

# The Operating System Machine Level

Wolfgang Schreiner  
Research Institute for Symbolic Computation (RISC-Linz)  
Johannes Kepler University

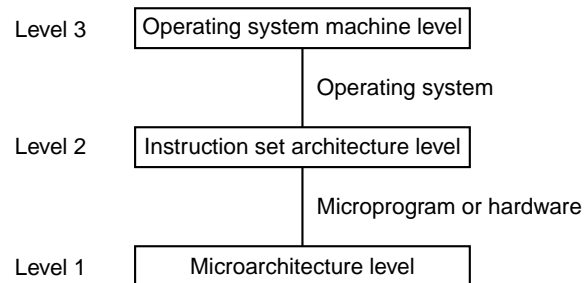
Wolfgang.Schreiner@risc.uni-linz.ac.at  
<http://www.risc.uni-linz.ac.at/people/schreine>

## The Operating System Machine (OSM)

The OSM is implemented by the operating system.

- **Operating System (OS):**

- Program that adds new instructions and features to the ISA.
  - \* The new instructions are called **system calls**.
- Implemented in software but can be also considered as a (virtual) machine.
  - \* OS is an **interpreter** for a system call.



Instruction set available to application programmers.

## Operating System Services

An OS provides important services to the application programmer.

- Process control:
  - Let a processor “simultaneously” execute multiple processes.
- Memory control:
  - Let machine appear to have more memory than it actually has.
- File control:
  - Pretend that files are linear sequences of bytes.

Prominent examples are Unix/Linux and Windows NT/2000/XP.

# Process Control

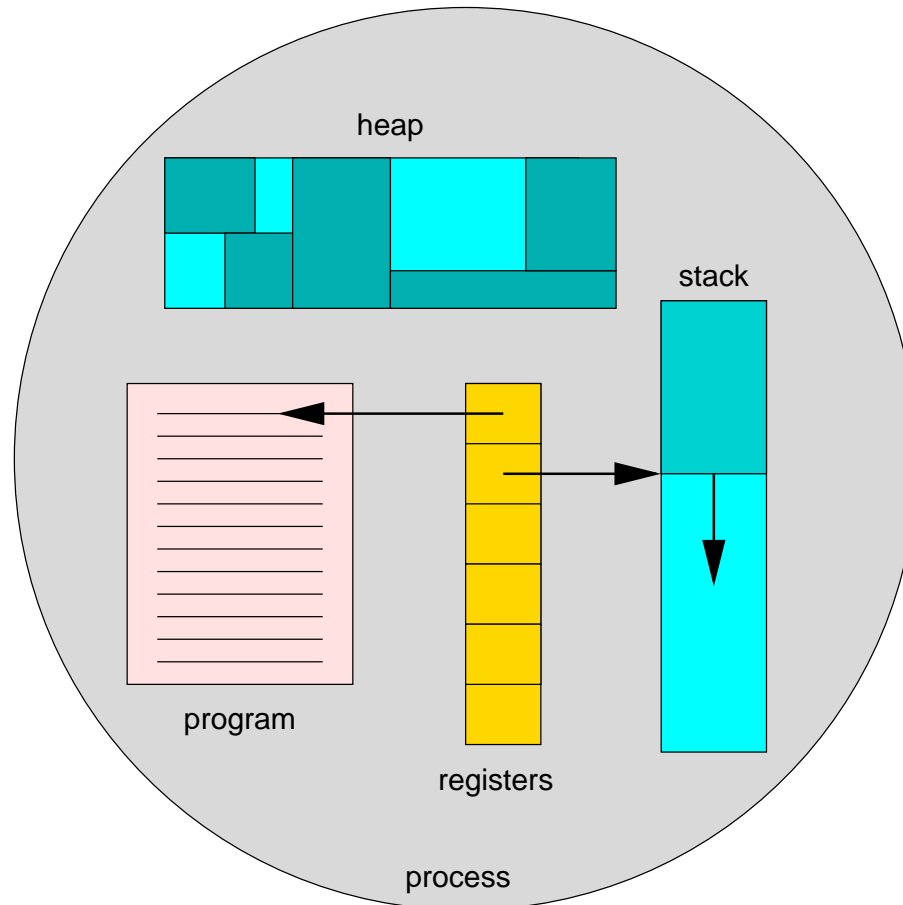
## Processes

**Process** = application program in execution.

- When the user starts a program, a new process is created.
  - The operation system manages process execution.
- Process consists of various components:
  - executable **code** loaded from disk into memory,
  - the **stack**, a memory area where program data are located,
  - the **heap**, a memory area where the program may allocate additional data space,
  - the (contents of the) program registers including the program counter and a **stack pointer**,
  - other information such as **user privileges**.

**Windows: CTRL-ALT-DEL ⇒ task manager.**

# Process



## Process Scheduling

Process may be in one of three states:

- **Executing:**

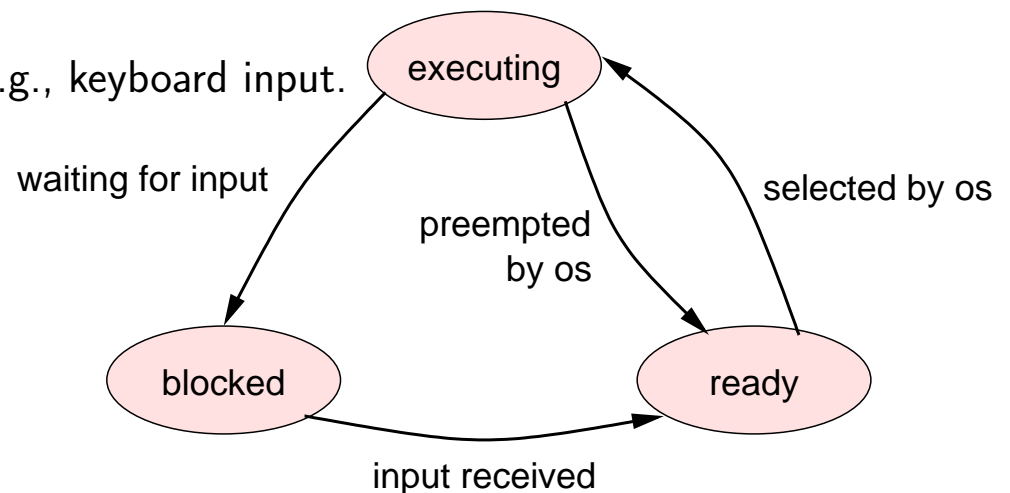
- The processor executes instructions of this process.

