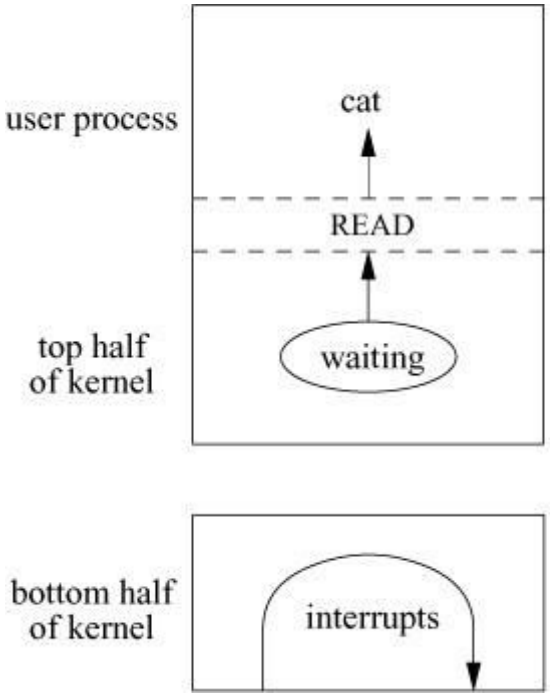
Systemy operacyjne

Wykład 15
Architektura jądra SO

Pierwotna struktura jądra

Górna i dolna połowa jądra



Podział niezależny od stopnia złożoności jądra. Reakcje na zdarzenia zewnętrzne i wykonywanie instrukcji programów żyją w różnych światach, a muszą się komunikować!

Dolna połówka

Dolna połówka asynchroniczna względem górnej połówki. Wykonuje kod obsługi przerwań. Informuje **górną połówkę** o pojawieniu się zdarzeń (np. koniec transmisji), które wznawiają wykonanie funkcji na rzecz programu użytkownika.

Nie może zatrzymać się nieokreśloną ilość czasu. **Dlaczego?**

Dolna połówka ma *“wyższy priorytet”*. W każdej chwili może przerwać działanie górnej połówki jeśli włączone przerwania.

Przypomnienie: Wyłączanie przerwań to instrukcja uprzywilejowana!

Pytanie: Co jeśli przerwania sprzętowe mają priorytety?

Górna połówka

Wielozadaniowość kooperacyjna – jeśli zadanie nie potrzebuje już procesora (np. czeka na przyjście zdarzenia) to woła **planistę** celem przełączenia na inne zadanie.

Górna połówka reprezentuje funkcje jądra wykonywane na rzecz programów użytkownika. Jeśli wiele programów, to każdy ma przypisany **kontekst** jądra. Przełączanie programów następuje w sposób kooperacyjny.

Narzędzia do zachowywania i wznawiania kontekstów: kanały oczekiwania, muteksy, skrzynki pocztowe, …

Wywłaszczalne jądro SO

Wywłaszczanie (ang. *preemption*) to odebranie zasobu programowi bez wpływu na jego poprawność.

Pojęcie najczęściej odnosi się do czasu procesora. Oznacza **przełączenie kontekstu** w wyniku zdarzenia zewnętrznego – np. upłynięcia **kwantu czasu**.

Jądro jest wywłaszczalne, jeśli w wyniku przerwań sprzętowych potrafi przełączać się między wykonywaniem funkcji na rzecz różnych procesów użytkownika.

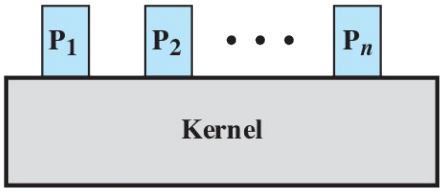
Pytanie: Czemu jest to pożądana cecha?

Gdzie wykonuje się kod jądra?

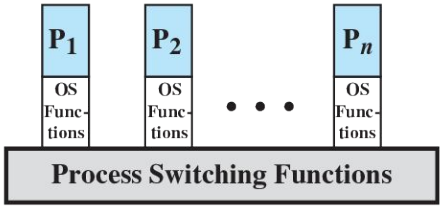
a. odrębna przestrzeń adresowa (wczesne SO dla PC)
b. jądra monolityczne
c. mikrojądra

Unikernels: jądro i program żyją w tej samej przestrzeni adresowej.

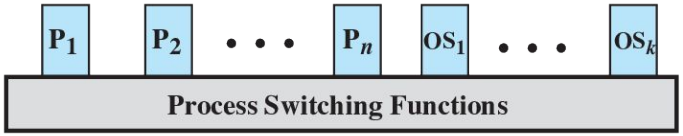
SO dla systemów wbudowanych również uruchamiają programy w tej samej przestrzeni adresowej.



