Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Languages, Levels and Virtual Machines

Γ. Γαροφαλάκης, Σ. Σιούτας

Languages, Levels and Virtual Machines

Language translation

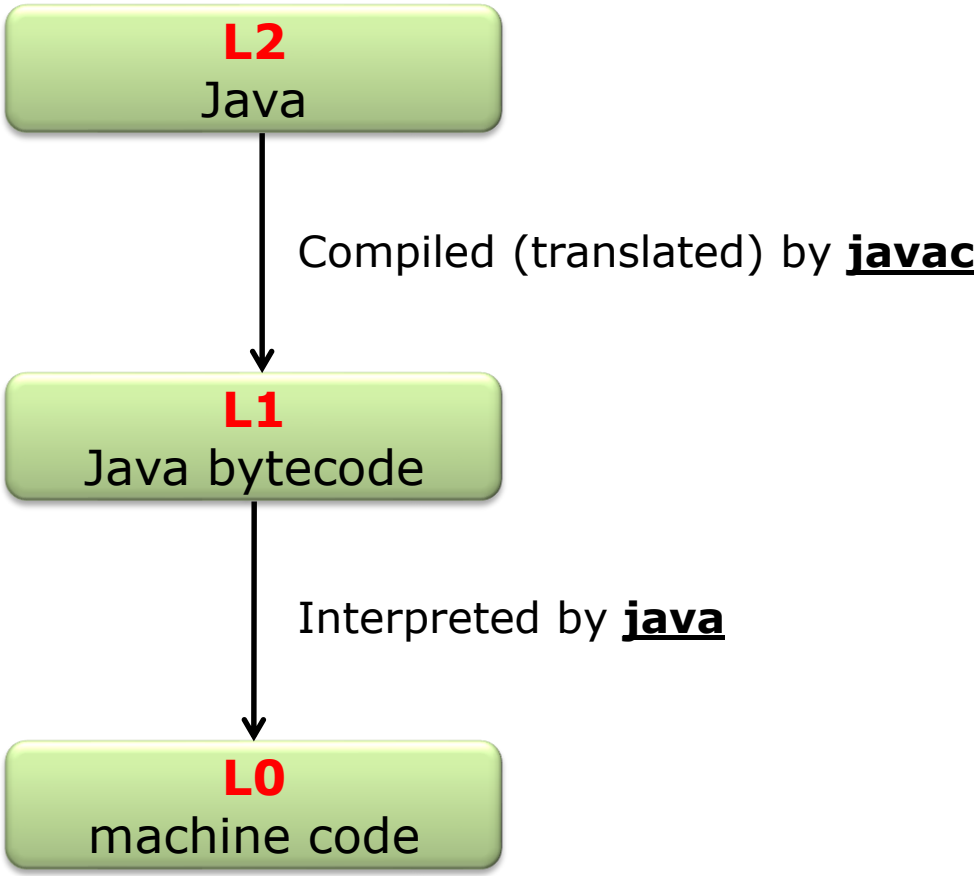
- Τι συμβαίνει όταν τρέχεις αυτό;

```
javac Foo.java
```

- Το πρόγραμμα **Foo.java** (γραμμένο σε Java) μεταγλωττίζεται (**compiled** ή **translated**) στο πρόγραμμα **Foo.class** (που είναι πρόγραμμα σε Java Bytecode)
  - Πρόκειται για δύο διαφορετικές γλώσσες
    - **Java:** υψηλότερου επιπέδου (L1) → καλή για να λύνεις προβλήματα
    - **Java Bytecode** χαμηλότερου επιπέδου (L0) → πιο λιτή, απλή, με λιγότερα abstractions, πιο κοντά στην λογική των transistors.
  - Κάθε εντολή L1 αντικαθίσταται από μία ή περισσότερες εντολές L0
  - Το πρόγραμμα τελικά εκτελείται στη γλώσσα L0

- Μια γλώσσα μπορεί επίσης να διερμηνευτεί (**interpreted**)
  - Διαβάζεις μία-μία τις εντολές της L1
  - Για κάθε μία εντολή L1 εκτελείς τις αντίστοιχες εντολές σε L0
  - Αυτό κάνει άλλωστε και το JVM (Java Virtual Machine), για την εκτέλεση Java Bytecode

