
CS2106

Introduction to OS

Office Hours, Week 4

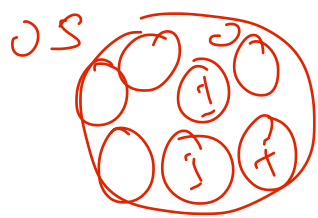
Agenda

- Interrupts
 - ▣ Questions + refresher
- Processes and Syscalls
- Fork()/Exit()
- Zombies/orphans
- C language questions
- Miscellaneous
- Logistics



Interrupts

- **Is an interrupt a syscall?** No.
 - ❑ Syscalls are OS services explicitly demanded by the programmer
 - ❑ Interrupts happen any and exist to notify of some external event.
- **Can you explain "exception is synchronous, interrupt is asynchronous"?**
 - ❑ In this context, asynchronous means that an event can happen at any time, regardless of what program is currently running and what instruction is currently being run.
 - ❑ Synchronous means that an event happens only at particular times with respect to the program being run. Which is when the program divides by zero, or similar.



Interrupts: Example

A process is currently executing the following instruction that loads a byte residing at memory address `addrA` into register `reg1`:

`load reg1, addrA;`

In the middle of the instruction, while the data is travelling on the bus, the highest priority interrupt is received. What will be the action of the operating system?

- 1 . The OS will send the data back to the memory to free up the bus for the high priority interrupt routine.
- 2 . The OS will save the data into a buffer next to the bus so that the transfer can continue after the high-priority interrupt is served.
- 3 . The OS will drop the data without saving it to minimize the interrupt latency.
- 4 . None of the above.**