(a) Separate kernel



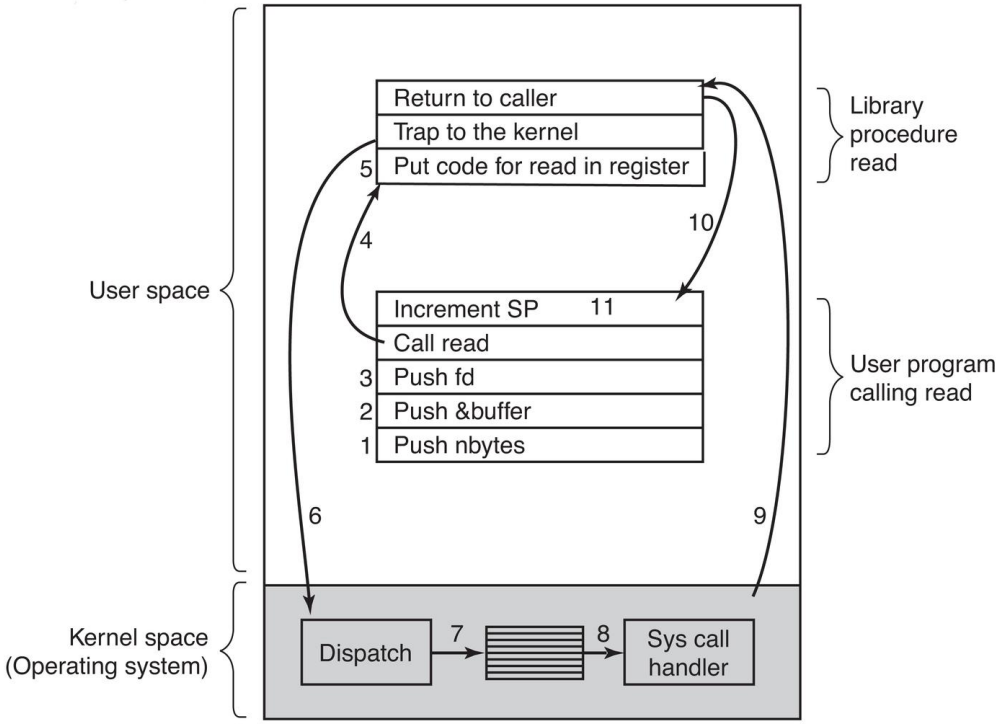
(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Obsługa wywołania systemowego we FreeBSD

- zapisanie kontekstu
- obsługa pułapki
- wektor wywołań systemowych
- copyin
- weryfikacja argumentów
- obsługa wywołania
- copyout
- ustawienie errno
- userret
- odtworzenie kontekstu
- powrót z pułapki



Architektura jądra

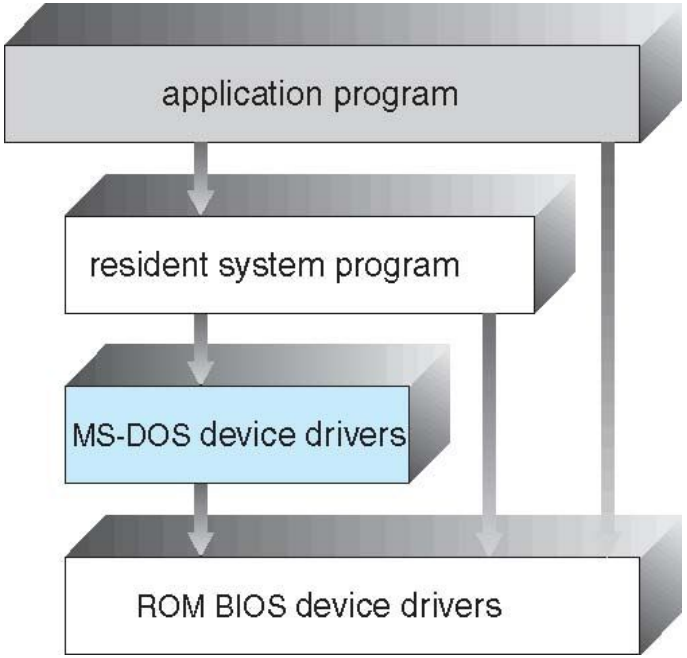
MS-DOS: Struktura ad-hoc

Podstawowe funkcje z minimalnym narzutem pamięciowym (min. 64K).

Wywołania systemowe dostarczone przez BIOS, interpreter poleceń, sterowniki i dodatkowe programy.

Brak izolacji, programy wykorzystują wiedzę o strukturach danych jądra.