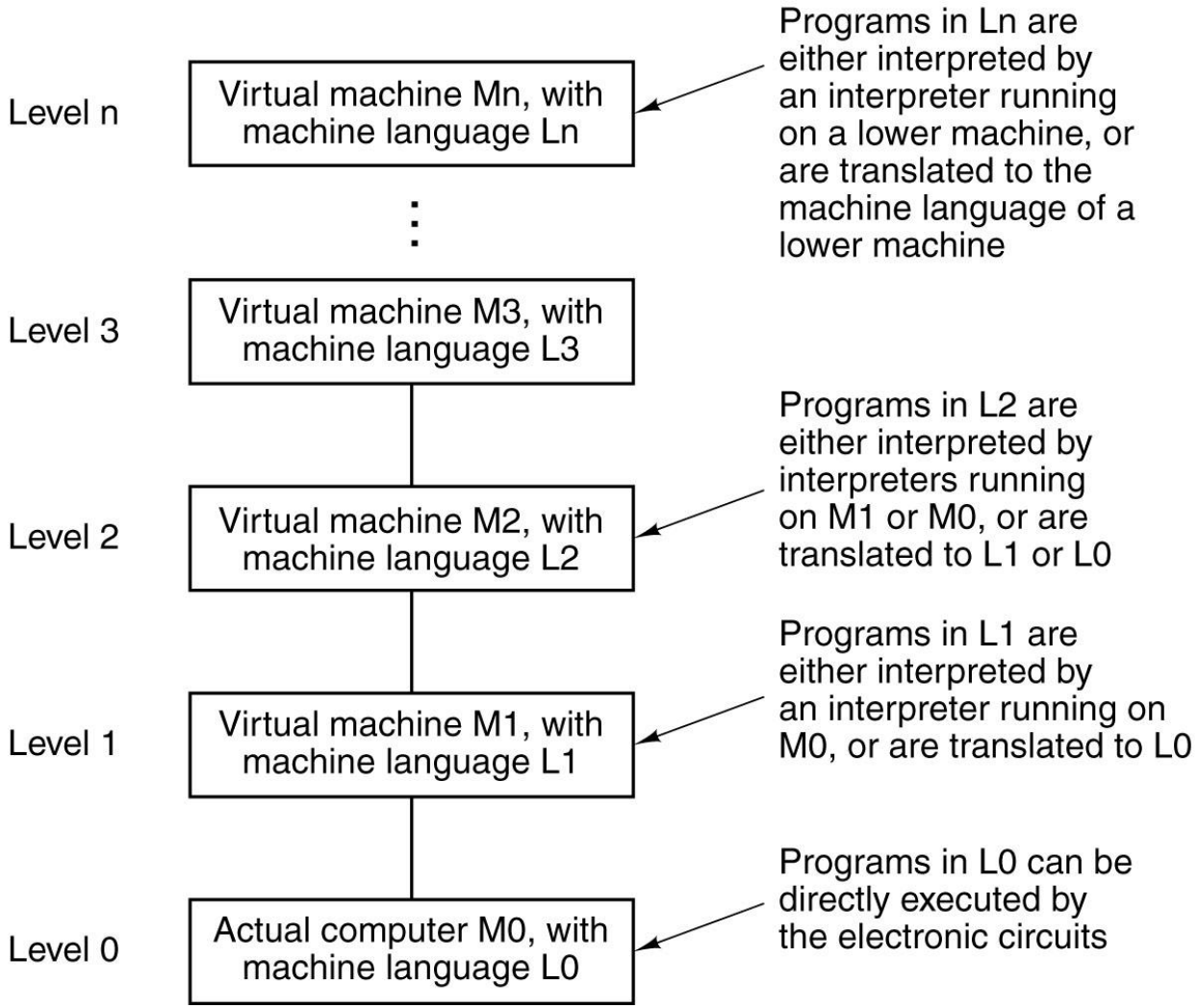
Compiling & Interpreting



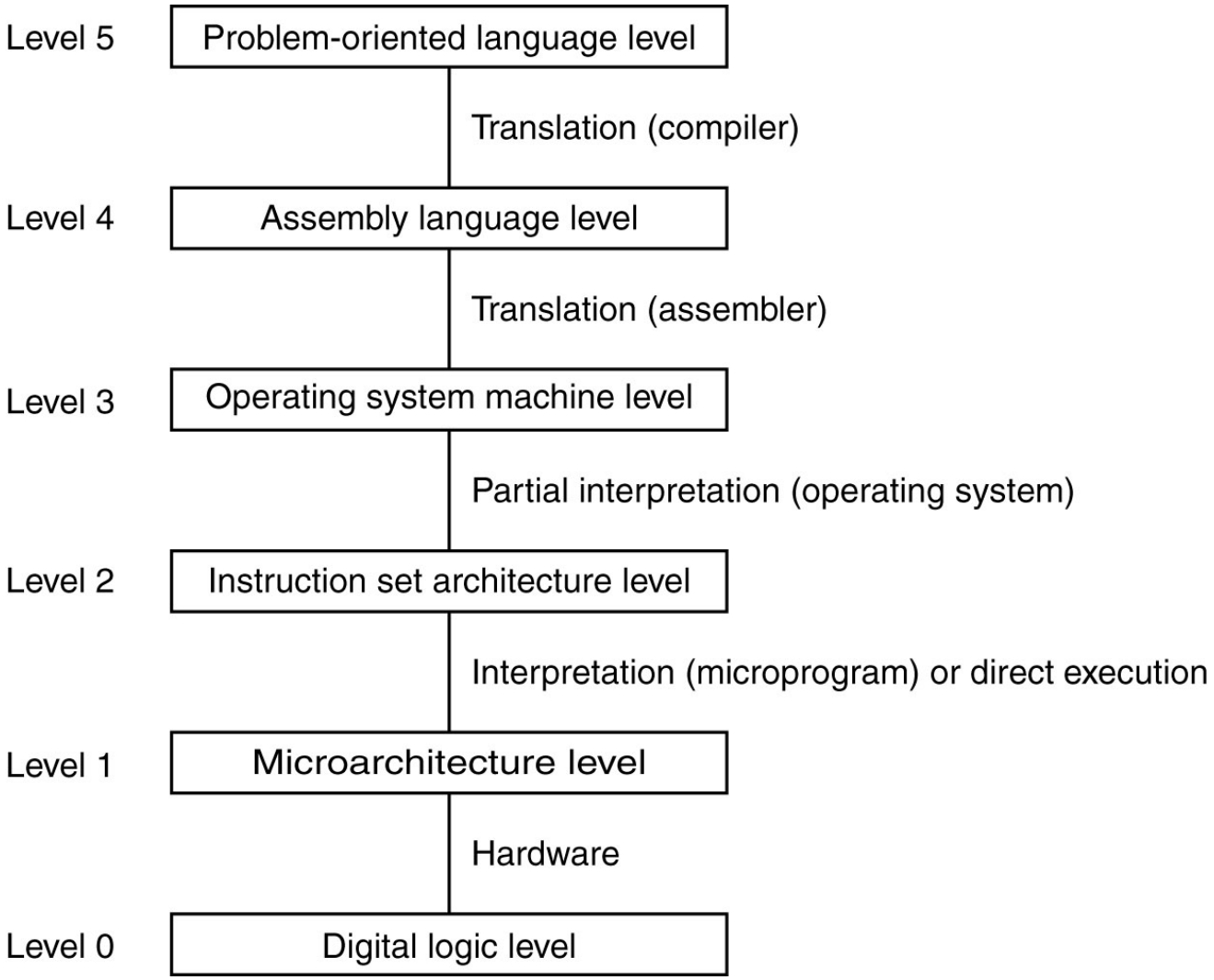
Virtual Machines

- Σκεφτείτε έναν υπολογιστή που εκτελεί προγράμματα σε γλώσσα L1
- Μπορεί να είναι κατασκευασμένος σε **hardware**
  - Αν η L1 είναι υψηλού επιπέδου → πολύ περίπλοκος ο σχεδιασμός του
  - Όποτε η L1 αλλάζει → Πρέπει να ξανασχεδιάζουμε το μηχάνημα από την αρχή
  - Όλα τα προγράμματα πρέπει να γράφονται σε L1
- Εναλλακτικά, θα μπορούσαμε να υλοποιήσουμε σε **software** ένα “virtual machine” που εκτελεί προγράμματα σε L1
  - Το L1 virtual machine τρέχει στο μηχάνημα L0
    - και το L0 μπορεί επίσης να είναι virtual!
  - Όταν αλλάζει η L1 → αλλάζουμε μόνο το virtual machine software
  - Για να υποστηρίξουμε τη γλώσσα L2, φτιάχνουμε απλά ένα νέο VM
    - Σε L0 ή ακόμα και σε L1
- Οι σημερινοί υπολογιστές έχουν πολλά επίπεδα “virtual machines”
  - Συνήθως περίπου 6

Layered view of computer organization



Layered view of computer organization



Από τη Java στα Transistors

Running a Java program

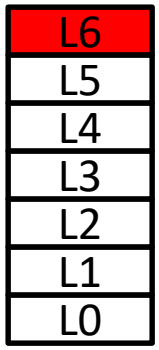
```
public class Foo
{
  public static void main(String[] args)
  {
    int i = 18;
    int j = 21;
    int k = i + j;
    System.out.println("18+21=" + k);
  }
}
```

```
$ javac Foo
$
```

```
$ java Foo
18+21=39
$
```

Magic!!!

