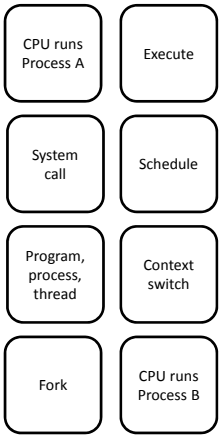
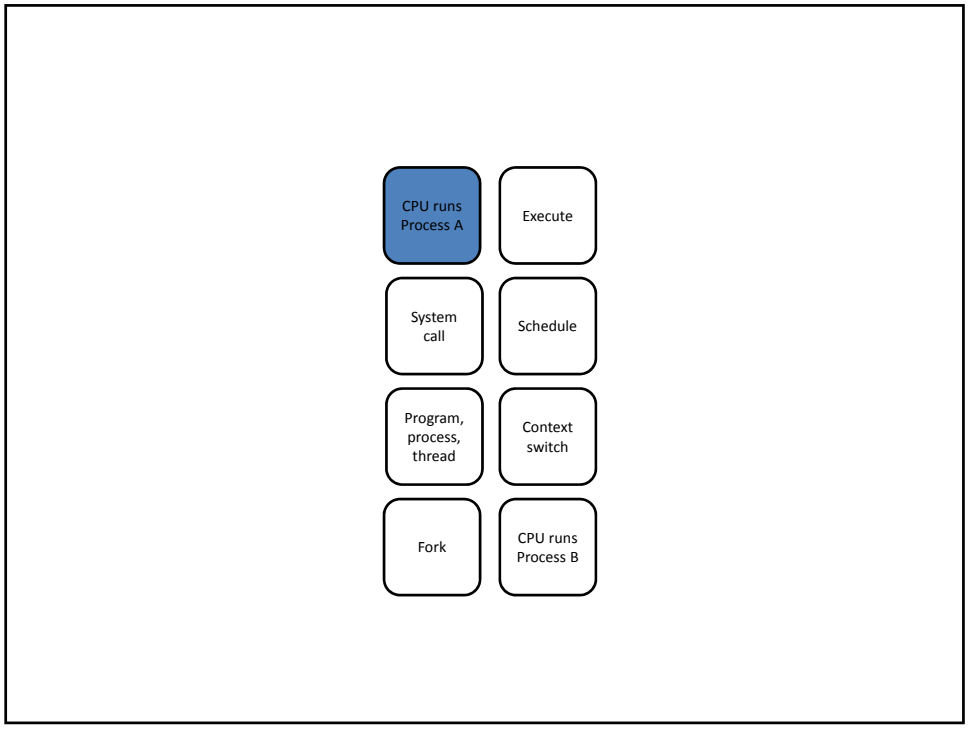
Operating System Design and Implementation