Brak modularności, awaryjność.



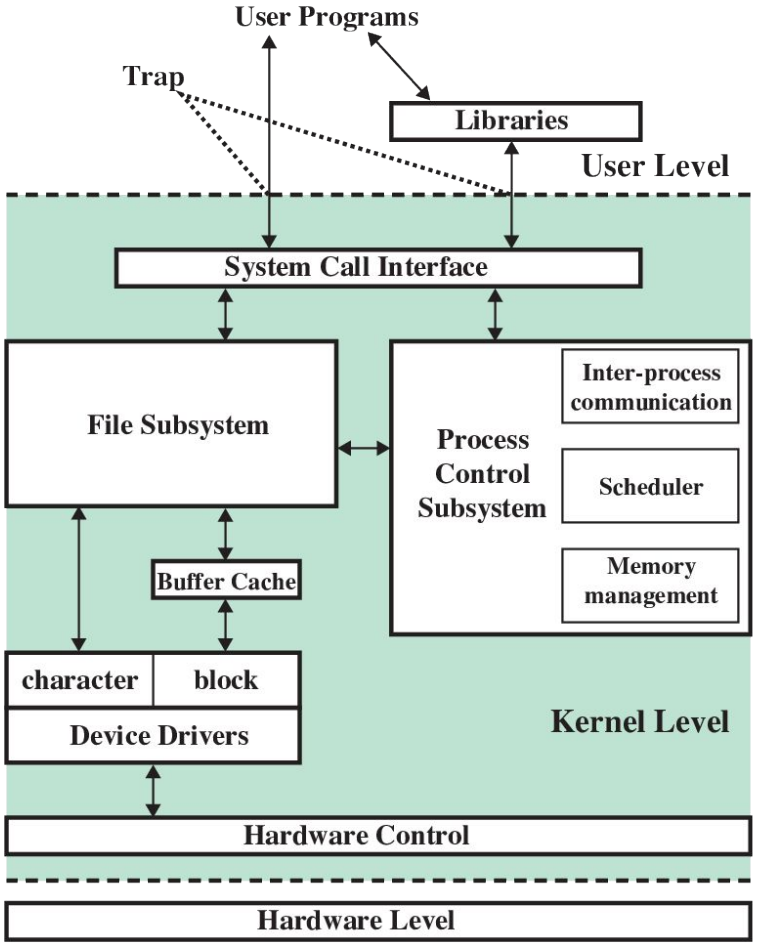
Unix: jądro monolityczne

Wymaga ochrony pamięci i trybu uprzywilejowanego. Błąd w jądrze kończy się załamaniem systemu.

Pierwotnie **niewywłaszczalne**.

Struktura modularna z elementami warstwowości.

Nierozszerzalne? Późniejsze systemy unikso-podobne dodają moduły i konsolidator jądra.



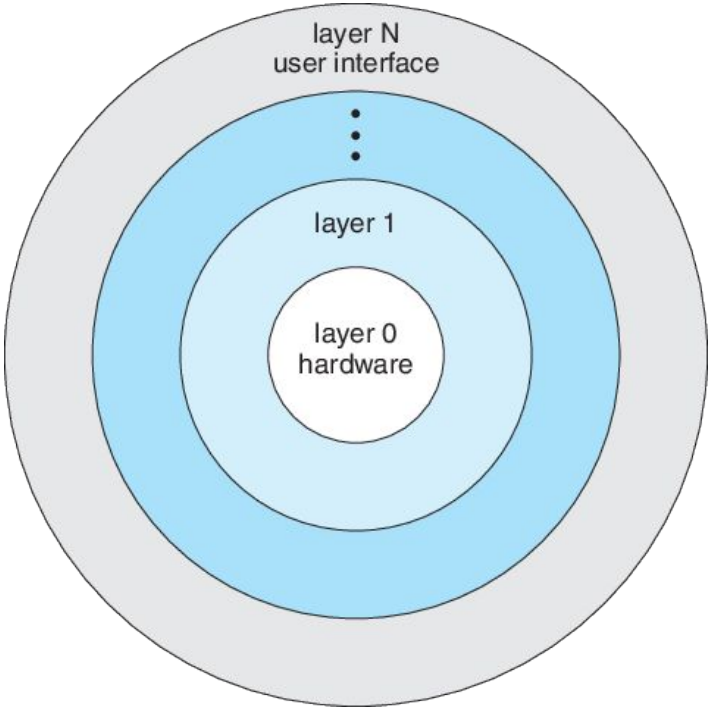
Struktura warstwowa

System operacyjny podzielony na warstwy, każda nadbudowana na niższej.

