Introduction to Operating Systems

Operating System Design – MOSIG 1

Instructor: Noel De Palma, Guillaume Huard

Class Assistant: Benjamin Negreverne

Slides heavily inspired from Fabienne Boyer, Arnaud Legrand, David Mazieres

Outline of the lectures

1. Introduction to Operating System
2. Memory management
3. Processes and threads
4. Synchronization and communication
5. Deadlock
6. File system and secondary storage
7. Nachos OS

Practical informations

- **Class web page:** http://sardes.inrialpes.fr/~depalma/enseignement/
- **References**
  - Operating System Concepts (8th ed, by Silberschatz, Galvin, and Gagne), Modern operating systems (2nd ed, by Tanenbaum)
- **Staff email address:** noel.depalma@inrialpes.fr, Guillaume.Huart@inrialpes.fr, Benjamin.Negrevergne@imag.fr
  - Add [M1-OSD] to the subject of your emails (otherwise, we may not read them)
- **Key dates:**
  - Lectures: Tuesday & Wednesday 13:30–15:00, F111
  - Practical Sessions: Wednesday 15:15–18:00, F111

Course goals

- Introduce you to operating system concepts
    - Hard to use a computer without interacting with OS
    - Understanding the OS makes you a more effective programmer
- The first minutes of the lecture can be devoted to re-explain some parts of the previous lecture.
- Prepare you to take graduate OS classes (M1 Principles of Computer Networks, M2 Parallel Systems, Distributed systems, . . .)

Programming Assignments

- Among the different practical sessions, some of them will be graded
- Implement projects in groups of up to 3 people
  - Working code or no credit here
- 33% of grade from projects
  - For each project, 50% of score based on passing test cases
  - Remaining 50% based on design and style
- 33% of grade from mid-term exam
- 33% of grade from final exam

Why study operating systems?

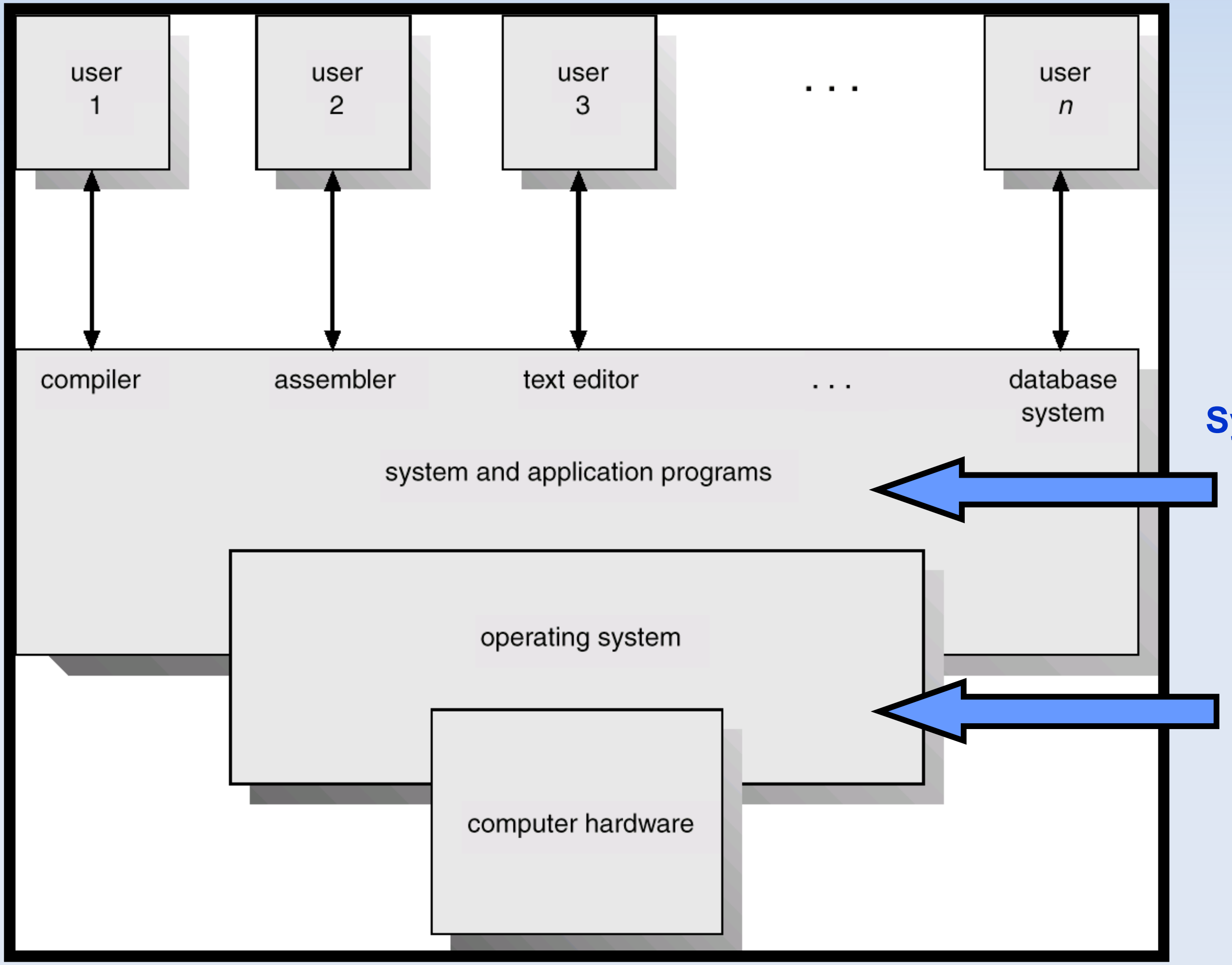
- Operating systems are a maturing field
  - New hardware or “smart” devices need new OSes
- Resource consumption is an OS issue
  - Many new metrics
  - Battery life, CUE, PUE ... => profile and optimize
- Security is an OS issue
  - Hard to achieve security without a solid foundation
- High-performance servers and web browsers are an OS issue
  - Face many of the same issues as Oses