L6: Problem-oriented language

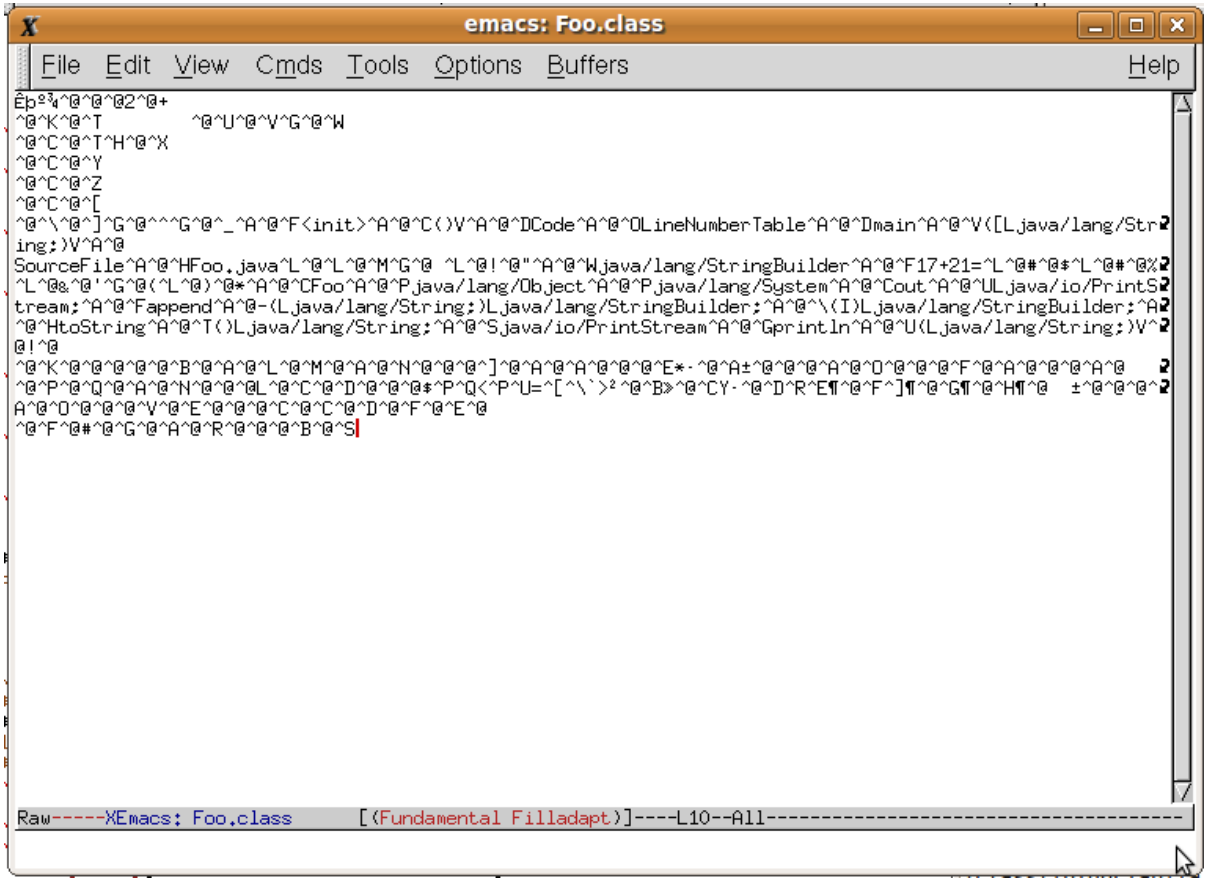


- Java

```
public class Foo
{
  public static void main(String[] args)
  {
    int i = 18;
    int j = 21;
    int k = i + j;
    System.out.println("18+21=" + k);
  }
}
```

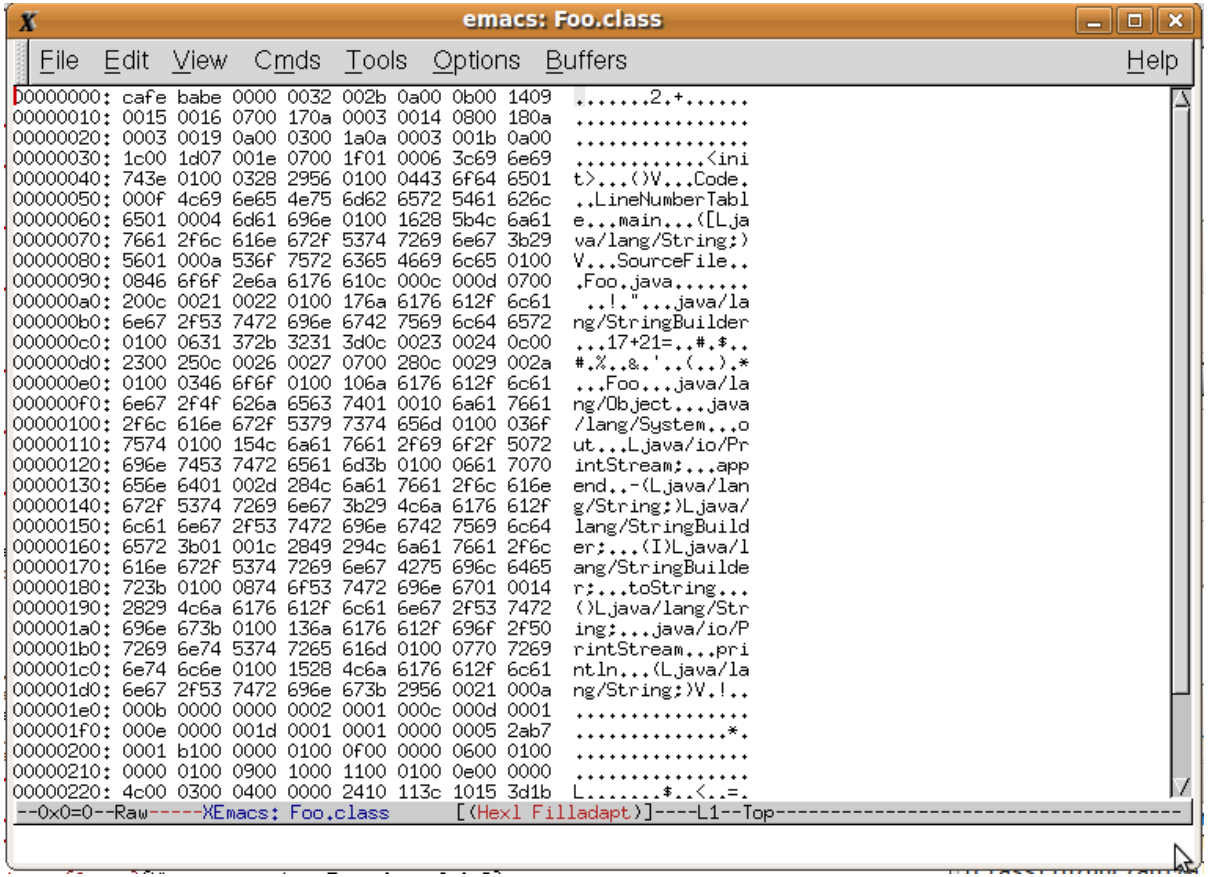
L5: Java Bytecode

- Open the Java Bytecode (class file) in a text editor:



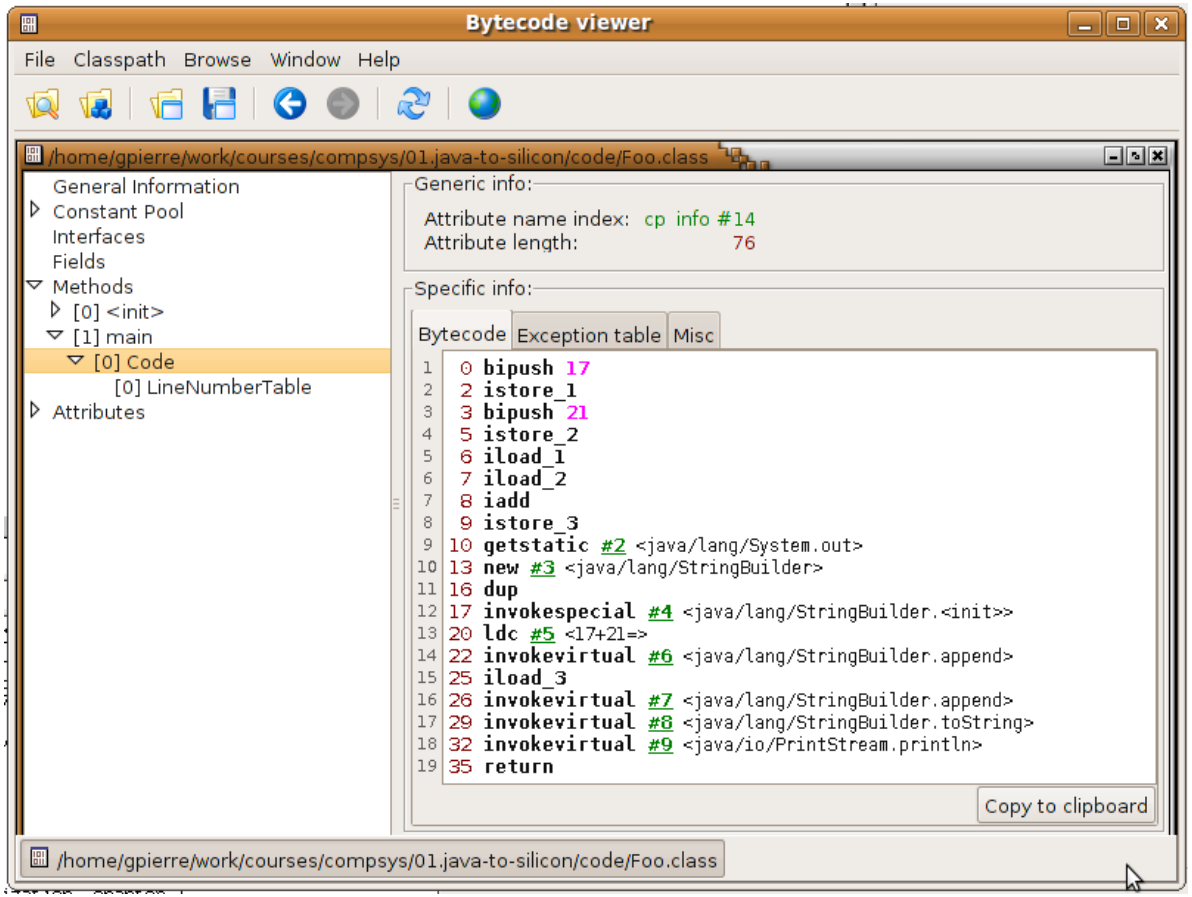
L5: Java Bytecode

- Open it in binary mode:



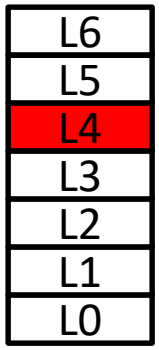
L5: Java Bytecode

- Use a Java Bytecode editor:



http://www.ej-technologies.com/products/jclasslib/overview.html

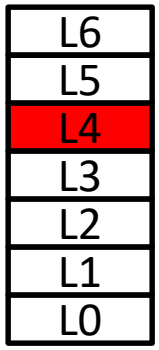
L4: Assembly Language



- Ποια η σχέση **machine code** και **assembly**;;;
  - Αν και είναι πολύ κοντά σε machine code, η assembly εξακολουθεί να είναι μια γλώσσα κατανοητή σε ανθρώπους (σχετικά ☺)
  - Ένα πρόγραμμα assembly πρέπει να γίνει compiled σε machine code από έναν assembler για να τρέξει

- Το JVM κάνει interpretation του προγράμματος “Foo.class” από **Java Bytecode** σε **machine code**
  - Το JVM είναι γραμμένο σε machine code (εκτελέσιμο)
  - Χαμηλότερου επιπέδου από το Java Bytecode
  - Πολύ χαμηλότερου επιπέδου απο την ίδια τη Java!