Dzięki odpowiedniemu zaprojektowaniu (modularność),warstwy korzystają tylko z funkcji warstw niżej.

W jądrze monolitycznym dużo większa pokusa, żeby łamać warstwowość celem optymalizacji!

Wyidealizowany obraz rzeczywistości!



Linux kernel map

2.6.32

functionalities

layers

human interface | system | processing | memory | storage | networking

user space interfaces

system calls and system files

HI char devices

cdev_add, input_fops, snd_fops, video_fops, console_fops, sys_capset, fb_fops, sys_syslog

interfaces core

kernel/ System Call Interface, system files, linux/syscalls.h, linux/uaccess.h, copy_from_user, /proc /sysfs /dev, sysfs_ops, register_chrdev, sys_ioctl, cdev_add, cdev_map, sys_epoll_create, cdev, sys_reboot, sys_init_module, km_dev_ucl

processes

kernel/ fs/exec.c, kernel/signal.c, sys_fork, sys_vfork, sys_clone, sys_kill, sys_signal, do_sigaction, sys_execve, linux_binfmt, sys_gettimeofday, sys_time, sys_times, sys_futex, sys_nanosleep

memory access

mm/ sys_brk, sys_mmap, shm_vm_ops, sys_shmctl, sys_shmat, sys_sysinfo, si_meminfo, sys_mprotect, sys_mincore, /dev/mem, mem_fops, mmap_mem, /proc/meminfo, /proc/self/maps, sys_msync

files & directories access

fs/ sys_chroot, sys_pivot_root, sys_mkdir, sys_chdir, sys_open, sys_read, sys_write, sys_tee, sys_poll, sys_pipe, sys_select, sys_flock, do_path_lookup, sys_readv, sys_getdents, sys_splice, sys_sendfile, sys_swapon, alloc_file, sys_chown, sys_chmod, fd, sys_newfstat, notify_change, fget, sys_fsync, fsnotify_change, inode_setattr, sys_mount, sys_sync, sys_sysfs

sockets access

net/ sys_socketcall, sys_connect, sys_accept, sys_bind, sys_listen, sys_sendmsg, sys_recvmsg, sys_setsockopt, sys_socket, /proc/net/, tcp4_seq_show, sg_proc_seq_show_dev, rt_cache_seq_show, sock_ioctl

virtual

security

security/ security, linux/security.h, security_capset, may_open, security_socket_create, inode_permission, security_inode_create, security_ops, selinux_ops

Device Model

drivers/base/ driver_init, kobject, linux/kobject.h, bus_register, kset, linux/device.h, bus_type, device_create, device, class, device_type, driver_register, device_driver, probe, load_module, module, module_param, kernel_param, kobject_uevent_init, kobject_uevent, kvm

threads

schedule_work, create_workqueue, INIT_WORK, queue_work, work_struct, workqueue_struct, kthread_create, kernel_thread, current, thread_info, do_fork

virtual memory

vmalloc_init, find_vma_prepare, vmalloc, vfree, vmlist, vm_struct, virt_to_page

Virtual File System

file, vfs_fsync, file_systems, vfs_read, vfs_write, vfs_fstat, vfs_getattr, vfs_create, inode, fsnotify, inode_operations, file_operations, file_system_type, get_sb, super_block, do_splice_direct, generic_file_aio_read, ramfs_fs_type

protocol families

inet_init, __sock_create, socket, inet_family_ops, inet_create, unix_family_ops, proto_ops, inet_dgram_ops, inet_stream_ops, socket_file_ops

bridges

cross-functional modules

debugging

sys_ptrace, log_buf, register_kprobe, printk, handle_sysrq, oprofile_start, kgdb_breakpoint, oprofile_init

synchronization

lock_kernel, mutex_lock_interruptible, mutex_unlock, kernel_flag, mutex, wait_for_completion, owner, complete, add_timer, timer_list, run_timer_softirq, down_interruptible, up, semaphore, wait_event, wake_up, msleep, spin_lock_irqsave, spin_unlock_irqrestore

memory mapping

mm/mmap.c, do_mmap, do_mmap_pgoff, kmem_cache_alloc, vma_link, mm_struct, vm_area_struct

page cache

address_space, bdi_writeback_thread, do_writepages

swap

si_swapinfo, swap_info, kswapd, do_swap_page, wakeup_kswapd

networking storage

nfs_file_operations, smb_fs_type, cifs_file_ops, iscsi_tcp_transport

socket splice

sock_sendpage, tcp_sendpage, udp_sendpage, sock_splice_read, tcp_splice_read

logical

functions implementations

HI subsystems

oss, alsa, video_device, mousedev_handler, tty

system run

init/ kernel/ boot, shutdown, power management, init/main.c, start_kernel, mount_root, do_initcalls, run_init_process, kernel_restart, kernel_power_off, hibernate, machine_ops