What is an operating system (OS)

- Layer between applications and hardware
- Main goals
  - Provide abstraction of hardware through APIs
  - Manage efficient resources sharing
  - Manage fair resources sharing
  - Ensure resources protection and access control

→ OS can be seen as a first layer of virtualization over hardware

General positioning



System service middleware

OS lecture

A. Silberschatz, Calvin and Gagne, 2002

Resources Managed by an OS

- **Runtime abstraction for programs**
- Processes — *Tasks*
- Threads (Lightweight processes)
- Driver (I/O management)
- **Runtime abstraction for memory**
- Primary memory — *Data*
  – RAM ...
- Secondary memory
  – Files ...

Task management

- Multi-processes
  - Manage process lifecycle
  - Manage processor allocation
  - Manage process isolation
- Multi-users
  - Protect from bad users

Data management

- Different level of abstraction (physical/logical)
- Primary memory = A byte array
  - Physical/virtual
- Secondary memory = permanent storage
  - Files : an abstracted unit of storage and structure
  - Block : a physical unit of storage (e.g disk block)
- *Operating system*
  - *Manage the primary memory allocation to processes*
    - *Manage the mapping between different memory abstractions*
  - *Manage the secondary memory*
    - *files creation/destruction/access*
    - *Manage the mapping of file to lower level abstraction*
  - *Manage access control*

