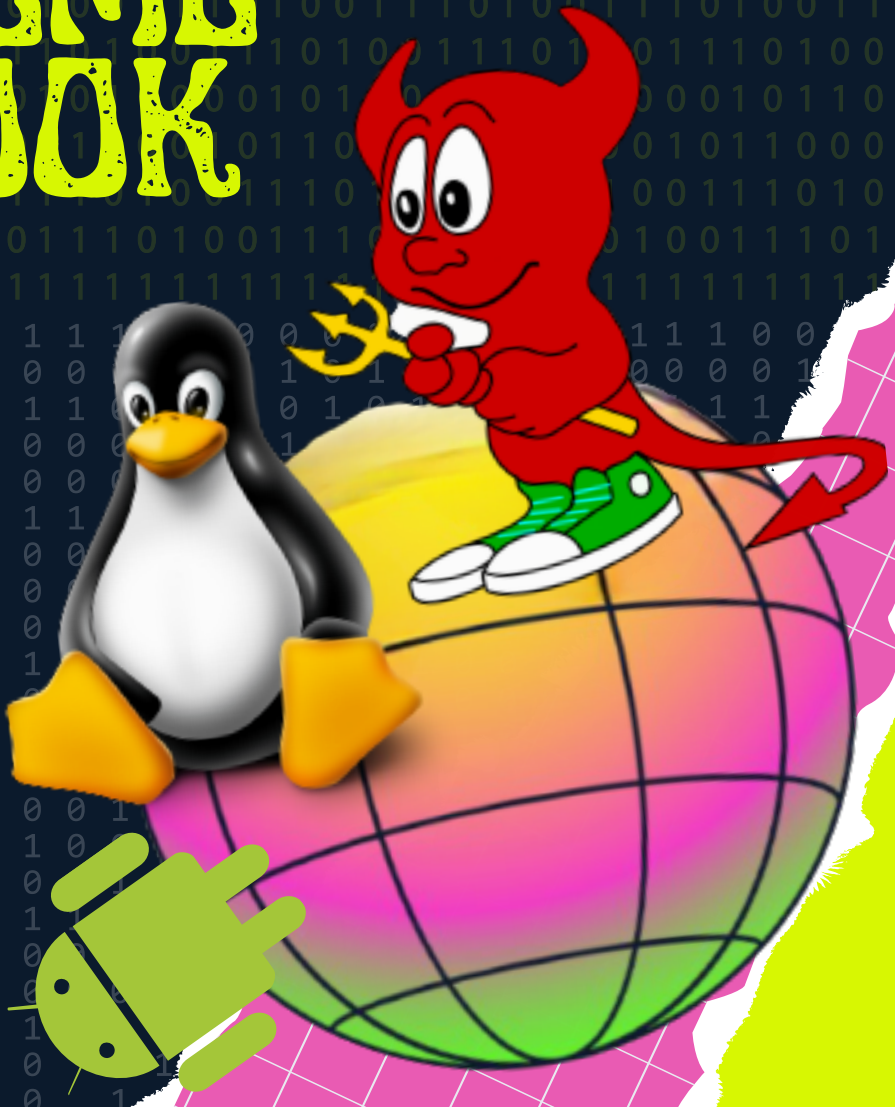
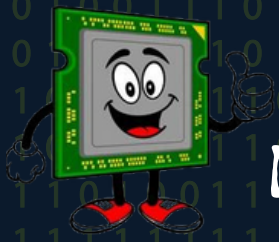
OPERATING SYSTEMS: MEME BOOK

