Memory Management Background:
1. Computer System Review
2. Address Binding & Linking

ECE595

Sep 27

Y. Charlie Hu

"Things related to memory" -- you have learned/heard so far

- What is a processor
- What are registers
- What is memory
- How's memory organized
- What's a heap?
- What's a stack?
- Globals, locals, etc.
- PC, SP

- All of above deal with logical memory!

- Hardware memory
- What's a cache and how's it organized
- Physical memory a whole new can of worms!

1

Warning! You May Be Bored

- This material may be redundant if
  - You've already had it (but may have forgotten)
  - You already hacked and found it
  - Your first language was assembly
- Feel free to …
  - I won't be offended
  - You'll still be held responsible for the material

2

Warning: Approximate Truth

- Some details for general info
- Most details ignored entirely
- Goals
  - Simplicity
  - Coverage
- C, Unix, Uniprocessors, No Threads

3

What does a Processor Do?

while (1)
  fetch (get instruction)
  decode (understand instruction)
  execute

Execute: load, store, test, math, branch

4

Logical Organization

Logically

F1 D1 E1 F2 E2 D2

5

Processor Operations

Logically

F1 D1 E1 F2 D2 E2

Pipeline

F1 D1 E1
F2 D2 E2
F3 D3 E3

What is the condition for a smooth pipelining?
What can happen to the E part of Load inst: LD R0, _Y

6

What Is Memory (Address Space)

- "Slots" that hold values
- Slots are "numbered"
- Numbers are called addresses
- Two operations – read or write
  - e.g., LD R1, _X
- What can you put in memory?
  - Anything. No "intrinsic" meaning

7

What is Cache?

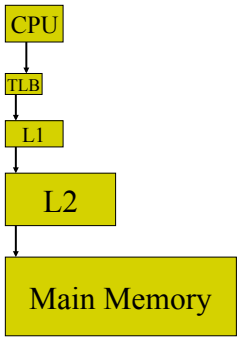
- Another kind of "memory", physically
- Closer to the processor
- More expensive, smaller, faster

- Operation is logically *transparent*
  - No naming/access by
    - program, compiler, linker, loader
    - OS?
    - CPU?

8

(Hardware) Memory Hierarchy

Where do we hope requests get satisfied?
e.g. LD R1, _X



9

What Are Registers?

- Places to hold information
- Built into the processor
- "Named" specially

Why?

- Need a place to put operands / temp values
  - e = (a+b) * (c+d) * (a-f)
- Highest level of memory hierarchy
- Register allocation problem – NP-complete
  - who does it?

10

What Is a Program?

```
int *totalPtr;
Init(void)
{
   totalPtr = calloc(1, sizeof(int));
}
AddToTotal(int y)
{
   int i;
  *totalPtr += y;
}
```

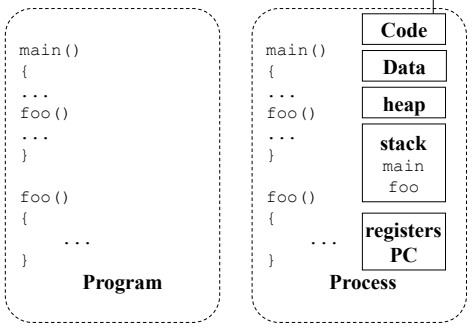
11

What Is a Program?

- Code
  - Main, subroutines (lib functions)
- Data
  - Static, global variables
  - Dynamically-allocated data *(e.g. ,malloc())*
  - Parameters, local variables
- What is a process?

```
int *totalPtr;
Init(void)
{
    totalPtr = calloc(1,
    sizeof(int));
}
AddToTotal(int y)
{
    int i;
   *totalPtr += y;
}
```


[Ice3] Program vs. Process



13

Everything Becomes Memory

- Various ranges of memory (addr space) are ***used*** for different purposes
  - Text/Code (program instructions)
  - Data (global variables)
  - Stack (local variables, parameters, etc)
  - Heap (dynamically allocated memory)

What Is a Stack?

