Operating Systems W3L1 - Processes & Threads: Part 1 (ctd)

Class	Operating Systems
Created	@Sep 14, 2020 9:12 PM
Materials	
Reviewed	☐
Type	Lecture

Regarding the *Materials*, we are continuing the notes from the last notes page. Also there was now W2L2 lecture because we didn't have two classes that week (labor day weekend).

Operating Systems W2L1 - Processes & Threads: Part 1

A Bit About Interrupts

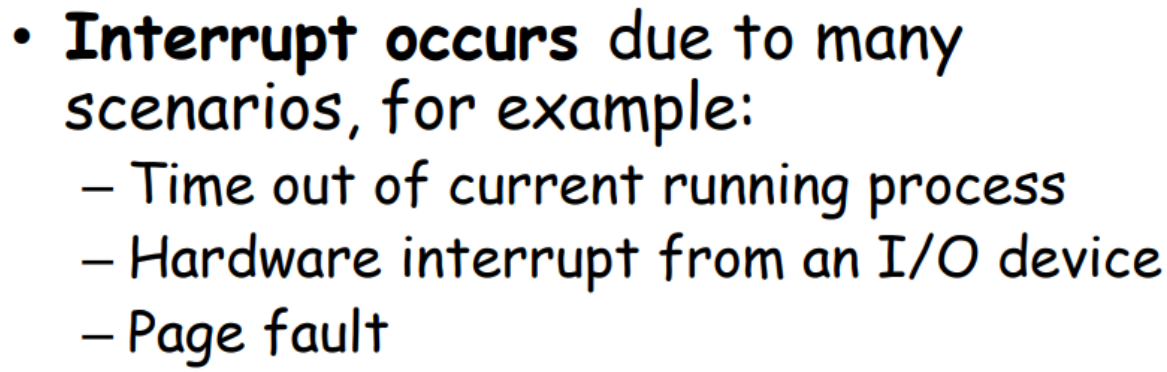
- All systems- big and small- have interrupts
- A `PIC` chip coordinates interruptions from processes
- Simple example of an interrupt is when you hti keys (and release keys) on your keyboard, since your OS needs to read the input
- **Interrupt:** Current running process is suspended, OS takes control

▼ When OS takes control, what modes switch?

Machine moves from **user mode** to **kernel mode**

> ❗ "Kernel" is just a fancy nickname for the OS

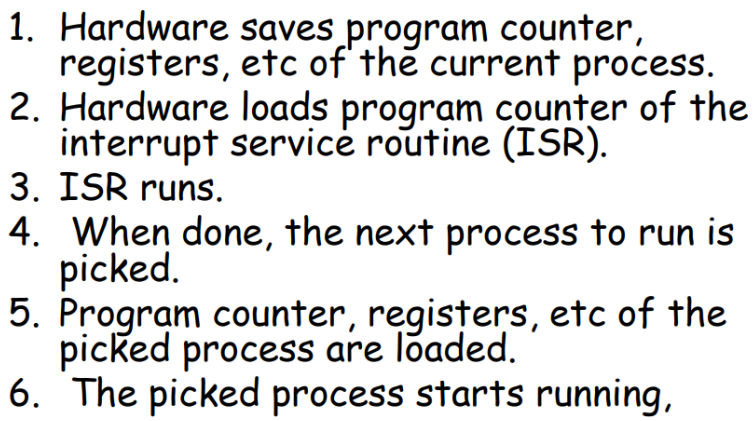
▼ Many scnearios where an interrupt can occur (some examples)



▼ Why might press and release be two interrupts?

1. Holding down the key is continuous
2. ???? F

▼ What is the order of events in an interrupt?



