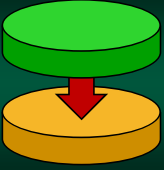


Subroutines & Operating Systems

Part 8

1



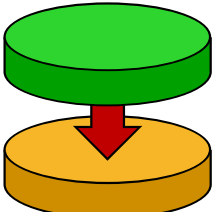
The System Stack

Pile of... Data

2

The System Stack

- The processor maintains a stack in memory
- It allows *subroutines*
 - analogous to the "functions" you use in Java and other third-generation languages
 - but, much more simple




Spring 2024 Backwards Stack - CS6.032.02 3

3

Examples of Stacks

- Page-visited "back button" history in a web browser
- Undo sequence in a text editor
- Deck of cards in Windows Solitaire



Spring 2024 Backwards Stack - CS6.032.02 4

4

Implementing in Memory

- On a processor, the stack stores integers
 - size of the integer the bit-size of the system
 - 64-bit system → 64-bit integer
- Stacks is stored in memory
 - A fixed location pointer (S0) defines the bottom of the stack
 - A *stack pointer* (SP) gives the location of the top of the stack

Spring 2024 Backwards Stack - CS6.032.02 5

5

Approaches

- Growing upwards
 - Bottom Pointer (S0) is the *lowest* address in the stack buffer
 - stack grows towards *higher* addresses
- Grow downwards
 - Bottom Pointer (S0) is the *highest* address in the stack buffer
 - stack grows towards *lower* addresses

Spring 2024 Backwards Stack - CS6.032.02 6

6

Size of the Stack

- As an abstract data structure...
 - stacks are assumed to be **infinitely** deep
 - so, an arbitrary amount of data can be stored
- However...
 - stacks are implemented using memory buffers
 - which are **finite** in size
- If the data **exceeds** the allocated space, a **stack overflow** error occurs

Spring 2024

Seckman@Stan - CS61B - CS103B

7

7

Subroutine Call Basics



Organizing Your Program

8

Subroutine Call

- The stack is essential for subroutines to work
- How?
 - used to save the return addresses for call instructions
 - backup and restore registers
 - pass data between subroutines



Spring 2024

Seckman@Stan - CS61B - CS103B

9

9

When you call a subroutine...

1. Processor pushes the instruction pointer (IP) – an address – on the stack
2. IP is set to the address of the subroutine
3. Subroutine executes and ends with a "return" instruction
4. Processor pops & restores the original IP
5. Execution continues after the initial call

Spring 2024

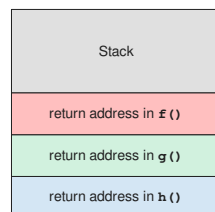
Seckman@Stan - CS61B - CS103B

10

10

Nesting is Possible

- Subroutines can call other subroutines
- **f()** calls **g()** which then calls **h()**, etc...
- The stack stores the return addresses of the callers
- Just like the "history button" in your web browser, you can store many return addresses



Spring 2024

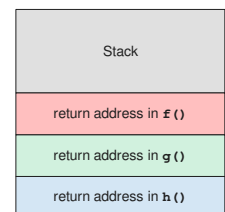
Seckman@Stan - CS61B - CS103B

11

11

Nesting is Possible

- Each time a subroutine completes, the processor pops the top of the stack
- ...then returns to the **caller**
- This allows normal function calls and recursion (a powerful tool)




Spring 2024

Seckman@Stan - CS61B - CS103B

12

12




x64 Subroutines

Organizing Your Programs ... with Intel

13

Instruction: Call

- The *Call Instruction* transfers control to a subroutine
- Other processors call it different names such as JSR (Jump Subroutine)
- The stack is used to save the current IP



Spring 2024
Seacrest@Stan - CS46 - CS235
14

14

Instruction: Call

CALL *address*

Usually, a label
(which is an address)

Spring 2024
Seacrest@Stan - CS46 - CS235
15

15

Instruction: Return

- The Return Instruction is used mark the end of subroutine
- When the instruction is executed...
 - the old instruction pointer is read from the system stack
 - the current instruction pointer is updated – restoring execution after the initial call

Spring 2024
Seacrest@Stan - CS46 - CS235
16

16

Instruction: Return

- Do not forget this!
- If you do...
 - execution will simply continue, in memory, until a return instruction is encountered
 - often is can run past the end of your program
 - ...and run data!

