# Introduction – About Minix

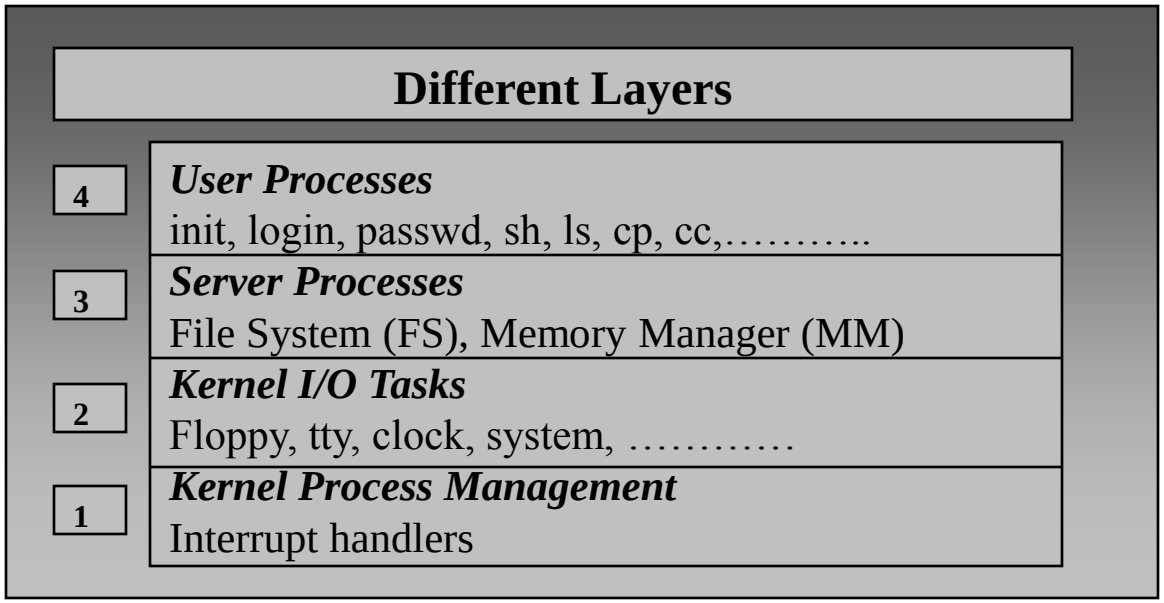
- Minix – Mini Unix (Minix) basically, a UNIX - compatible operating system.
- Minix is small in size, with microkernel-based design.
- Minix has been kept (relatively) small and simple.
- Minix is small, it is nevertheless a preemptive, multitasking operating system.

# Continued

- Modularity in Minix
- Source Code – C language
- Networking support – TCP/IP protocol

# Architecture – Which You must be familiar..

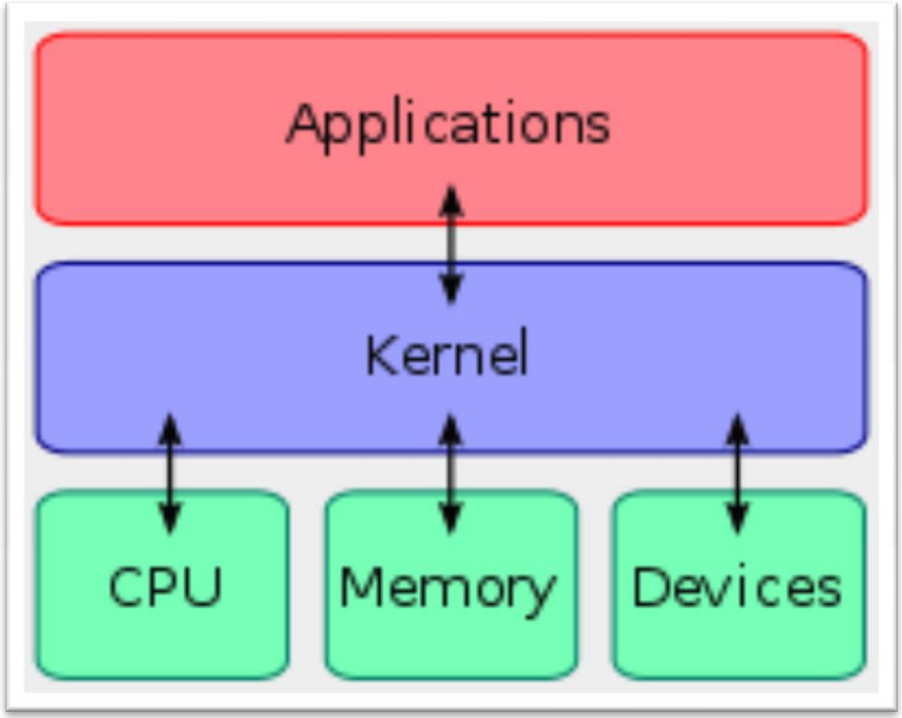
- The figure represents the internal architecture of Minix.



