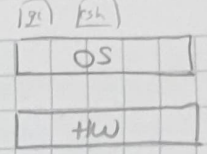
Kitap : Modern OS Tanenbaum and Bos 4th edition.
Yüzdeler (Approximately): 0/030 Midterm, 0/040 final, 0/030 HW 01.03.2021

OPERATING SYSTEMS



OS is a piece of software.

* That abstracts the computer HW through interfaces.

(HW abstraction)
- Hides the messy details underlying HW
- Present users with a resource abstraction that is easy to use
- Extends or virtualizes the underlying machine

* Manage the resources

(resource manager)
- Processors, memory, timers, disc - - -
- Allows multiple users and programs to share the resources and coordinates

OS is managing who is running now (timer, memory, syscall...)

OS → kernel mode

* Performs services
  - Abstract HW
  - Provides protection (gcc, csh memory isolation etc.)
  - Manages resources

MSKI → M interrupt (Kernel'de interrupt olmazsa crash yapabilir.)

As RM, ✓ sharing resources in 2 different ways:
1- In time (Time sharing ex. printer, CPU, Network Adapter
2- In space (Disc, Memory)

✓ allows multiple programs run at the same time
✓ Manage and protect memory

Without OS, we can't run so much HW

OS vs. Kernel

- Wide view
  Windows, Linux are OS
  Includes system programs, libraries...

- Narrow Definition
  OS often equated with kernel
  The Linux kernel; the Windows executive - the special piece of sw that runs with special privileges and actually controls the machine.

2.3.2021

OS not always running.

ls, ps etc. is part of the standart bash shell

So they are not part of the OS.

* for Linux, drivers are in kernel mode.
  memory management " "
  signals " "

History of OS

1- 1st Generation (1945-55)

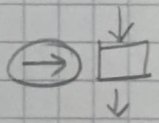
Vacuum Tubes.

Her 20 dakikada bir patlayan tüpler olduğu için veriniz.

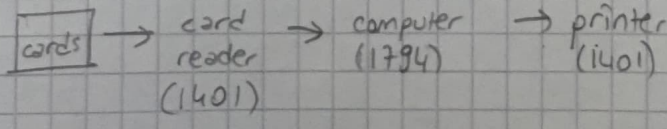
2 - 2nd Generation (1955-65)

Transistor and Batch System

Transistor controls the flow of current



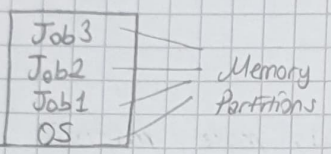
Punchcard : 80 columns 12 rows cards



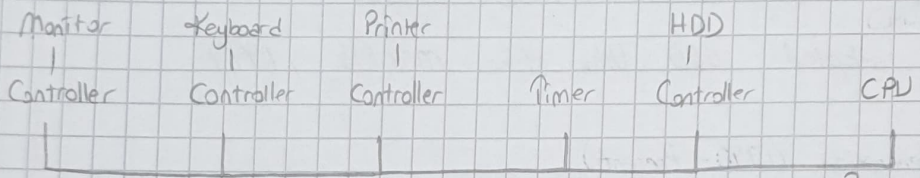
Computer was very expensive

3. 3rd Generation (1965-1980)

ICs and Multiprogramming



1401s were no longer needed



Timesharing

Still cost of million dollars

- Multiprogramming

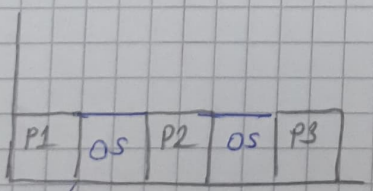
Run several programs at once

Protect each one's memory against the others

When one program needs to do I/O, let another program uses the CPU

As necessary pre-empt a CPU bound program to let another run

Use OS "daemons" for spooling → Uzun süre çalışan arkaplan işlemi



timer interrupt
syscall

4- 4th Generation (1980 - Present)

Personal Computers

* Large Scale Integration Unit
* IBM PC were designed
* Bill Gates paid 75000 $ for DOS, made MS-DOS
* Engelbart invented GUI, mouse, window
* Apple adopted BSD Unix for macOS, 1999

5- 5th Generation (1990 - Present)

Mobile Computers

* Nokia released N9000 (mid 1990)
* Ericsson coined term smartphone (1997)
* RIM's Blackberry OS (2002)
* Apple IOS (2007)
* Android Linux based OS by Google (2008)

Security and battery management

4th (Gen... Present)

Large Scale Integration Circuit

IBM designed the IBM PC

Bill Gates paid 75000 for DOS, made MS-DOS

Engelbart invented GUI, mouse, windows

Apple adopted BSD Unix for macos, 1999

5th Generation (1990 Present)

Nokia released N9000 (mid 1990)

Ericsson coined term smartphone (In 1997)
RIM's Blackberry OS (in 2002)
Apple IOS 2007

