PART 1

System Administration

init reads configuration file `/etc/inittab`

id:runlevels:action:command

```
ap::sysinit:/sbin/autopush -f /etc/iu.ap
fs::sysinit:/sbin/rcS sysinit
is:3:initdefault:
p3:s1234:powerfail:/usr/sbin/shutdown -y -i5 -g0
sS:s:wait:/sbin/rcS
s0:0:wait:/sbin/rc0
s1:1:wait:/sbin/rc1
co:234:respawn:/usr/lib/saf/ttymon -g -h -p "console
login: " -d /dev/console
s2:2:wait:/sbin/rc2
s3:3:wait:/sbin/rc3
s5:5:wait:/sbin/rc5
s6:6:wait:/sbin/rc6
```

wait – run process, waits until it terminates, then reads next line
nowait – run process in the background (like & in shell scripts)
respawn – run process in the background.Whenever it dies, restart it.
sysinit – run only once, after init starts
initdefault – default runlevel definition, no process is executed

Booting, startup, shutdown

- **booting** = the phase between computer turned on and OS kernel running in memory
- **startup** = the phase between OS kernel started and full operating system functionality

`s2:2:wait:/sbin/rc2`

`/sbin/rc2` is a shell script:

```
for f in /etc/rc2.d/K*
do
    /sbin/sh $f stop
done
for f in /etc/rc2.d/S*
do
   /sbin/sh $f start
done
```

What is the content of `/etc/rc2.d/` ?

```
# ls -l /etc/rc2.d
lrwxrwxrwx  1 root  root   13 May 3 2003 K05atd -> ../init.d/atd
lrwxrwxrwx  1 root  root   14 May 3 2003 K06innd -> ../init.d/innd
lrwxrwxrwx  1 root  root   14 May 3 2003 S09isdn -> ../init.d/isdn
lrwxrwxrwx  1 root  root   14 May 3 2003 S55sshd -> ../init.d/sshd
lrwxrwxrwx  1 root  root   13 May 3 2003 S60lpd -> ../init.d/lpd
...
```

BOOT phases:

1. firmware (BOOT ROM)
   - contains: **P**ower **O**n **S**elf **T**est, general drivers, user interface, boot loader
   - selects boot disk
   - boot loader loads and executes boot block
2. boot block (`bootblk`)
   - small loader in the first sector(s) on the disk
   - contains pointer to filesystem reader
   - loads and executes filesystem reader
3. filesystem reader (`ufsboot`)
   - longer than boot block
   - understands to the structures of FS (i-nodes, data blocks, fragmentation)
   - finds and loads kernel (`unix` + `genunix`)
   - possibly finds and loads kernel configuration (`/etc/system`)
   - possibly finds and loads kernel modules
   - pass execution to the kernel

all subsystem starting scripts are in `/etc/init.d/`
they accept arguments `stop` and `start`

```
#!/sbin/sh
case "$1" in
'start')
        if [ -x /usr/sbin/cron ]; then
                /usr/bin/rm -f /etc/cron.d/FIFO
                /usr/sbin/cron &
        fi
        ;;

'stop')
        /usr/bin/pkill -x -u 0 -P 1 cron
        ;;

*)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
```

STARTUP phases:

1. kernel internal processes are started (`sched`, `fsflush`, `pageout`)
   PID starts from 2, they have no any file with task image (executable)
2. root directory is mounted by kernel (using kernel parameter)
3. console is opened (file descriptors 0,1,2)
4. init (PID=1) is executed (`/sbin/init`)
   - reads configuration `/etc/inittab`
   - executes all other processes, especially starting scripts `/sbin/rc`*n* and console port monitor (`ttymon`, `agetty`)
5. startup scripts `/sbin/rc`*n* (defined for each runlevel)
   - search in runlevel directories `/etc/rc`*n*`.d/` for subsyst. start/stop scripts
   - all subsystem scripts `Knnxxxx` are run with argument stop (`K42ntpd`)
   - all subsystem scripts `Snnxxxx` are run with argument start (`S50sshd`)
6. subsystem starting scripts (e.g. S50sshd)
   - start daemons in correct order, with correct arguments
7. daemons read configurations and serve requests
8. console port monitor initialises console and waits for user login

Runlevels

change:

`# init` *n*

display:

`# who -r`

runlevel	Solaris	RH Linux
0	shutdown, run firmware	shutdown and halt
s	singleuser, maintainance	singleuser, maintainance
1	similar to s	similar to s
2	multiuser w/o net servers	multiuser w/o NFS
3	full multiuser	text mode multiuser
4	not used	not used
5	poweroff	multiuser + Xwindow
6	reboot	reboot

Console login

```
co:234:respawn:/usr/lib/saf/ttymon -p "console login: " -d /dev/console
```

1. Portmonitor (`ttymon`) is started after boot, resets console terminal parameters, opens console device and waits for user login name. When user enter his name, executes `/bin/login` by `exec()` system call without `fork()` system call – login continues as the same process with the same PID, UID, open files, etc.

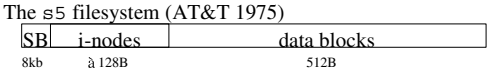
2. Login reads and checks password (against /etc/passwd), sets its own UID, changes the current directory and executes shell by `exec()`.

3. Shell executes global (`/etc/profile`) and per-user (`~/.profile`) settings and prompts user for command. Commands are forked and executed until command `exit` or Ctrl-D, which terminates the shell.

If any problem, wrong password, user logout, user shell is killed or dies then the process with the PID of portmonitor/login/shell terminates. Its parent (init) is signalled and respawn immediately new pormonitor process.

The UNIX filesystem

History: The `s5` filesystem (AT&T 1975)



The UFS filesystem (BSD 1986)

- superblock (SB), size 8kB
  - static information
    - magic number (filesystem type), filesystem size, number of inodes, block and fragment sizes, percentage of disk space reserved for privileged user, root i-node number
  - dynamic information
    - total number of free blocks, total number of free i-nodes
    - filesystem clean flag FS_CLEAN
  - ...
  - superblock listing `fstyp -v /dev/rdsk/c0t4d0s3`
    `tune2fs -l /dev/hda1`

2. Disk subsystem

Storage (disk) may be divided into regions called logical disks, partitions or slices

1. No partitions – we can use whole disk `/dev/dsk/c0t4d0`

+ no wasted space

- growing directories may fill up whole disk

2. Fixed size partitions – like `/dev/dsk/c0t4d0s3`

The first sector contains table of sizes and boundaries (VTOC or partition table) defining each partition. It is created by utility `fdisk` (Linux,Windows) or `format` (Solaris)

i-node

i-node size (s5 64B, UFS 128B, vxfs 256B)

display i-node number: `ls -i` i-node content: `ls -l`

```
# ls -l /etc/passwd
-rw-r--r--  1 root  root  2602 Oct 16 2005 /etc/passwd
```

i-node content	changed by command	notice:
type	-	types: f,d,l,b,c,p,s,D
permissions	`chmod`	12bits: sst rwx rwx rwx
link count	`ln` or `rm`	
owner	`chown`	stored UID only
group	`chgrp`	stored GID only
file size	-	
access time	`touch -a`	`ls -lu`
data modif. time	`touch -m`	`ls -l`
i-node modif. time	`touch -c`	`ls -lc`

pointers to allocated data blocks (typically 13-15 pointers)

REMEMBER: file name is not stored in i-node !

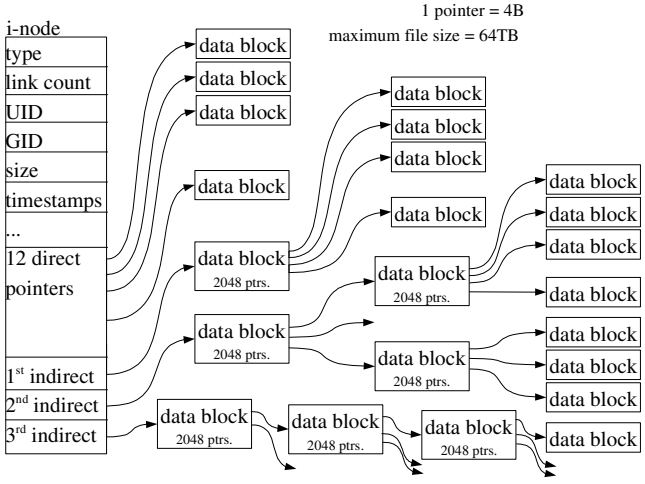
+ each filesystem has limited space
+ filesystems may differ (UFS, FAT, ...)
+ each filesystem may be mounted using different attributes (ro, noatime, nosuid)
+ backup utilities (ufsdump) may be used only for selected files
- size of partition cannot be changed dynamically
- partitions cannot exceed size of physical disk
- mounted partitions cannot be shifted without data loss

