CS 461 / ECE 422
Discussion #2

AppSec Checkpoint 1

Paul Murley – pmurley2@illinois.edu

9/4/19

Overview

- MP1 Checkpoint 1: Overview and Setup
- Review: Stacks
- Pointers and Strings in Assembly
- GDB
- System Calls

Setting Up Your Git Repo

- If you haven't already, **create a git repo** by following the link on the course website
- If you have done so by this morning, you should see a new branch, AppSec (otherwise wait tonight for the handout script to be re-run)
- Clone your repo into the VM
  - Install git by doing sudo apt-get update && sudo apt-get install git
- Steps to merge the AppSec branch (demo)
  - git pull
  - git merge origin/AppSec
  - git push origin master

MP1 Setup and Checkpoint 1 Read-Through

MP Reminders

- If you haven’t already, you SHOULD find a partner

MP Reminders

- If you haven’t already, you SHOULD find a partner

3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

MP Reminders

- If you haven’t already, you SHOULD find a partner

> 3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

- Do not modify the files you will not be submitting (c files, etc.)

MP Reminders

- If you haven’t already, you SHOULD find a partner

> 3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

- Do not modify the files you will not be submitting (c files, etc.)
- Commit and push to the master branch only

Assembly Practice – Argument Ordering

```
int foo(int a, int b);
```

main:

```
foo(4,12);
```

How should the stack look before `call foo`?
How do you pass arguments onto the stack?

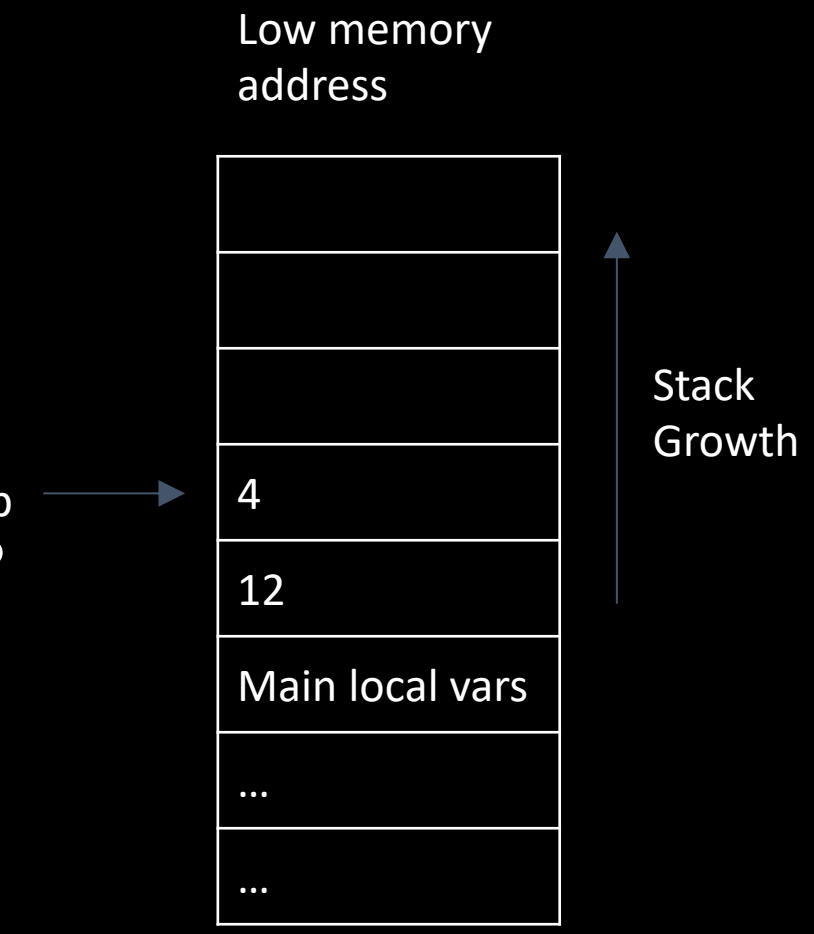
Assembly Practice – Argument Ordering

```
int foo(int a, int b);
```

main:

```
foo(4,12);
```

How should the stack look before `call foo`?
How do you pass arguments onto the stack?