Android Linux based OS by google 2008

Security and Battery mangement

8.3.21

MMU → Memory Management Unit

~~Why are we~~ Responsible for making vmemory possible. Without vm, OS doesn't work. makes our processor think that ~~CPU~~ it has always

HD controller has register to read and write

---

* Processor

Brain of the computer
fetch, decode, execute → pipelining
CPUs have registers
PC
Stack Pointer
Program Status Word (PSW) : condition bits (flags checking equality), kernel mode, CPU priority
A user program must take a syscall, which traps into the kernel and wake OS

global doesn't use stack

gcc -S test.c
less test.s

* Pipelining

F → D → E

F → D → E
F → D → E
Holding Buffer

8.03.21

* Multithreading / Multicores

Moore's Law → x2 transistors every 18 months

Single processor can process more than one thread officially (multithreading)

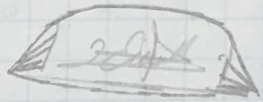
Multicores: Real multiple CPUs on the same chip

GPUs are extreme examples of multicores (thousands of cores) simpler than CPU

* Memory

Should be fast, large, cheap
Not all possible with the current technology

* Registers and Cache



(register) 32x32 bits for 32 bit system
64x64 " " 64 " "
No delay accessing them
L1 cache 16 KB
L2 cache a few megabytes

* Disks

Larger, cheaper, slower
Slow because the mechanical (spinning)
Load OS and apps from disk
Manage disk space allocation
Store and retrieve files by name
Provide some protection for files

* I/O Devices

piece of HW
piece of SW

Controller and the device
Very complicated, controller hides this
Driver: Middle layer sw for device controller by manufacturer

Three types of I/O

1- Busy waiting → CPU, Disk: becker (I/O: slaves)

2- Interrupts → CPU asks the driver to get the data vs. process blocks. When data is ready on int. process is generated and sent to queue

3- Direct Memory Access (DMA) → CPU sets up the DMA chip telling it how many bytes to transfer, the device and memory addresses involved, and the direction, and lets it go

1- interrup
2- Dispatch
3- Return

Interrupts vs traps

(hw) → 1st way

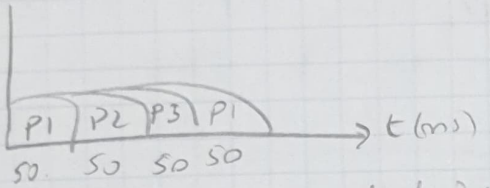
(sw) → 2nd way of switch → syscall

read(...,...,...)

1- push byte
2- " buffer
3- " fd
4- call read
5- Put code for read in reg.
6- Dispatc ↓syscall handler → trap kernel
8- Syscall hand
9- Return to caller
10- Increment SP

9.3.21

Thread context switch costs much



Lets 1000 1000 1000 1000 → bad interactivity, less switch

OS zoo

Mainframe, Server, Mul. Processor, PC, Handheld, Embedded, server node, Real time, Smart Cards

Mainframe → Server OS

Provide server some of them large. It can provide print service, file service
Typical are Solaris, Free BSD, Linux, Win 20xx
Serve multiple users

Multiprocessor OS

More than one processor
Job of OS is more difficult
~~the hard part~~

PC
Linux, Win, Appli
Interactivity important

Handheld
Android, Apple's IOS smartphone and tablets

Embedded
run on computers that control devices
TV's, microwaves
No need for protection
Don't accept user-installed software

Smart card OS
JVM
credit card
Smallest OS
Rom holds an interpreter

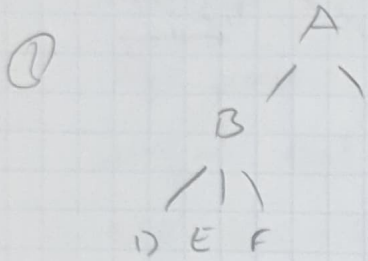
Sensor Node OS
Is a real comp with CPU, RAM, ROM
Small, real OS, event driven

Real-Time OS
Most occur at a certain moment (or range), we have a hard real time sys
There is not a necessity to wait
50 ms

Multiprocessing → Doesn't need multi processor

Processes

pid = fork() return 0 if it is child



C → child process. pid = waitpid (pid, &statloc, options)