L4: Assembly Language



spyros@chaos: ~

File Edit View Search Terminal Help

```
#include <stdio.h>
int main()
{
  int i = 17;
  int j = 21;
  int k = i+j;
  printf("i+j=%d\n", k);
  return 0;
}
                    1,1           Top
```

Note -- You can get the Assembly language of a C program by:

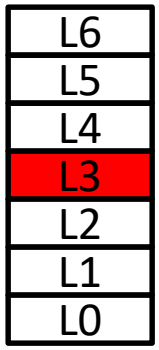
```
$ gcc -S foo.c
$
```

spyros@chaos: ~

File Edit View Search Terminal Help

```
        .file   "foo.c"
        .section .rodata
.LC0:
        .string "i+j=%d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $17, -12(%rbp)
        movl    $21, -8(%rbp)
        movl    -8(%rbp), %eax
        movl    -12(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    $.LC0, %eax
        movl    -4(%rbp), %edx
        movl    %edx, %esi
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
        .section .note.GNU-stack,"",@progbits
                                              1,2-11        All
```

L3: Operating System

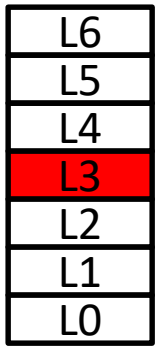


- "user" and "system" both take large percentages of CPU time

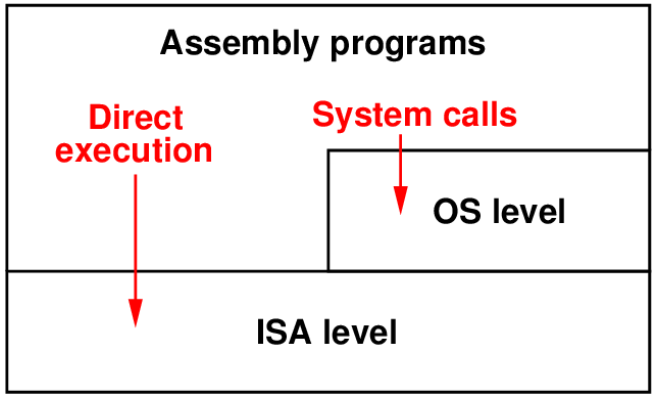
```
spyros@chaos: ~
File  Edit  View  Search  Terminal  Help
top - 19:58:41 up  8:20,  7 users,  load average: 1.49, 1.49, 1.22
Tasks: 173 total,   1 running, 172 sleeping,   0 stopped,   0 zombie
%Cpu(s): 57.8 us, 42.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   5978908 total,  3783148 used,   195760 free,   281732 buffers
KiB Swap:  4124668 total,     1372 used,  4123296 free,   959780 cached

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+  COMMAND
 9903 spyros    20   0 2018m   15m 8996 S 196.5  0.4   0:40.13 java
 2589 spyros    20   0 1824m 259m   35m S   1.3  6.7  10:34.32 gnome-shell
 1599 root      20   0  196m   34m   17m S   0.3  0.9   1:18.44 Xorg
 2889 spyros    20   0  749m   29m   13m S   0.3  0.8   0:05.63 gnome-terminal
 2959 spyros    20   0  123m 2412 1736 S   0.3  0.1   0:26.14 synergyc
 9545 spyros    20   0 24792 1604 1148 S   0.3  0.0   0:00.85 top
 9866 spyros    20   0 24800 1608 1148 R   0.3  0.0   0:00.25 top
    1 root      20   0 24604 2592 1384 S   0.0  0.1   0:00.51 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.01 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:03.26 ksoftirqd/0
    6 root      rt   0     0    0    0 S   0.0  0.0   0:00.02 migration/0
    7 root      rt   0     0    0    0 S   0.0  0.0   0:00.14 watchdog/0
    8 root      rt   0     0    0    0 S   0.0  0.0   0:00.11 migration/1
   10 root      20   0     0    0    0 S   0.0  0.0   0:01.86 ksoftirqd/1
   11 root      rt   0     0    0    0 S   0.0  0.0   0:00.16 watchdog/1
   12 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 cpuset
   13 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 khelper
```

L3: Operating System



- Τα εκτελέσιμα προγράμματα «πατάνε» σε δύο χαμηλότερα επίπεδα:
  - Στο Λειτουργικό Σύστημα
  - Και στο επίπεδο ISA (Instruction Set Architecture)



L3: Operating System

Note – You can see which system calls a program makes by:

```
$ strace <prog>
```

spyros@chaos: ~/tmp

File Edit View Search Terminal Help

```
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

6,2 Top

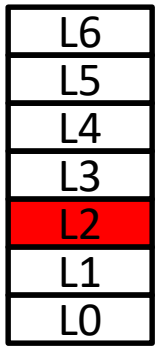
spyros@chaos: ~/tmp

File Edit View Search Terminal Help