- **Ready:**

- The process is ready for execution but the processor executes instructions of another process.

- **Blocked:**

- The process waits for some event, e.g., keyboard input.



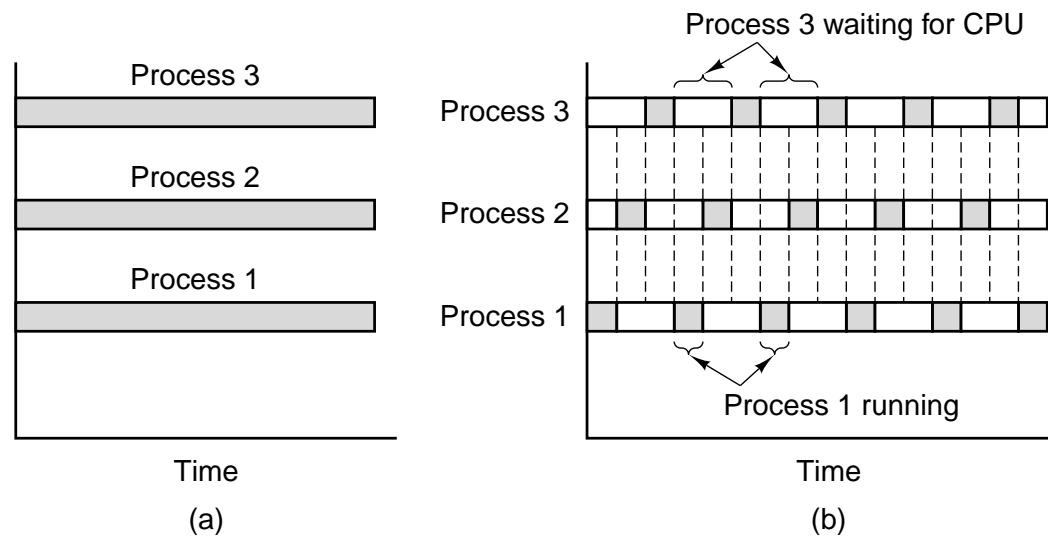
## Process Management

- At any time, the OS holds a pool of ready processes.
  - At most one process is executing.
- **Preemptive Scheduling:**
  - Executing process receives a certain **time slice**.
  - After the time slice has expired, OS **preempts** process.
  - Process is put into ready pool, another ready process is scheduled for execution.
  - Rapid switching (say every 50 ms) creates the **illusion** that processes are simultaneously active.
- To request an OS service, a process performs a **system call (trap)**:
  - Special processor instruction that gives control to the OS.
  - OS takes data from registers and determines which service to perform (e.g. output).
  - OS invokes system service that interacts with hardware device.
  - OS returns control to application.



## Multi-Processing

The OS schedules the CPU among multiple processes.



Multi-processing: multiple processes may run at the “same” time.

# Memory Control

## Virtual Memory

Program may need more memory than computer has.

- Tradition solution was the use of **overlays**.
  - Programmer divided a program into a number of overlays (pieces).
  - Overlays could be stored on secondary memory (disks).
  - Only one overlay was in computer memory at a time.
- Programmer was in charge for managing the overlays.
  - Reading overlays from disk to memory, writing overlays from memory to disk.
  - Difficult and error-prone task.
- Still used in the 1990s for DOS/Windows 3.x programs.

**Virtual memory emerged from the automation of overlay management.**

## Address Spaces

Idea: separate the address space from physical memory locations.

- **Virtual address space.**

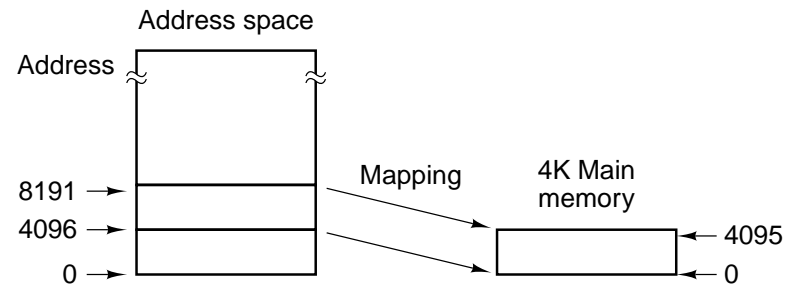
- Set of addresses to which program can refer.
- Size depends on needs of program.

- **Physical address space.**

- Set of addresses where data can be stored.
- Size restricted by cost of hardware.

- **Pages:** blocks of addresses of fixed size.

- E.g. 4096 bytes: page 0–4095, page 4096–8191, page 8192–12287, ...
- Virtual address space is organized in pages.