Scheduler

kernel/sched.c, task_struct, schedule_timeout, schedule, setup_timer, process_timeout, activate_task, rq, context_switch

logical memory

physically mapped memory, mm_init, kmalloc, kfree, pgd_t, pmd_t, pte_t

logical file systems

ext4_file_operations, ext4_get_sb, ext4_readdir

protocols

proto, /proc/net/protocols, udp_prot, tcp_prot, udp_recvmsg, tcp_recvmsg, udp_sendmsg, tcp_sendmsg, udp_rcv, tcp_v4_rcv, tcp_transmit_skb, NF_HOOK, nf_hooks, ip_queue_xmit, ip_local_deliver, ip_push_pending_frames, ip_route_input, alloc_skb, ip_rcv, sk_buff, ip_output

device control

drivers/

abstract devices and HID class drivers

drivers/input/ drivers/media/ sound/, console, kbd, fb_ops, drm_driver, mousedev

generic HW access

request_region, pci_driver, request_mem_region, pci_register_driver, pci_request_regions, usb_driver, usb_submit_urb, usb_hcd_giveback_urb, ioremap, usb_hcd

interrupts core

request_irq, tasklet_struct, jiffies_64++, tasklet_action, do_timer, setup_irq, tick_periodic, timer_interrupt, do_softirq, do_IRQ, irq_desc, softirq_init

Page Allocator

/proc/slabinfo, mm/slob.c, mm/slub.c, mm/slab.c, kmem_cache, __free_pages, kmem_cache_init, kmem_cache_alloc, __free_one_page, __get_free_pages, page, __alloc_pages, vm_stat, totalram_pages, try_to_free_pages

block devices

block/ gendisk, block_device_operations, request_queue, init_scsi, scsi_device, scsi_driver, sd_fops, usb_storage_driver, usb_stor_host_template

network interface

linux/netdevice.h, netif_receive_skb, dev_queue_xmit, netif_rx, register_netdev, net_device, dev_ioctl, alloc_netdev_mq, ieee80211_alloc_hw, ether_setup, ieee80211_rx, netif_carrier_on, ieee80211_xmit, drivers/net/

hardware interfaces

drivers, registers and interrupts

HI peripherals device drivers

uvc_driver, vga_con, ac97_driver, atkbd_drv, i8042_driver, psmouse, drivers/media/video/

device access and bus drivers

ehci_irq, writew, readw, ehci_urb_enqueue, outw, inw, usb_hcd_irq, pci_read, pci_write

CPU specific

include/asm/, arch/x86/, setup_arch, x86_init, trap_init, early_trap_init, start_thread, native_init_IRQ, switch_to, /proc/interrupts, set_intr_gate, system_call, show_regs, interrupt, pt_regs, atomic_t, cli, sti

physical memory operations

arch/x86/mm/, mem_init, get_page_from_freelist, zonelist, zone, free_area, free_list, out_of_memory, die, num_physpages, do_page_fault