```
execve("./foo", ["./foo"], [/* 52 vars */]) = 0
brk(0)                                  = 0x112d000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a5e000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=104790, ...}) = 0
mmap(NULL, 104790, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd830a44000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\30\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1811160, ...}) = 0
mmap(NULL, 3925240, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd83047f000
mprotect(0x7fd830634000, 2093056, PROT_NONE) = 0
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a43000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a42000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a41000
arch_prctl(ARCH_SET_FS, 0x7fd830a42700) = 0
mprotect(0x7fd830833000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7fd830a60000, 4096, PROT_READ) = 0
munmap(0x7fd830a44000, 104790)          = 0
fstat(1, {st_mode=S_IFREG|0664, st_size=1777, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd830a5d000
write(1, "Hello, World!", 13Hello, World!)          = 13
exit_group(0)                           = ?
```

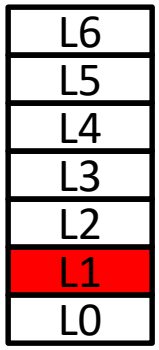
28,24 All

L2: Instruction Set Architecture



- Ο κάθε επεξεργαστής έχει ένα σετ εντολών μηχανής (machine code), που λέγεται Instruction Set Architecture.
  - Π.χ., x86, SPARC, ARM, ...
- Οι compilers μεταγλωτίζουν ένα πρόγραμμα υψηλότερου επιπέδου σε machine code του εκάστοτε επεξεργαστή

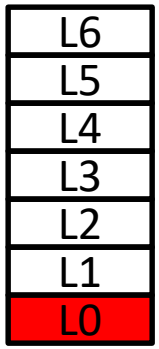
L1: Microarchitecture



- Πως «ξέρει» ένας επεξεργαστής να εκτελέσει μια εντολή;
  - Οι εντολές μηχανής είναι υλοποιημένες σε ένα χαμηλότερο επίπεδο, αυτό του μικροπρογράμματος (**micro-program**)
  - Το micro-program επίπεδο ουσιαστικά υλοποιεί ένα **virtual machine** που επιτρέπει την εκτέλεση machine code (ISA level)

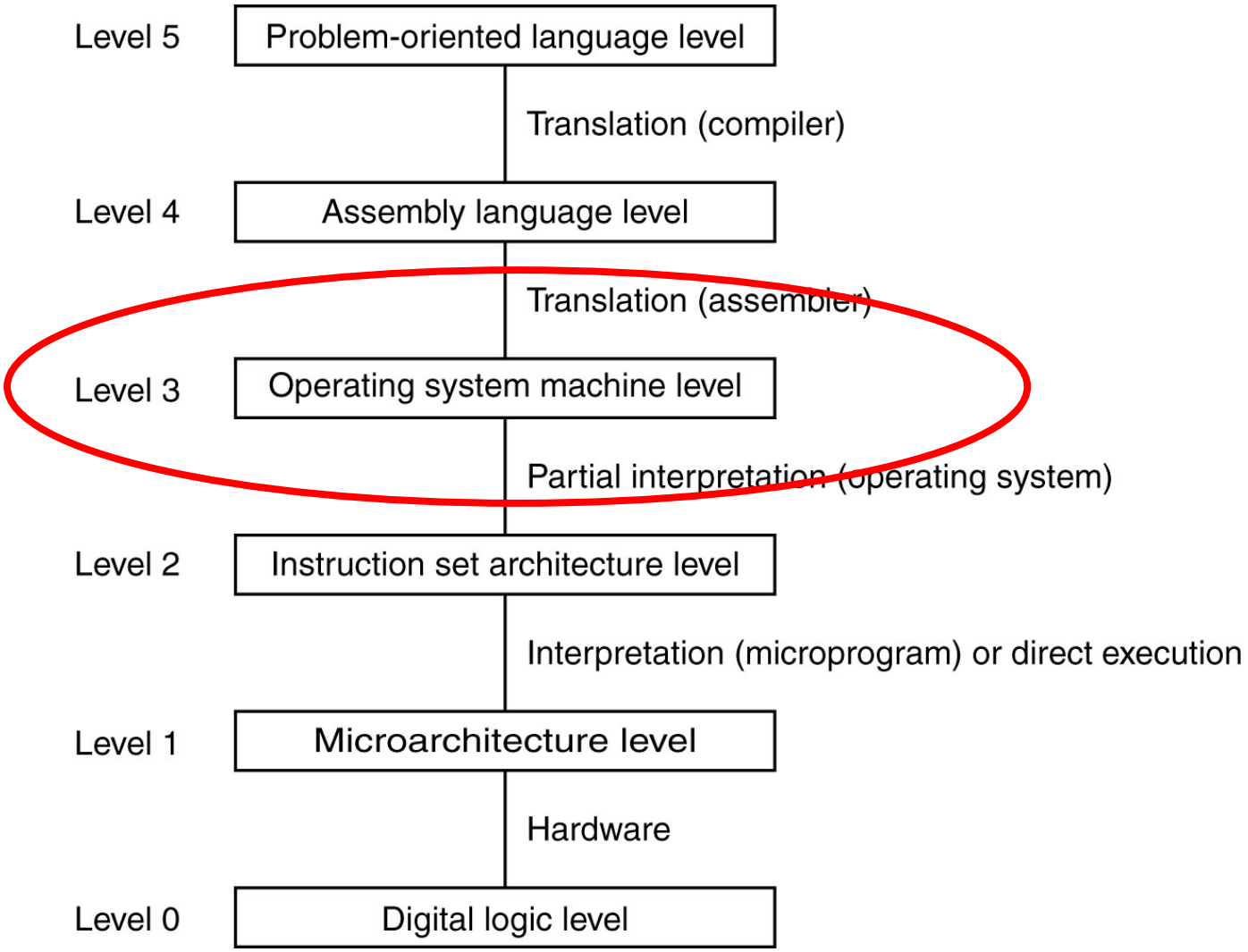
- Γιατί να μην υλοποιήσουμε τις εντολές machine code απευθείας σε hardware;
  - Γιατί είναι απλούστερο να έχουμε ένα ακόμα επίπεδο!
  - Επίσης βοηθάει στην υλοποίηση CPU pipelinining
  - Αλλά γι' αυτά παρακολουθήστε ένα μάθημα αρχιτεκτονικής υπολογιστών ☺

L0: Digital gates



- Το hardware κάνει όλο το **hard work**!
  - Store data in memory, read/write memory, copy data
  - Arithmetic operations (add, subtract, multiply, …)
  - Issue I/O operations
- Αυτά εκτελούνται με ψηφιακές πύλες (**digital gates**)
  - Digital gates manipulate only **boolean values** (either 0 or 1)
  - a **AND** b
  - a **OR** b
  - **NOT** a
- Οι ψηφιακές πύλες υλοποιούνται με **transistors**
- Transistors are magic! ☺
  - Ρωτήστε τον Ηλεκτρολόγο Μηχανικό της γειτονιάς σας για το πως δουλεύουν...!

Operating System



Λειτουργικό Σύστημα

- Το ΛΣ είναι ουσιαστικά ένας **διαχειριστής πόρων του συστήματος**!