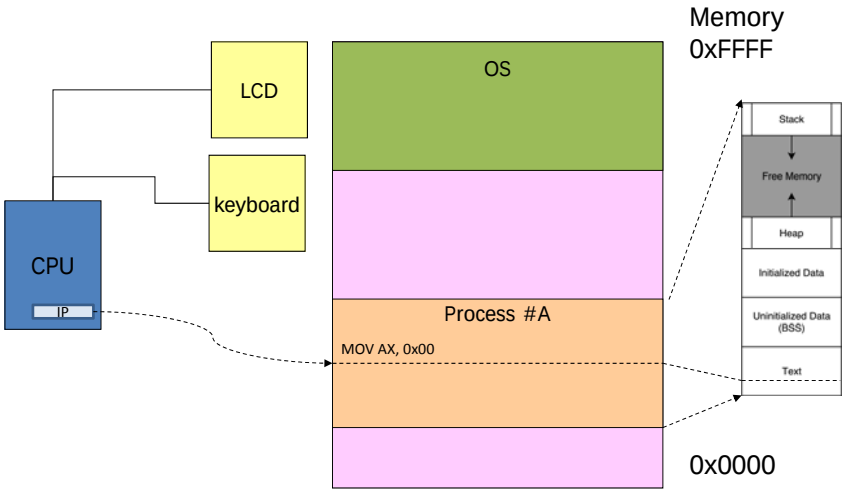
Process Management – Part I

Shiao-Li Tsao

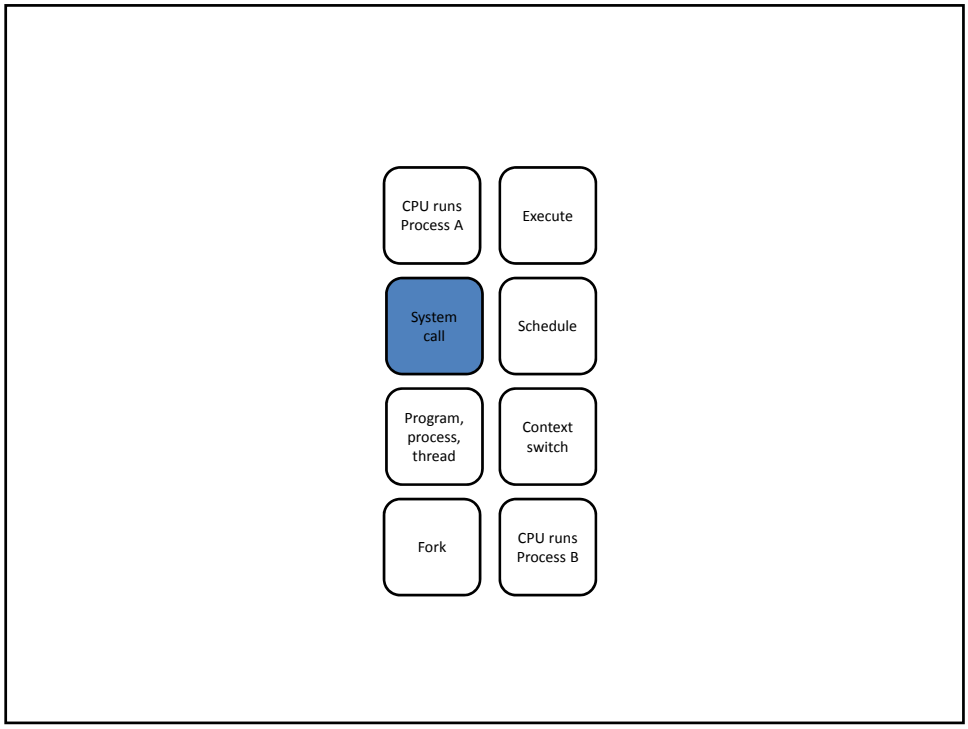




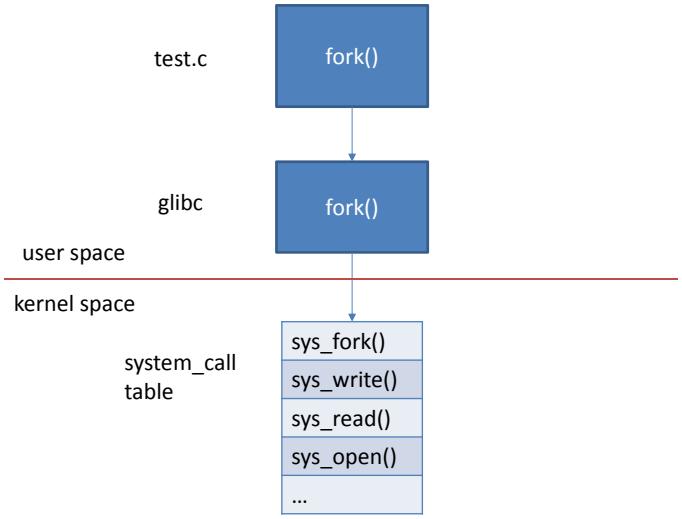
OS Memory Layout with Process A



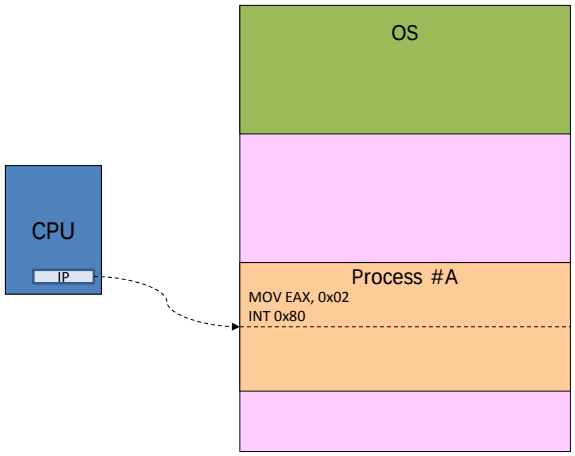
Let's assume physical memory first



Process A Calls Lib and then Traps

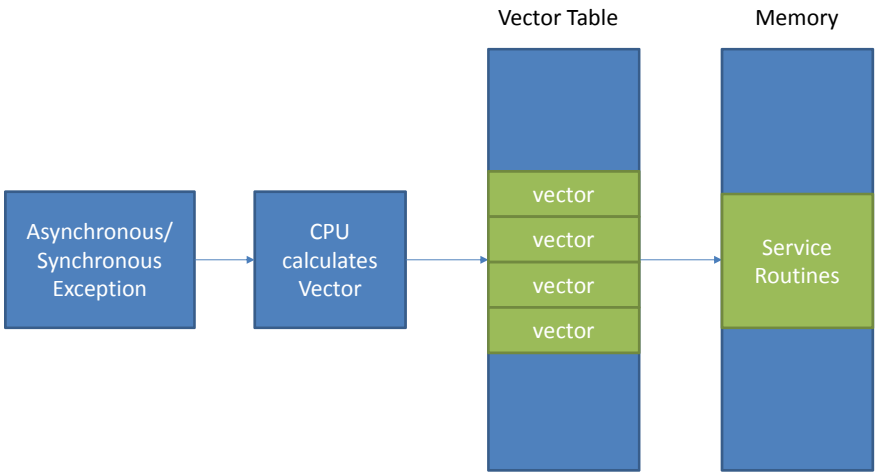


Traps Kernel