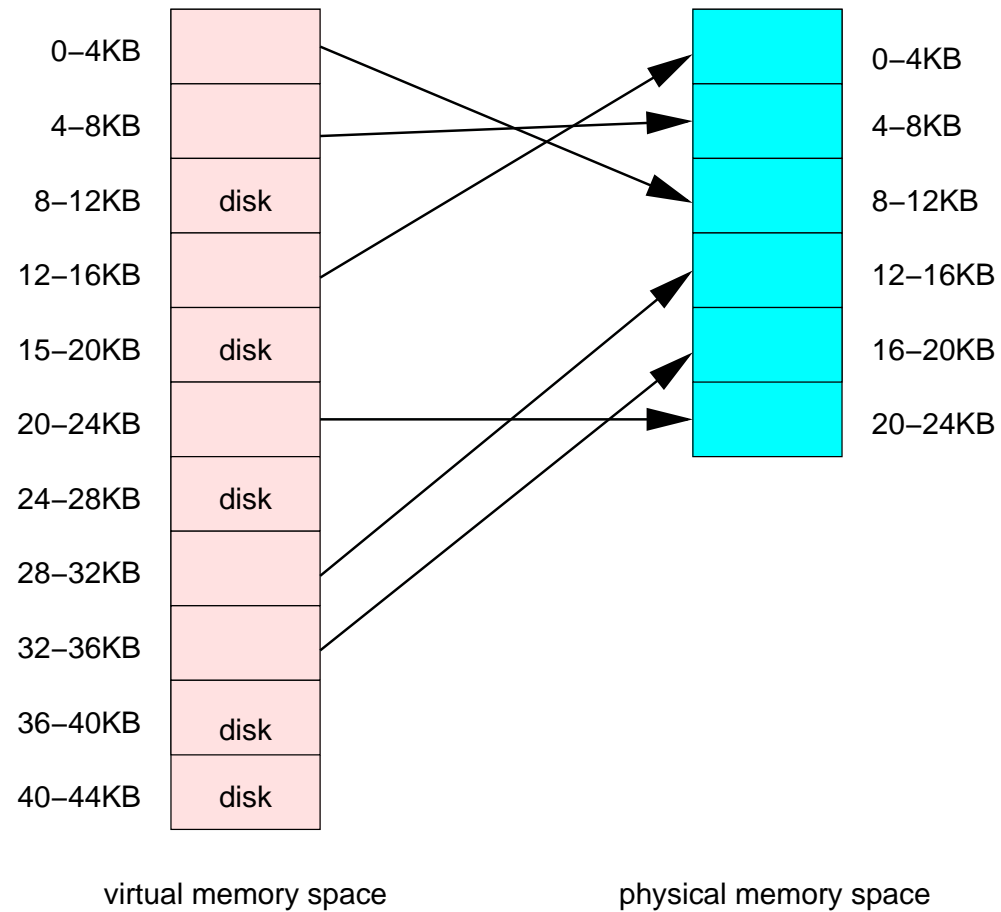
Virtual addresses are mapped to physical addresses.

## Pages and Page Frames

Physical memory has frames that can hold virtual memory pages.

Page	Virtual addresses	Page frame	Physical addresses
15	61440 – 65535		
14	57344 – 61439		
13	53248 – 57343		
12	49152 – 53247		
11	45056 – 49151		
10	40960 – 45055		
9	36864 – 40959		
8	32768 – 36863		
7	28672 – 32767	7	28672 – 32767
6	24576 – 28671	6	24576 – 28671
5	20480 – 24575	5	20480 – 24575
4	16384 – 20479	4	16384 – 20479
3	12288 – 16383	3	12288 – 16383
2	8192 – 12287	2	8192 – 12287
1	4096 – 8191	1	4096 – 8191
0	0 – 4095	0	0 – 4095

# Page Table



## Example

- Program needs 16 KB memory with addresses 0–16383 ( $=2^{14} - 1$ ).

Virtual Address	Physical Address
0	65536
4096	61440
8192	32768
12288	4096

- Page table maps page at address 4096 to physical address 61440.
- Process references memory word at virtual address 4201.
- Memory management unit looks up table and computes:
  - $(4201 - 4096) + 61440 = 105 + 61440 = 61545$
- Physical address 61545 is sent via system bus to main memory.

## Paging

The virtual address space is much larger than the physical space.

- Program may refer to virtual address not in memory.
  - Trap is generated that interrupts normal program execution.
  - Page currently loaded in main memory is saved to disk.
  - Referenced page is loaded from disk into main memory.
  - Address map is changed to map virtual page to main memory page.
  - Program execution continues in the normal way.
- Programs may assume arbitrarily large virtual address space.
  - Pages are automatically loaded/stored from/to disk.
  - **Transparency:** programs need not be aware of virtual memory at all.

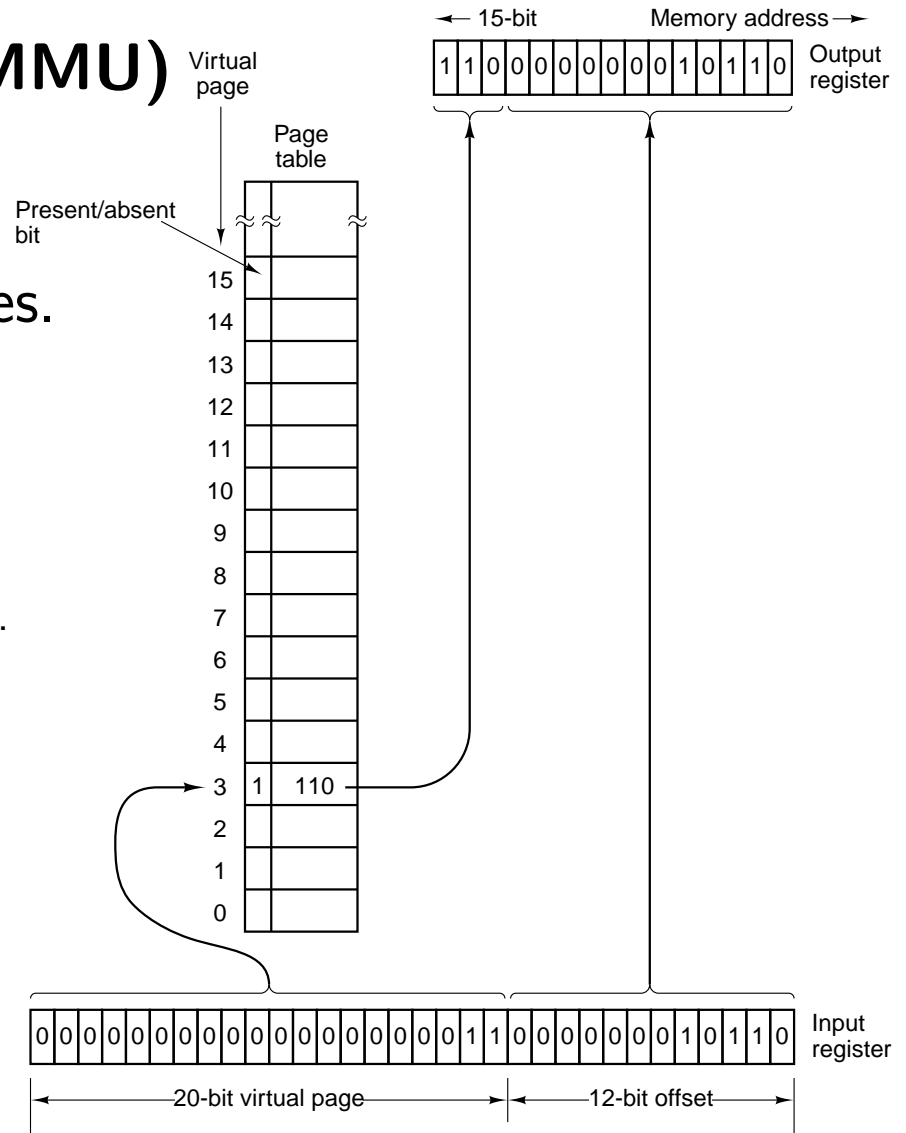
Paging frees program writers from memory constraints.