3. virtual disks based on volume management software (VM)

Solaris: Sun Storage Manager, or Veritas Volume Manager
Linux: RAID MetaDevices
HP-UX: Logical Volume Manager
HP-UX: up to 256 physical disks are grouped to *Volume Group*. Each disk is divided to 4MB peaces – *extends*. Extends are grouped to *Logical Volume*.
Example:
Volume group `/dev/vg00` contains two 2GB physical disks `/dev/rdsk/c0t3d0` and `/dev/rdsk/c0t4d0`. We can create logical volumes `/dev/vg00/lvol1` (3GB) and `/dev/vg00/lvol2` (1GB).

Data allocation:

example: 1 block = 8kB
1 pointer = 4B
maximum file size = 64TB



+ each filesystem has limited space
+ filesystems may differ (UFS, FAT, ...)
+ each filesystem may be mounted using different attributes (ro, noatime, nosuid)
+ backup utilities (ufsdump) may be used only for selected files
+ size of partition can be changed dynamically
+ partitions can exceed size of physical disk
+ additional redundancy can be achieved
+ parallel disk access may speed up data transfer

RAID 0 – striping
– concatenation
RAID 1 – mirror
RAID 2 – bit-level striping with ECC
RAID 3 – byte level striping with dedicated parity disk
RAID 4 – block level striping with dedicated parity disk
RAID 5 – block level striping with distributed parity
RAID 6 – block level striping with dual distributed parity
RAID 7 – asynchronous, cached striping with dedicated parity

Directory

- occupies one i-node and multiple data blocks as regular file
- data blocks contain a table of directory items. Each item has its name and associated i-node number

Block/character device

- occupies one i-node and no data blocks
- pointers to data blocks in i-node are replaced by MAJOR and MINOR numbers.

Named pipe and unix domain socket

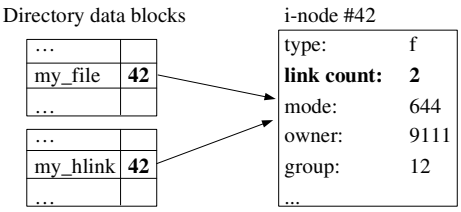
- occupies one i-node and no data blocks
- no data is written to disk. During open, kernel allocates small buffer in memory

Symbolic link

- occupies one i-node and one data block
- data block contains a string = target path
- some implementations may improve speed by storing short paths into i-node

Hardlink
- hardlinks are other names of the same i-node
- hardlinks are alowed only whithin a single filesystem
- hardlinks cannot be created on directories. Directory entries . and .. look like hardlinks, but they cannot be deleted without deleting of the directory
- each i-node contains a hardlink reference counter
- all hardlinks pointing to the same i-node are equivalent

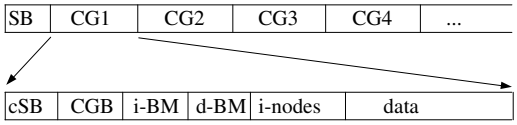
Example: A file with 2 hardlinks:



cylinder groups
During random file access, UFS reads i-nodes and their data blocks. In the model FS layout i-nodes are in the beginning of the disk while data blocks are spread over the whole disks, which leads to intensive head movement.
To speed-up operation both i-nodes and data blocks were divided into the same number of groups forming cylinder groups (CG).

Each CG contains: copy of superblock, cylinder group block, bitmap of free i-nodes, bitmap of free blocks, array of i-nodes and array of data blocks.