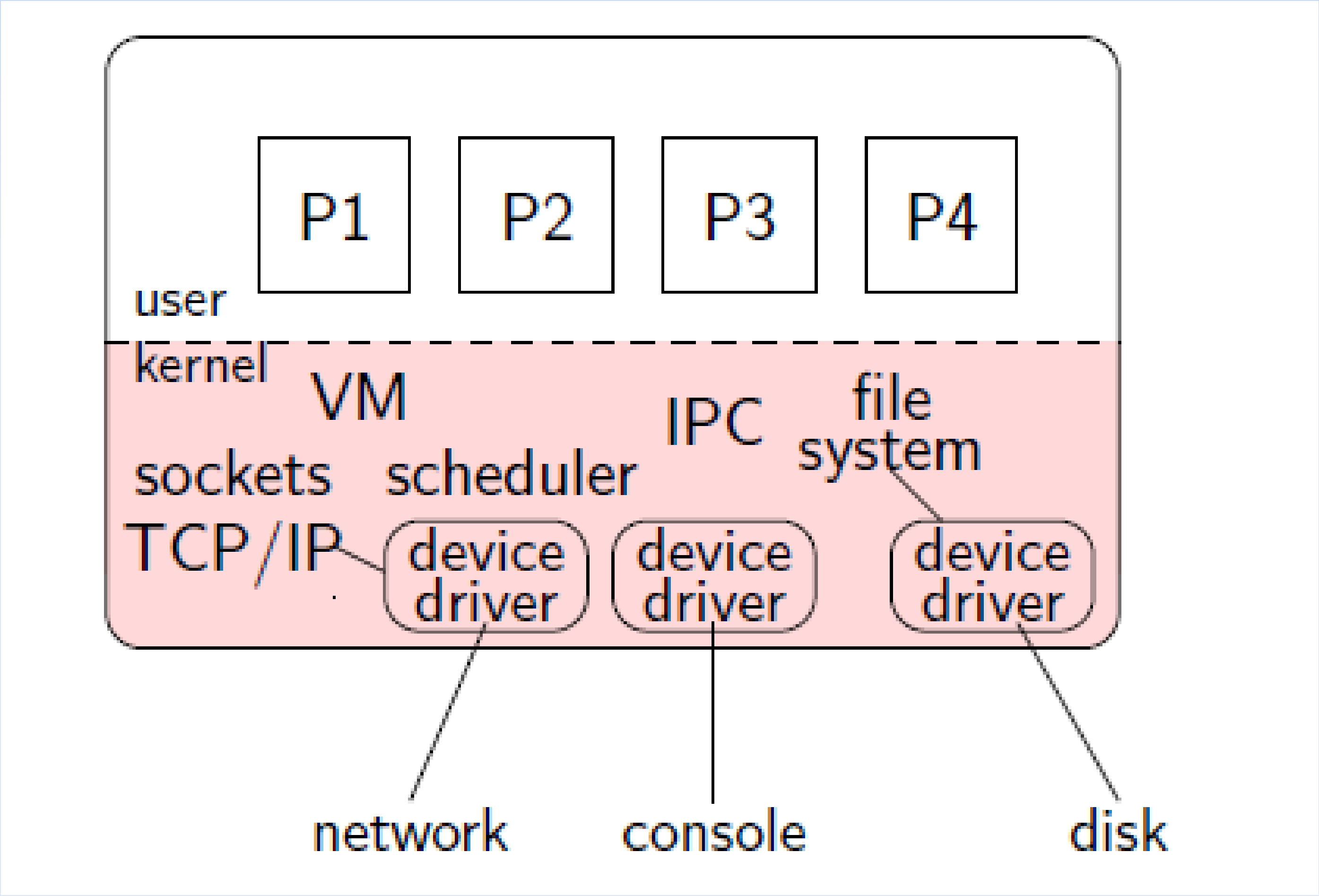
OS Structure

- Kernel
  - Always in central memory
    - Run in supervisor mode
    - Maintains data structure for users and application
- System services
  - Part of the system that can be swap in/out from memory if necessary
- Drivers
  - Low level hardware management
  - IT-based programming

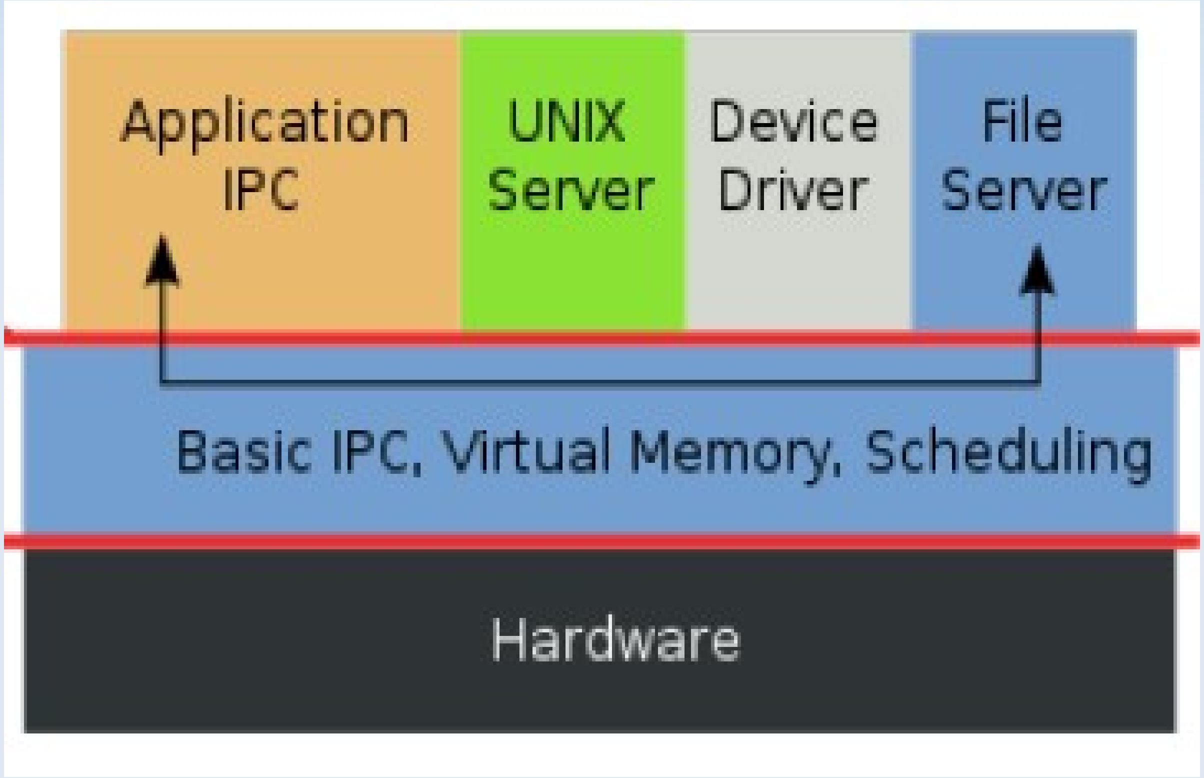
OS Structure

- Minimal kernel (micro-kernel / client-server)
  - Mach / Chorus / L4
  - Maximize the OS functions implemented outside the kernel
  - Better extensibility et adaptability
  - Better faillure isolation (separate processes)
  - … But comes with overheads
- Monotlihic Kernel
  - Unix, Linux, Windows XP
  - Better performances
  - The OS is a set of functions. Direct call induces less IPC (Inter Processus Call)