# Memory Management Unit (MMU)

Chip for address translation.

- Maps virtual to physical addresses.
  - Virtual: 32 bit.
  - Physical: 15 bit (e.g.).
- Virtual address is decomposed.
  - Virtual address = virtual page : page offset.
  - Page size: 4096 bytes (e.g.)
  - 12 bit page offsets ( $2^{12} = 4096$ ).
  - 20 bit virtual page ( $32 - 12 = 20$ ).
- Page table gives page address:
  - Indexed by virtual page.
  - Gives higher order bits for physical address.

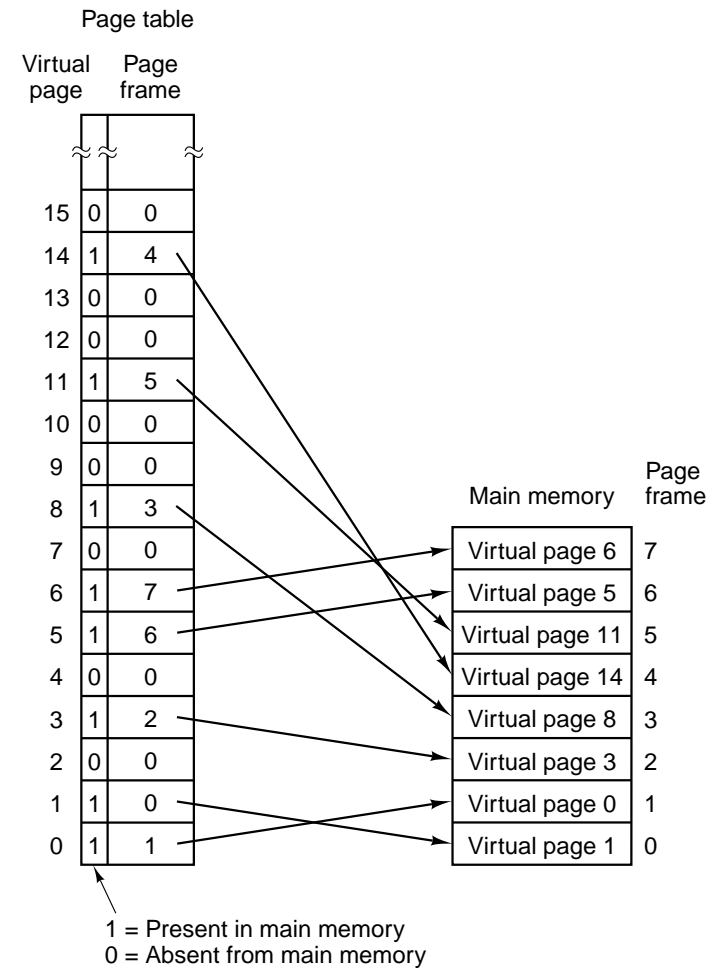


# MMU Mapping

Not all pages are in memory.

- Presence/absence bit.
  - Tells which table entries are in memory.
  - Other entries are located on disk.

One-to-one mapping of present pages to page frames.



## Demand Paging

When a referenced page is not in memory, a page fault occurs.

- OS must handle page fault.
  - Read required page from disk.
  - Enter its physical memory location in the page table.
  - Repeat the instruction that caused the fault.
- Initially: no page in memory.
  - All present bits are set to 0.
  - When CPU tries to fetch instruction, first page is loaded.
  - If program refers to other pages, these pages are also loaded.
  - The set of pages required by the program (its **working set**) is eventually loaded.

Pages are only loaded by page faults (i.e., on demand).

## Page Replacement

For each page to be loaded, another one must be stored on disk.

- A **page replacement** policy is needed.
  - Algorithm that tries to predict which page in memory is least useful.
    - \* Its absence would have the smallest adverse effect on the running program.
  - Select page that is not needed for the longest time in the future.
    - \* Problem is that OS cannot look in the future.
- **Least Recently Used (LRU)** algorithm.
  - Select page that was least recently used.
  - Probability that this page is not in the program's current working set is high.
  - Nevertheless pathological situations may occur.

**A variant of LRU is used in most operating systems.**

## Paging

- Paging frees applications from the limits of physical memory.
  - Virtual address space of a process may be much larger than physical address space.
  - Only the available disk space limits the size of a program.
  - Have some swap space available in your PC (e.g., 128 MB swap space for 256 MB main memory).
- But paging is slow:
  - Memory access time: 1 ns; disk access time: 10 ms.
  - Reading a page from disk is one **million** times slower than reading a memory cell.

Buy more memory rather than a faster processor.