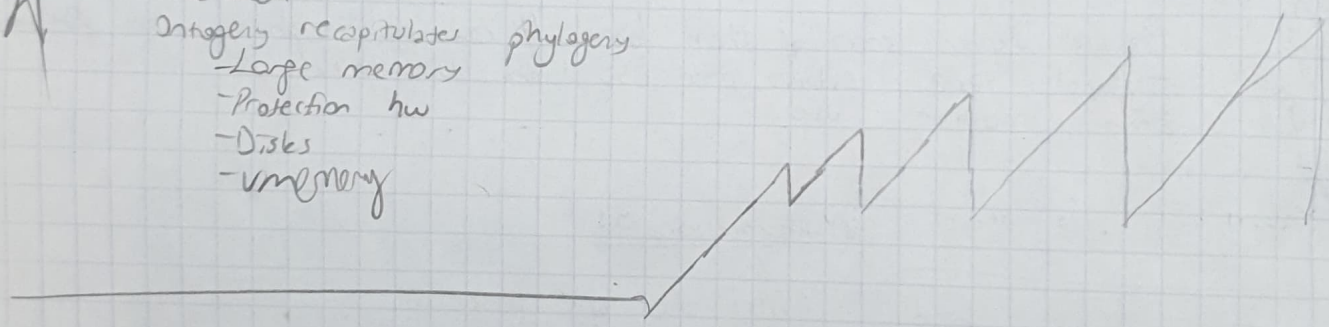
s = execve(name, argv, environp)

exit()

ps -ax

OS Concepts

1- Processes
Address Spaces
Files
I/O
Protection
The Shell
Ontogeny recapitulates phylogeny
- Large memory
- Protection hw
- Disks
- unemployed



1- Processes

It is basically a program in execution

Process table, array of structures, one for each process currently in existence

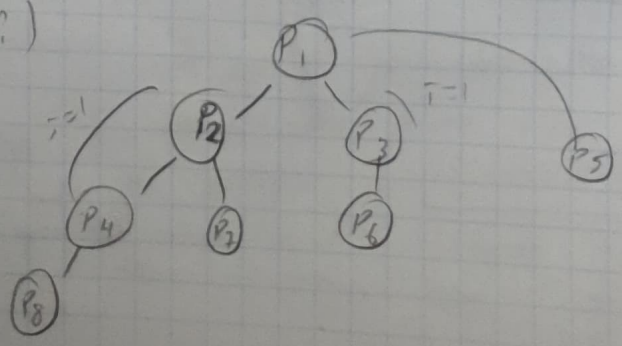
- Registers, open files, other data

If parent is terminated, OS kills child process.

```
for (i=0; i<5; ++i)
    fork()
```

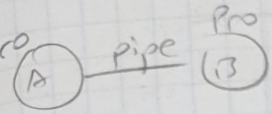
$2^5 - 1$

1+2+4+8+16



Files

most PC OS have the concept of a directory as a way of grouping files
- Path name, root directory, working dire., file descripto



connect two processes

ls -alRt | wc -l

Ontogeny - —

Develop. of an embryo (ontogeny) repeats (rec...) the evolution of the species (ph.)

Each new species of computer

POSIX has about 100 procedure calls

// proc calls onto system calls is not one-to-one
OS calls are too expensive (user to kernel, datas etc)

Proce. and sys calls are different

exit has no path. But it runs. How? if (strcmp(command, "exit") == 0) exit(0)
Syscalls

16.03.2021

Syscalls for File Management

fd = open (file, how, ...)

s = close (fd)

n = read (fd, buffer, nbytes)
n = write

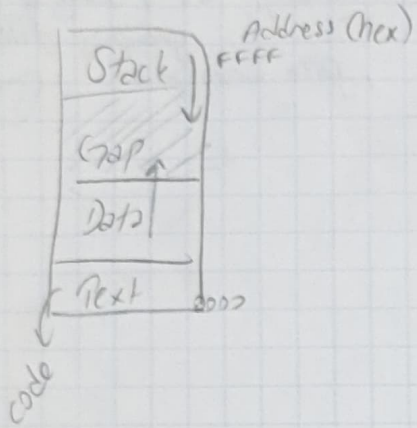
link (name1, name2)
unlink (name)
mount (special, name, flag)
umount (special)

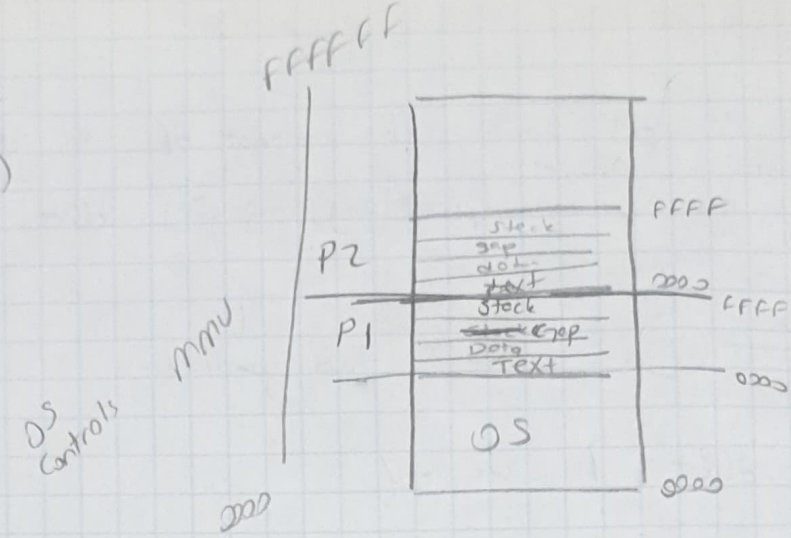
s = chdir (dirname)
= chmod (name, mod)
s = kill (pid, signal)
seconds = time (&seconds)