UFS:



sparse files
- not all data blocks are allocated
- the total file size is greater than total number of allocated blocks multiplyed by the block size
- sparse block (a hole) is a result of seek() behind the end of file and subsequent write()
- total size of files may be greater than the disk capacity

file size is reported by
`ls -l file`

allocated space is reported by
`du -k file`

allocation strategy
- data blocks are allocated in the same CG as their i-node => limits head moving
- the first block of any file is allocated in the middle of the biggest continuous free space=> each file can grow without fragmentation

suballocation
- if there is no free block, growing file may share the last block with another file
- the last block is divided into 8 fragments, each file uses different fragments
- only the last block may be shared
- each i-node contains a bitmap (1byte) of used fragments of the last block
- if a suballocated file grows it may require additional copy out of the last block (overhead) to satisfy free space. Without suballocation the operation would fail on unsufficient disk space.

journal filesystem (transaction filesystem)
- journal can be turn on during mount
- makes metadata changes (mkdir, rmdir, unlink, file create) atomic
- each metadata change is logged in journal (dedicated area on the filesystem) to be able in case of system crach do (after reboot) either commit all operations or roll back the operations.
- prevents errors: wrong link count, unconnected i-node, unused data block, unused i-node, …
- commit or roll back is finished by fsck after reboot

filesystem check (fsck)
- filesystem consistency check = 5 phases
- during systém startup it runs in non-interactive mode repairing minor problems
- interactive mode can repair all inconsistences
- in case of superblock loss, `mkfs` can be used to report data block numbers of backup superblocks, then any copy can be used to repair superblock using `fsck`

filesystem snapshot
- when it is created on mounted filesystem, kernel saves the content of each block prior to its first change
- original content (before changes) is stored in a sparse file on different filesystem
- kernel introduces a new block device /dev/fssnap/* which content is constructed as original content from sparse file (if the block is present) or appropriate block from original device (because still it has not been changed)
- /dev/fssnap/* provides a view of the content of the original device at the time of filesystem snapshot creation
- FS snapshot is used for backup with garanteed consistency
- /dev/fssnap/* can be dumped by dump
- /dev/fssnap/* can be read-only mounted to any other directory and backed up by any backup utility (tar, cpio,...)
- the size of sparse file is equal to the size of original block device
- the sparse file allocates as many blocks as many blocks were changed

3. OS Installation

1. Firmware (so-called BIOS/CMOS/BootROM) can be setup from local keyboard+display or serial console.
2. Host can boot from any disk, CD/DVD or network.
3. Installation media contains both: pre-installed OS and spooled OS (packages).
4. Installation program can be
   - interactive (controlled from local keyboard or from serial console)
   - noninteractive (controlled by configuration file)

Booting from network
1. get network details (IP address, netmask,router) – RARP, BOOTP, DHCP
2. download boot-loader (mostly TFTP)
3. optional: get boot parameters (who is install server, configuration, etc.) DHCP, BootParam protocol, NFS
4. download kernel (NFS, TFTP)
5. mount root filesystem – NFS

note: modern operating systems can boot (steps2-5) using HTTP

Noninteractive installation benefits:
- fast and error-free recovery method
- fast method for future major upgrades of OS
- can be parametrized to install different groups of machines
- can install and configure OS + software + run any post-install script

Packages
- whole OS is divided into set of packages
- package contains
  - binary files
  - configuration templates
  - documentation files
  - package information (author, name, description, hotline contact,...)
  - install and uninstall script
  - package dependence information
- operating system contains database of installed packages. Database stores relation between all installed packages and all their files. Some files/directories may be shared among several packages.
- packages may be installed/uninstalled in arbitrary order fulfilling dependency rules
- each OS have its package format (RPM)

Multilevel incremental backup
- each backup has level
- RULE: store all files with more recent timestamp than timestamp of previous backup with lower level
- timestamps of all previous backups are stored in backup-history file

Example: multilevel backup schema A
- do full backup each month
- each Friday do incremental backup relative to full backup
- each day do incremental backup relative to previous Friday

Example: multilevel backup schema B
- do full backup each month
- each Friday do incremental backup relative to previous week
- each day do incremental backup relative to previous day

4. Backup and Recovery

Cost of backup ~ cost of loosen data

cost of backup = cost of media + cost of drive + cost of work
(cost of 1kB: the cheapest are tapes, CD/DVD, disks, the most expensive are redundant systems)
- it should contain also cost of recovery and recovery testing

cost of data = cost of acquiring of new (equivalent) data (common binary files like /bin/ls are cheap, unique snapshots or measured values may be expensive)