## Segmentation

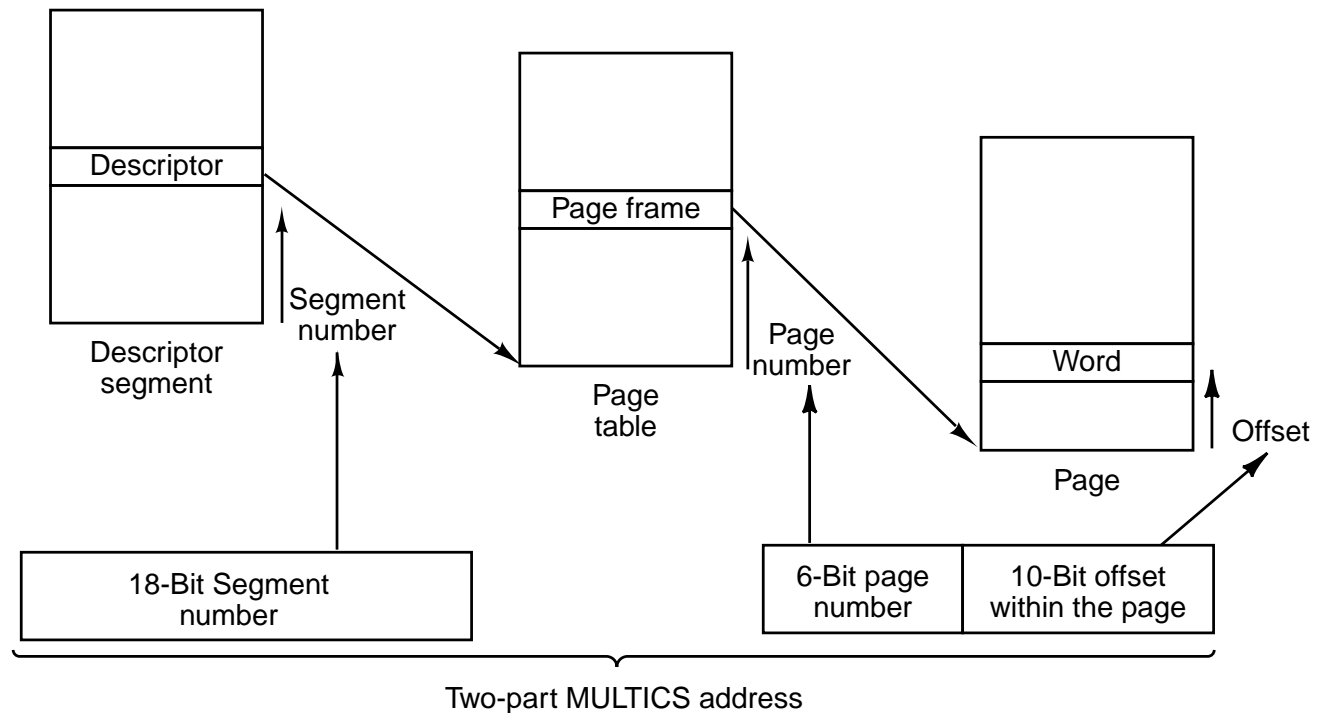
Virtual memory discussed up to now is one-dimensional.

- Typically **multiple address spaces** are needed.
  - Multiple tables that may independently grow and shrink.
  - E.g. a program's call stack and its heap space.
- Solution: use of **segments**.
  - Independent address spaces from 0 to some maximum.
  - Different segments may have different lengths; segment lengths may change.
  - Implementation by paging:
    - \* Each segment is organized as sequence of pages and loaded by demand paging.
    - \* A separate page table is needed for each segment.

**By multiple page tables, segments can be supported.**

# Segmentation

Example: MULTICS scheme, 1965.



## Virtual Memory on the Pentium II

Virtual memory with demand paging and segmentation.

- LDT (Local Descriptor Table)

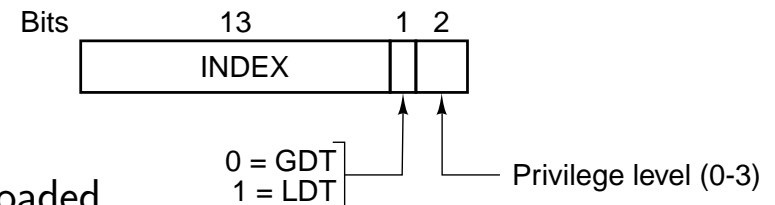
- One LDT for each program describing segments local for the program.
- Code, data, stack, ...

- GDT (Global Descriptor Table)

- Single GDT shared by all programs.
- System segments, e.g., operating system data.

- Pentium **segment registers**:

- CS for code segment, DS for data segment, ...
- Into each segment register, a **segment selector** is loaded.
- Every address is translated with the help of some loaded segment selector.

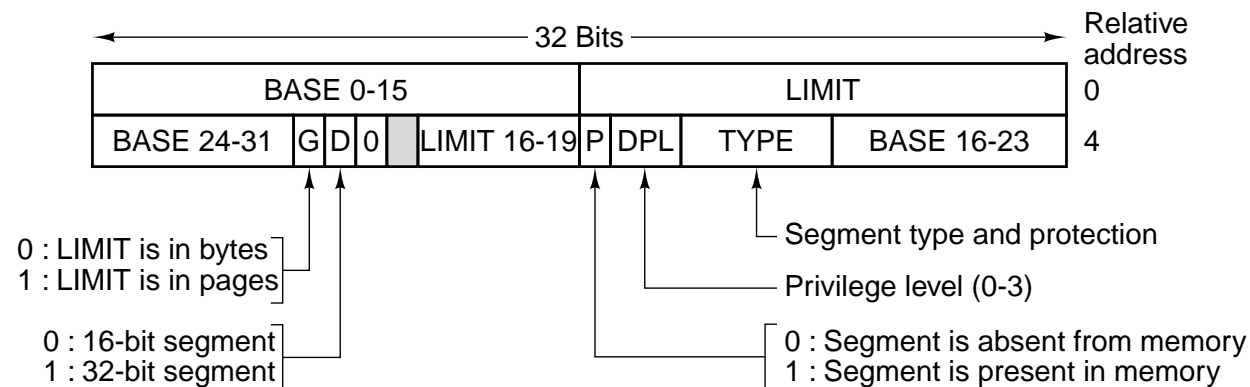


Segmented address is converted to physical address in two steps.



## Segment Descriptors

A segment is described by an 8 byte descriptor.

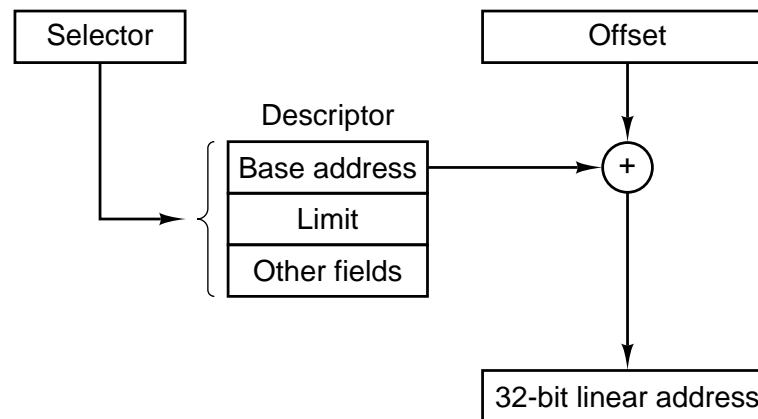


- Segment descriptor is stored in MMU register.
  - Segmented address consists of (segment selector, offset).
  - Segment selector references segment descriptor in MMU.