PART 1



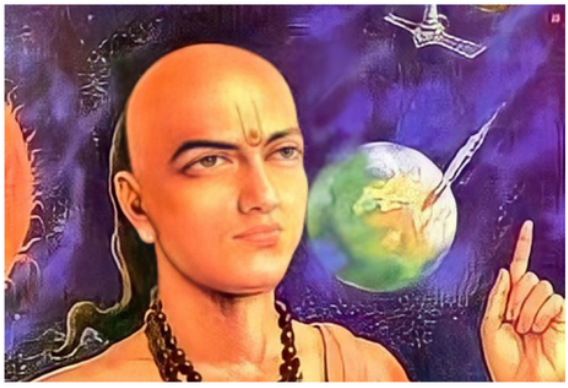
BHUMIKA & SAPTARISHI

CONTENTS



The Process Abstraction and the CPU

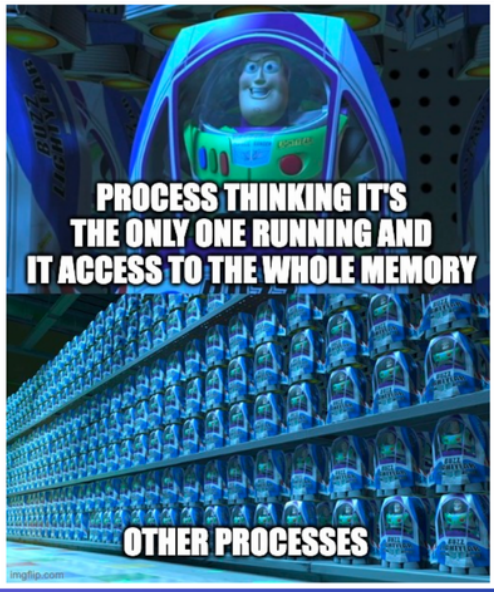
Number of times exec() returns



exec() returns **0** times

Return value in parent after the successful fork()

fork() returns the pid of the child in the parent



Kernel gives the illusion to the process that they are the only one on the system

fork creates a child that is a complete copy of the parent

Kernel cannot trust any process due to risks of it being malicious and harming the system

exec completely changes the child process depending on the content of the executable given as an argument

The executable is just a dead entity stored somewhere. To become alive it needs to actually execute and become a process

CPU is orders of magnitude faster than other system components like memory, disk and network

CPU being the fastest component on the system

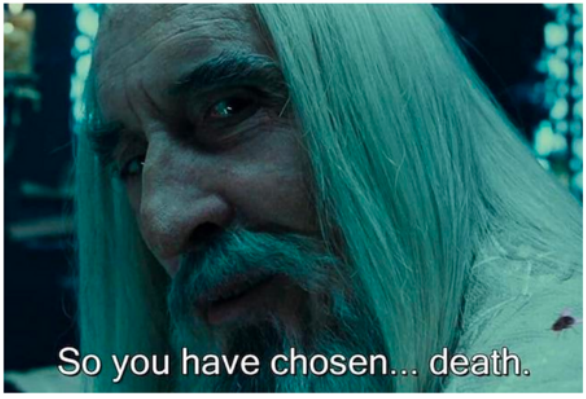
Kernel when the user process tries to access any info without having required permission

Kernel maintains permissions for all data for security purposes

A fork bomb leads to a process continuously replicating itself without end

```
while(1){
fork();
}
```

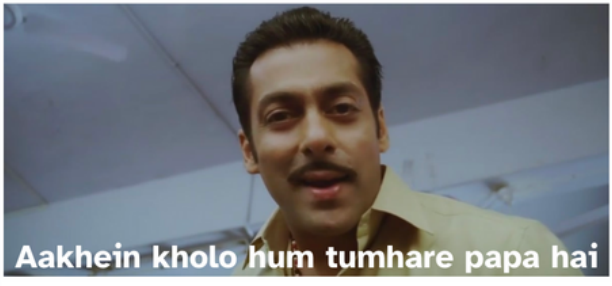
Kernel to process when it divides by 0



Division by 0 leads to a software exception, usually leading to process termination

According to one policy decision, when a parent dies all it's children get reparented to init

Parent process dies.
Child process needs a new parent
Init be like:



Processes after calling blocking wait()

Process enters blocked state until the child exits in case it has not exited yet

Life of a process

Birth

Please please please give me the CPU please please please

Death

The kernel managing all the processes, being the ultimate arbiter and in general having complete control over the whole system

Processes are made of one or more threads

When people use processes and threads interchangeably, but the processes be like

Kernel to the user mode after the interrupt handler is fired due to some interrupt/exception