load 41, A
 $\frac{64 \text{ KiB}}{16} = 64 \text{ K} = 2^{16}$

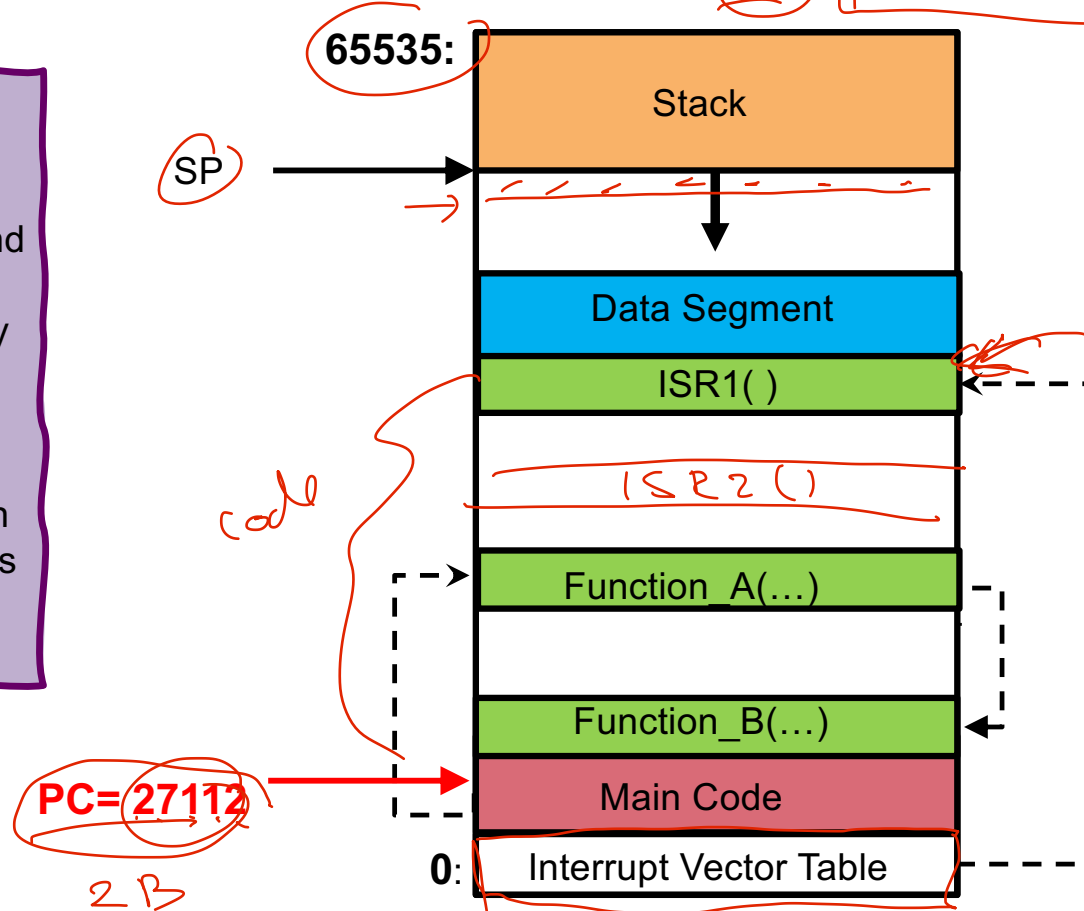
Interrupt: Example

Assume 64KB byte-addressable memory

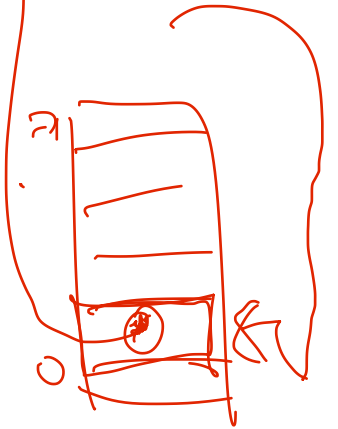
Stack semantics:

PUSH (X) → decrements SP and puts X to the location pointed by new SP

POP (X) → reads X from the location pointed by SP, puts it into X, and increments SP



$108 \left(\frac{64 \text{ KiB}}{16} \right)$
 16 bits = 2B



Executing an instruction in hardware

PC: **load reg1, memA**

- Address of the instruction in memory: 27112 (content of PC)
- Content of memory at address 27112 (3 bytes):

0110 0001 1001 1001 0101 100

code for load reg1 address memA

Execution happens in the following **hardware** stages:

- Step 1:** read one byte from address found in PC (27112)
 - put the byte into IR; $PC = PC + 1$
- Step 2:** decode instruction from IR → it's a **load**, needs an address!
- Step 3:** read two more bytes (address *memA*)
 - put the two bytes into IR; $PC = PC + 2$

$PC \leftarrow PC + 3$

Executing the quiz instruction in hardware

- **Step 4:** read the source operands (the byte found at memory address *memA*). **Interrupt 1 arrives!**
- **Step 5:** perform the ALU operation (nothing to do for load)
 - Interrupt 1 pending
- **Step 6:** store the result (the read byte) into the destination operand (reg1)
 - interrupt 1 pending
- **Step 7: check if any interrupts are pending?**
 - No? Go to step 1
 - **Yes: Interrupt handling**

interrupt
pending

$PC = PC + 3$

PSW
program status word

Interrupt Handling Steps

1. PUSH(status register) -- hardware
2. PUSH (PC) -- hardware
3. Disable interrupts in the status register -- hardware
4. Read Interrupt Vector Table entry (ISR1) -- hardware
5. PC ← ISR1 -- hardware
6. ISR1 execution -- **software**

PC ← ISR1 address

ISR 1 execution

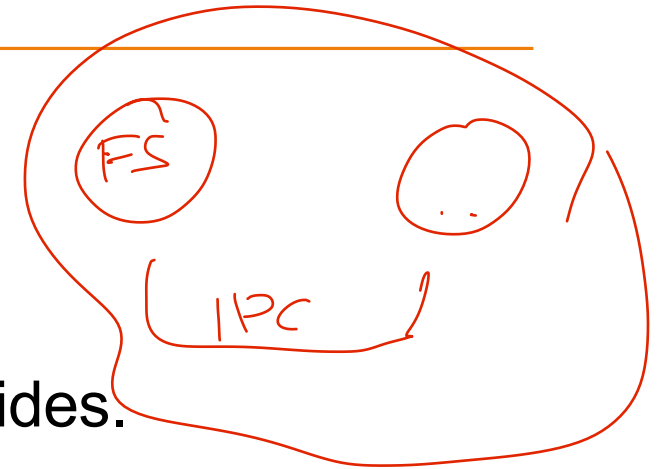
1. PUSH(registers to be modified, if any) -- software
2. Optionally enable interrupts (all or some) -- software
3. Do the necessary stuff
 - Allowed to modify the pushed registers
4. POP (modified registers, if any)
5. IRET -- specialized interrupt return instruction. Semantics:
 - POP(status register) -- enables interrupts -- hardware
 - POP(PC) -- returns to the main code (next instruction) -- hardware
 - **Step 1** of the next instruction after load (PC=27115)

Side note: some instructions do not react to interrupts (hardwired to skip step 7)

Processes And Syscalls

- **Is the OS itself a process?**

- No. Process is an abstraction that OS provides.
- But modern operating systems that feature many functionalities that are implemented as processes.
- These functionalities can be essential (in case of microkernels, the file system can be a process), or non-essential (e.g., Solitaire).