Descriptor is used to convert segmented address to linear address.

## Linear Address Computation

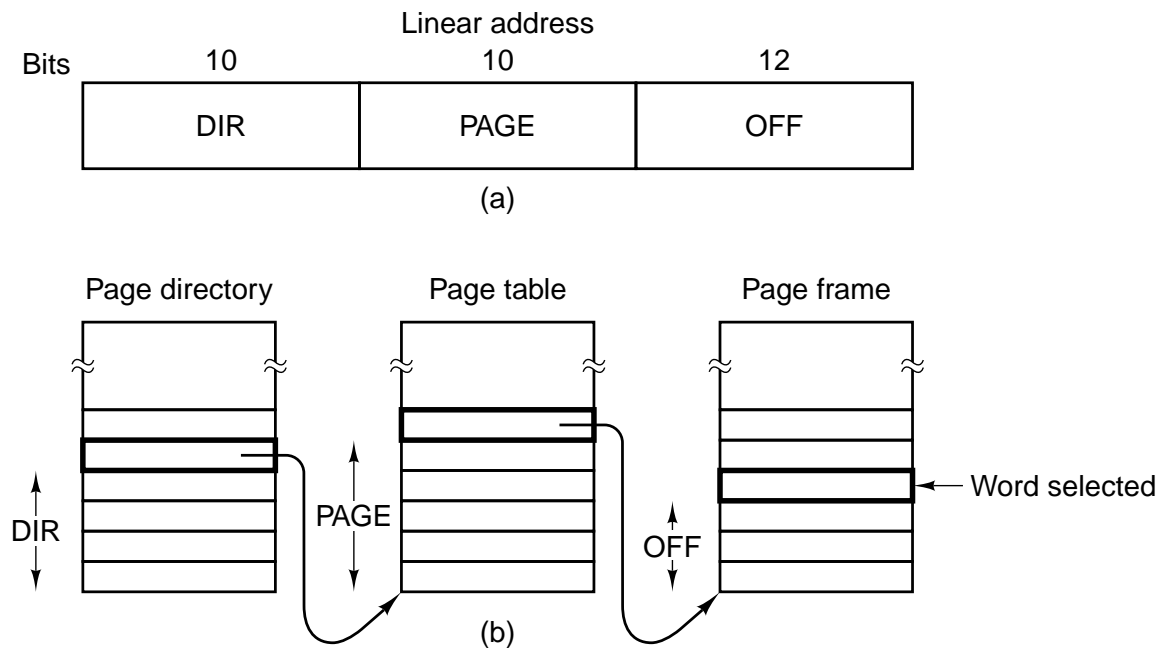
32 bit BASE field in descriptor is added to segment offset.



Linear address is interpreted as virtual address for paging scheme.

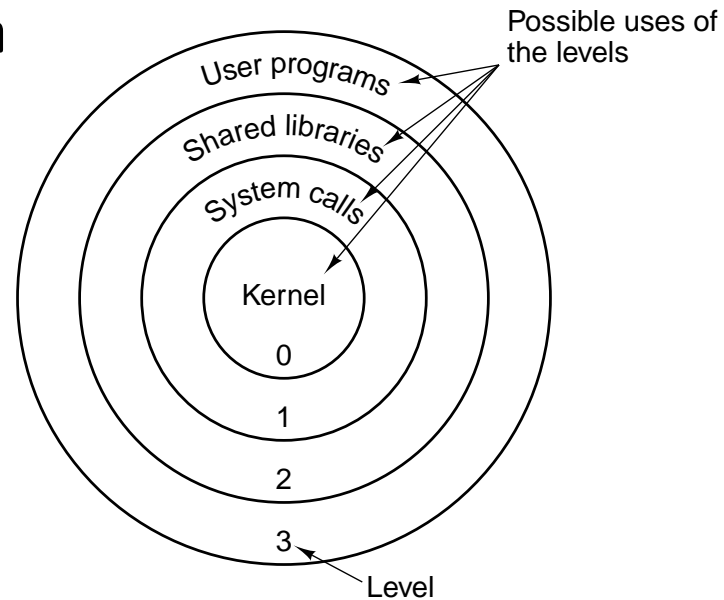
## Physical Address Computation

Physical address is computed by 2-level mapping scheme.



A program's page directory is located by global register.

## Memory Protection



- At each instant, a program operates in one protection level.
  - Indicated by 2-bit field in **PSW (Program Status Word)** register.
  - Access to data at lower level are illegal.
    - \* A trap is generated.
  - Only procedures at lower levels may be called.
    - \* Only access to official entry points in lower level.

# File Control

## Files

A file is a core abstraction of the virtual I/O.

- **File:** sequence of bytes written to an I/O device.
  - Device may be a disk: data can be read back later.
  - Device may be a printer: data cannot be read back.
  - Further file structure is up to application programs.
- **File I/O:** sequence of system calls.
  1. Open a file: locate file on disk and bring into memory information necessary to access it.
  2. Read or write data from/to file.
  3. Close the file: free space used to hold file information.

OS provides abstraction from concrete hardware control.

## Reading a File

Copy data from file to memory.