Assembly Practice - Pointers

```
int foo(int a, int *b);
```

main:

```
int x = 10;
int y = 20;
foo(x, &y);
```

How can we push a pointer onto the stack?

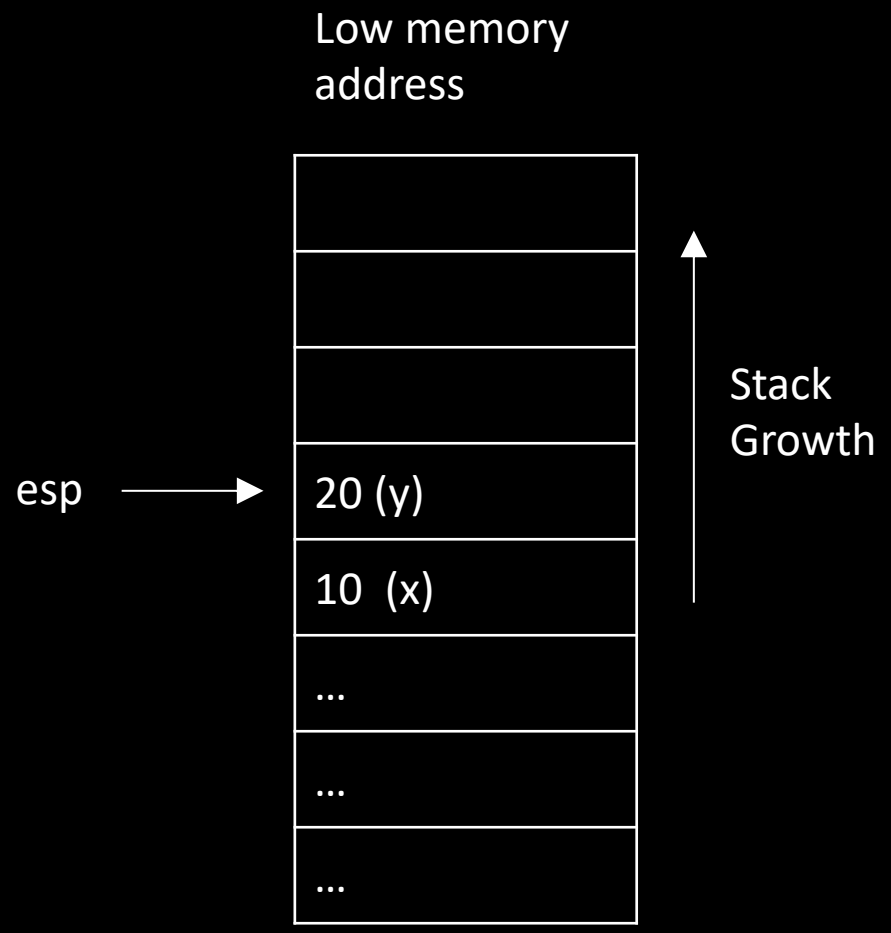
Assembly Practice - Pointers

```
int foo(int a, int *b);
```

main:

```
int x = 10;
int y = 20;
foo(x, &y);
```

How can we push a pointer onto the stack?