disk controller drivers

scsi_host_alloc, libata, Scsi_Host, ahci_pci_driver, aic94xx_init

network device drivers

usbnet_probe, ipw2100_pci_init_one, zd1201_probe, e1000_xmit_frame, e1000_intr

electronics

user peripherals

keyboard, camera, mouse, graphics card, audio

I/O

I/O mem, I/O ports, ACPI, USB controller, PCI controller

CPU

registers, APIC, interrupt controller

memory

RAM, DMA, MMU

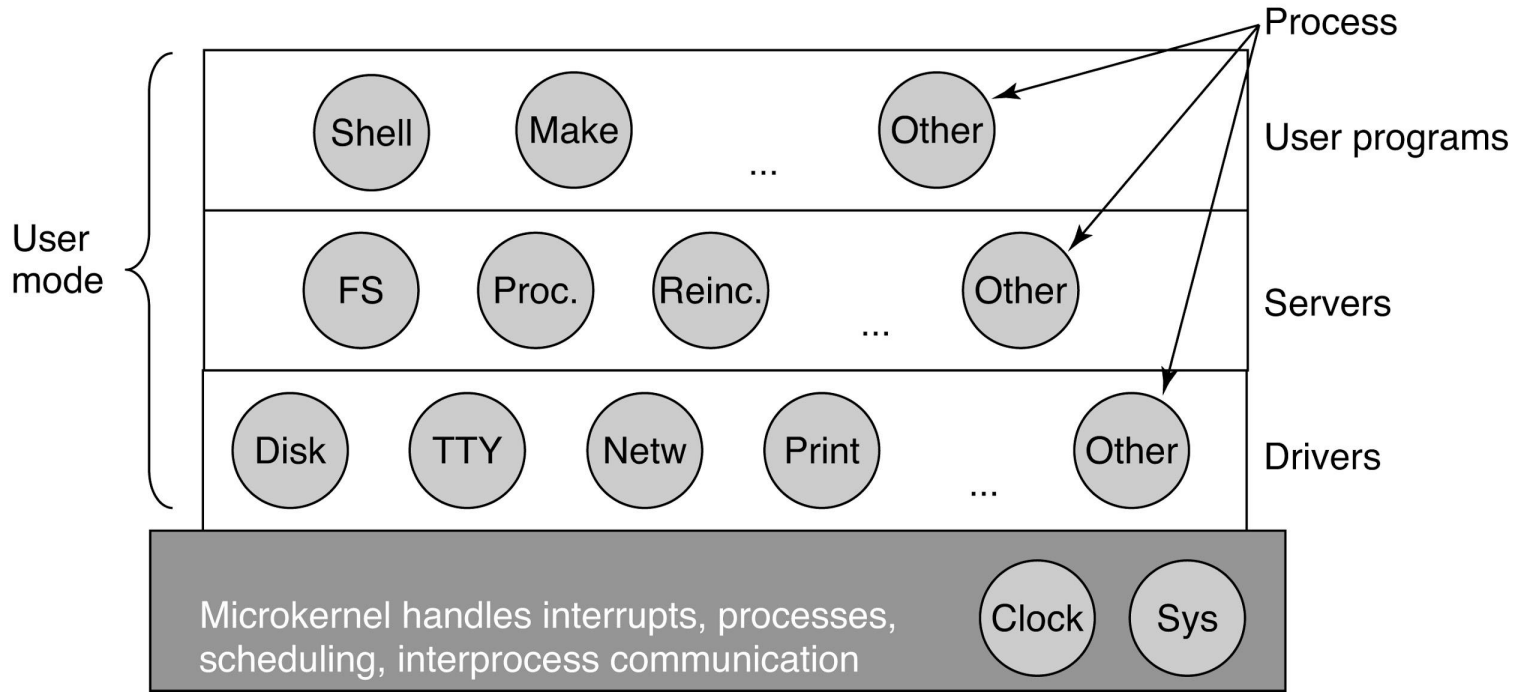
disk controllers

SCSI, SATA

network controllers

Ethernet, WiFi

Mikrojądro

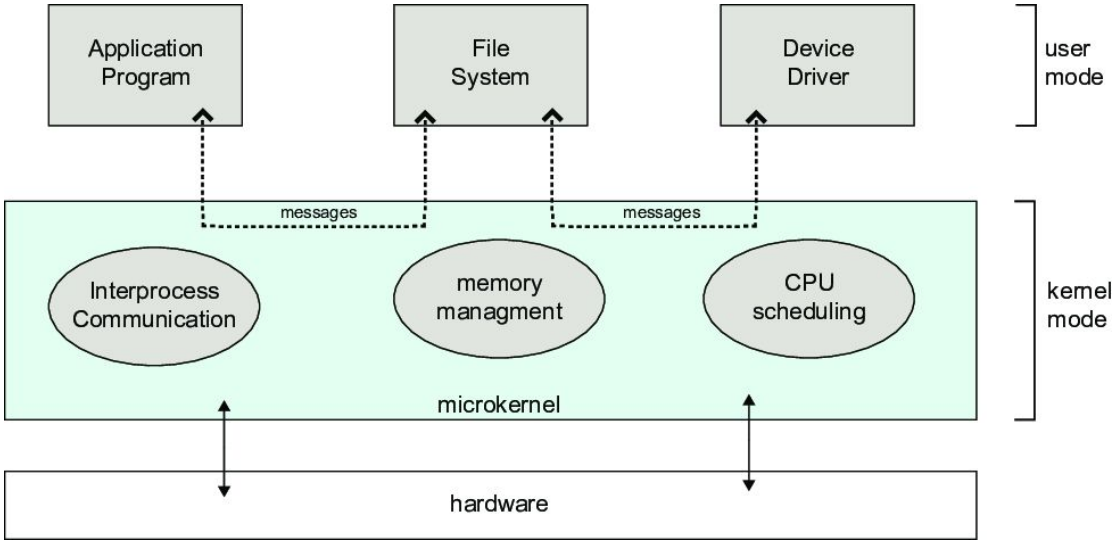


W jądrze minimalny zestaw funkcji. Usługi, sterowniki, systemy plików – działają na tych samych zasadach co programu użytkownika. Można je restartować – serwer reinkarnacji Minix 3.