OS Structure

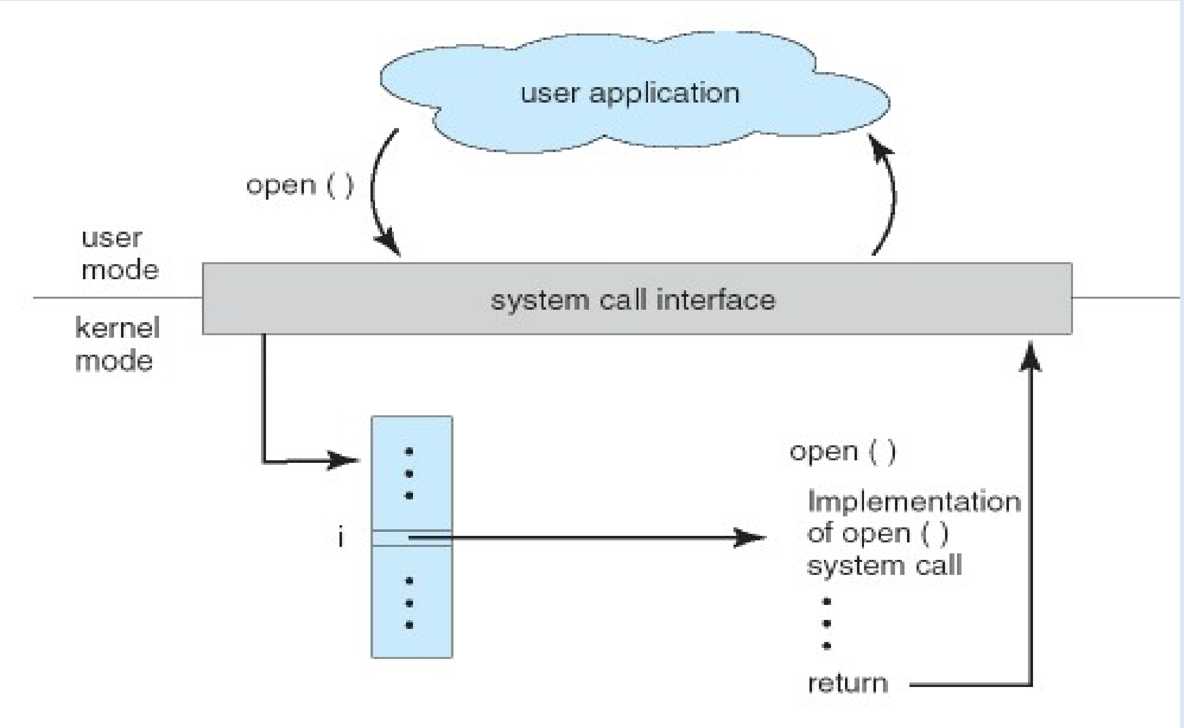


Micro kernel



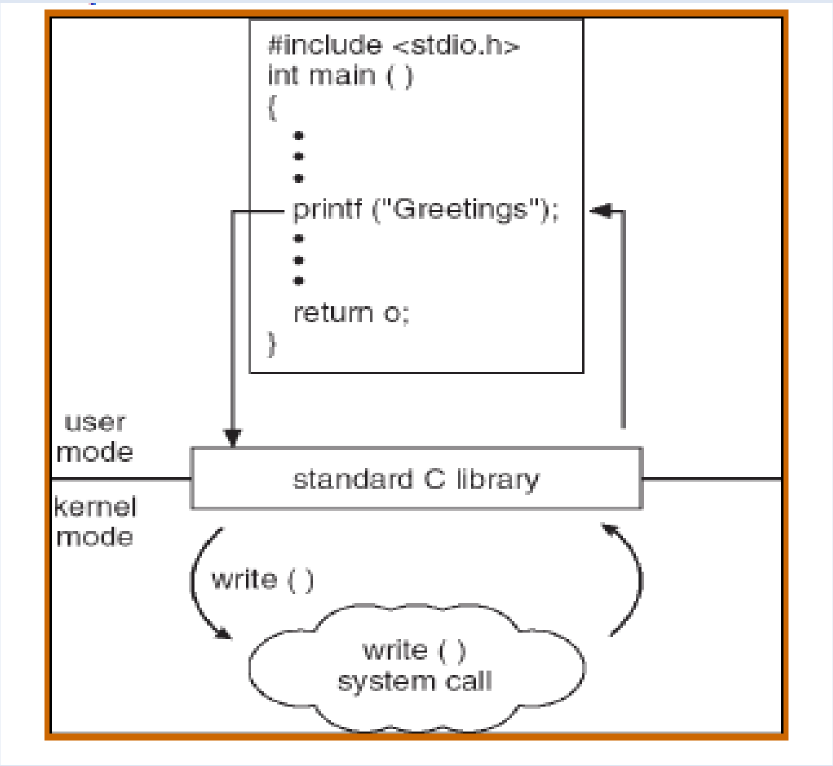
OS API—system call

- **Applications can invoke kernel through system calls**
  - **Special instruction transfers control to kernel**
  - **. . . which dispatches to one of few hundred syscall handlers**
- **Goal: Do things app. can't do in unprivileged mode**
  - **Like a library call, but into more privileged kernel code**



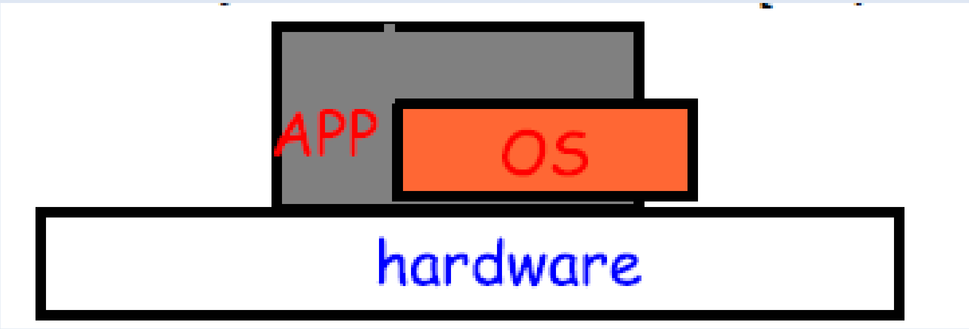
System call example

- Standard library implemented in terms of syscalls
  - printf – in libc, has same privileges as application
  - calls write – in kernel, which can send bits to output