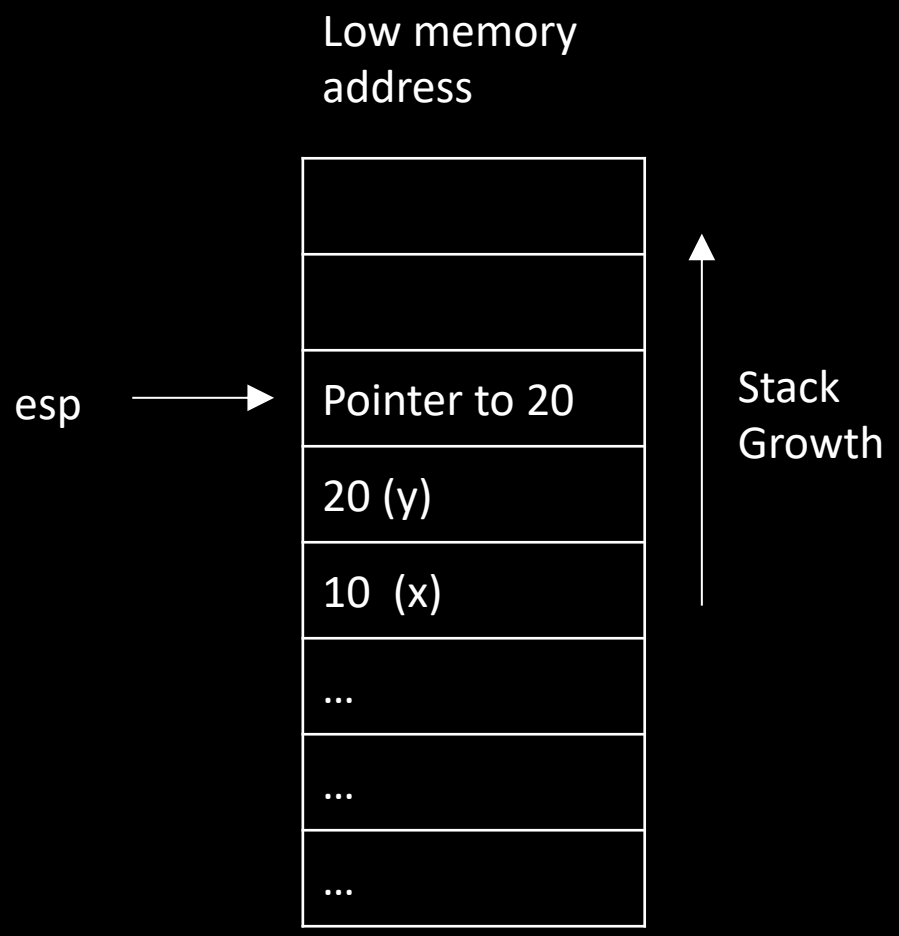
Assembly Practice - Pointers

```
int foo(int a, int *b);
```

main:

```
int x = 10;
int y = 20;
foo(x, &y);
```

How can we push a pointer onto the stack?



