CS 354 - Machine Organization & Programming
Tuesday, October 1, 2019

Midterm Exam - Thursday, October 3rd, 7:15 - 9:15 pm

- **Lec 1 (2:30 pm):** room 3650 of Humanities
- **Lec 2 (4:00 pm):** room B10 of Ingraham Hall
- UW ID required
- #2 pencils required
- closed book, no notes, no electronic devices (e.g., calculators, phones, watches)
- see "Midterm Exam 1" on course site Assignments for topics

Project p2B (3%) DUE: 10 pm, Monday, October 7th

Homework hw2 (1.5%) DUE TOMORROW: 10 pm, Wednesday, October 2nd

Last Time

Pointers to Structures
Standard & String I/O and `stdio.h`
File I/O and `stdio.h`
Copying Text Files

Three Faces of Memory
Virtual Address Space
C's Abstract Memory Model

Today

C's Abstract Memory Model (from last time)
Meet Globals and Static Locals
Where Do I Live? (from last time)
Linux: Processes and Address Spaces
---------- END of Exam 1 Material ----------
Meet the Heap

Next Time

Exam Mechanics
Heap Allocator Design

Meet Globals and Static Locals

What?

A *global variable* is A unit of storage declared outside of any functions

-
- accessable to all function after its declaration
- allocated in the data segment.

A *static local variable* is a unit of storage declared inside the function with modifier/data.

-
- accessible only within the function after its declaration,
- allocated in data segment.

Why? For storage that exists during the entire program's execution

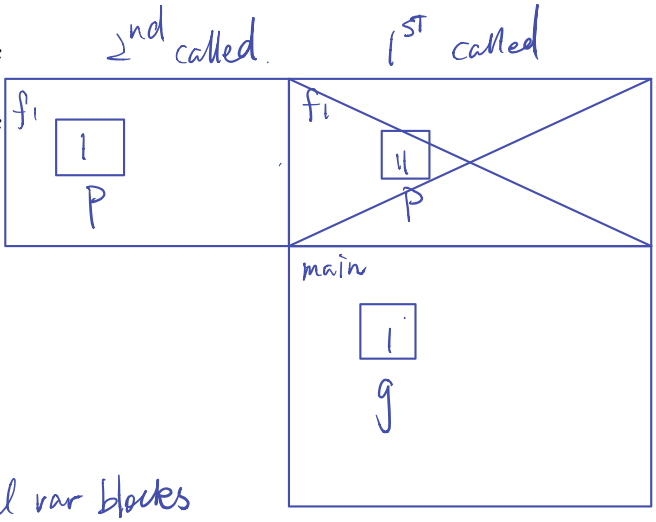
❋ *In general, global variables* should not be used.

How?

```
#include <stdio.h>
int g = 11;  ← global variable

void f1(int p) {
  static int x = 22;
   x = x + p * g;
   printf("%d\n", x);
}

int main(void) {
   int g = 1;
   f1(g);
   g = 2;
   f1(g);
   return 0;
}
```



DATA

2 x 22 143 145

g f1:x

output 143
145

shadowing: when a local var blocks accesses to a global var of the same name

❋ *Don't use the same identifier for local variables* and global vars to avoid shadowing

Linux: Processes and Address Spaces

Process and Job Control

- Linux is a multitasking OS where you can concurrently run multiple processes

`ps` all the process ps -e : show everything , ps -e | more : page by page .

`jobs` show jobs

`&` run in the background

ctrl+z suspend running process

`bg`

`fg` back to running process

ctrl+c terminate the process

`kill` kill process e.g. forefox, chrome's PID

`top` show the list for kill

Program Size

size <executable or object_file>

Displays the size in Bytes of program's code, .Data .Bss , mem areas

Data segment

```
$gcc -m32 myProg.c
$size a.out
   text    data     bss     dec     hex filename
   1029     276       4    1309     51d a.out
```

Virtual Address Space Maps

- Linux enables you to see the process's VAS

```
$pmap <pid_of_process>

$cat /proc/<pid_of_process>/maps

$cat /proc/self/maps
```

/proc: virtual file system that reveals kernel data as text files.

```
$cat /proc/loadavg
```

ctrl z : suspend running process 暂停

fg : back to running process

ctrl c : terminate running process.

loop & : loop in the background

jobs : show jobs.

ctrl c. fg enter, fg enter, ↓ not work with background.

bash.
ps: all the processes. ps -e : everything
ps -e | more : pages by pages
— & ← in background ? output : 1D

Meet the Heap

What? The heap is

- a segment of a processing VAS used for dynamic allocated mem.

  dynamically allocated memory: is allocated in runtime to satisfy uncertain mems. needs:

pmap - ↑ processing mem map.

- A collection of various-size mem blocks that are managed by an allocator

  block: a continious chunk of mem containing a payload and overhead.

  payload: part of block usage by process requesting heap mem.

  overhead: part of block used by the allocator to manage the heap's structure.

  allocator: code that allocates and frees blocks as well as spitts and merge them.

Allocator Approaches

Implicit: Java.

- "new" implicitly determines bytes needed.
- garbage collector: implicitly recycles heap blocks

Explicit: C

- "malloc" must explicitly be told how many bytes needed.
- "free" must explicitly be called to recycle heap blocks.