Procedures I
CSE 351 Winter 2024

Guest Instructor:
Will Robertson

Teaching Assistants:
Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson
Relevant Course Information

- Lab 2 due next Friday (2/2)
  - Can start in earnest after today’s lecture!
  - See GDB Tutorial Lesson and Phase 1 walkthrough in Section 4 Lesson

- Midterm (take home, 2/8–2/10)
  - Make notes and use the midterm reference sheet
  - Form study groups and look at past exams!
Procedures I
Lesson Summary (1/3)

- Memory is organized into 5 segments based on data declaration and lifetime
  - Goals: maximize use of space, manage data differently, apply separate permissions

- A segmentation fault is caused by an impermissible memory access
Lesson Summary (2/3)

- **The Stack** is the memory segment with the highest addresses and grows downward
  - Stack “top” (lowest address) is defined by the value of the stack pointer (%rsp)
  - Can manipulate using add, sub, push, and pop

- Procedure calling conventions for passing control and data
  - call and ret pass control using %rip and a return address on the stack
  - Arguments: %rdi, %rsi, %rdx, %rcx, %r8, %r9, Stack
  - Return value: %rax
Lesson Summary (3/3)

- Stack organized into **stack frames** that hold a procedure instance’s data
  - Size will vary based on procedure specifics
  - Space gets allocated as procedure executes, deallocated by the time it returns
Lesson Q&A

❖ Learning Objectives:
  ▪ Determine the location/segment in memory that a piece of data will be stored based on the nature of that data (*i.e.*, static, literals, etc.).
  ▪ Trace stack frame movement and creation.

❖ What lingering questions do you have from the lesson?
  ▪ Chat with your neighbors about the lesson for a few minutes to come up with questions
Procedures I – Practice
Practice Questions (1/2)

- How does the stack change after executing the following instructions?
  
  ```
  pushq  %rbp
  subq  $0x18, %rsp
  ```

- For the following function, which registers do we know *must* be used?
  
  ```
  void* memset(void* ptr, int value, size_t num);
  ```
Practice Questions (2/2)

Answer the following questions about when `main()` is run (assume `x` and `y` stored on the Stack):

- **Higher/larger address**: `x` or `y`?
- How many total stack frames are created?
- What is the maximum depth (# of frames) of the Stack?

```c
int main() {
    int i, x = 0;
    for (i=0; i<3; i++)
        x = randSum(x);
    printf("x = %d\n", x);
    return 0;
}
```

```c
int randSum(int n) {
    int y = rand()%20;
    return n+y;
}
```

A. 1  B. 2  C. 3  D. 4
Procedures I – Context
**Simplified Memory Layout**

### Address Space:
- **Stack**
- **Dynamic Data (Heap)**
- **Static Data**
- **Literals**
- **Instructions**

### What Goes Here:
- **Local variables and procedure context**
- **Variables allocated with new or malloc**
- **Static variables (including global variables)**
- **Immutable literals/constants (e.g., "example")**
- **Program code**
x86-64 Linux Memory Layout

- **Stack**
  - Runtime stack has 8 MiB limit

- **Heap**
  - Dynamically allocated as needed
  - `malloc()`, `calloc()`, `new`, ...

- **Statically allocated data (Data)**
  - Read-only: string literals
  - Read/write: global arrays and variables

- **Code / Shared Libraries**
  - Executable machine instructions
  - Read-only

Hex Address:
- 0x00007FFFFFFFEFFFF
- 0x000000
- 0x400000
- 0x000000

This is extra (non-testable) material
Stack Overflow

- When the stack pointer exceeds the stack bounds (segmentation fault)
  - In theory: when it collides with the Heap
  - In x86-64 Linux, when it exceeds 8 MiB limit

- Causes?
  - Infinite/deep recursion
  - Very large local variables

- Fixes?
  - Use iterative solution, compiler tail-call optimization
  - Allocate large variables elsewhere (more on the Heap later this quarter)
Aside: Stack Overflow

- Has nothing to do with actual stack overflow – named based on poll of blog users; some of the non-winning options:
  - algorithmical
  - bitoriented
  - dereferenced
  - fellowhackers
  - humbleprogrammers
  - privatevoid
  - shiftleft1
  - understandrecursion

- Crowd-sourced their logo for $512
Discussion Questions

- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others’ opinions and experiences

- Naming/etymology plays a big role in learning
  - Which new terms in this class have been the most intuitive for you to learn vs. the most difficult?
  - What do you think goes into a good vs. bad name more generally in computer science?
Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support