Miscellaneous SysCalls

Sends signals

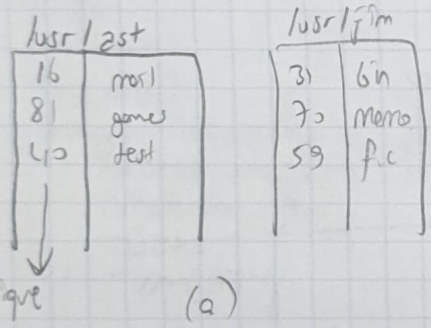
Memory Layout



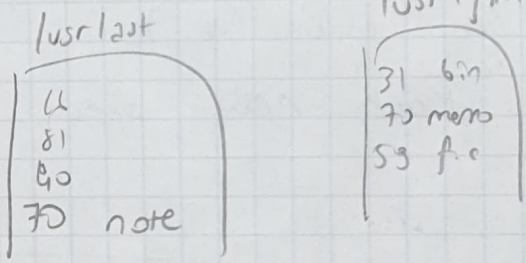


Linking

i-node

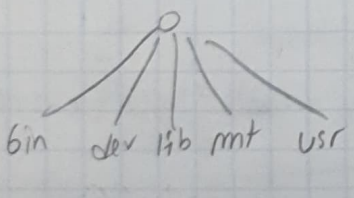


(a)

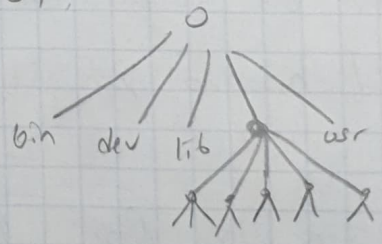


link("/usr/jim/memo","/usr/ast/note")

Mounting

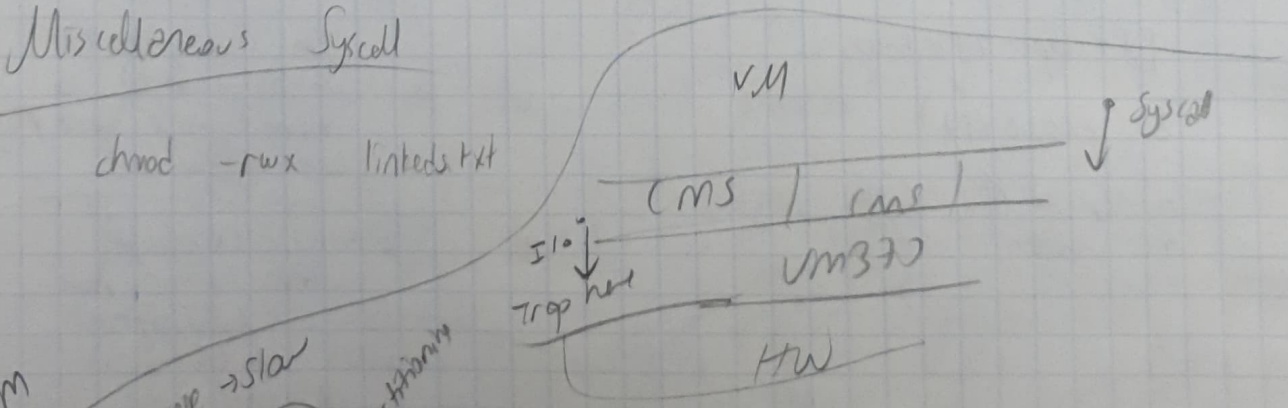


mount("/dev/sdb0","/mnt",0);



Miscellaneous Syscall

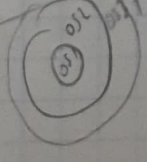
chmod -rwx linkedstxt



The JVM

Disadvantage → slow

Exokernels

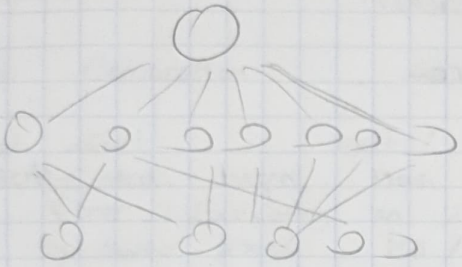


If it wants each HW, VM can handle in real OP.S. in VM process we have supervisor

hypervisor

OS Structure

- Monolithic Systems



Main Procedures

Service procedures (file I/O etc.)