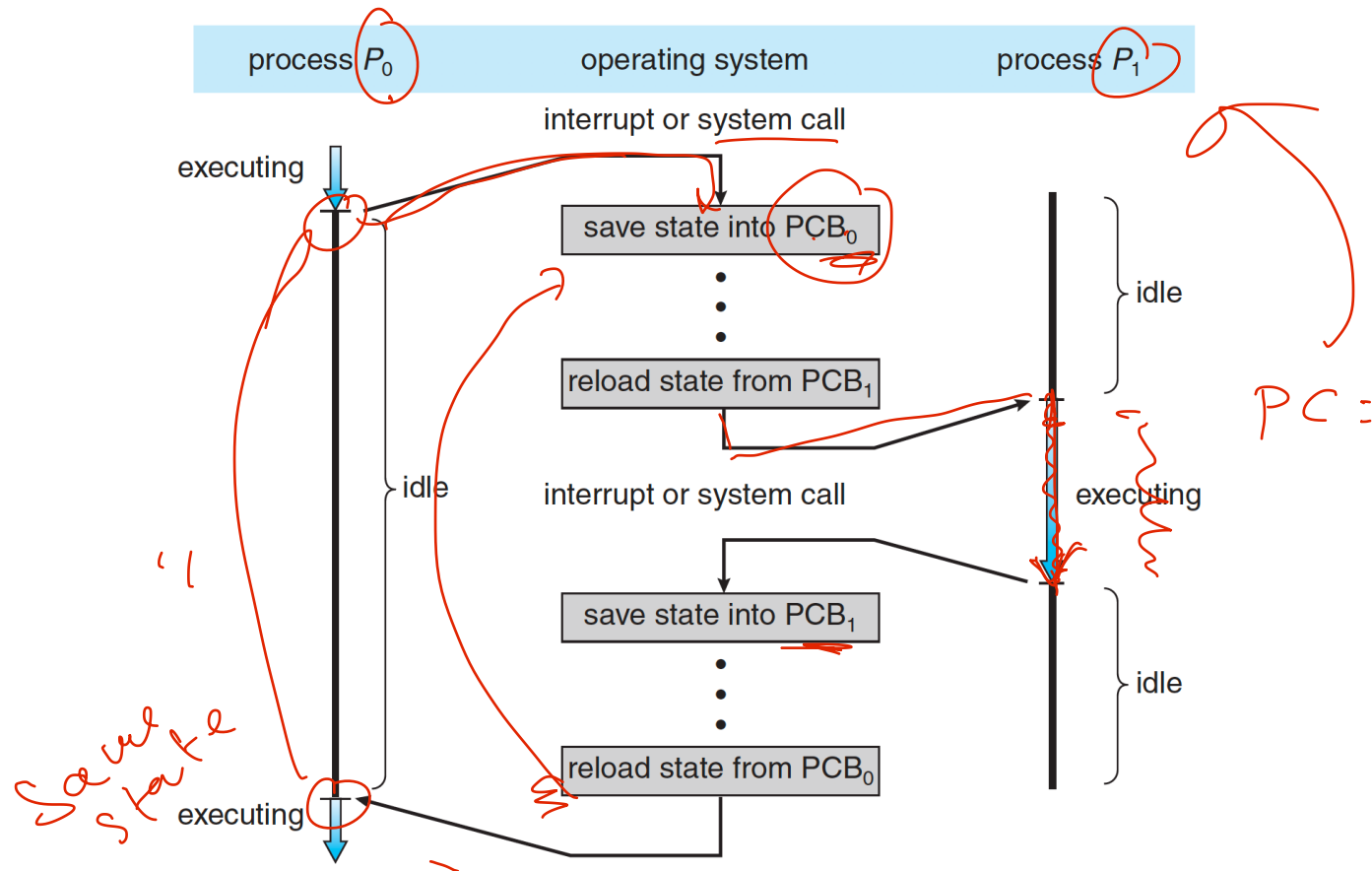


Processes And Syscalls

- **What is the difference between the sys call library-os and user program-os?**
- No difference.
- However, the user cannot execute a syscall directly, at least not from high-level programming languages, such as C.
- In assembly language you can do a syscall yourself, but it's not convenient.

try
syscall

Can you re-explain context Switch?