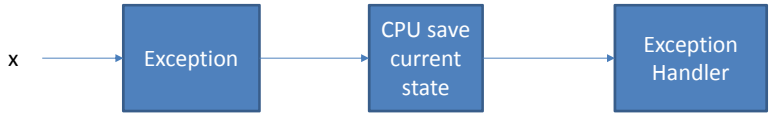


Let's assume physical memory first

Exceptions and interrupts



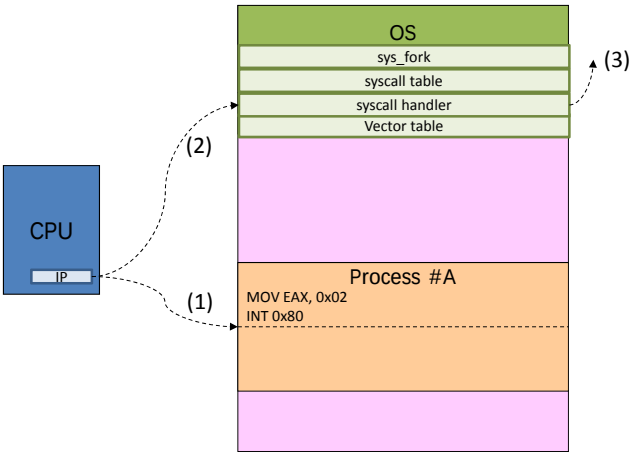
Exceptions and interrupts (Cont.)



x86 Interrupts and Linux Usage

Vector range	Use
0-19	Nonmaskable interrupts and exceptions
20-31	Intel-reserved
32-127	External interrupts (IRQs)
128 (0x80)	Programmed exception for system calls
129-238	External interrupts (IRQs)