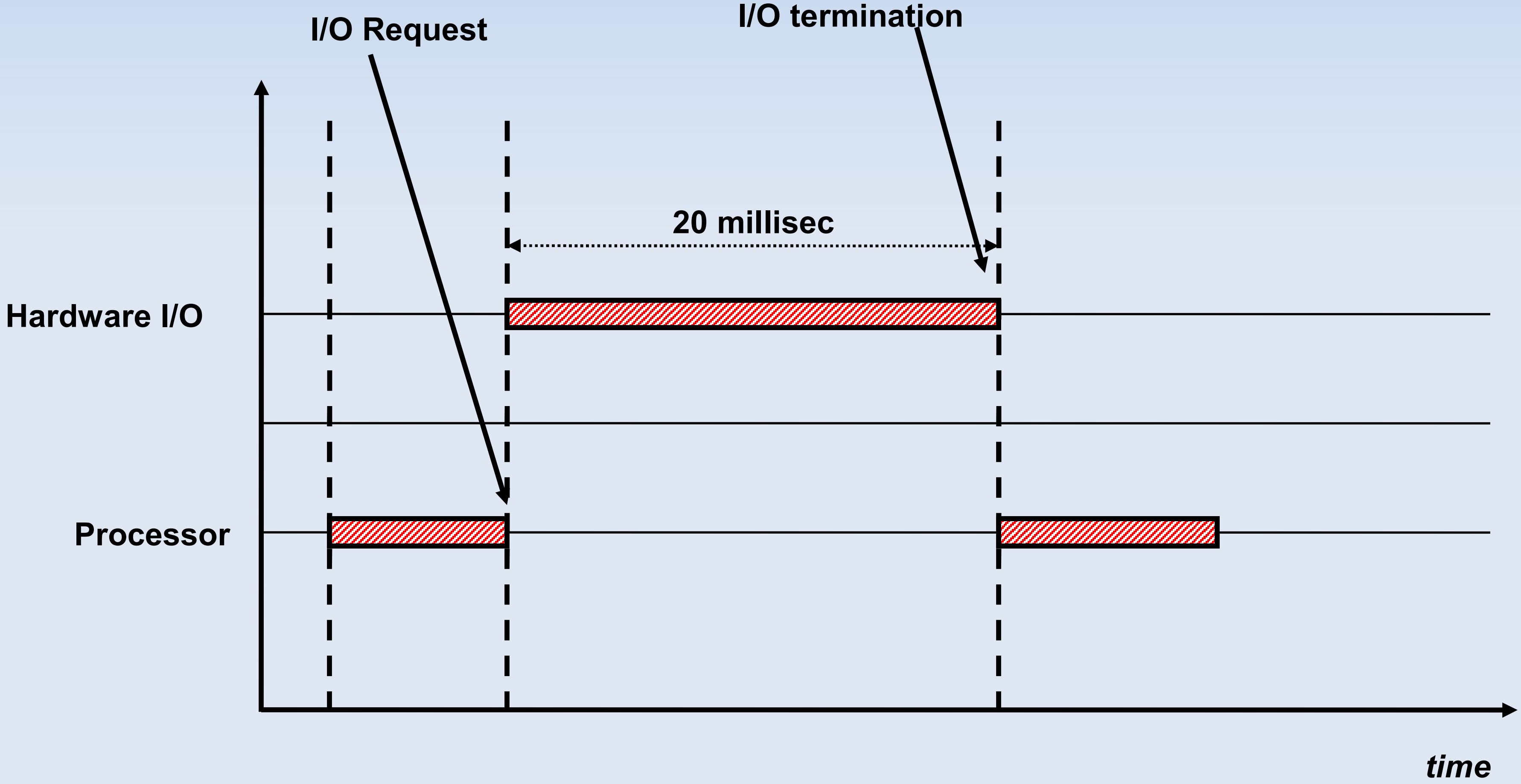


Primitive Operating systems: Monoprogramming (1950)

- Just a library of standard services [no protection]
  - Standard interface above hardware-specific drivers, etc.
- Simplifying assumptions : Monoprogramming
  - System runs one program at a time
  - No bad users or programs (often bad assumption)
- Problem: Poor utilization
  - . . . of hardware (e.g., CPU idle while waiting for disk)
  - . . . of human user (must wait for each program to finish)