Processes And Syscalls

- **I am confused. In L2b slide 18, first point says "Status is returned to the parent process" but the third point says "The exit() function does not return".**
 - "The exit() function does not return" means the control flow does not return to the callee (unlike "normal" functions).
 - But the exit status of the process **is** communicated to the parent!

Processes And Syscalls

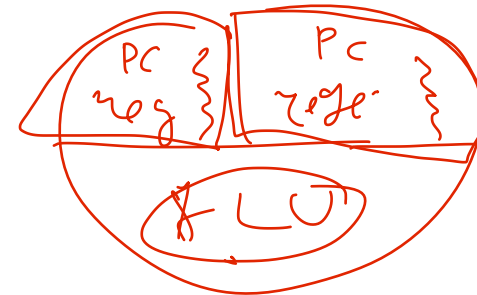
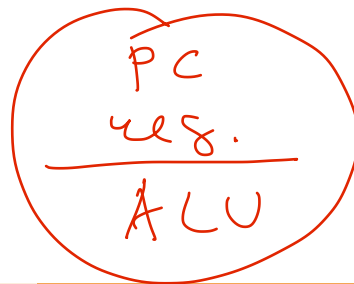
- **Is `exit(10)` at the end of the function and `return(10)` the same?**
- Only if we are talking about the end of the `main()` function, then the two are functionally equivalent.
- Better use `return 10`, because it doesn't require you to include an extra header file.

Processes And Syscalls

- **Do we need to understand the concept of cores for this module?**
- Yes! No other module will teach you that, and it's important in 2021, when even the cheapest smartphone has many cores.

Processes And Syscalls

- Is it true that one core allows only one process to be run at any point in time?
- Which means it is not possible for 2 processes to be running on the same core simultaneously?
- Correct. Means that only one process can occupy the core (ALU etc) and the registers. However, some cores can run 2 processes simultaneously.
 - But they have to be special cores, so called hyper-threaded cores.
 - Such cores must have 2 sets of all registers to store the context of 2 processes.
 - Many other resources (ALU, caches) are shared between the two processes



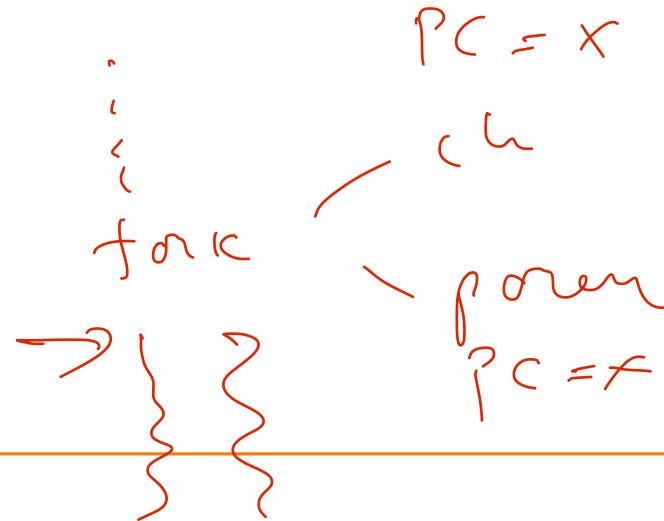
Processes And Syscalls

- I am still very confused about the question on "context of a process". What is the definition of context of a process and how to we decide whether or not it is part of the context of a process?
- The context of a process consists of all the state that must be saved, and later restored, while not affecting the correctness of the process execution.
- Extra state that only contributes to performance improvement, e.g., caches, is not part of the context. Saving and restoring cache content would be too impractical and would defeat the purpose of improving the performance.

Fork()/Exec()

- So fork duplicates the remaining part of the process?
- For duplicates the entire process, with all its context.
 - That includes the program counter (PC) too, which in the parent points to the instruction after fork(), and therefore in the child as well. Because of this, the child will continue from that point on.
- Remember: the only difference between the child and the parent
 - Return value of fork()
 - PID

man fork



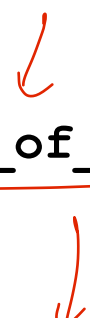
Fork()/Exec()

- When do we use fork() and exec() and for what use?
- You use fork() and exec() when you want your program P1 to execute another program P2 in parallel.
- E.g., a master process can create many worker processes that run in parallel and do same or different things.