Traps Kernel

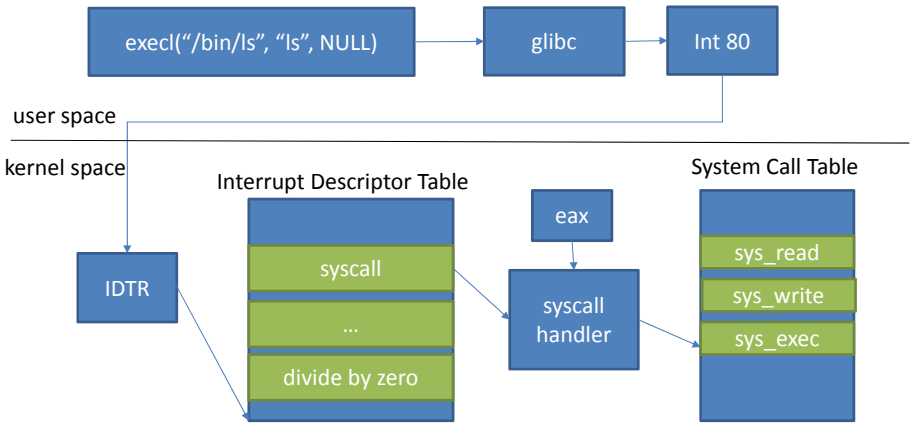


Let's assume physical memory first

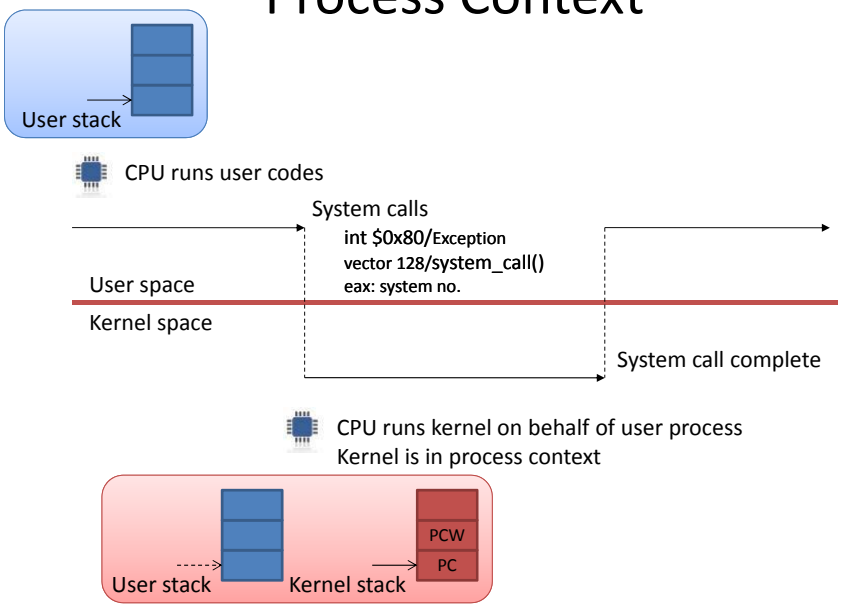
Linux System Call Table

%eax	Name
1	sys_exit
2	sys_fork
3	sys_read
4	sys_write

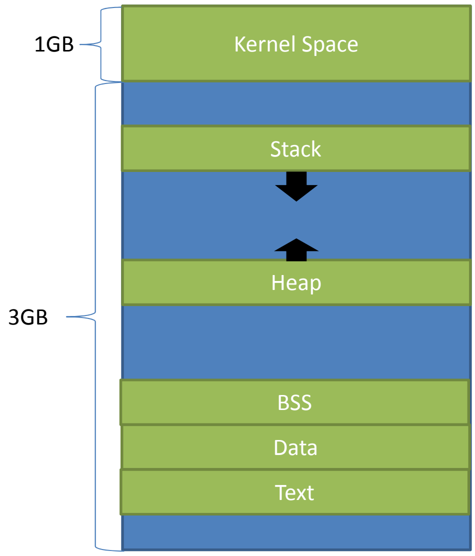
Linux System Call Procedures



Process Context



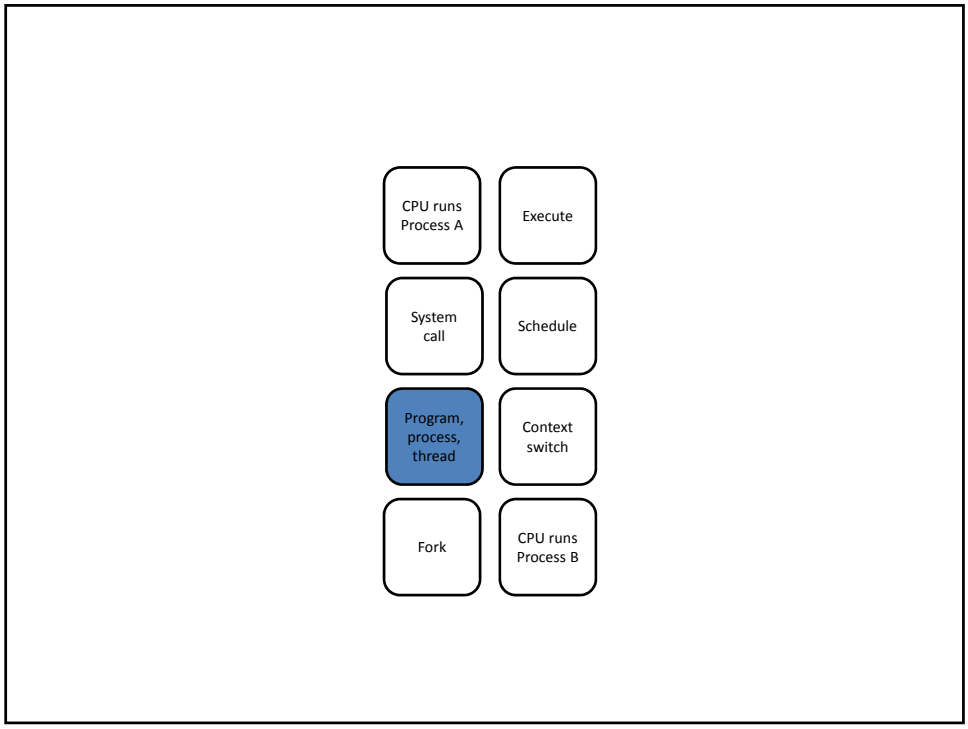
Kernel Space vs. User Space



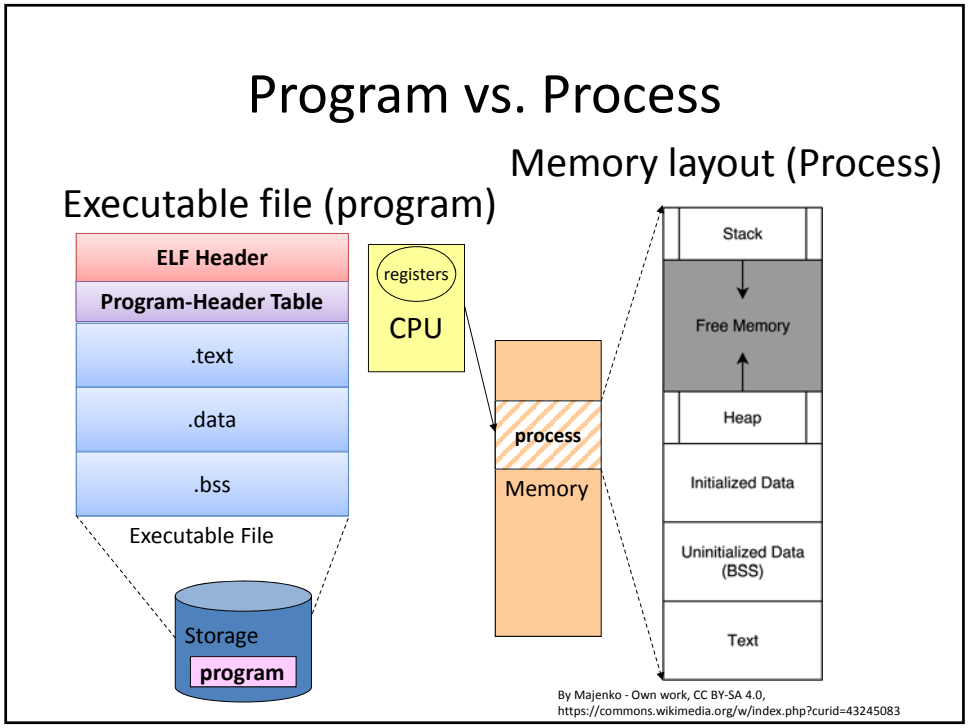
Kernel Mode vs. User Mode

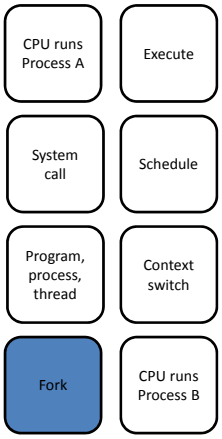
f	po	so	o	proc	st	m	rl	x	mnemonic	op1	op2	op3	op4	iext	tested f	modif f	def f	undef f	f values	description, notes
	00		r					L	ADD	**r/m8**	r8					o..szapc	o..szapc			Add
	01		r					L	ADD	**r/m16/32**	r16/32					o..szapc	o..szapc			Add
	02		r						ADD	**r8**	r/m8					o..szapc	o..szapc			Add
	03		r						ADD	**r16/32**	r/m16/32					o..szapc	o..szapc			Add
	04								ADD	**AL**	imm8					o..szapc	o..szapc			Add
	05								ADD	**eAX**	imm16/32					o..szapc	o..szapc			Add
	06								PUSH	ES										Push Word, Doubleword or Quadword Onto the Stack
	07								POP	**ES**										Pop a Value from the Stack
	08		r					L	OR	**r/m8**	r8					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	09		r					L	OR	**r/m16/32**	r16/32					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	0A		r						OR	**r8**	r/m8					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	0B		r						OR	**r16/32**	r/m16/32					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	0C								OR	**AL**	imm8					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	0D								OR	**eAX**	imm16/32					o..szapc	o..sz.pc	.....a..	o......c	Logical Inclusive OR
	0E								PUSH	CS										Push Word, Doubleword or Quadword Onto the Stack