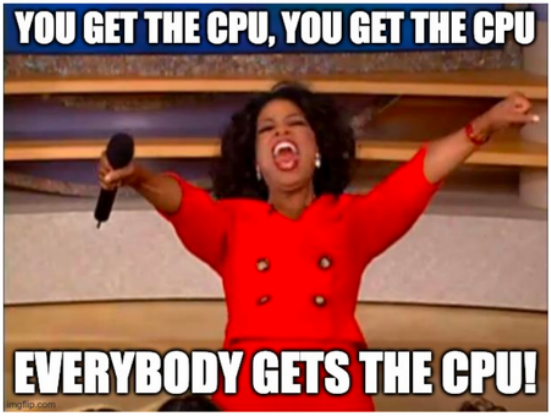
Kernel runs whenever an interrupt occurs

Maskable interrupts can be ignored by the kernel

Maskable interrupts after failing to get attention from the kernel

Scheduling

Round Robin Scheduling to processes



Round Robin gives all ready processes on the queue a chance to run

In co-operative scheduling there is no preemption which can lead to a process hijacking the CPU

Kernel in co-operative scheduling model when a process hijacks the CPU and doesn't yield

The world if oracular scheduling was possible

Oracle scheduling needs knowledge of future

Due to the probabilistic nature of lottery scheduling, it is possible for high priority processes to not get the CPU as well

Processes with many tickets when they don't get the CPU in lottery scheduling

Preemptive scheduling to the processes when they don't yield within the epoch time

Pre-emptive scheduling removes the process from the CPU at the end of epoch length

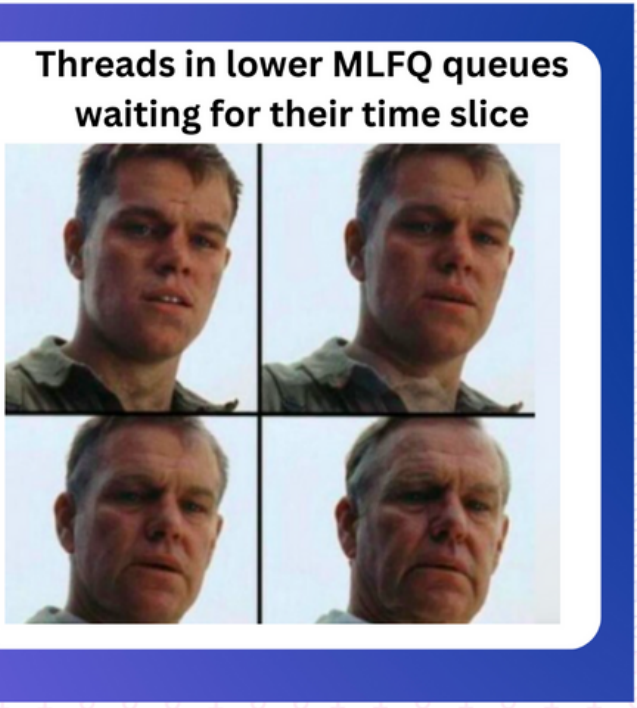
Basic MLFQ would not demote such processes despite their malicious nature

Malicious process yielding at 99% of the epoch time just to stay in the highest priority level in MLFQ

Completely fair scheduler be like

Due to the probabilistic nature of lottery scheduling, it is possible for high priority processes to not get the CPU as well

Processes with many tickets when they don't get the CPU in lottery scheduling



MLFQ first checks the higher queues before moving on to the lower queues

MLFQ demotes processes which use up their whole time quantum

Processes in the lowest priority of MLFQ to higher priority processes

High priority processes getting CPU multiple times because of tickets

Memory and disk

Process waiting for the disk read to retrieve the information

accessing data from a disk typically requires more time

Disk read operations are generally slower compared to memory read operations.