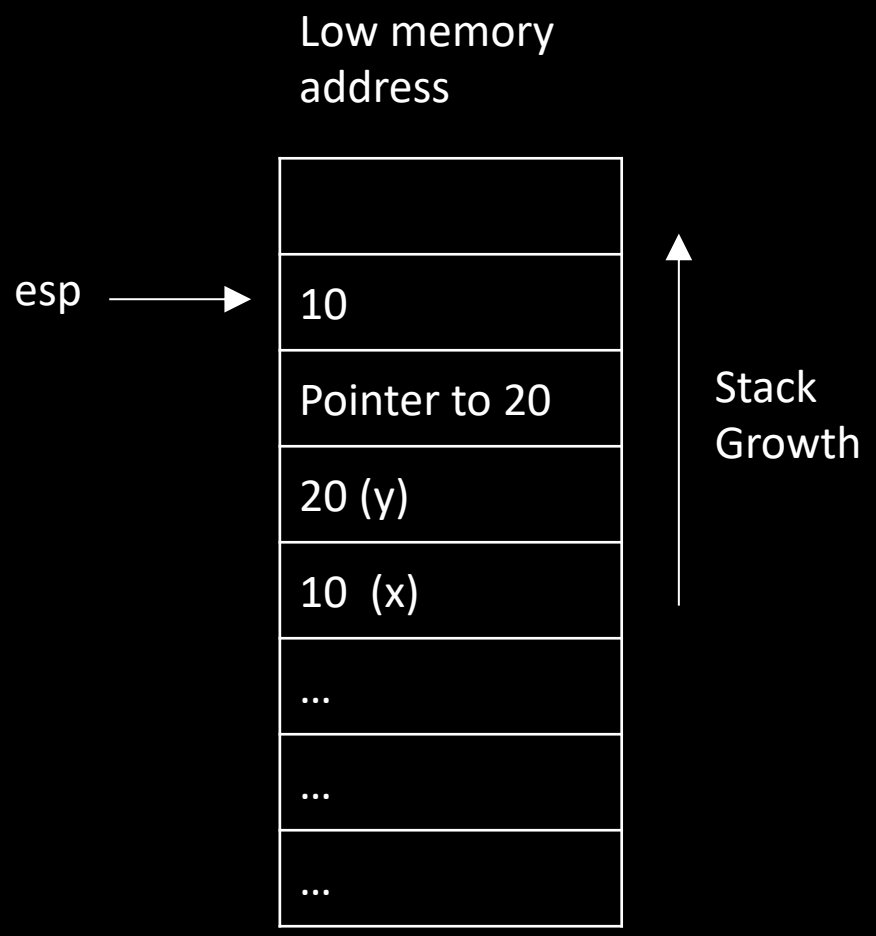
Assembly Practice - Pointers

```
int foo(int a, int *b);
```

main:

```
int x = 10;
int y = 20;
foo(x, &y);
```

How can we push a pointer onto the stack?



Assembly Practice - Strings

```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

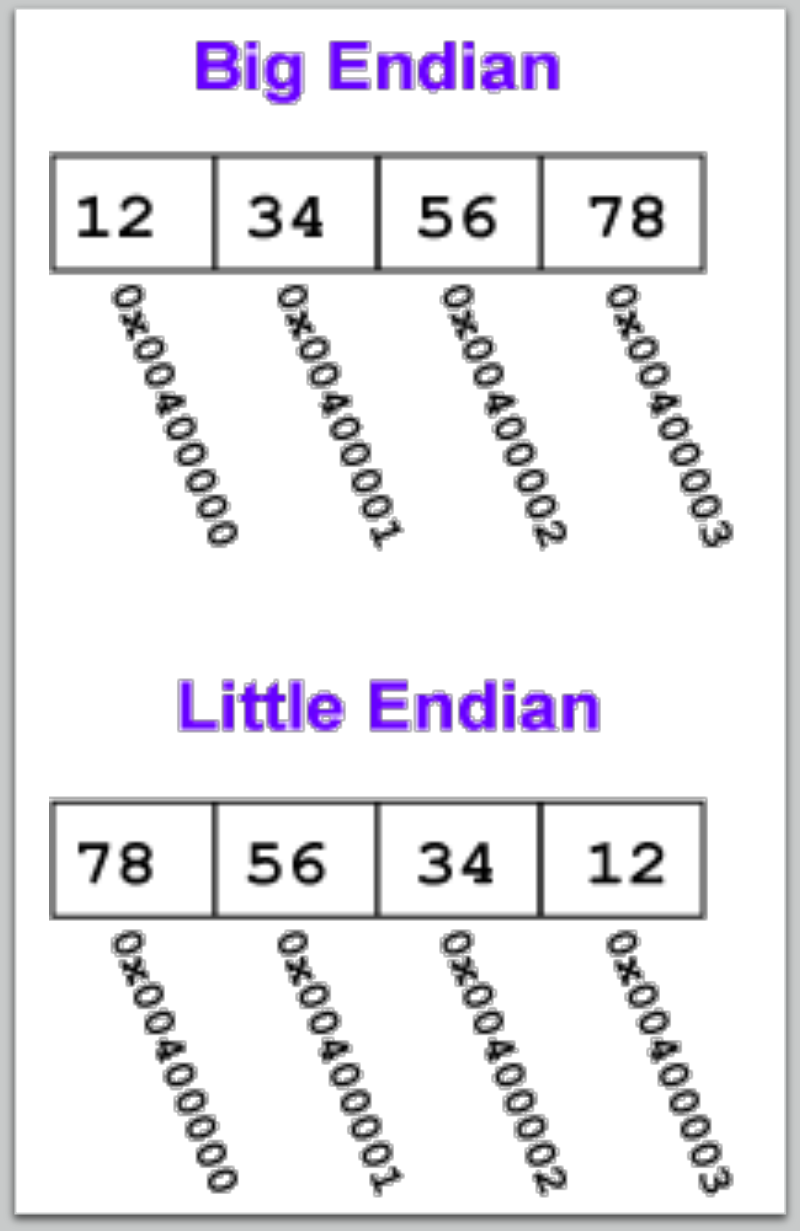
Tips:

- 1. The byte order on x86 is **little endian**.
- 2. Characters are read on the stack from **top to bottom** (low address to high address).
- 3. What character/value indicates the **end of a string**?

x86: Little Endian

- The **least significant byte** (the "**little** end") of the data is placed at the byte with the lowest address.

- Reference: https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html



Assembly Practice - Strings