	0F			02+					Two-byte Instructions											
	10		r					L	ADC	**r/m8**	r8				c	o..szapc	o..szapc			Add with Carry
	11		r					L	ADC	**r/m16/32**	r16/32				c	o..szapc	o..szapc			Add with Carry
	12		r						ADC	**r8**	r/m8				c	o..szapc	o..szapc			Add with Carry
	13		r						ADC	**r16/32**	r/m16/32				c	o..szapc	o..szapc			Add with Carry
	14								ADC	**AL**	imm8				c	o..szapc	o..szapc			Add with Carry
	15								ADC	**eAX**	imm16/32				c	o..szapc	o..szapc			Add with Carry
	16								PUSH	SS										Push Word, Doubleword or Quadword Onto the Stack

f	po	so	o	proc	st	m	rl	x	mnemonic	op1	op2	op3	op4	iext	tested f	modif f	def f	undef f	f values	description, notes
F	00		0	02+		P			SLDT SLDT	**m16** **r16/32**	*LDTR* *LDTR*									Store Local Descriptor Table Register
F	00		1	02+		P			STR STR	**m16** **r16/32**	TR TR									Store Task Register
F	00		2	02+		P	0		LLDT	***LDTR***	r/m16									Load Local Descriptor Table Register
F	00		3	02+		P	0		LTR	**TR**	r/m16									Load Task Register
F	00		4	02+		P			VERR	r/m16						....z...	....z...			Verify a Segment for Reading
F	00		5	02+		P			VERW	r/m16						....z...	....z...			Verify a Segment for Writing
F	01		0	02+					SGDT	**m**	*GDTR*									Store Global Descriptor Table Register
F	01	C1	0	P4++	D[24]	P	0		VMCALL					vmx		o..szapc	o..szapc			Call to VM Monitor
F	01	C2	0	P4++	D[24]	P	0		VMLAUNCH					vmx		o..szapc	o..szapc			Launch Virtual Machine
F	01	C3	0	P4++	D[24]	P	0		VMRESUME					vmx		o..szapc	o..szapc			Resume Virtual Machine
F	01	C4	0	P4++	D[24]	P	0		VMXOFF					vmx		o..szapc	o..szapc			Leave VMX Operation
F	01		1	02+					SIDT	**m**	*IDTR*									Store Interrupt Descriptor Table Register
F	01	C8	1	P4++			0		MONITOR	*m8*	*ECX*	*EDX*		sse3						Set Up Monitor Address