Spring 2024
Seacrest@Stan - CS46 - CS235
17

17

Instruction: Return

RET


No arguments!

Spring 2024
Seacrest@Stan - CS46 - CS235
18

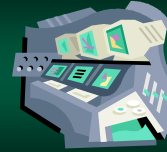
18

Subroutine Example

```
_start:
  mov rcx, 4
  mov rbx, 12
  call AddIt
  add rbx, 1
  ...
AddIt:
  add rbx, rcx
  ret
```



19



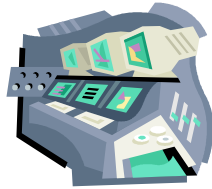
Operating Systems

The master software

20

What is an operating system?

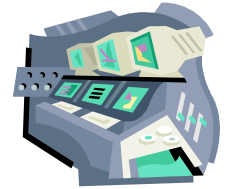
- The operating system is simply a series of programs
- These programs, however, run with special privileges which are needed by the OS
- Processors support two modes for executing programs



21

Execution Modes

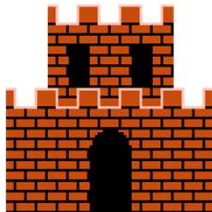
- *Privileged (supervisor) mode*
 - can run special instructions
 - can talk to all the hardware
 - etc...
- *User mode*
 - can only execute certain instructions
 - can't talk to all the hardware



22

Vector Tables

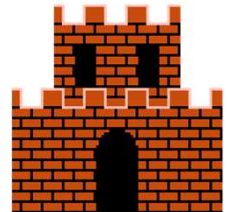
- Programs (and hardware) often need to talk to the operating system
- Examples:
 - software needs talk to the OS
 - USB port notifies the OS that a device was plugged in



23


Vector Tables

- But how does this happen?
- The processor can be *interrupted* – *alerted* – that something must be handled
- It then runs a special program that handles the event



24

Vector Table

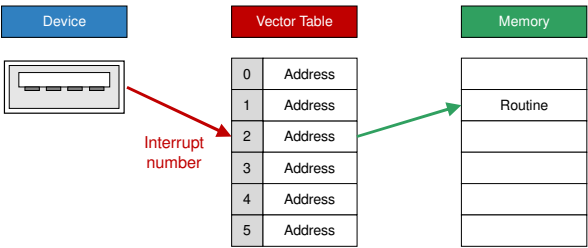


- During an interrupt, the device sends the processor an *interrupt number*
- The processor looks up the number in the *vector table*
- Table contains the address of *Interrupt Service Routine (ISR)* to execute

Spring 2024 Sacramento State - CS&E - CSJ 35 25

25

How It Works



Vector Table	
0	Address
1	Address
2	Address
3	Address
4	Address
5	Address

Spring 2024 Sacramento State - CS&E - CSJ 35 26

26

The Processor Actions



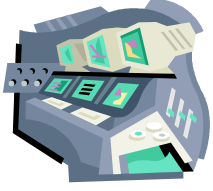
- Backup the register file
- Backups the instruction pointer
- Executes the ISR
- Restores the register file
- Restores the Instruction Pointer

Spring 2024 Sacramento State - CS&E - CSJ 35 27

27

The Kernal

- All these Interrupt Service Routines belong to the *kernal* – the core of the operating system
- Vast majority of the operating system is hidden from the end user



Spring 2024 Sacramento State - CS&E - CSJ 35 28

28



Interact with Applications


How do WE talk to the OS

Spring 2024 Sacramento State - CS&E - CSJ 35 29

29

Interact with Applications

- Software also needs to talk to the operating system
- For example:
 - draw a button
 - print a document
 - close this program
 - etc...




Spring 2024 Sacramento State - CS&E - CSJ 35 30

30

Interact with Applications

- Software can interrupt itself with a specific number
- This interrupt is *designated specifically for software*
- The operating system then handles the software's request



Spring 2024
Sechenovs, Stein - CS46 - CSU 20
31

31

Application Program Interface

- Programs "talk" to the OS using *Application Program Interface (API)*
- Application → Operating System → IO
- Benefits:
 - makes applications faster and smaller
 - also makes the system more secure since apps do not directly talk to IO

Spring 2024
Sechenovs, Stein - CS46 - CSU 20
32

32

Instruction: syscall (64-bit)