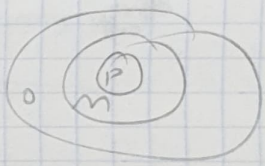
Utility Process

Layered Systems

→ ancestor of UNIX

Instead of layers, MULTICS was described as having a series of concentric rings

5 operator
4 user prog
3 I/O
2 op-process com.
1 Memory and drum man.
0 Processor allocation



Microkernels

1000 - 1,3UG

Monolithic OS of 5 million line codes is contain between 10k-50k kernel bugs.

Keep kernel small!

MINIX 3 microkernel is 12 000 lines C and some 1400 lines assembler

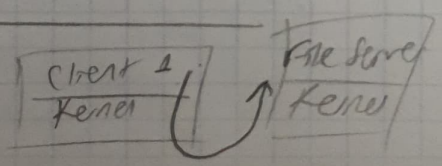
- Policy usr mode (→ complicated All algorithms called policy) Microkernel handles interrupts, processes, scheduling, IPC
- Mechanism kernel (Reading disk) (Not smart)

Other of them is in user mode

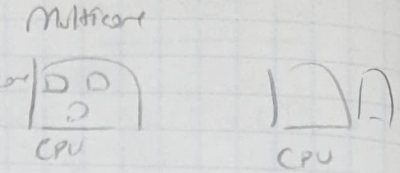
Overclocking: Increasing the clock rate of CPU 1 GHz → 1.2 GHz

Drivers in usr mode is smart. Not crash whole OS

Client-Server Model



CHAPTER 2
PROCESSES AND THREADS

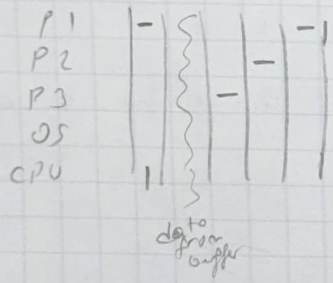


- Most central concept in any OS is process:
  An abstraction of a running program

- Pros are one of the oldest and most import abstr. that OS provide

Single Program vs. MProgramming

Each pro runs to completion, but intermixed with other pro--



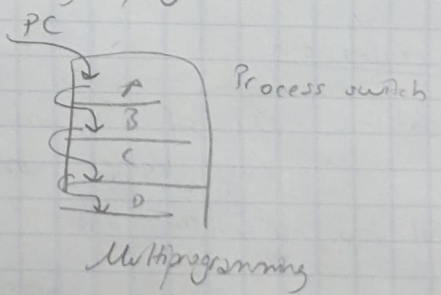
16.03.2021

Process Model

A process is just an instance of an executing program

Conceptually each process has its own virtual CPU

We will assume there is only one CPU
Increasingly however that assump is not true, since new chips are often multicore



Process Creation

1- System initialization; Process that stay in bg to handle some activity mail etc and so on are called (daemons) → long running bg processes that answers requests for services

2- Execution of a process creation syscall by a running process (fork)

3- A user request to create a new proces

4- Initialization of a batch job

Create new pro fork
change its memory img execve
Windows 100 commands

Proce Termination

1- Normal Exit exit / ExitProcess
2- Error Exit
3- Fatal Error : Executing an illegal instruction, referencing nonexisting memory, divide by 0
4- Killed by another process: Kill / Terminate Process

Proce. Hierarchies

- In some systems, parent and child continue to be associated in certain ways
- In UNIX process and all of its children together form a process group. A special process, called init, is present in the boot image
- Windows has no concept of a process hierarchy

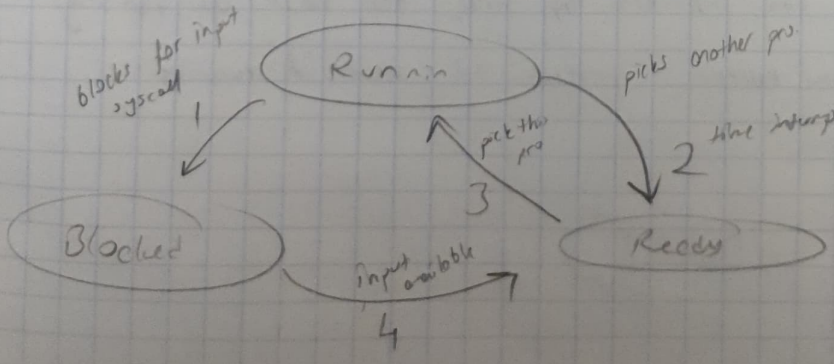
Process States

cat chapter1 ch2 ch3 | grep tree

Depending on the relative speeds of 2 proce., it may happen that that grep is ready to run, but there is no input waiting for it. It must then block until some is available