Program vs. Process



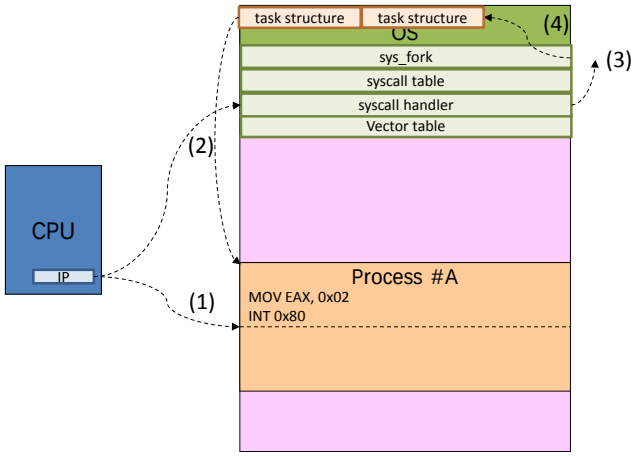


Process creation and destroy in Linux

- Process creation flow
  - fork()
    - creates a child process that is a copy of the current task
    - it differs from the parent only in its PID, its PPID, and certain resources
    - copy-on-write
  - exec()
    - loads a new executable into the address space and begins executing it
  - fork()+exec()=spawn()

20

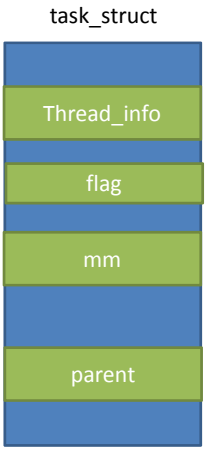
Traps Kernel



Let's assume physical memory first

Process descriptor in Linux

- Linux process descriptor



22

Process descriptor in Linux

- Where Linux store the process descriptor
  - Of course memory
  - In kernel space – why ?
  - Per process (lightweight process)
  - Allocated by slab allocator (will be described in MM) – why ?
  - Co-located with process kernel stack
    - What is process kernel stack?
    - Why co-located with process kernel stack?

23

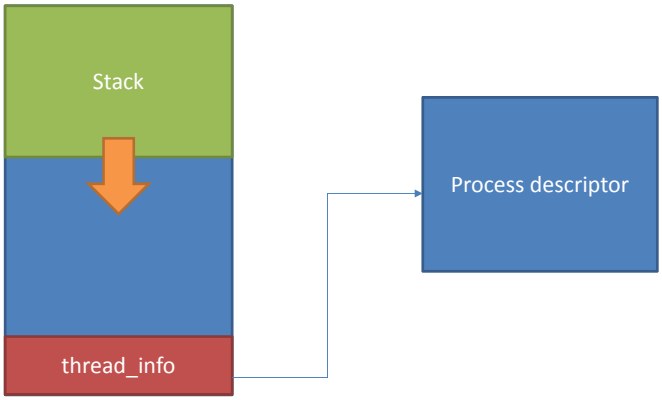
Process descriptor in Linux

- What is process kernel stack?
  - A stack space used while process is running at kernel mode – why separated with stack used in user mode?
- Why co-located with process kernel stack
  - Easy to access by stack pointer

24

Process descriptor in Linux

- Storing process descriptor and kernel stack



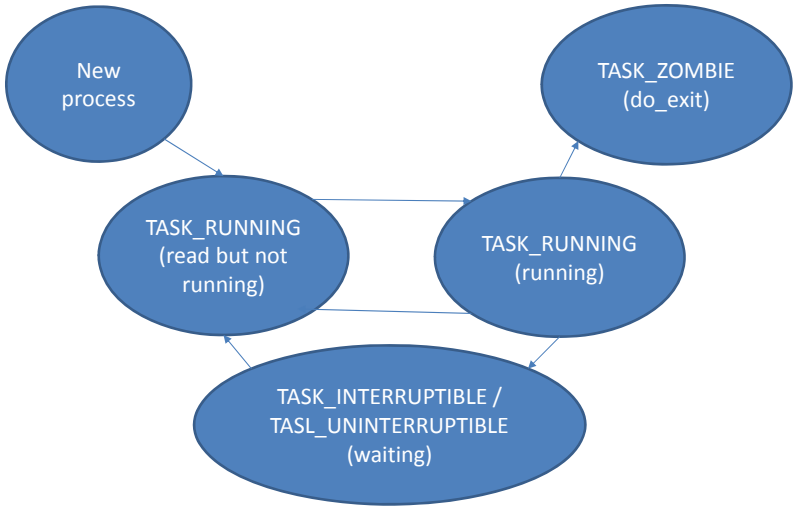
25

Process descriptor in Linux

- How these process descriptors are managed
  - Who & when create process descriptor?
    - "Parent" creates children & process descriptor
    - While (more precisely should be "before") the process is created
  - Who & when destroy process descriptor?
    - People destroy themselves, "Parent" or "ancestor" destroy the process descriptor
    - While (more precisely should be "after") the process is terminated
  - Who & when use process descriptor?
    - Scheduler
    - While scheduler is invoked