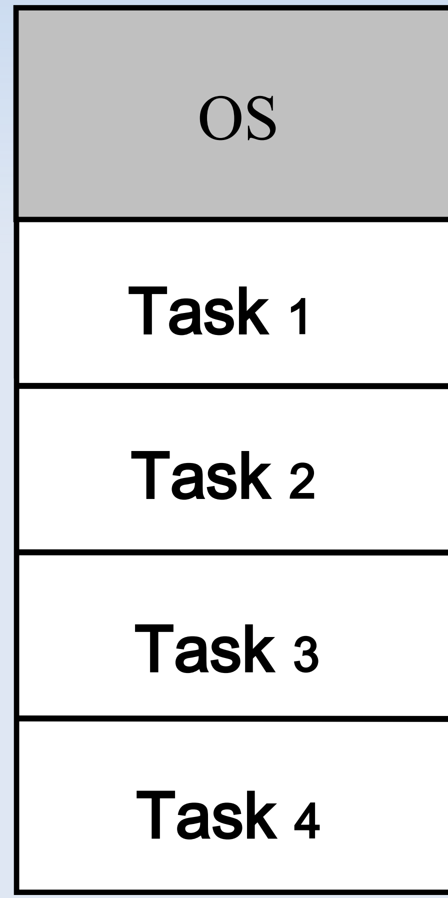


Mono-programming

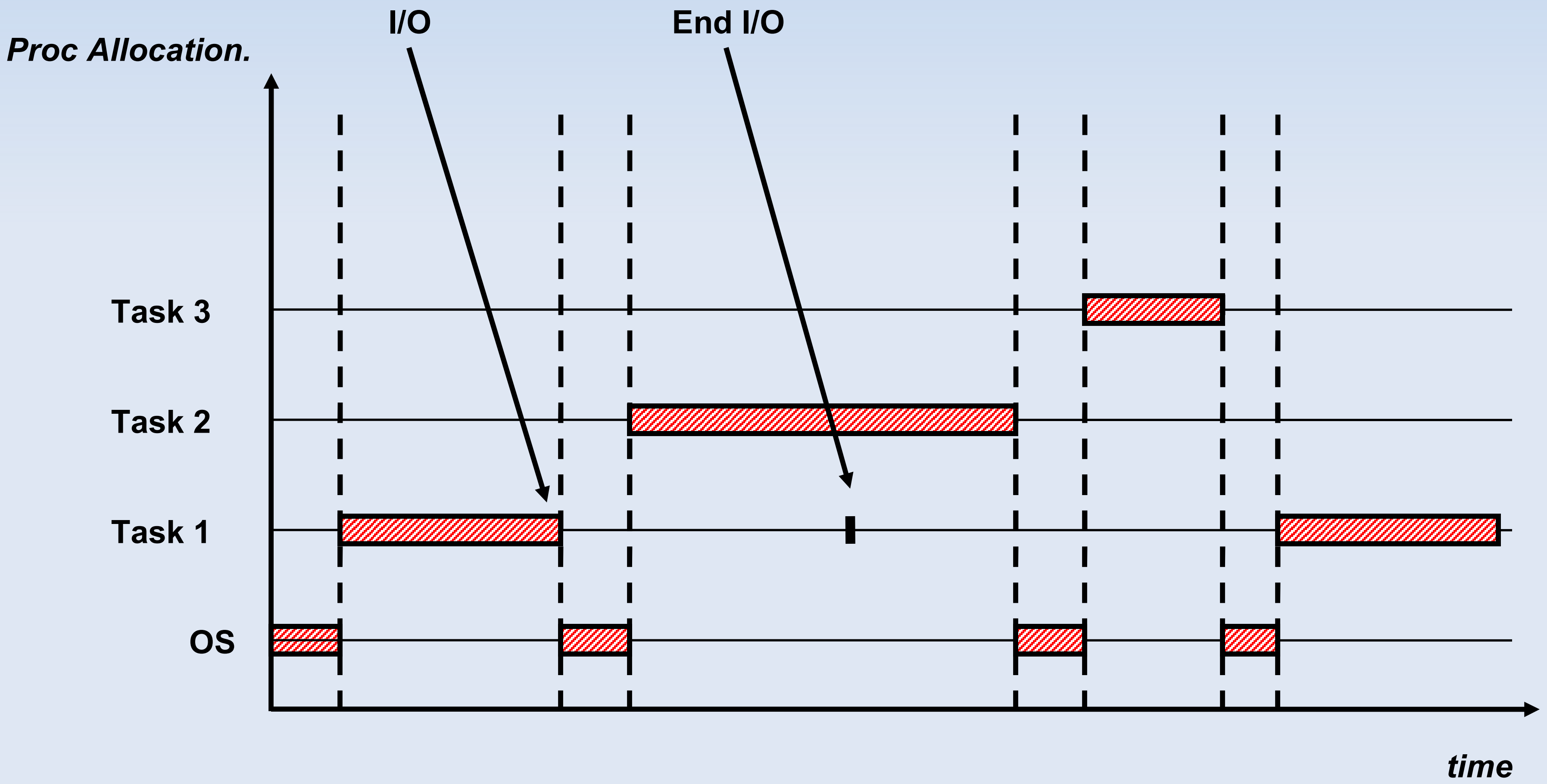


Multi-programming (1960/1970)

- Multiple tasks in memory at the same time
- Run more than one process at once
  - Need a basic scheduler
  - When one process blocks on I/O run another process
- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
- Advantages
  - Better CPU utilization
- Disadvantage
  - Still not very efficient
  - Need Protection



Multi-programming



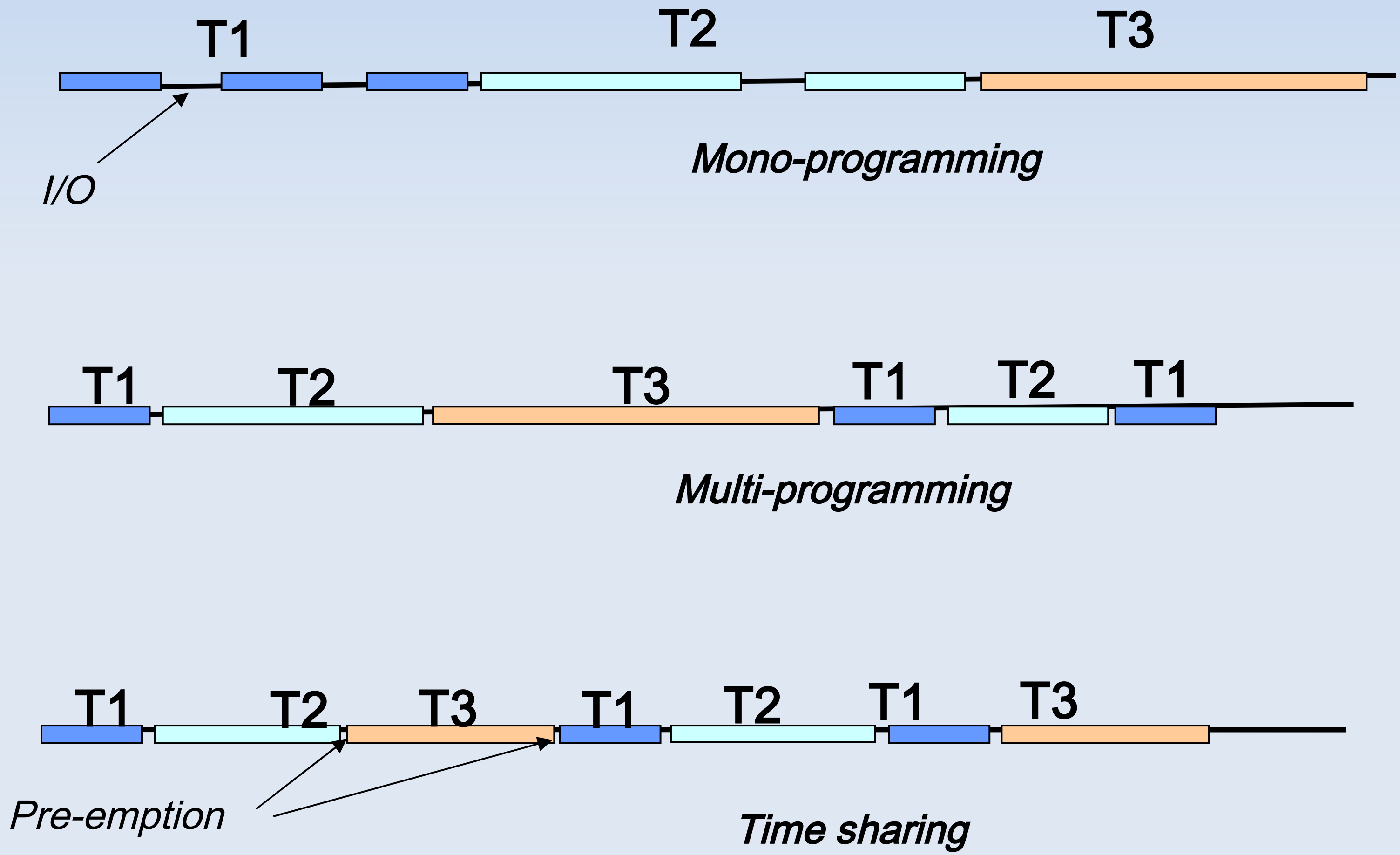
Time sharing (1970)