When a process needs to read data from disk

Kernel in the pre-MMU era to every memory access asking for virtual address translation

Hardware involvement in translation can make it faster : enters (MMU)

TLBs caches virtual to physical address translations in a hardware cache

TLBs making Memory Management faster

Kernel looking at the processes who are under the illusion that they have access to the entire, contiguous address space

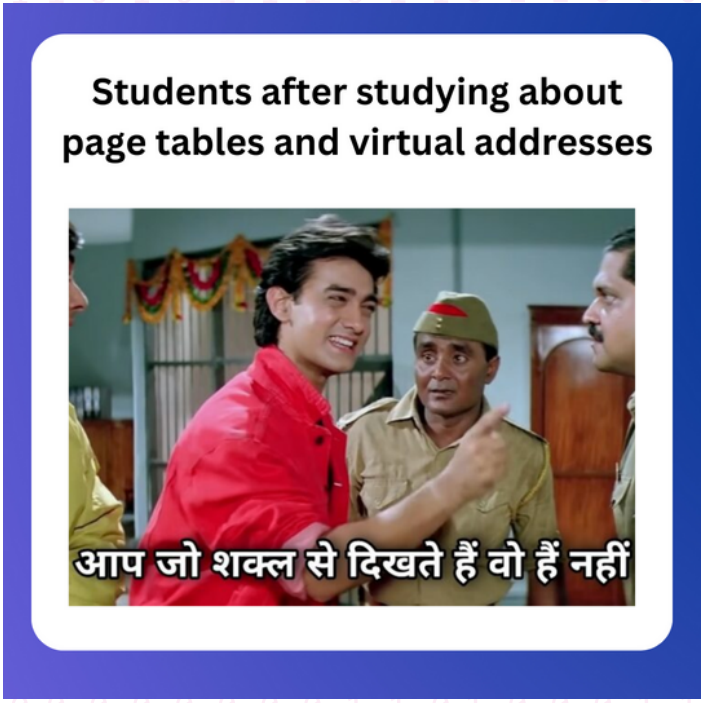
address spaces are intended to provide a private view of memory to each process

A virtual address points to something that acts like memory but isn't the true address

When you see what is "stored" at a virtual address

MMU asks the kernel for the translation for addresses not in MMU

There is a lot of illusion involved as the true physical addresses aren't shared due to security reasons



storages like DRAM loses its contents when power is turned off

HDD uses spinning disks made of magnetic material and have historic significance

SNEAK PEEK

(part 2 is subject to the funding and grade received by the authors)

Developers looking at critical section in a multithreaded program

Multithreaded program when you run it on a system with just one core

Threads when entering critical section

I had a problem I wanted to solve using multithreading, but I didn't want to use locks.

Two now prob lems. due to rac econditions Ihave

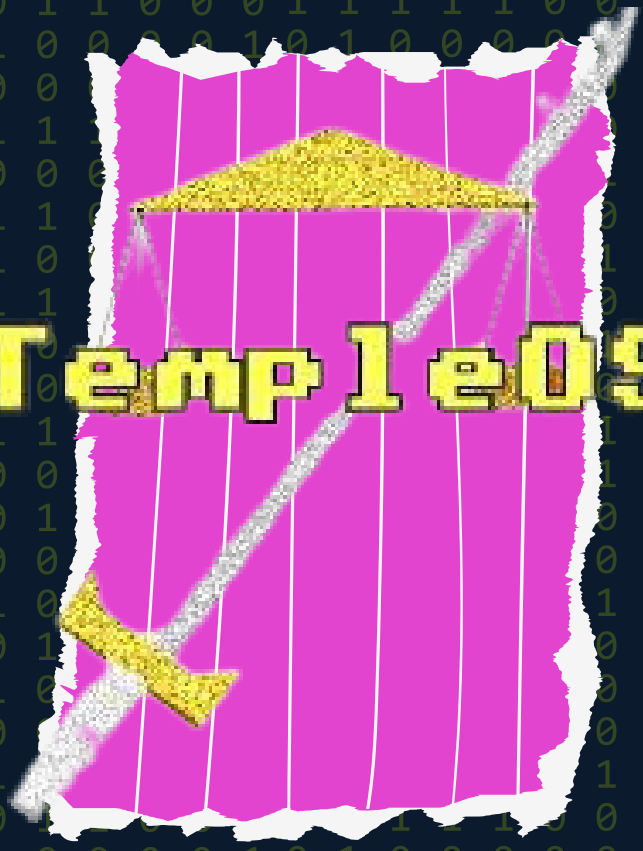
When the critical section is short

When the critical section is long

Kernel virtualizing the entire memory

Completely fair scheduler be like

The End



Brought to you by Team Losethos (iykyk).