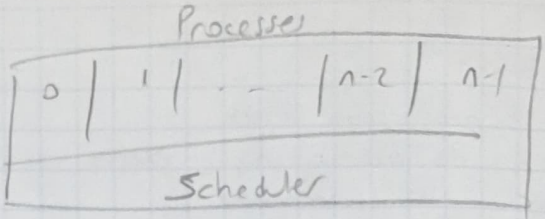
(1) ode! 1- Running (using CPU at that instant)
(several) 2- Ready (runnable, temporarily stopped to let another process run)
3- Blocked (unable to run until some external event happens)

sleep(200)ms



Lowest level of OS is scheduler

All the interrupt handling and details



of actually starting and stopping process are hidden away in what is here called sched

Imple of Processes

A table (an array of structures) called process table, with one entry per proces

Process Management	Mem. Man	file m
Registers PC SP PID CPU time used Alarm ⋮	Pointer to - seg. info stack data text	Root directory Working File descriptor User ID Group ID

User process in running when a disk interrupt

HW saves only → minimal register PC or sometimes determines which one to use

HW(1- HW stacks pc, prog. status word (not all registers, why?)
2- HW loads new pc from interrupt vector.

3- Assembly-lang. procedure saves registers. All interrupts start by saving the registers often in the p.table entry for current process

4- " " " sets up new stack. Saving the registers and setting the stack pointer cannot even be expressed in C!

5- C interrupt service runs (typically reads ad buffers input)
6- Scheduler decides which process is to run next
7- C procedure returns to asm code
8- Control is passed back to the asm lang. code to load up the regs and memory map for the new current process ad start it running

Address of handling

main mem. buffer or interrupt register

Interrupt HW

22.03.2021

Modeling Multiprogramming

P_1 30% wait I/O → Efficiency 70%

P_2 30% wait →

P_1	RU	B/.	RE/B	
P_2	B/RE	B	RU	

0,3 × 0,3 = 0,09

CPU utilization = $1 - p^n$ n process are waiting for I/O is p^n

$1 - 0,09 = 91\%$ Efficiency

For 3 pro $1 - (0,3)^3 \cong 0,98$

Ex.

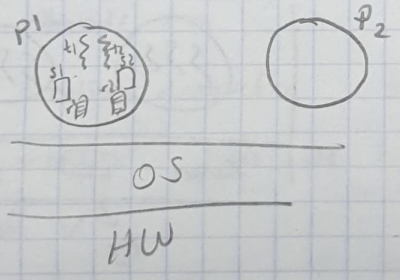
OS takes 2GB
Each user prog 2GB

in 8GB with 60% wait
We can run 3 programs at once CPU ut. $= 1 - (0,8)^3 = 49\%$

THREADS

Like processes: programs in parallel

- Share the same address and data



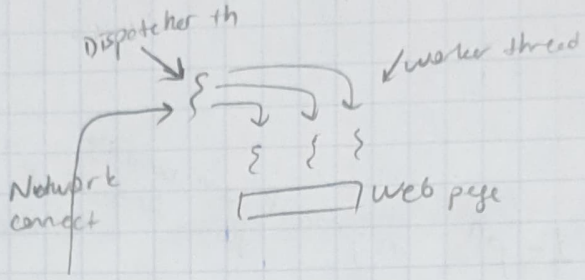
- Lighter and faster to use
- There is no protection bw threads
- Threads co-operate, they do not compete

∝ Main reason is in many app., multiple activities are going on at once

Some of them may block from time to time.

for ex. → MS word has 3 threads

" → Web Server



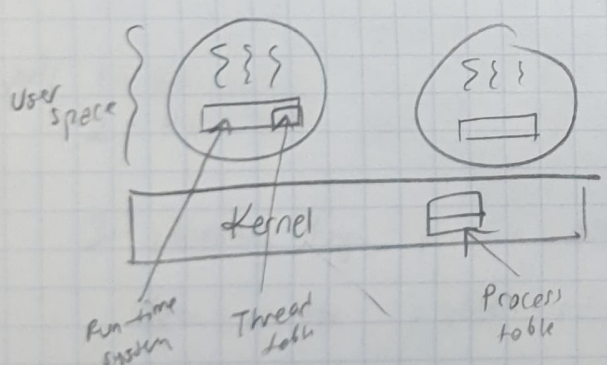
Dispatcher thread

```
while TRUE
    get next request (&buf)
    handoff work (&buf)
```

Worker Thread

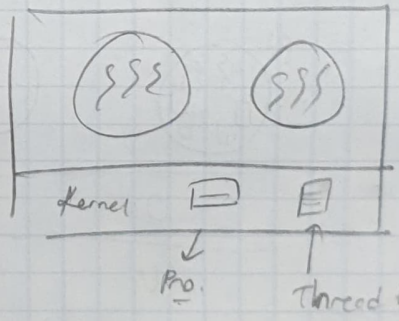
```
while TRUE
    wait for work (&buf)
    look for page in cache (&buf, &page)
    if page not in cache (&page)
        read page from disk (&file, &page)
    return page (&page)
```