IPC z użyciem przekazywania komunikatów

Jądro tylko pośredniczy w wymianie komunikatów między serwerami usług.

Izolacja między usługami polepsza niezawodność i elastyczność systemu



Koszt? Więcej kopiowania danych. Większa liczba zmiany kontekstów. Tylko małe komunikaty można kopiować efektywnie! Co zrobić z dużymi?

Mniejsza wydajność niż jądra monolityczne. Trudności pokonane, ale niesmak pozostał → jądra hybrydowe. Za to L4 się udało!

Windows NT: jądro hybrydowe

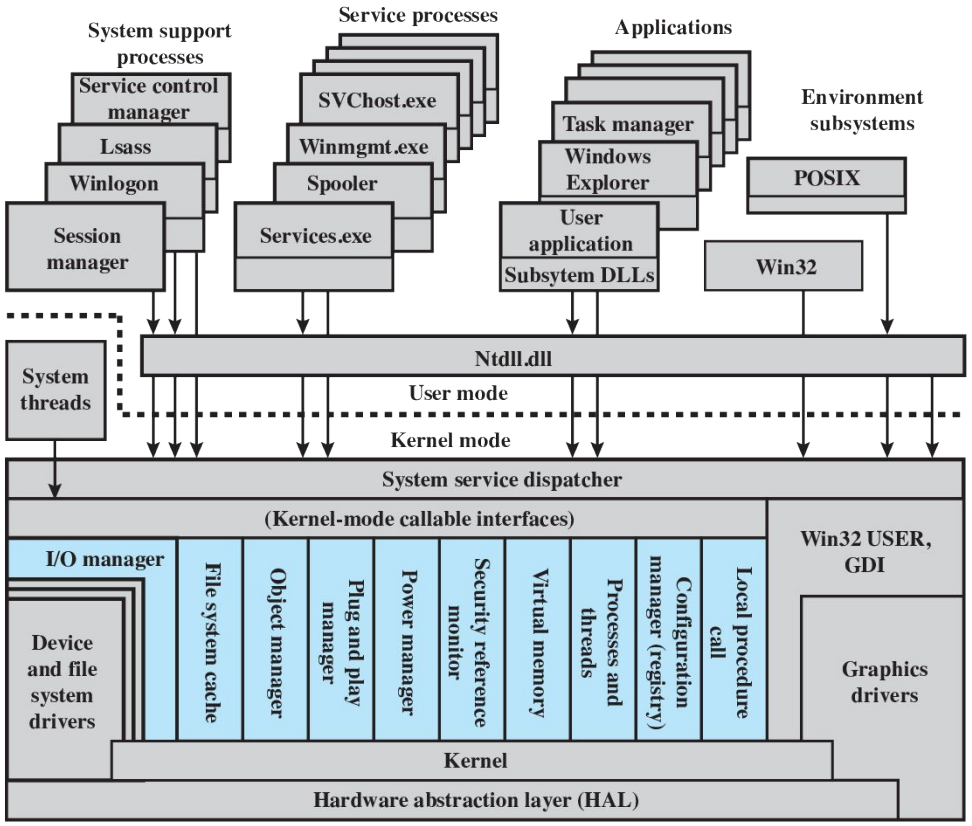
Zmiana kontekstu zbyt droga. Krytyczne wydajnościowo komponenty w jądrze.

Kod jądra wykonuje się poza kontekstem wątków jądrowych.

HAL unifikuje dostęp do części zasobów sprzętowych.

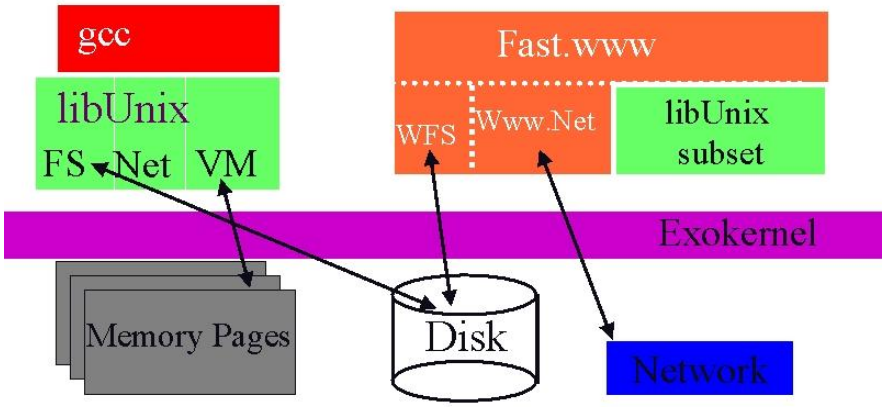
Windows Executive dostarcza podstawowych usług dla programów użytkownika.