- ISR is considered a *privileged process*

- If memory is not big enough, a *swap space* is used on the disk

Moving From User to Kernel Mode

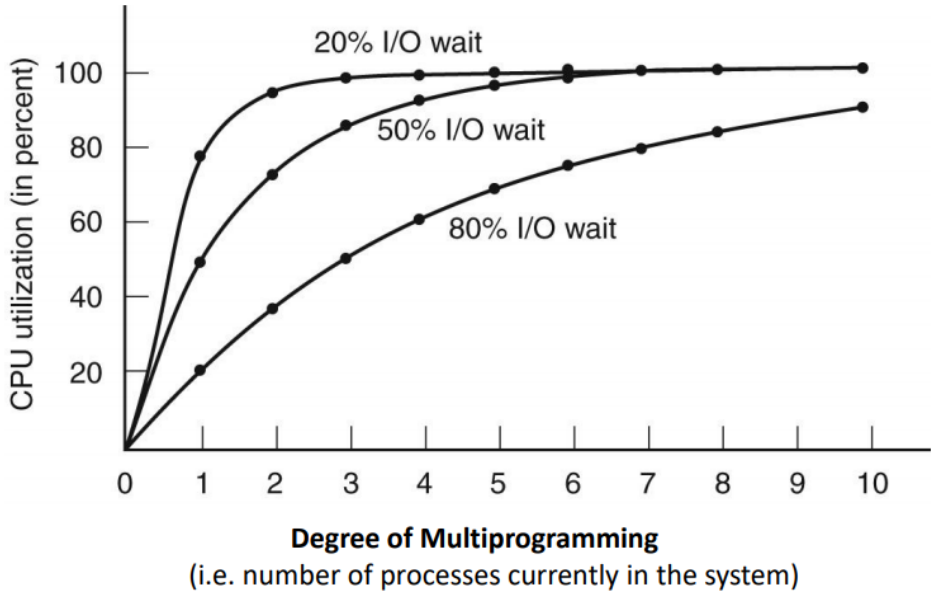
- Interrupts is an example of a process moving from user mode to kernel mode, as is a system call (which is voluntary)

Multiprogramming

Simple Model

- Suppose `n` processes in memory at once and that a process spends a fraction of `p` waiting for I/O
- Probability that all processes are waiting for I/O is p^n at a given moment
- Thus, CPU utilization is $1 - p^n$

▼ Graph - CPU utilization nased pm a number of processes



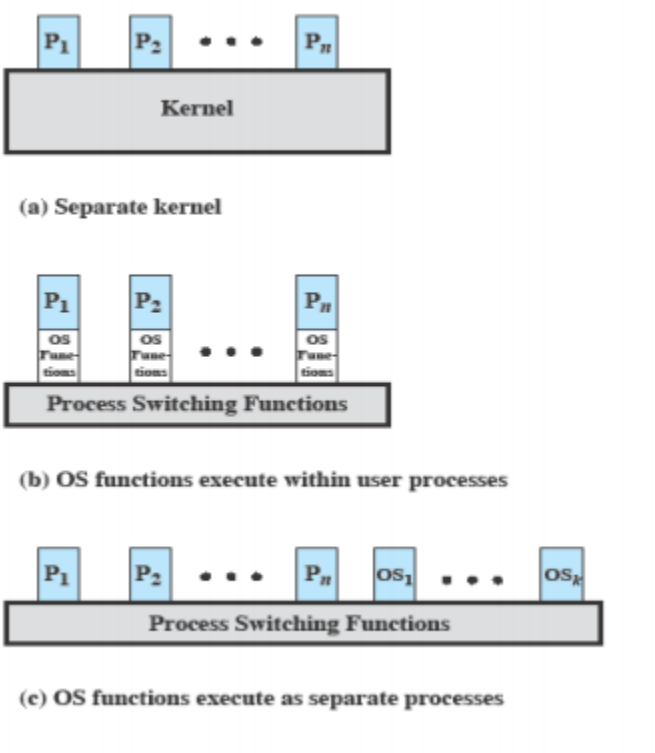
Multiprogramming lets processes use the CPU when it would otherwise become idle.

▼ What don't we want CPU to be idle?

We are wasting time and resources by having processes just sitting and being idle

Looking at OS

▼ Processes related to OS in different situations



- `(b)` and `(c)` are used in the [literal] biggest scenarios → biggest machines use `(c)` due to huge number of processes running and using lots of memory
- OS processes have privilege over regular processes, but not necessarily more priorities

Threads

▼ Multiple *threads of control* in ap rocess, and all threads share same address space. Elaborate.

Process with four threads, the threads can share data. However, each thread needs its own program counter

- Thread of control is the execution of a program
- Simply speaking, threads are used to execute programs in parallel

▼ What does multithreading entail?

Process that can be broken up and operated in parallel

Why Threads?