- Run more than one process at once
  - The cpu is shared between processes
  - Time slices
  - CPU pre-emption (on I/O or end of the time slice) and context switch
- Processes can be in memory or swapped on disk
  - Total memory usage greater than in machine (must virtualize the memory)
  - Improve the number of managed processes
- Better resource management and better mean response time

Time sharing (1970)

- Issues
  - Fair CPU sharing (Need policy)
  - Total memory usage greater than in machine (must virtualize)
  - Super-linear slowdown with increasing demand (thrashing)
  - Protect process’s memory from one another (Memory isolation)
  - Protect users (access control)

Mono/multi programming/time sharing



Protection

- A task must not read/write in the memory zone of another task
- A task must not impact the kernel memory excepts using SVC
- A task must not read/write I/O data of another task

➔ Need isolation (memory …)

Evolutions from the 70's

- Hardware evolutions
  - Personal computer and laptop
  - Specialized architecture: Real time, embeded, mobile device
  - Multi-processors
  - Virtualization
- Networking evolutions
  - Ethernet (30 Gb swiched network ...), Internet (broadband ...)
- Distributed systems
  - Cluster / Grid / Cloud
- Many new criteria to optimize
  - Consistency, Performances, Availability, Security, Energy consumption
- Many levels to optimize

Real time system

- *Time constraint*
  - *Bounded execution time*

- *Hard* real-time systems
  - Strong SLA guarantee
  - Few or no secondary memory
  - No or Short context switch
  - Specific OS (Plane, robotics ...)
- *Soft* real-time systems
  - Used for multimedia or virtual reality
  - Soft time constraint—no SLA guaratee
  - Task priority
  - Specific memory management

Mobile systems

- Phone, Personal Digital Assistants (PDAs)
- Specific OS (e.g android, windows CE)
- More and more powerfull (Cpu, memory ...)
- Constraints
  - Subject to disconnection
  - Small screen
  - Energy consumption

Parallel multiprocessor system (1/2)

- SMP (Symetric Multi Processeurs)
  - Classical OS with multi-processor support (DB, Web, NFS, …)
  - Standard Processors
  - Full memory sharing
- Parallel system machine
  - Specialized Architectures
    - Specific processors for vectorial operations
    - Specialized network
    - Full or partial memory sharing

Clustered systems

- Multiple nodes (hundred and more)
  - Homogeneous
  - Shared disk or share nothing
- Fast network interconnection (SCI, Ethernet, …)
  - LAN
- 2 characteristics :
  - Scalability through partitionning or load balancing
  - High availability through master/slave or active replication

Grid

- Thousand of nodes and more
  - Cluster interconnexions through internet
  - Heterogeneous nodes
  - Grid5000
- Grid OS for ressources reservation, Task scheduling and protection
- Mainly parallele calculus

Cloud computing

- Deliver IT ressources and service on demand over the network
  - Virtualized OS and network
  - Auto-scalability (scale up/scale down)
  - Pay as you use: Low and fast deployment cost
  - IT is managed by the cloud provider
- 3 layers
  - IaaS (EC2, microsoft AZURE) : provides VM
  - PaaS (Google Apps) : provides application servers
  - SaaS (Salesforces) : provides applications
- 3 infrastructures
  - Public, Private, Hybrid