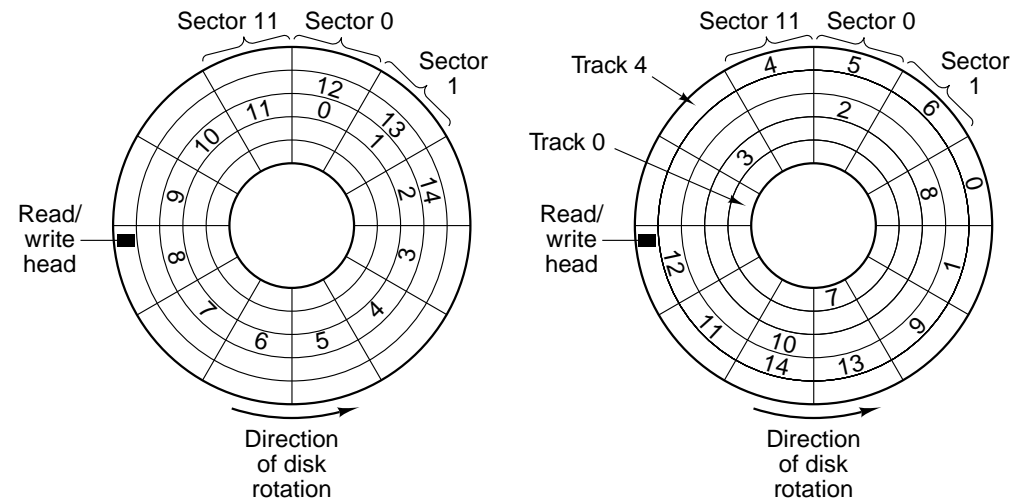
- System call `read`:
  - Indication of which (open) file is to be read.
  - Pointer to a memory buffer in which to put the data read.
  - Number of bytes to be read.
- Each open file has a pointer to the next byte position to be read.
  - `read` puts a certain number of bytes into buffer.
  - Pointer is advanced by the number of bytes read.
  - `read` returns number of bytes read.

Subsequent `read` calls read consecutive blocks of data.

## File Organization

How is the space for a file organized on a disk?

- File may be organized in consecutive sectors.
  - Used for CD-ROM file systems (file size known in advance).
- File may be organized random sectors.
  - Used for hard disk file systems (file grows dynamically).





## File View

OS sees file different than application programmer.

- OS sees a file as a collection of blocks on disk.
  - Application programmer sees a linear sequence of bytes.
- **File index** holds disk addresses of file blocks.
  - Typically organized as a list of disk block addresses.

OS maps file index information to linear byte sequence.

## Free Blocks

OS must know which sectors on disk are free for allocation.

- **Free List:** a list of all “holes” on disk.
  - Position and size of each hole.
- **Bit map:** one bit per file block.
  - Bit 1 indicates that file block is in use.

Track	Sector	Number of sectors in hole
0	0	5
0	6	6
1	0	10
1	11	1
2	1	1
2	3	3
2	7	5
3	0	3
3	9	3
4	3	8

(a)

Track	Sector											
	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0
2	1	0	1	0	0	0	1	0	0	0	0	0
3	0	0	0	1	1	1	1	1	1	0	0	0
4	1	1	1	0	0	0	0	0	0	0	0	1

(b)

## Block Sizes

How large should a file block be?

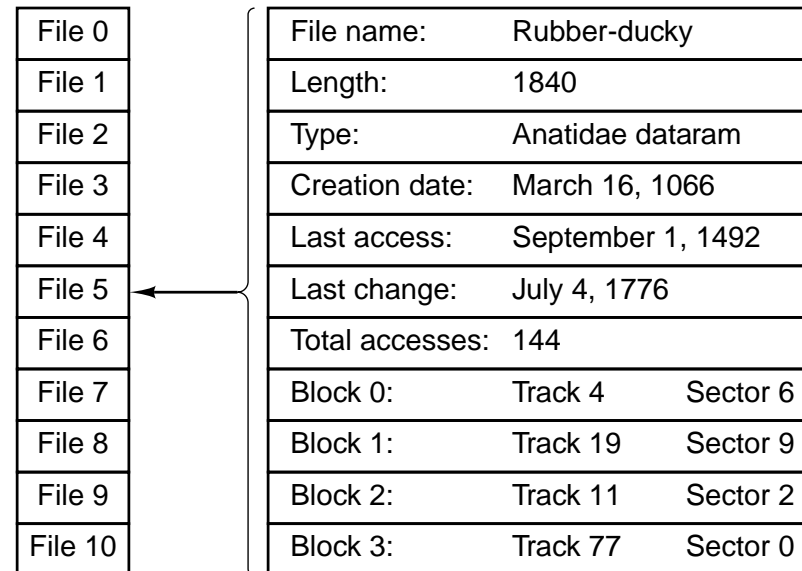
- Advantage of larger file blocks:
  - About 10 ms needed to seek file block.
  - Reading 1KB block takes about 0.125 ms, reading 8 KB block takes about 1 ms.
  - It is much better to read 8 KB at once than 8 times 1 KB.
- Advantage of smaller file blocks:
  - Minimum file size is 1 block.
  - Also for larger files, half of the space of the last block is wasted in average.
  - If files are small, much disk space may be wasted.

Today, larger blocks are used, because transfer efficiency is critical.

## Directory Management

Files are grouped in directories.

- Various system calls:
  - Create a file and enter it in a directory.
  - Delete a file from a directory.
  - Rename a file.
  - Change protection status of a file.
- Directory itself is a file.
  - May be listed in another directory.
  - Tree of directories emerges.



Directory may keep various pieces of data on a file.