- Generally speaking, concurrency
  - Number of threads can change dunamically
  - Programming is "easier" with threads (and by easier, we mean not impossible because it's still very hard)
  - Benefits processes using lots of I/O and processes that overlap
- Lighter weight → faster to create and restore; much cheaper

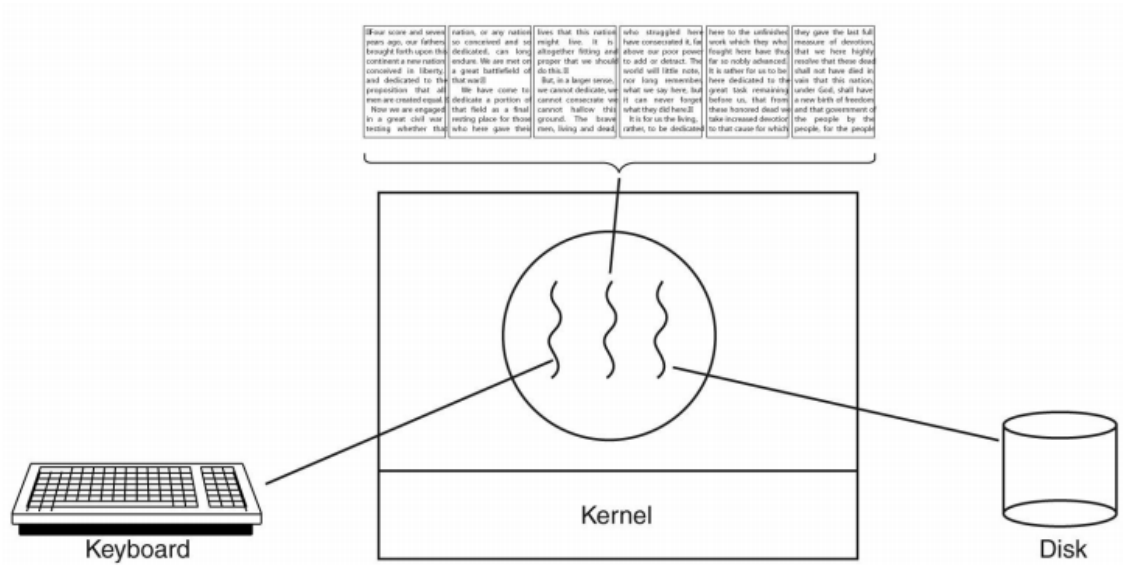
▼ Two strategies for dealing with something slow

1. Speed it up
2. Hide it → Do something while something else is ongoing

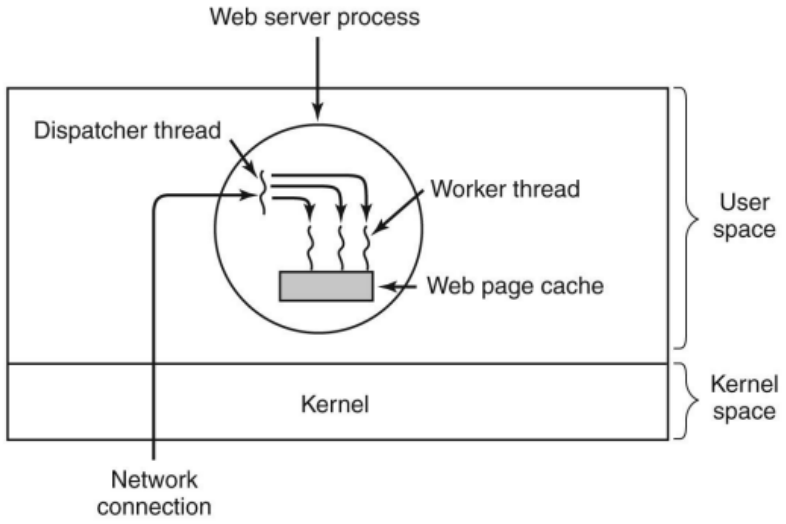
> ❗ Threads shape data and such, but each have their own stack

▼ Examples

A Word Processor



Multithreaded Web Server



- Even sequential processes are threaded → just one thread
  - Adding a new thread is just a unique *stack* and *program counter*
  - Threads are parts- contained by processes → **NOT a process within a process**

Processes vs. Threads

- **Process** group resources, and **threads** are entities scheduled for eecution (one of those process resources)
  - Threads sharing all fellow resources

▼ Why no protection between threads?

Unlike processes being protected from other processes, threads are not since threads may need to send info, share resources, etc. → they are part of the same program, so they need to work together. This process is faster without protecting threads from other threads.