Osobowości: Win32, POSIX.



Exokernel

Abstrakcje powodują spadek wydajności. Interfejs jądra powinien być dostosowany do wymagań aplikacji.



Egzojądro nie zarządza zasobami! Dba jedynie o bezpieczne dzielenie zasobów między aplikacjami.

System operacyjny udostępnia zestaw bibliotek (libOS) realizujących różne warstwy abstrakcji. Każda aplikacja wybiera sobie czego potrzebuje.

Unikernel szczególnym przypadek architektury exokernel.

Mikrojądro L4

Wprowadzenie do L4

Małe (< 10kLOC)! Bardzo szybki mechanizm komunikacji. Nie ma planisty zadań ani zarządzania pamięcią!

- wątki
- przestrzeni adresowe
- zadania (grupowanie wątków)
- komunikacja międzyprocesowa

Wątek posiada zestaw wirtualnych rejestrów (**`TCR`**, **`MR`**, **`BR`**), do komunikacji z µ-jądrem i innymi wątkami. Rozbudowany protokół komunikacji z jądrem.

Komunikacja w L4

Jądro niewywłaszczalne poza ustalonymi punktami. Przesyłanie komunikatów blokujące → schadzki / randki (ang. *rednez-vous*).

Komunikat składa się: z nagłówka, pewnej liczby *untyped word* i *typed word*, co wpisujemy do ***M**essage **R**egisters*. Żeby odebrać należy przygotować miejsce i poinformować jądro przez wypełnienie ***B**uffer **R**egisters*.

Elementy o ustalonym znaczeniu:

- **`StringItem`** ciąg bajtów kopiowany przez jądro do odbiorcy
- **`MapItem`** wysłanie stron z zachowaniem prawa własności
- **`GrantItem`** przekazanie praw własności do stron

Protokoły L4

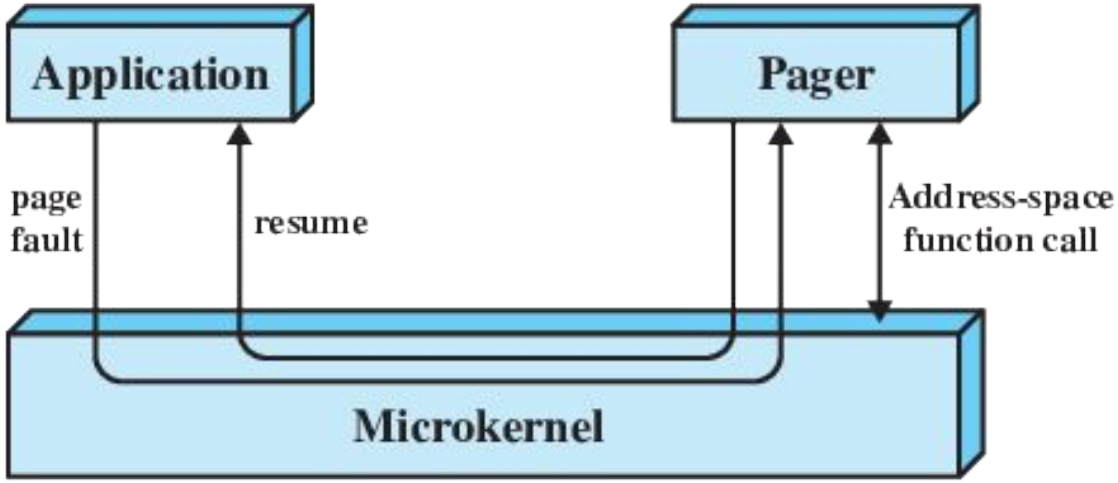
Zestaw specjalnych komunikatów jądra:

- **ThreadStart** początkowe PC i SP wątku
- **Interrupt** koduje zgłoszenie i obsłużenia przerwania
- **PageFault** informuje program stronicujący o błędzie strony wątku i pozwala podłączyć brakującą stronę
- **Preemption** zgłasza wywłaszczenie wątku programowi planisty
- **Exception** zgłasza powstanie wyjątku procesora w zadanym wątku i pozwala na wznowienie jego wykonania
- σ_0 obsługa początkowej przestrzeni adresowej

Planista o obsługa błędów stron w L4

Każdy wątek X ma przypisany wątek:

- **planisty S**, który wysyła informacje do jądra na ile czasu uruchomić wątek X,
- **programu stronicującego P,** który obsługuje błędy stron dla wątku X.



Pytania?