- Data structure that supports push/pop
- Uses?
  - Anything w/ LIFO (last-in first-out behavior)
  - Only care about recent behavior
- Example?
  - DFS
  - Procedure calls!

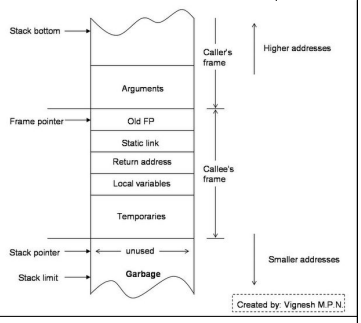
Procedure calls

- Incoming parameters from caller
  - Don't even know who caller is
- Local variables survive only when in use
- Temporary variables (a+b) * (c+d)

```
void Loop(int N)
{
    int a,b,c,d,e,f,g;
    ...
    g = (a+b*(c+d);
}
```


Stack Frames

- Frame = info for one procedure call
- Incoming parameters
- Return address for calle
- New local variables
- New temporary variable
- Size of frame



- Where are the instructions that perform these? Who generate them if you program in C?

Stack Is Just Memory

- Defined, used by convention
  - Agreement among OS, compiler, programmer
- How does OS manage stack (and heap)?
  - Allocate chunk of memory
  - Have pointer into chunk
- Problems?
  - Must know maximum size of stack?
- How to stay efficient despite uncertainty?

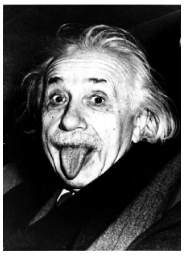
What Does Memory Look Like?

- Logical memory
  - Code+data, stack, heap
  - Which ones grow?
  - How do you give them the most flexibility
- Physical memory?
  - Another can of worms, entirely
- We will move on to Memory Management

Fun Trick

- What does this program do?

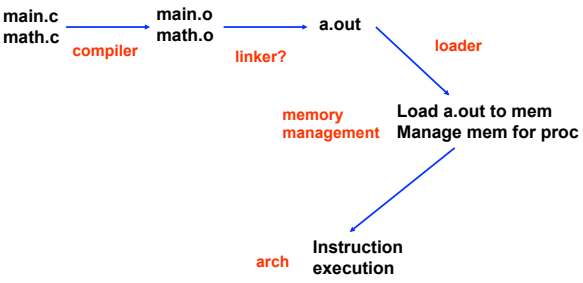
```
static void Loop(void)
{
    static char *startAddr;
    char local;
    printf("locations are %d %d\n", (&startAddr), (&local));
    startAddr = &local;
    Loop( );
}
int main(int argc, char *argv[ ])
{
    Loop( );
}
```



Fun Trick

- Recursive function
- One static variable, one local variable
- Print difference between static variable and address of local variable
- Store address of local into static, recurs
- What does this tell you?
  - Address of local ?
  - Address of static ?
  - Address of main/Loop ?

A gap among Architecture, Compiler and OS courses



Example

Main.c:

```
main( )
{
  static float x, val;

  extern float sin( );
  extern printf( ), scanf( )

  printf("Type number: ");
  scanf("%f", &x);
  val = sin(x);
  printf("Sine is %f", val);
}
```

Math.c:

```
float sin(float x)
{
  static float temp1, temp2,
    result;

  – Calculate Sine –

  return result
}
```

Example (cont)

- Main.c uses externally defined sin() and C library function calls
  - printf()
  - scanf()
- How does this program get compiled and linked?

24

Tasks of a Linker

- Read in object files produced by the compiler
- Produce a self-sufficient object file (a.out)
  - Involves
    - segment relocation
    - address translation

25

Back to the Example

- Do the main/sin example with the following segment sizes
  - Main: code 420, data 42
  - Math: code 1600, data 12
  - Library: code 1230, data 148
- Output: code 3250, data 202
- In reality segment starts on a page (4 Kbytes) boundary

26

Memory Layout – Division of Responsibility

- Compiler: generates object file
  - Information is incomplete
  - Each file may refer to symbols defined in other files
- Linker: puts everything together
  - Creates one object file that is complete
  - No references outside this file (usually)

27

Division of Responsibility (cont)

- OS (next major topic of this class)
  - Allow several different processes to share physical memory
  - Provide ways of dynamically allocating more physical memory

What could the compiler not do?