```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

Tips:

- 1. The byte order on x86 is **little endian**.
- 2. Characters are read on the stack from **top to bottom** (low address to high address).
- 3. What character/value indicates the **end of a string**?

Assembly Practice - Strings

```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

How do you write “abcd” in little endian?

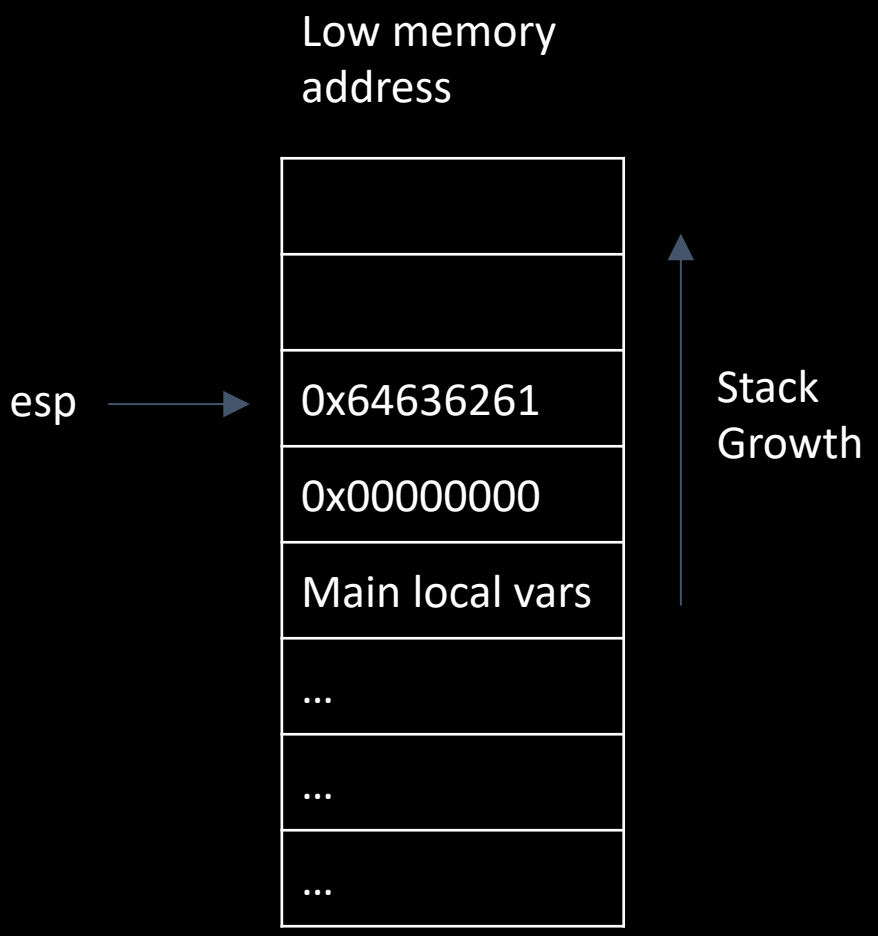
Assembly Practice - Strings

```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

How do you write “abcd” in little endian?