# Kernel

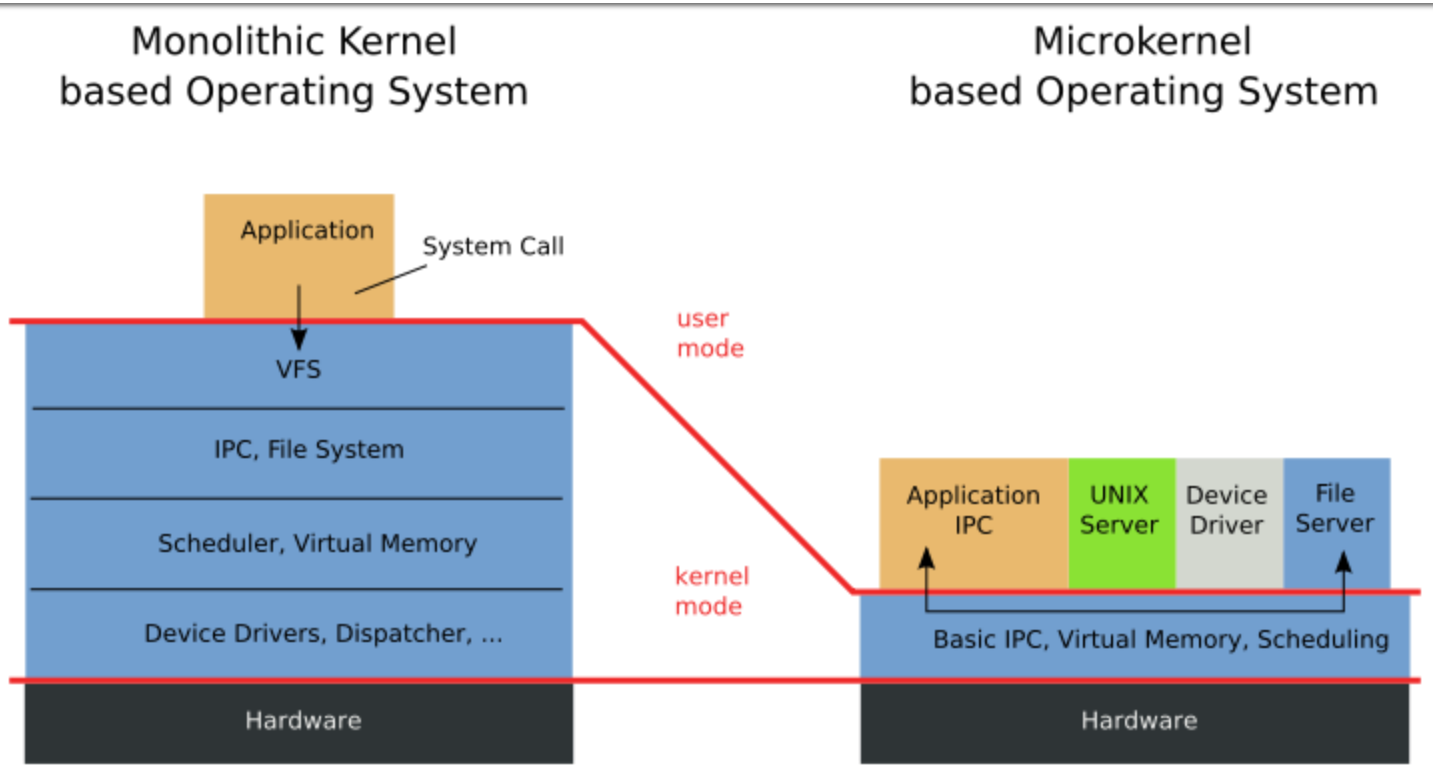
- central component of operating systems
- Manages the system's resources (the communication between hardware and software components).
- lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that application software must control to perform its functions.

# Kernel




A kernel connects the application software to the hardware of a computer.

# MicroKernel




- horizontal stucture
- System services are obtained by executing an IPC system call addressed to a particular server.

# System Calls – Flow of Control

- The flow of control in minix from the user level to the kernel level is handled by system calls.
- Because of its microkernel structure, it actually only has three system calls: send, receive, and sendrec.
  - user processes are only allowed to use the last one
- If you want a subsystem to do something for you, you send a message to the subsystem you have in mind.

# Processes

- Inter-process Communication
  - Is handled by the kernel
  - A process sends a destination and a message to the kernel, which then copies the message to destination process
  - A process must be waiting for the message in order to receive

# System Calls – Flow of Control

- Minix has a set of system calls which are located in the table.c file.
- The file system (fs) has its own table and so does the memory manager(mm).