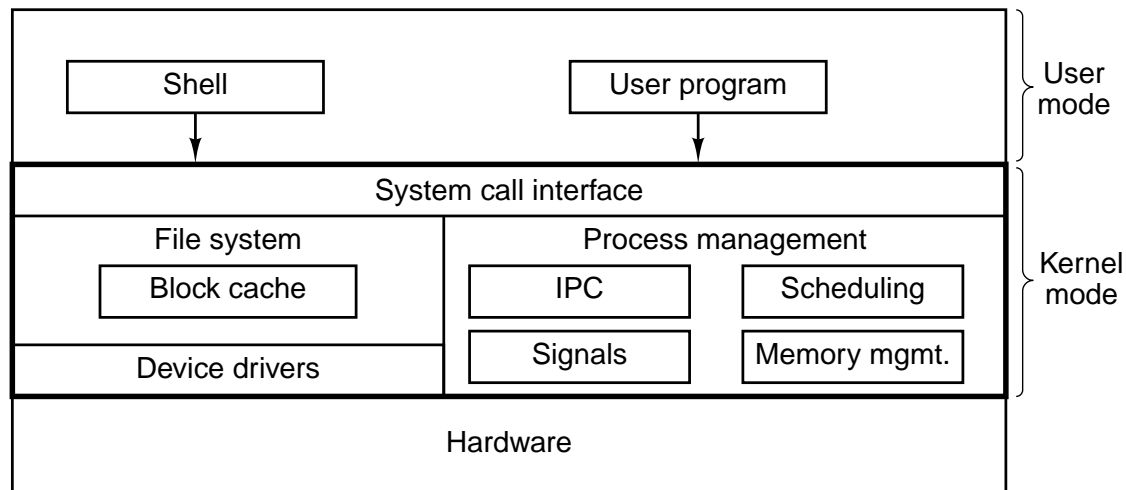
## Example: Unix

## Unix

- 1970: Ken Thompson, Dennis Ritchie at the AT&T Bell Labs.
  - Written for the PDP-7 in assembler; new version for the PDP-11 in C.
  - 1974 landmark paper in the Communications of the ACM.
- BSD Unix (Berkeley System Distribution).
  - Unix version by the University of California at Berkeley.
  - Inclusion of the TCP/IP protocol (later chosen for the Internet).
- 1984: System V Unix by AT&T.
  - Split in the Unix world between BSD and System V.
- IEEE P1003 standard: POSIX (Portable Operating System-IX).
  - Supported (and extended) by all Unix systems including Linux.
- 1990s: Linux (Linus Torvalds and many others)
  - Based on the GNU environment (Richard Stallman and many others).

## Unix Architecture

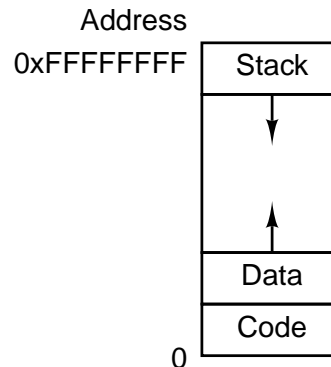
Small kernel with a modular layer of device drivers at the bottom.



GUIs (X-Windows based KDE or GNOME) operate in user mode.

## Unix Virtual Memory

Linear address space (no segments) divided in three parts.



- Entire address space is paged.
  - Program may be larger than machine's physical memory.
  - (Portions of) files may be mapped into the address space.
    - \* Memory-mapped files may be used for inter-process communication.



## Unix Virtual I/O

A file is a linear sequence of bytes.

```
/* open the files */
in = open("infile", 0);
out = creat("outfile", PROTECTION);

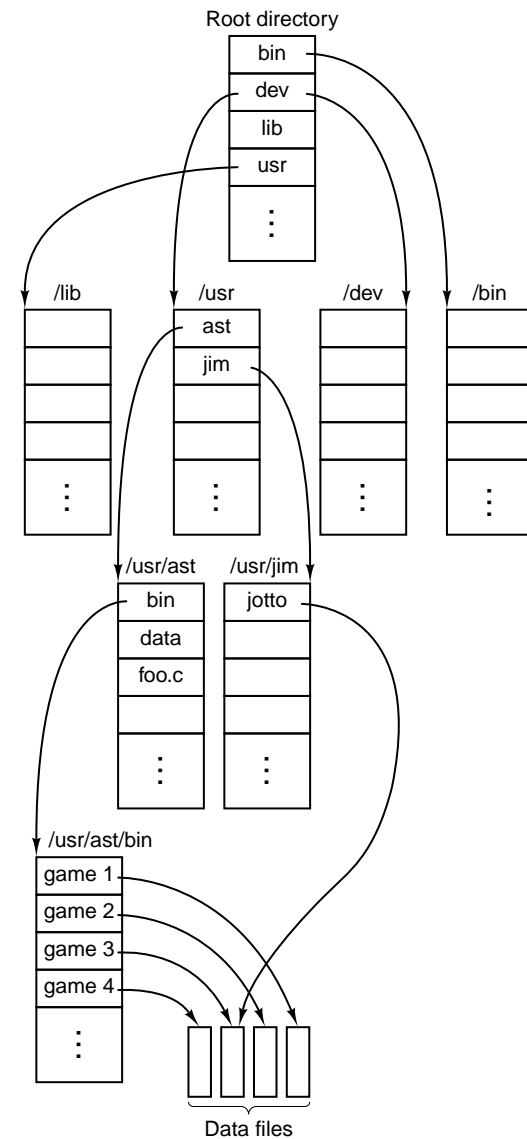
/* copy data from one file to the other */
do {
    count = read(in, buffer, BSIZE);
    if (count > 0) write(out, buffer, count);
} while (count > 0);

/* close the files */
close(in);
close(out);
```

# Unix Directory System

All disks are mounted in a single directory hierarchy.

- All files can be reached from the **root**:
  - An **absolute path** lists all directories from root to a file.
    - `/usr/jim/jotto`
  - A **relative path** lists all directories from the **working directory** of a process.
    - \* Process working directory is `/usr/ast`.
    - \* Relative path `bin/game3`.
  - A file may be **linked** to another file.
    - \* Both paths `/usr/ast/bin/game3` and `/usr/jim/jotto` refer to the same file.



## Unix File Systems

To each file, a 64-byte i-node is associated.

- **I-node** (index node).

- File type and protection, number of links to the file, owner's identity and group, file length.
- The time the file was last read and written; the time the i-node was last changed.
- 13 disk addresses.
  - \* The first 10 addresses point to data blocks.
  - \* Remaining addresses point to **indirect blocks** (blocks that point to data blocks or other indirect blocks: double indirect and triple indirect blocks).

- I-nodes are located at the beginning of the file system.

- Given an i-node number, the i-node can be located.

**Directory entries consist of file names and i-node numbers.**

## Unix Process Management

Unix supports multiple processes and multiple threads within a process.

- **Processes:**

- A process can create a replica of itself (`fork`).
- Both processes have separate address spaces.
- The newly created child process can replace its program by any other program (`exec`).
- Processes may communicate respectively synchronize via signals, pipes, semaphores, messages, shared memory.

- **Threads:**

- Within a process, multiple threads (light-weight processes) may execute.
- Threads share the data space of the process.
- Threads within a process can communicate via shared variables and synchronize via mutexes and condition variables.