Assembly Practice - Strings

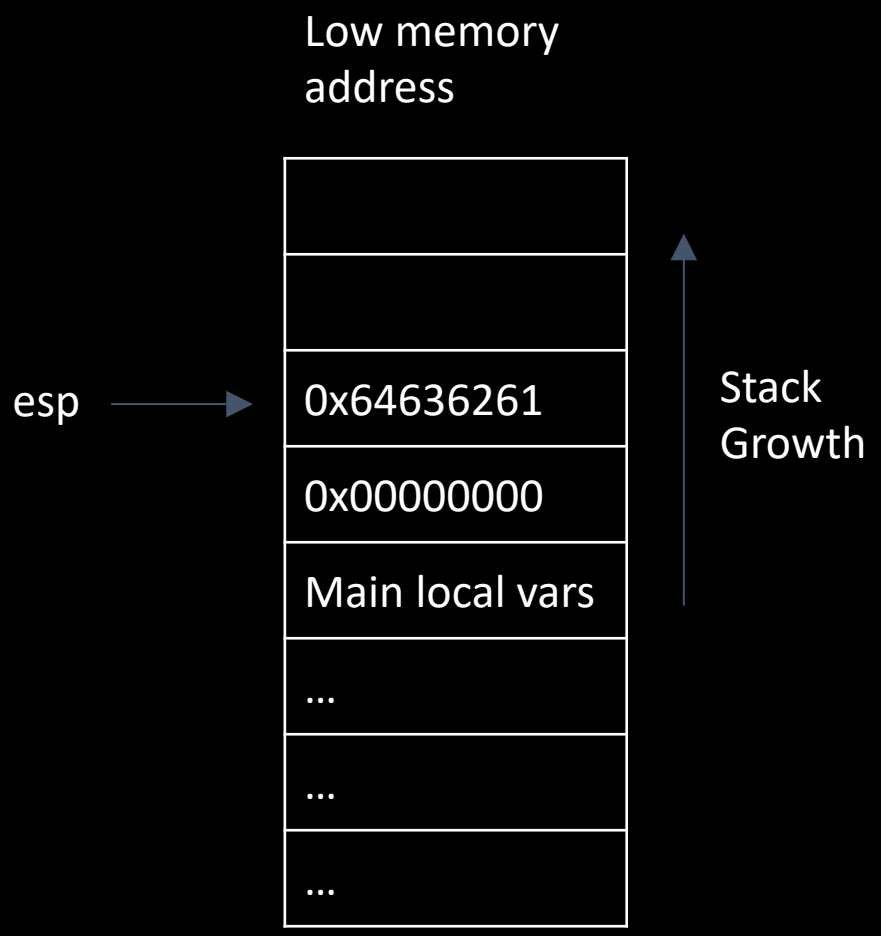
```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

How do you write “abcd” in little endian?