Imp. Threads in User Space



1st

OS doesn't know thread



2nd

Main Adv.: Speed ∴ no context switch
2- Handle everything in user (flexible)

Disadv: If one of threads is blocked, all threads will be blocked

Model			
Threads	Paral ✓	Blocking syscall	
Single Thread Processes	No paral	Block	
FSMachine	Paral	NonBlock	Interrupts

Lightweight processes

If one thread open a file, others can access them

Each thread has own PC, Registers, Stack, State

23.02.2021

Kernel threads → Linux, Windows etc

In modern OS → Parallelism is important (Run threads ~~at~~ in same time)

User Level Threads

- Thread creation, destruction, scheduling done at user level
- Thread scheduler finds the highest priority thread

No syscall ✓

No context switch ✓

No copying data safely accross a protection boundary ✓

No need to flush the hw memory cache ✓

Allow customized scheduler

Disadvantages

Blocking syscall

Page faults

Inability to benefit from multiprocessor CPUs

Kernel can't automatically grow per-thread stacks

Imp. Threads in Kernel space

Kernel manages thr

Similar thr table but in kernel

No problem with blocking syscall

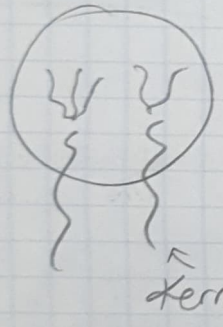
Handles multi CPU easily

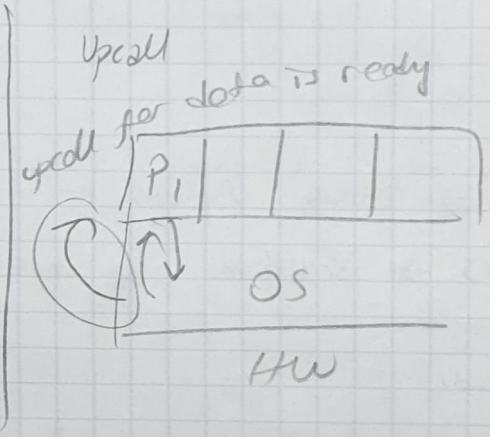
Thread switching is expensive

Still cheaper than process switches

Preemptive Scheduling -> Pro. uses too much CPU, I am going to take the CPU away

Hybrid





Scheduler Activations

Thread creation is done at user level

When A is ready, kernel does an upcall

Careful about non-blocking

Making Single Thread Code Multithreads

Local variables are not cause any trouble

One solution is to prohibit global variables altogether

Another -> It is possible to allocate a chunk of memory for the globals and pass it to each procedure in thread as parameter

create_global / set_global

Static shouldn't use

Must use reentrant routines

can be invoked more than once simultaneously

can only use local or dyn.a

must not use static

many c libraries routines do use static buffer

Interprocess Communication (IPC)

First issue: One process can pass info to another

2nd issue: 2 process airline reservation system to grab last seat

3rd issue: Process A produced data, B prints. B has to wait until A has produces

Race Conditions

2 or more processes are reading or writing some shared data and the final result depends on who run precisely

Critical Regions

Mutal exclusion: One process shared variable (allocation, dispeler) will be excluded

Part of the prog. where the shared memory is accessed is critical region

Requirements to avoid race conditions:

1- No 2 processes may be simultaneously inside their critical reg.

2- No assumptions may be made about speeds or the number of CPU

3- No process running outside its critical region may block other processes

4- " " should have to wait forever to enter its critical

Mutual Exclusion With Busy Waiting

Continuously testing a variable until some value appears

- Proposals for achieving mutual exclusion
  - Disabling Interrupts
  - Lock variables
  - Strict alternation
  - Peterson's solution
  - TSL instruction

Disabling Interrupts

Usuable only to the kernel

Better not be available to user proces!

Doesn't work well on multiprocessor

Special case solution for kernel only

Lock Variables

global variable lock

P1

```
while (lock != 0)
    ;
Start critical
lock = 1
    End
lock = 0
```

there is a window check 0 to 1 and set it to 1

P2

```
some
```

~~Mutex~~

Busy waiting

```
while (TRUE) {
    while (turn != 0); //loop!
    critical_region();
    turn = 1;
    noncritical;
}
```

```
while (TRUE)
    while (turn != 1);
    critical();
    turn = 0;
    noncrit;
}
```

Strict Alternation

a

a is blocked enter region

a lock is called a spin lock

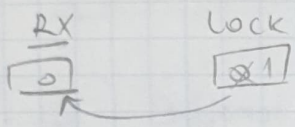
30.03.2021

— Peterson's Solution

```
while (1)                    P0
    enter region (0)
    critical region
    leave region (0)
    non critical
```

→ P1 same

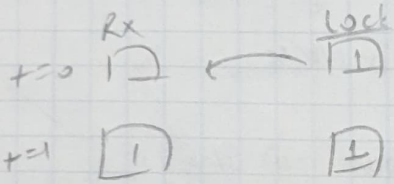
— TSL Instruction (Test and Set Lock)



lock'taki değeri Rx'e al, sonra lock'u değiştir.