- Threads can be in any of several states

▼ Per process vs. per thread items

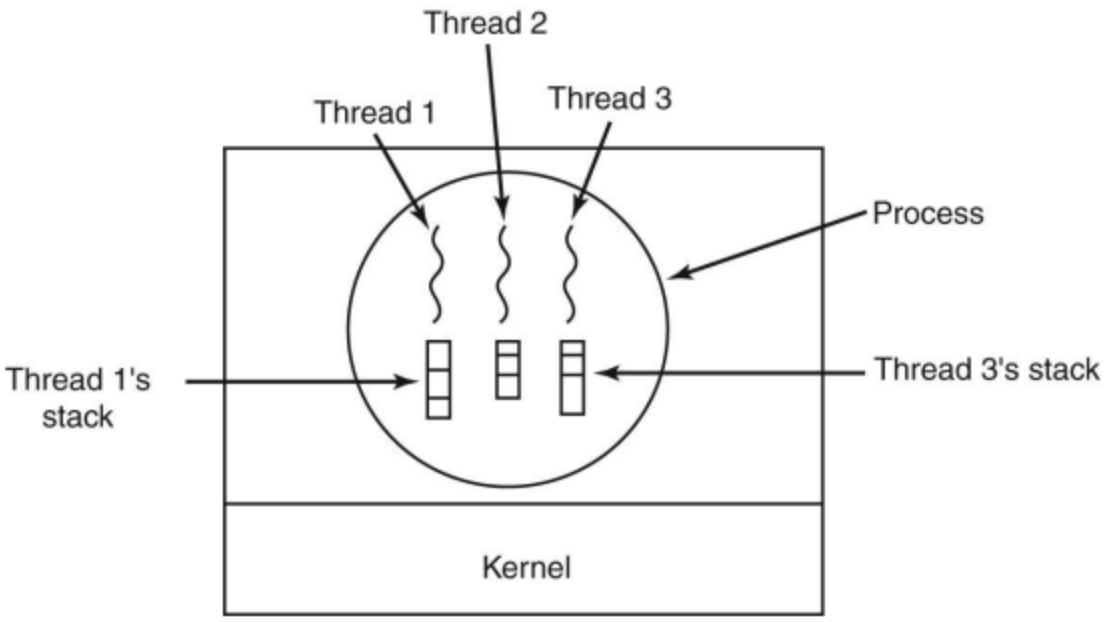
Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

▼ What is an **address space?**

You can find the wiki page here, but it's- as you know from CSO- the range of discrete addresses of memory for a various number of variables and data

- Time slices allocated only to processes, not individual threads

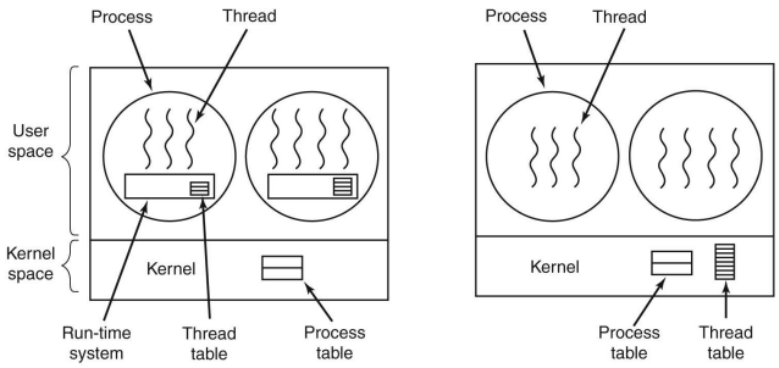
▼ Processes / Threads diagrams



Where to Put The Thread Package?

User space

Kernel space



- A **thread package** is a library that helps allocate the needed "items" for threads
  - In either user space or kernel space

▼ Implementing threads in **user space**

Implementing Threads in User Space

- Threads are implemented by a library.
- Kernel knows nothing about threads.
- Each process needs its own private thread table.
- Thread table is managed by the runtime system.

Implementing Threads in User Space

Advantages

- Very fast thread scheduling
- Each process can have its own thread scheduling algorithm
- Scale better

Disadvantages

- Blocking system calls can block the whole process
- Page fault blocks the whole process
- No other thread of the process will ever run unless the running thread voluntarily gives up the CPU

▼ Implementing threads in **kernel space**

Implementing Threads in Kernel Space

- Kernel knows about and manages the threads
- No thread management is needed by the runtime.
- Creating/destroying/(other thread related operations) a thread involves a system call

Implementing Threads in Kernel Space

Advantages

- When a thread blocks (due to page fault or blocking system calls) the OS can execute another thread from the same process

Disadvantages

- Cost of system call is very high

▼ What is the *runtime system*?

The environment in the OS that leaves space for programs to run