```
mov     %esp, %eax
push    %eax
```



Assembly Practice - Strings

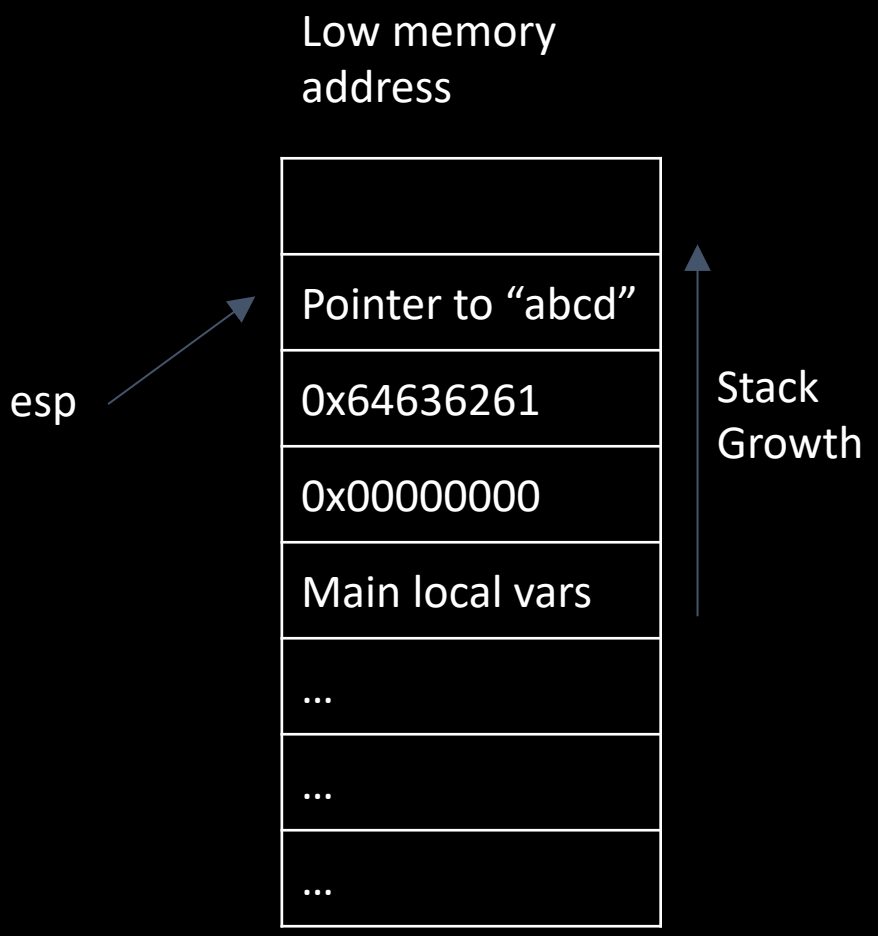
```
int foo(char* str);
```

main:

```
foo(”abcd”);
```

How do you write “abcd” in little endian?

```
mov     %esp, %eax
push    %eax
```



GDB – The GNU Debugger

- Debugger: A program that lets you examine another program as it runs
  - Can stop or pause execution using **breakpoints**
  - Can report values of memory and registers while paused
- Uses:
  - Find bugs
  - Find vulnerabilities(!!!)

GDB Commands

- Disassemble: *disas function_name*
- Set breakpoints:
  - *break function_name*
  - *b *0xbffebee0*
- Examine: *x $eax*, *x/s 0xdeadbeef*, *x/2wx 0x5adface5*
- Look at register values: *info reg*
- Run: *run (r)*
- Continue: *continue (c)*
- Step(one instruction): *si*
- Show current instruction: *display/i $pc*

GDB Demo

System Calls

- Different from user function calls (ask the kernel to do something for you)
- Unique system call numbers stored in register %eax
- Arguments are stored in %ebx, %ecx, %edx…
- Invoke system call with instruction “int $0x80”
- Check https://syscalls.kernelgrok.com/

System Calls

- Take system call sys_mkdir as example
- Sys_mkdir: 0x27,   const char __user *pathname,   int mode
- Goal: eax contains 0x27, ebx points to a place contains the directory name, ecx contains mode (which could be 0 here)
- You need to complete similar task in MP 1.1.5

Final Reminders

- Read the handout carefully!
- Start Checkpoint 2 early!
- Office Hours: M-F 5:00pm-7:00pm in SC 4405
- Contact:
  - Paul Murley
  - pmurley2@Illinois.edu
  - 445 CSL