It's guaranteed to be indivisible

or



— Spin Locks & Priority Inversion

I have 2 processes. H is on 1 ~~something~~

L never runs if H is runnable

While H sleep, L grabs the spin lock

H wakes up and tries to get the lock

It spins, waiting for L to free it

But L can't get the CPU, so it doesn't progress

One example of deadlock

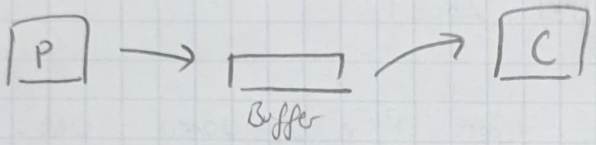
- Blocking Instead of Busy Waiting

Prevent wasting time busy waiting

Simplest way sleep + wakeup

Syscall sleep causes to block (suspend until another process wake it up)

Classical Problem: Producer - Consumer



Race Condition

Producer

```
while (1){
    item = produce()
    if (count == N) sleep() // empty
    insert item()
    if (count++ == 0)
        wake (consumer)
}
```

begin
end

Consumer

```
if (count == 0) sleep()
item = remove
if (count-- == N)
    wake(producer)
consume item
```

begin
end
full

Semaphore

down - up

Binary semaphore = mutex 0 or 1

down → sleep → decrement if it > 0

up → wakeup → increment

Implement

Typically done by kernel

Mask interrupts while manipulating semaphore

On multiprocessor, use TSL as well

Both of these are for only a few ms - not a serious problem

semaphore mutex = 1
semap empty = 100
semap full = 0

while(1)
item = produce
down(&empty)
down(&mutex)
insert
up(mutex)
up(full)

down full
down mutex

up mutex
up empty

---

2 Different Uses of Semaphores

1- Counting semaphores (full, empty) → Sleep / wait

2- Mutual Exclusion semaphores (mutex) → Make sure just 1 process in critical region

Mutexes are special case of semaphore. Useful when no counting

If TSL is available, easy to implement at user level

Try to grab lock, if unsuccessful let another thread run.

Different Processes?

If they have disjoint address spaces, how can they share semaphores or a common buffer?

1- Shared vars in kernel

2- Many OSes offer a way for processes to share some portion of their address spaces with other processes.

05.04.2021

Futex (fast User Space mutex)

Busy wait or lock faster?

Block only if necessary and switch to kernel space

Condition Variable

Second synch mechanism (Mesela switch için 2 cond lazımsa)

Cond. var. allow threads to block due to some condition not being met

cond_wait(&condp, &the-mutex) → needs both of them

Risks of Semaphores and Mutexes

Using correctly is difficult

If you get wrong, deadlock !

Testing is difficult, because it's all timing-dependent

We need higher level construct.

- Monitors

A pl construct

Java has, C/C++ not

Like a class, but only 1 thread can be executing in it at a time

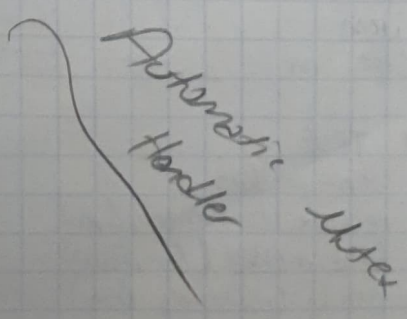
Like language implementation of mutexes

They are good for mutex and you cannot go wrong with mutex

Ex:

```
monitor example
    integer i;
    condition c

    procedure producer();
    .
    .
    end
    procedure consumer()
    end
end monitor
```



Monitors and Application Blocking

We still need a way to block

2 new operations: wait and signal

Java

```
static our_monitor mon = new our_monitor()
```

Disadvantages of Monitors

Only available in a very few languages

They don't work well without some shared memory

Şimdiye kadar user or kernelda shared memory varsa diye konuştuk

spin locks
semaphores
mutexes
cond variable
monitors

What if memory space is not shared? Processes are running on 2 different machines

Message Passing

send (dest, &msg) — Sender blocks if receiver's buffer is full

recv (src, &msg) — Receiver blocks if there is no data ave.

-Consumer Produce örneği:

```
producer (void) {
    int item
    message m

    while TRUE
        item = produce
        receive (consumer, &m)
        build_message (&m, item)
        send (consumer, &m)
```

```
consumer (void)

N empty buffer
for (i=0 i<N ) send (producer, &m)

receive (producer, &m)
item = ...
send ( . . )
consume_item (item)
```