- Compiler does not know final memory layout
  - It assumes everything in .o starts at address zero
  - For each .o file, compiler puts information in the symbol table to tell the linker how to rearrange outside references safely/efficiently
    - For exported functions, absolute jumps, etc
  - Linker needs to rearrange segments
  - What makes rearrangement tricky?
    - Addresses!

What couldn't the compiler do? (cont)

- Compiler does not know all the references
  - e.g. addresses of functions / variables defined in other files
  - Where it does not know, it just puts a zero, and leaves a comment (relocation info) for the linker to fix things up
- These are called *cross references*

Components of Object File

- Header
- Two segments
  - Code segment and data segment
  - OS adds empty heap/stack segment while loading
- Size and address of each segment
  - Address of a segment is the address where the segment begins

Components of Object File (cont)

- Symbol table
  - Information about stuff defined in this module
  - Used for getting from the name of a thing (subroutine/variable) to the thing itself
- Relocation information
  - Information about addresses in this module linker should fix
    - External references (e.g. lib call)
    - Internal references (e.g. absolute jumps)
- Additional information for debugger
  - e.g. at breakpoint, "list"

32

Components of Object File (cont)

- Type "man 5 a.out" on UNIX for more information on UNIX object files

33

Linker functionality

- Three functions of a linker
  - Collect all the pieces of a program
  - Figure out new memory organization
    - Combine like segments
    - Does the ordering matter?
  - Touch up addresses
- The result is a runnable object file (e.g. a.out)

34

Linker – a closer look

- Linker can shuffle segments around at will, but cannot rearrange information within a segment

35

Recap: Tasks of a Linker

- Read in object files produced by the compiler
- Produce a self-sufficient object file (a.out)

- Implementation:
  - *How many times does linker need to scan the object files?*

Linker requires at least two passes

- Pass 1: decide how to arrange memory

- Pass 2: address touch-up

Pass 1 – Segment Relocation

- Pass 1 assigns input segment locations to fill up output segments
  - Read and adjust symbol table information
  - Read relocation info to see what additional stuff from libraries is required

Symbol Table

- Symbol table info:
  - Segments: name, size, old location, new location
  - Symbols: name, input segment containing it, offset within the segment

Pass 2 – Address translation

- In pass 2, linker reads segment and relocation information from files, fixes up addresses, and writes a new object file
- Relocation information is crucial for this part

40

Relocation Information

- Contains function/variable address and offest values to be relocated
- Examples of how to relocate:
  - "Place final address of symbol here"
    - LDW R1, _Y
    - _Y is external
  - "Add difference between the final and original address of segment to the contents of this location"
    - J _X
    - _X is in the original segment
  - "Add final address of symbol to contents of this location"
    - C reference of the form x = y.q;
    - y is external struct with {int p; int q}

41

Putting It Together

- Pass 1:
  - Read symbol table, relocation table
  - Rearrange segments, adjust symbol table
- Pass 2:
  - Read segments and relocation information
  - Touch-up addresses
  - Write new object file

42

Course schedule (tentative)

Week 1	Intro / OS components	
Week 2	Processes	Lab 1
Week 3-4	Process Synchronization	Lab 2
Week 5-6	CPU scheduling, Deadlock, MM background	Lab 2
Week 7	Thread, Midterm	Lab 3
Week 8	Fall break, Memory management	Lab 3
Week 9-11	Memory Management	Lab 3
Week 12-15	File system	Lab 4
Week 16	Distributed file sys. / Course review	
Dec	Final exam	

43

Reading assignment

- All the prerequisite for ECE595

44