SYSCALL

Calls interrupt number reserved for programs needing attention

Spring 2024
Sechenovs, Stein - CS46 - CSU 20
33


33

Subroutine vs. Interrupt

Subroutine	Interrupt
Executes code	Executes code
Returns when complete	Returns when complete
Called by the application	Executed by the processor
Part of the application	Handles events for the OS

Spring 2024
Sechenovs, Stein - CS46 - CSU 20
34

34



Linux System Calls

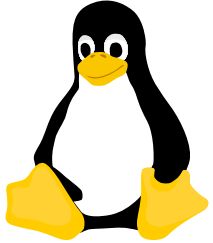
How software and hardware "talk"

Spring 2024
Sechenovs, Stein - CS46 - CSU 20
35

35

Interrupts on the Linux

- Linux, like other operating systems communicate with applications using *interrupts*
- Applications do not know where (in memory) to contact the kernel – so they ask the processor to do it

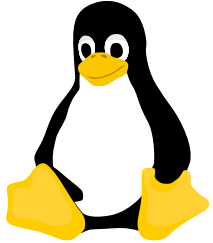


Spring 2024
Sechenovs, Stein - CS46 - CSU 20
36

36

How It Works

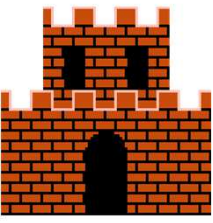
1. Fill the registers
2. Interrupt using `syscall` (or `INT 0x80` if on 32-bit)
3. Any results will be stored in the registers



Spring 2024 Backwards Book - C&C - CSU 25 37

37

How to Call Linux – 64 bit

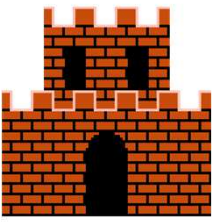


- The `rax` register must contain the *system call number*
- This number indicates what you are asking the OS to do
- There are only **329** total calls in the entire 64-bit UNIX operating system!

Spring 2024 Backwards Book - C&C - CSU 25 38

38

How to Call Linux – 64 bit



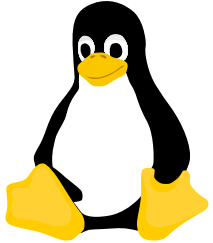
- Different registers are used to hold data
- The order is also quite odd: *rdi, rsi, rdx, r10, r8*

Spring 2024 Backwards Book - C&C - CSU 25 39

39

Kernels are Simple!

- Linux only has **1** write and **1** read system call
- The location, number of bytes, and device only change
"write x many bytes from address y to device z"
- So, writing to the screen, a file, a port, etc...use the **same** call!



Spring 2024 Backwards Book - C&C - CSU 25 40

40

Some Linux 64 Calls

System Call	rax	rdi	rsi	rdx
read	0	file descriptor	address	max bytes
write	1	file descriptor	address	count
open	2	address	flags	mode
close	3	file descriptor		
get pid	39			
exit	60	error code		

Spring 2024 Backwards Book - C&C - CSU 25 41

41

Linux 64: Sys Write

```

mov rax, 1
mov rdi, 1
lea rsi, address
mov rdx, length
syscall

```



Spring 2024 Backwards Book - C&C - CSU 25 42

42

Linux 64: Sys Read

```
mov rax, 0
mov rdi, 0
lea rsi, address
mov rdx, maxBytes
syscall
```


Linux command for READ

0 = Keyboard

Maximum number of bytes to read

Call Linux

43



Saving Registers & Lost Data

Avoiding horrible side-effects

44

Saving Registers & Lost Data

- Each subroutine will use the registers as it needs
- So, when a sub is called, *it may modify the caller's registers*
- Some processors have few registers – so its *very* likely
- This can lead to hard-to-fix bugs if caution is not used – e.g. loop counter gets changed



45

Two Solutions

- Caller saves values
 - caller saves **all** their registers to memory before making the subroutine call
 - after, it restores the values before continuing
 - not recursion friendly – it pushes all of them!
- Subroutine saves the values
 - push registers (it will change) onto the stack
 - before it returns, it pops (and restores) the old values off the stack

46

Saving Registers... How nice! :-)

```
DoSomething:
push rax
push rbx
push rcx
...
pop rcx
pop rbx
pop rax
ret
```

Save registers

Your code

Restore them. Note the reverse order

47