xyz;
```

Packages

```
class Point {
   public int x;
   public int y;
```

Classes

Methods

```
   public int foo(int a) {
      ...
      Point p;
   }
```

References

C

Header files

```
#include "xyz.h"
```

Structs
- all members public

```
struct Point {
   int x;
   int y;
};
```

Functions

```
int foo(int a) {
   ...
   Point* p;
}
```

Pointers

Pointers

```
int a = 5;
int b = 6;
int *pa = &a; // declares a pointer to a
              // with value as the
              // address of a

*pa = b;      // changes value of a to b
              // (a == 6)

pa = &b;      // changes pa to point to
              // b's memory location (on
              // stack)
```

Pass-by-value vs. Pass-by-pointer

```
int foo(int x) {
  return x + 1;
}
```

```
void bar(int* x) {
  *x += 1;
}
```

```
void main() {
  int x = 5;
  int y = foo(x);
        // x==5
        // y==6
  bar(&x);
        // x==6
        // y==6
}
```

by-value

by-pointer

What can pointers point at?

- Local (“stack”) memory

```
void foo() {
  int a;
  int* p = &a;
```

< exists until the function returns

- Global memory

```
int g;
void foo() {
  int* p = &g;
```

< always exists

- Dynamic (“heap”) memory **(more on this later)**

```
void foo() {
  int* p = malloc(sizeof(int));
  free(p);
```

^ exists until free()’ed

Function Pointers

```
int some_fn(int x, char c) { ... }
    // declares and defines a function

int (*pt_fn)(int, char) = NULL;
    // declares a pointer to a function
    // that takes an int and a char as
    // arguments and returns an int

pt_fn = &some_fn;
    // makes pt_fn point at some_fn()’s
    // location in memory

int a = (*pt_fn)(7, ‘p’);
    // calls some_fn and stores the result
    // in variable a
```

Arrays

- Arrays are just pointers

```
void foo() {
  int a[100];    // allocates a 100 elem array;
                 // a is a pointer to the
                 // beginning of the array

  a[1] = 5;      // the second elem in the
                 // array is set to 5

  *(a+1) = 5;    // same as the above, but uses
                 // pointer arithmetic
```

- Don't use pointer arithmetic unless you have a good reason to!

Common C Pitfalls (1)

- What's wrong and how to fix it?

```
char* city_name(float lat, float lon) {
   char name[100];
   ...
   return name;  < name is invalid after return
 }
```

- **Problem:** returning pointer to local (stack) memory

Common C Pitfalls (1)

- **Solution:** allocate "name" on the heap

```
char* city_name(float lat, float lon) {
   char* name = malloc(100 * sizeof(char));
   ...
   return name;
 }
```

Common C Pitfalls (2)

- What *could* be wrong? (similar to prior example)

```
void foo() {
  int tmp[100];
  int y = some_fn(&tmp);
  ...
  return;
}
```

< tmp is invalid after return

- **Problem:** some_fn() might save the address of tmp in a global:

```
int* g;
int some_fn(int* a) {
  g = a;
```

Common C Pitfalls (3)

- What's wrong and how to fix it?

```
void foo() {
   char* buf = malloc(32);
   ...
   print(buf);
   return;        < didn't free buf
}
```

- **Problem:** memory leak

Common C Pitfalls (3)

- **Solution:** call "free(buf)" before "return"

```
void foo() {
   char* buf = malloc(32);
   ...
   print(buf);
   free(buf);   // fix memory leak
   return;
 }
```

Common C Pitfalls (4)

- What's wrong and how to fix it?

```
void foo() {
   char* buf = malloc(32);
   ...
   free(buf);
   print(buf);
   return;
 }
```

< called free() too soon

- **Problem:** use-after-free

Common C Pitfalls (5)

- What's wrong and how to fix it?

```
struct Foo {
  int x,y;
}
void foo() {
   Foo* foo = malloc(sizeof(Bar));
   foo->x = 1;
   foo->x = 2;
   ...
 }
```

^ used wrong type in sizeof

- **Problem:** bad allocation

Common C Pitfalls (5)

- **Suggested idiom:** use sizeof(*foo)

```
struct Foo {
  int x,y;
}
void foo() {
   Foo* foo = malloc(sizeof(*foo));
   foo->x = 1;
   foo->x = 2;
   ...
 }
```

Project 1

- Goals
  - get acquainted with Virtual PC
  - get acquainted with the NT kernel
- Done alone
  - Projects 3 and 4 can be done in groups of 2
- **Don't** use local hard disks of the lab machines for permanent storage!
  - use Z:
  - if you run out of space (probable: virtual disks get big), make a directory for yourself in

    `o:\unix\projects\instr\11wi\cse451`

Project 1

- Making a VM image
  - walkthrough posted on the course website
- Editing the virtual disk
  - you can drag/drop from Explorer running on your workstation to Explorer running on Virtual PC (really cool)
- What if you can't boot your VM due to a kernel bug?
  - use the "mount" command (see `project1/Wrk.cmd`)
  - allows you to mount virtual disks on your workstation
    - .... should show up as a drive (e.g., "E:")
    - .... currently doesn't work (stay tuned)

Project 1

- Debugging
  - use the "winbag" command
  - this allows you to debug the NT kernel using a Visual Studio-like debugger (really cool)