Fork()/Exec()

- Can fork be useful without exec? E.g., to speed up things by doing things in parallel?
- Yes! Example:

```
if (fork() == 0) {
    do_the_first_half_of_some_work(); //child
}
else{
    do_the_second_half_of_some_work(); //parent
}
```



- Without `exec()`, the code of `some_work()` must be written in the program

Fork()/Exec()

- After exec() allows the child process to take over the current task, how does the parent process accept another request?

```
if(fork()==0) { exec(new_task_info); } //first child
else //parent
    if(fork ==0) exec(another_new_task); //second child
    else {
        // the rest of parent
    }
}
```

fork();
fork();

Fork()/Exec()

- What's the difference between exec() and execl()?
 - Just the format of the parameters (minor).

man exec XY

Zombies/Orphans

- **What is worse, a zombie or an orphan?**

- Zombie! Orphan processes get a new parent (`init` process).

- **How does `wait()` actually clean up the zombie processes?**

- It doesn't! Zombie happens when the parent didn't wait (or didn't wait on time).
- Timely `wait()` can prevent zombies from happening.

Zombies/Orphans

- How does wait() actually clean up the zombie processes?
- It doesn't! Zombie happens when the parent didn't wait (or didn't wait on time). Timely wait() can prevent zombies from happening.

defunct
PS

Zombies/Orphans

- Is there any way to free the PIDs of the zombies?
- Manually. E.g.,
- `$ kill -s SIGCHLD PID`

Zombies/Orphans

- How to ensure that children ~~does~~ not exit early?
- It's quite complicated and would require more complex synchronization.
- Fortunately, modern versions of `fork()`, such as `clone()`, can be parametrized in a way that doesn't require a parent to wait for the child.

- Homework: `man clone`

```
if (fork() == 0) { child }  
else { parent; wait()
```

2 output
26

Zombies/Orphans

- **Why can't the child terminate by itself without the parent?**
- It can, if you use `clone()` instead of `fork()` with proper parameters.

Miscellaneous

- **What exactly is address space?**
- The set of all addresses a program can address.

- **Can you explain what is the PCB and where is it used?**
- PCB = program control block. Please check lecture 2a.

Miscellaneous

- When we return from a function call, it is right to say we move the stack pointer back to the previous function but the memory still retains the data of the function until the data is overwritten?
- Correct!
- At any point in time is it correct to assume that there will be only be one FP pointing to some location in the stack? Hence we only need to take into account the FP which points to the most recently added stack frame?
- If you have a single-core system, than at any time you have one currently active stack frame, and it's pointed to by the value of the FP register.

C

- Why do we initialize variables in C first? E.g., `int pid; pid = getpid();` instead of `int pid = getpid();`

- These two statements are equivalent in C. You can initialize any variable at the time of declaration. Old compilers had a restriction about where in the code you can declare a variable. You couldn't declare a variable in the middle of the code, only at the beginning, only at the beginning of a new scope(`{...}`). It made it easier for early compilers to figure out how much stack space to allocate for local variables. Example:

```
if(...) {  
    int a, b; // 0  
    a = 3;  
    b = a + 2;  
    int d; // not allowed  
    ...  
}
```

C

- When u declare a variable in C, does it equals to zero/NULL before initialization?
- No. When running on Linux, the **Operating System** will initialize all variables in BSS to zero. Other operating systems do not. C standard does not specify the content of uninitialized variables. You should never assume it is zero/NULL. In fact, many compilers will throw a compilation error if you try to read an uninitialized variable.

C

- **Why in c programming in main() there's always return 0 at the end?**
- This gives the programmer the way to indicate the exit status of the program. You can omit “return 0”, the compiler will insert it automatically.

C

- Why is it a good practice to have the function declaration first, then implement it later in the code?

```
void f(int n); // declaration
```

```
void g(int m); //declaration
```

```
void f(int n){ //implementation
```

```
...
```

```
g(n-1); //can use because already declared, although not implemented
```

```
void g(int m){
```

```
...
```

```
f(m-1); //can use, because it's declared (and implemented);
```

```
}
```

Logistics

- **How is the quiz being graded?**
- Each question 1 mark.
- For MCQ, all or nothing.
- For MRQ, partial marks given. Each of N options is evaluated independently as a true/false question, whose weight is $1/N$ marks.



Thank you!