Kernel has access to all resources
Kernel programs not subject to any constraints for memory access or hardware access
⇒ faulty kernel programs can cause system crash

Kernel provides its functions only via special functions, called system calls
standard C-library provides them
Have strict separation of kernel data and data for user programs
⇒ need explicit copying between user program and kernel

Simplified structure of kernel:
initialise data structures at boot time;
while (true) {
    while (timer not gone off) {
        assign CPU to suitable process;
        execute process;
    }
    select next suitable process;
}
In addition, have interrupts:
kernel asks HW to perform certain action
HW sends interrupt to kernel which performs desired action

Key points:
- No user context available while interrupts are processed
- Interrupts must be processed quickly
  ⇒ any code called from interrupts must not sleep

Kernel modules

- can add code to running kernel
- useful for providing device drivers which are required only if hardware present
- `modprobe` inserts module into running kernel
- `rmmod` removes module from running kernel (if unused)
- `lsmod` lists currently running modules

Concurrency issues in the kernel

- Consequence for handling concurrency in the kernel:
  Manipulation of data structures which are shared between
  - code running in user mode and code running in interrupt mode
  - code running in interrupt mode
  must happen only within critical regions
- In multi-processor system even manipulation of data structures shared between code running in user context must happen only within critical sections
Achieving mutual exclusion

Two ways:
- **Semaphores**: when entering critical section fails, current process is put to sleep until critical region is available ⇒ only usable if all critical regions are in user context
- **Spinlocks**: processor tries repeatedly to enter critical section
  Usable anywhere
  Disadvantage: Have busy waiting

Programming data transfer between userspace and kernel

Linux maintains a directory called proc as interface between user space and kernel
Files in this directory do not exist on disk
Read-and write-operations on these files translated into kernel operations, together with data transfer between user space and kernel
Useful mechanism for information exchange between kernel and user space

Using proc-interface

In the kernel, do the following:
- `create`

A tour of the Linux kernel

Major parts of the kernel:
- Device drivers: in the subdirectory `drivers`, sorted according to category
- file systems: in the subdirectory `fs`
- scheduling and process management: in the subdirectory `kernel`
- memory management: in the subdirectory `mm`
- networking code: in the subdirectory `net`
- architecture specific low-level code (including assembly code):
  in the subdirectory `arch`
- include-files: in the subdirectory `include`