# Continued…System calls…

- A system call in minix is similar to a system call in any system.
- A user-level process cannot directly access a disk. Instead it asks the kernel to obtain data from a file for it (the read system call).
- A user-level process cannot create another process. Instead, it asks the kernel to create one for it.
- **User Program ==> Library ----> Kernel ==> MM/FS**

# Continued…System calls…

- System calls are typically made via library routines. Library routines are normal procedure calls. They perform the setup for the system call.
- In minix the system call functions in a similar fashion.
- The prototype of the libraries are defined in /usr/include/unistd.h or other header files according to the system call

# Continued….System Calls….

- To implement a system call in minix one needs to do the following steps:

- Look for a free slot in the table.c file
- Follow the standard convention

```
 no_sys,      /*  0 = unused  */
do_exit,      /*  1 = exit    */
do_fork,      /*  2 = fork    */
do_XXX        /* 77 = XXX     */ → your system call entry
```

# Continued ….. System Calls…

- The system call would be named in the following manner

  do_XXX
- Once the system call method has been written, its declaration should be mentioned in the function declaration header file: **“proto.h”**
- **_PROTOTYPE ( return type do_XXX, (arguments if any) );**

# Continued….System Calls…

- The library should also be informed about the system call that would be called by the user.
- The library file for the system call is written in the lib/posix directory. The file naming convention is followed here it starts with an underscore ( _XXX )
  Eg: - _fork.c (this code is defined in /lib/posix)

```
#include <lib.h>
#define fork _fork
#include <unistd.h>
PUBLIC pid_t fork() {
        message m;
        return(_syscall(MM, FORK, &m));
}
```

When you add a new system call, you need to add code as above.

# Continued….System Calls…

- It calls _syscall (MM, FORK, &m);.
- The first parameter is the destination. This is always MM/FS. When kernel receives a system call request, what the kernel does is to send a message to MM/FS. Actual work is done by MM/FS.
- The second parameter specifies the type of service. The ID is defined in /usr/include/minix/callnr.h .
- User program cannot call _syscall() directly. The structure of a message is defined in /usr/include/minix/type.h.

# Continued….System Calls…

- Once the code for the system call is written in the "lib/other" directory a system file needs to be created. The system file is present in the "/src/lib/syscall" directory and instructs to jump to our _XXX code.
- A standard naming convention is followed here too. The filename extension is ".s".

```
.global XXX

XXX:
ba      _XXX
nop
```

# Minix Messages

- 6 different types of message structs in Minix
- All part of a union

- Check: /usr/include/minix/type.h.

# Minix Messages

- To pass a message a variable of a specific data type is defined, and the variable is added to the message structure as shown in the example below.

- To add a message struct:

- The message structure defined is as follows:
- Typedef struct { datatype mnYY; } mess_n;

  n => message number of the message
  YY => variable name

# Minix Messages

- Typedef struct { char *m7sb; } mess_7;
  #define m7_sb m_u.m_m7.m7sb;

- To add data in the message variable the following needs to be done:
  message m;
  m.m7_sb = data;

# Minix Messages

Γενικά:

message m;

m.m*x*_*yz*, *όπου:*

*x = 1..6, δηλώνει τη δομή που έχουμε επιλέξει*

*y = {i, p, l, f, ca, c}, δηλώνει τον τύπο δεδομένων του συγκεκριμένου μέλους*

i : int

p: pointer

l : long

f: function

ca: character array

c: char

*z : αύξων αριθμός του τύπου y στο συγκεκριμένο είδος μηνύματος*

# Get Minix

- Linux
  - Copy the minix archive from ~hy345/minix/minix204.tar.gz to your directory
  - tar zxvf minix204.tar.gz
- Windows
  - Copy the minix archive from ~hy345/minix/minix204_win32.zip to your directory
  - uncompress

# Run minix over QEMU

- Linux
  - type:  ./startminix.sh
- Windows
  - execute: qemu-win.bat

- emulator will launch in a new window
- hit '=' to boot minix without network support
- log in as root (no password required)

# CS-345 Assignment 3

- http://www.csd.uoc.gr/~hy345/docs/assignments.htm
  - How to run minix (over QEMU or bochs)
  - How to re-compile the kernel

# CS-345 Assignment 3

- Create 2 new system calls
- getpnr()
  - Dexetai san parametro ena process ID (PID) kai 8a epistrefei to process number pou einai apo8hkeumeno ston process table tou kernel
- prproct()
  - 8a emfanizei ta kathleimena slots tou process table.
- Write a simple program that uses the 2 system calls

# CS-345 Assignment 3

- Paradeigma e3odou:

```
PID   P_NR    P_NAME
0      -10      TTY
0      -9       DP8390
0      -8       SYN_AL
0      -7       IDLE
...

29      4        sh
18      5        update
30      6        getty
30      7        getty
30      8        getty
33      9        project5
```

Process number of PID #33 is 9 Process number of PID #29 is 4

# Info / Tips

- vi like editor:  elvis
- pico like editor:  elle

- Search for file:  find . -name idle.c
- Use grep
- Update Makefiles.
- MUST check source code of similar calls (HINT: check getset.c code)

# Tip..Baby steps…

i. υλοποίηση ενός "dummy" system call που εκτυπώνει απλά ένα μήνυμα

ii. επέκταση του (i) έτσι ώστε να εκτυπώνονται τα ορίσματα που περιέχονται στο "μήνυμα έκδοσης" του system call

……

# Sources

- *web.syr.edu/~ondsouza/SunOS-**minix.ppt***
- ***Wikipedia***