

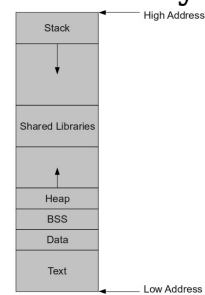
Lecture 05

- Program layout and segments
- The stack
- Calling conventions

Program vs. process

- A program is an executable file that contains a set of instructions
 - Usually stored on disk or other secondary storage
- A process is a program in execution
 - It resides, at least partially, in memory

Process memory layout



32-bit vs 64-bit

- 32-bit systems usually have shared libraries at the lowest address, followed by the text segment
- Starting addresses differ
 - Text or code usually starts 0x400000 on 64-bit, 0x8047000 on 32-bit

Text segment

- Also called the code segment
- Contains actual program instructions and any statically linked libraries
- Often read only and executable
 - Self-modifying code

Data segment

- Initialized global variables and static strings



Demo

- `hello.c`

```
#include <unistd.h>
int main(int argc, char *argv[])
{
    int ret = 0xbeefbeef;
    ret = write(1, "Hello\n", 6);
    return 0;
}

$ gcc -masm=intel -S hello.c
$ objdump -M intel -dl a.out
```



BSS

- Block started by segment
- Holds uninitialized global variables
 - By C convention are initialized automatically to 0



Heap

- Dynamically allocated memory
 - i.e., obtained via `malloc()`
- Grows upward as memory is requested
 - Upward → increasing addresses



Stack

- Holds temporary, or automatic, variables
- Arguments passed during a function call
- Information needed to return to a previous point in the program
- Grows downward (decreasing addresses)



Stack

- A stack is a very common data structure used in programs and architectures
- Stacks are generally LIFO queues
 - Last in, first out
- Two operations
 - PUSH - add something to the stack
 - POP - retrieve the most recent item



x86_64 hardware support

- RBP, base pointer
- RSP, stack pointer, points to next available address
- PUSH and POP instructions
 - Automatically decrement/increment RSP by 8



Function calls

- CALL
- RET



Base pointer

- Represents currently active region of the stack
- Used in combination with an offset to reference all local variables
 - Accesses are relative to RBP
- RBP updated any time a function is called or returns

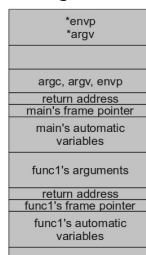


Stack frame

- Frame is pushed on function call
- Popped on function return
- Contains data for function calls
 - Parameters
 - Return address
 - Return value
 - Automatic (local) variables
- Structure is defined by calling convention



General stack frame layout



Stack frames form a linked list

- RBP always points to the start of the previous stack frame
 - Which contains. ... the previous RBP



Register preservation

- Functions share one set of registers
- Calling convention dictates how they are shared
 - Caller-saved: the calling function is responsible for saving them

```
void foo() {  
    // push regs  
    bar();  
    // pop regs  
}
```



- Callee-saved: the called function is responsible for saving and restoring

```
void bar() {  
    // push regs  
    do_things;  
    // restore regs  
    return;  
}
```



Calling conventions

- Dictate how registers are shared
- How the stack is managed when a function is called
 - Return address location
 - RBP, etc
- Also dictate how a process interacts with the kernel

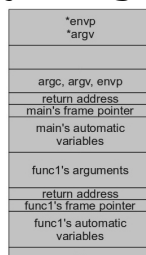


cdecl - "C declaration"

- Function parameters pushed onto stack right to left
- RAX for (primitive) return values
- Caller-saved stack



General stack frame layout again



How did we get here?

- Arguments to func1() were pushed onto the stack
- func1() was called
- RBP was pushed onto the stack
- RSP was moved to RBP
- Space for local variables was allocated
- Local variables set to initial values
 - If provided



Interacting with the kernel



System V AMD64 ABI

- Used on Solaris, Linux, FreeBSD, and macOS
- RDI, RSI, RDX, RCX, R8, and R9
 - integer or pointer arguments
 - R10 instead of RCX for kernel
- XMM0-7
 - Floating point
- Additional arguments on the stack
- Return value in RAX and RDX



- Callee must save and restore RBP, RBX, R12-R15 if used
 - Not for system calls
- Lots of details
 - <http://refspecs.linuxbase.org/elf/x86-64-abi-0.99.pdf>



Example



Questions?