26

Process descriptor in Linux



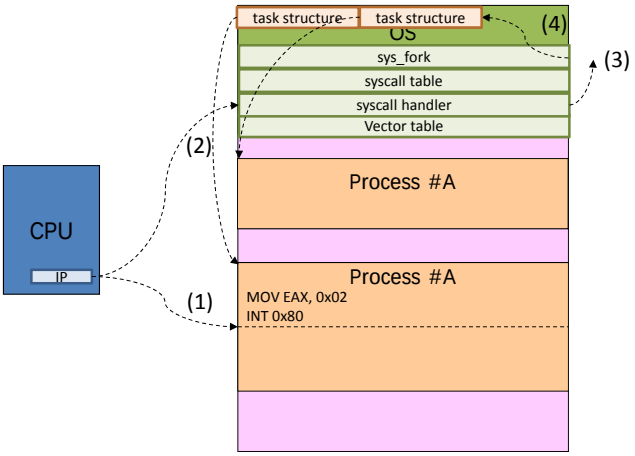
27

Process creation and destroy in Linux

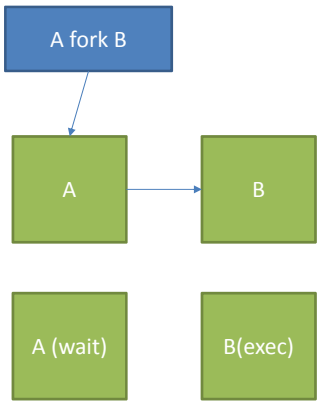
- User process/thread
  - create by parent
  - run alternatively in Kernel Mode and in User Mode
  - run kernel function through system calls
- Kernel process/thread
  - normally create while OS booting
  - run only in Kernel Mode
  - run specific kernel function
  - use only linear addresses (will be discussed later in MM)
  - process 0: **init** (executed during start_kernel())