  - Ίσως ο πιο «ασφαλής» ορισμός
  - Εξασφαλίζει ότι πολλά προγράμματα (διεργασίες) μπορούν να τρέχουν ταυτόχρονα
    - χωρίς να παρεμβαίνει το ένα στο άλλο
    - έχοντας το καθένα την «ψευδαίσθηση» πλήρους κυριότητας του μηχανήματος
  - Διαχειρίζεται τις περιφερειακές συσκευές
    - Επιτρέποντας τη χρήση τους στις διεργασίες που τις χρειάζονται
    - Παρέχοντας ένα ομοιόμορφο interface για την προσπέλασή τους

Processes (Διεργασίες – Προγράμματα - Υποπρογράμματα)

- Ένα process είναι το πλαίσιο στο οποίο εκτελείται ένα πρόγραμμα
  - Διαθέτει ιδιωτικό memory address space (ξεχωριστό από άλλες διεργασίες)
  - Διαθέτει ιδιωτικό stack, PC & SP registers, κ.τ.λ.

- Το ΛΣ αναλαμβάνει το process scheduling (χρονοπρογραμματισμό διεργασιών)
  - Ανά πάσα στιγμή μόνο ένα process μπορεί να τρέχει σε ένα CPU core
  - Το ΛΣ φροντίζει να εναλλάσσονται πολλά processes στα διεθέσιμα cores
  - Δίνει την ψευδαίσθηση ότι όλα τρέχουν ταυτόχρονα, ακόμα κι όταν είναι περισσότερα από τον αριθμό των cores (το οποία πρακτικά συμβαίνει πάντα)
  - Φροντίζει να κατανέμει τους πόρους και τον χρόνο εκτέλεσης δίκαια

- Οι διεργασίες είναι isolated
  - Κάθε διεργασία τρέχει «μόνη της»
  - Έχει την ψευδαίσθηση ότι έχει όλο το μηχάνημα στην αποκλειστική διάθεσή της
  - Προαιρετικά, οι διεργασίες μπορούν να επικοινωνήσουν μεταξύ τους
  - Αυτό γίνεται μέσω τεχνικών Inter-Process Communication (IPC) που προσφέρονται επίσης από το ΛΣ

Input/Output

- Οι διεργασίες θέλουν να αλληλεπιδράσουν με διάφορες συσκευές
  - Οθόνη, πληκτρολόγιο, **δίσκο**, εκτυπωτές, δίκτυο, κάρτα ήχου, USB, κ.τ.λ.

- Κάθε συσκευή είναι διαφορετική!
  - Όχι μόνο οι διαφορετικού τύπου (π.χ., εκτυπωτής και δίσκος)
  - Αλλά και οι συσκευές ίδιου τύπου
    - Π.χ., εκατοντάδες μοντέλα εκτυπωτών
    - Π.χ., εκατοντάδες μοντέλα δίσκων
    - κ.τ.λ.
  - Διαφορετικοί κατασκευαστές
  - Διαφορετικές ταχύτητες

- Το ΛΣ προσφέρει
  - **Ομοιογενές interface** σε όλα τα είδη διαφορετικών συσκευών
  - **Διαιτησία** (**arbitration**) ανάμεσα στις διεργασίες που θέλουν να χρησιμοποιήσουν την ίδια συσκευή ταυτόχρονα

Memory Management

- Οι διεργασίες χρειάζονται μνήμη
  - Αλλά θέλουμε η κάθε διεργασία να είναι απομονωμένη από τις άλλες
  - Καλύτερα: Δώσε σε κάθε διεργασία την ψευδαίσθηση ότι έχει όλη τη μνήμη του υπολογιστή στη διάθεσή της
  - Ακόμα καλύτερα: Δώσε σε κάθε διεργασία την ψευδαίσθηση ότι έχει στη διάθεσή της όλον τον χώρο διευθύνσεων (address space), άσχετα με το πόσο λιγότερη μνήμη έχει ο υπολογιστής στην πραγματικότητα

- Για να πετύχει τα παραπάνω, το ΛΣ πρέπει να παρέχει:
  - Memory isolation
  - Paging (δηλ., να μεταφέρει κομμάτια της μνήμης στον δίσκο όταν δεν χωράνε όλα στη μνήμη)
  - Virtual Memory (δηλ., να δίνει στην κάθε διεργασία την ψευδαίσθηση πλήρους κυριότητας σε όλο το address space)