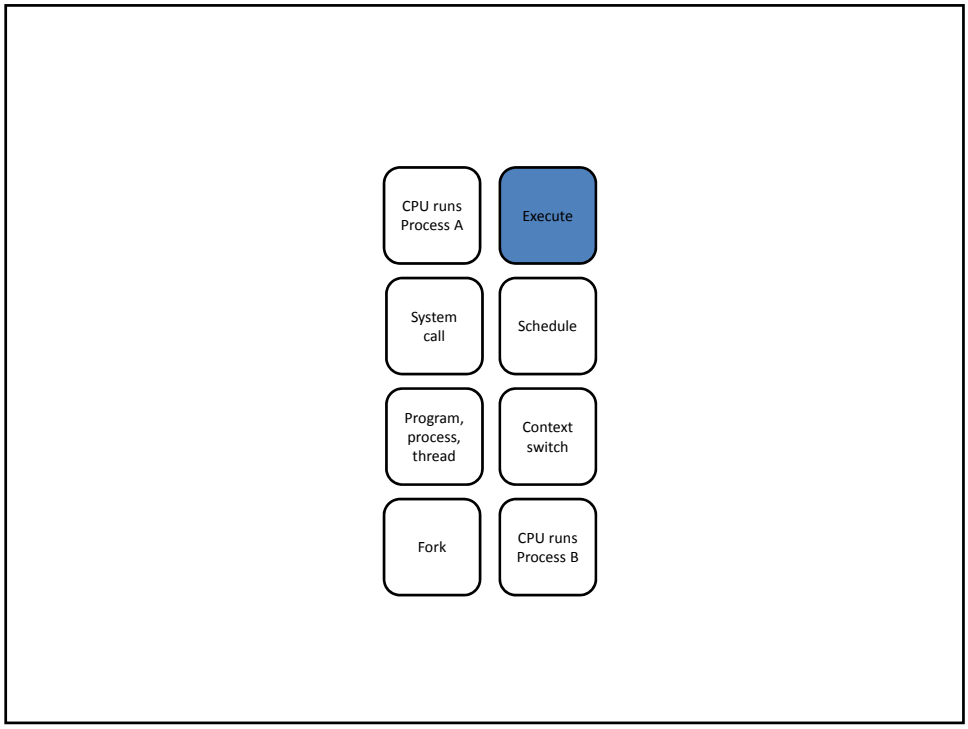
28

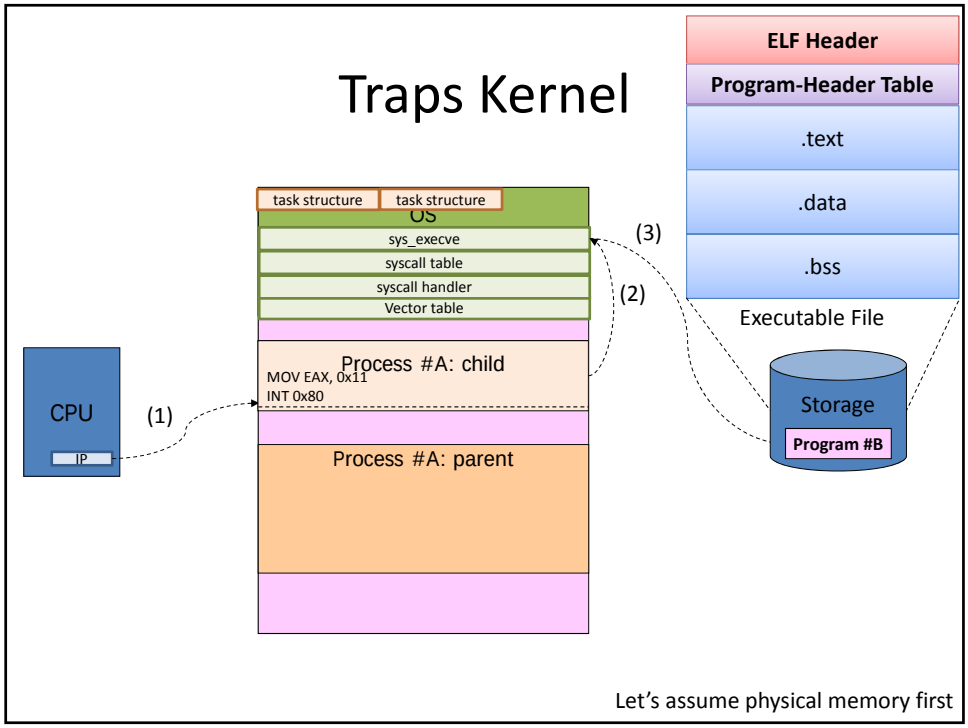
Traps Kernel

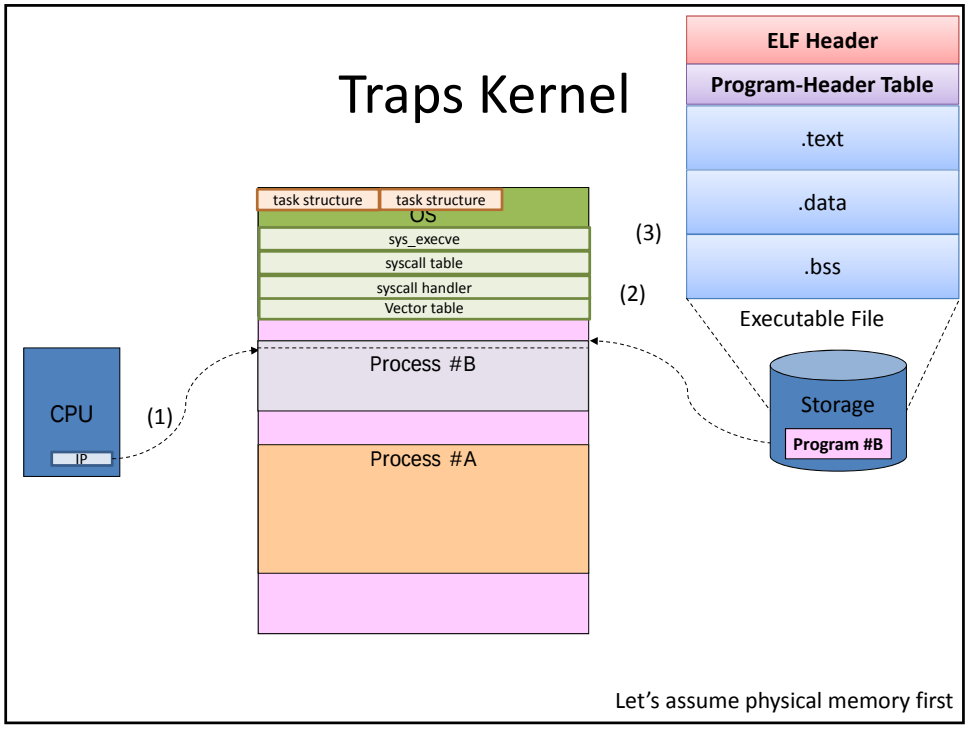


Let's assume physical memory first




CPU runs Process A
Execute
System call
Schedule
Program, process, thread
Context switch
Fork
CPU runs Process B


Traps Kernel
ELF Header
Program-Header Table
.text
.data
.bss
Executable File
task structure
task structure
OS
sys_execve
syscall table
syscall handler
Vector table
(3)
(2)
Process #A: child
MOV EAX, 0x11
INT 0x80
Process #A: parent
CPU
IP
(1)
Storage
Program #B
Let's assume physical memory first


Traps Kernel
ELF Header
Program-Header Table
.text
.data
.bss
Executable File
task structure
task structure
OS
sys_execve
syscall table
syscall handler
Vector table
(3)
(2)
Process #B
Process #A
CPU
IP
(1)
Storage
Program #B
Let's assume physical memory first