B is faster during backup
A is faster during recovery
A needs less media
B can recover to more history points

schema A:

day	level	performed backup
Mo 1	L0	full backup (whole filesystem)
Tu 2	L2	changes in filesystem since day 1
We 3	L2	changes in filesystem since day 1
Th 4	L2	changes in filesystem since day 1
Fr 5	L1	changes in filesystem since day 1
Mo 6	L2	changes in filesystem since day 5
Tu 7	L2	changes in filesystem since day 5
We 8	L2	changes in filesystem since day 5
Th 9	L2	changes in filesystem since day 5
Fr 10	L1	changes in filesystem since day 1
Mo 11	L2	changes in filesystem since day 10
Tu 12	L2	changes in filesystem since day 10
We 13	L2	changes in filesystem since day 10
Th 14	L2	changes in filesystem since day 10

Data loss reasons:
- hardware failure (disk, bus, power supply, CPU, RAM, ...)
- software failure (bug in SW, wrong configuration, wrong version, ...)
- accidental removal by user (mis-spelling, space in argument, quoting, ...)
- targeted removal by user (hacker joke, revenge, war of competitors, ...)
- physical damage (fire, water flooding, hurricane, terrorism, war)

Backup & recovery plan
- what should be backed up
- where to store backups
- how frequent should be backup created
- kind of backup (full, incremental, ...)
- how long history of backups is kept
- how to do backup readability test
- how to restore data

Filesystem crash on day 13 – we recover
1) full backup – level 0 (from day 1)
2) last increment on level 1 (from day 10)
3) last increment on level 2 (from day 12)

we need 3 media sets (or better 6 – why?)

Backup utilities
- file archiving (tar, cpio, dump/restore)
- compression (compress/uncompress, gzip, bzip2)
- data storing and manipulation (dd, mt)

can be used separately (tar ... ; gzip ... ; dd ....)
or pipelined (tar ... | gzip | dd > storage_device), which is faster and does not require extra temporary storage space

Do not use non-unix filesystem utilities (zip, rar, arc, ...) Why?

Incremental (delta) backup
- based on “archive” file attribute bit (each application may mark changed files which have to be backed up - must be supported by applications)
- based on file modification timestamp (requires monotonic system time, allows multilevel incremental backup)

Backup must store all necessary information for future restore to be able restore exactly the same state as in time of backup.
- full recursive copy of files (including attributes) IS correct backup
- copy of whole filesystem (content of block device) IS correct backup
- copy of files with non-zero size IS NOT correct backup
- copy of files without hardlink information IS NOT correct backup
- incremental backup cannot be done by copying files with newer timestamps (it does NOT store information about file delete):
  find / -mtime 7 -print | do_backup_files_from_stdin

tar (= BSD tape archiver)
archive = file1_header+file1_data+file2_header+file2_data+...+EOF
- there is no archive header, only file header (header ~ inode)
- traverses filesystem and reads data by open() read() and close() syscalls
- list of files for backup may be specified on command line
- always recursive
- older versions had problem with initial / in file path
- replae/update option does not actually replace specified file in the middle of the archive but simply appends it at the end of archive (=> stored twice).

cpio (= SVR4 based archiving utility)
- similar to tar (but not compatible)
- the list of files for backup is read from stdin

Example: backup all PDF files
tar -cvf backup.tar `find . -type f -name ´*.pdf´ `
find . -type f -name ´*.pdf´ | cpio -ov > backup.cpio

dump/restore
archive = archive_hdr, file1_hdr, file2_hdr,...,file1_data, file2_data...
- supports 10 levels of multilevel incremental backup
- opens and reads raw filesystem directly from block device
- reads i-nodes into memory, sorts them by their data location, then writes headers and data => reading from disk is very fast (almost sequentially)
- filesystem specific (must implement filesystem reader)
- needs root access rights (to be able open block device)
- does not modify access time of files (does not read files via kernel open())
- list of content is present at the beginning of backup archive
- supports interactive restore
- timestamps of backups are stored in /etc/dumpdates

Kernel characteristics: *(thing about why?)*
- there is a permission test at the beginning of each syscall
- kernel offers minimal number of syscalls
- each syscall is as simple as possible (satisfying security and functionality)
- arguments of syscalls are mainly integers or strings. If struct is required (either for input or output) it is allocated in process space and process pass a pointer to it. Kernel never allocates/deallocates these structures.
- kernel never do what can be done in process or library
- any blocking syscall may be interrupted if signal arrives (the syscall returns error code EINTR meaning "interrupted syscall, try it again" to give a process a chance to response to the event.

compress/uncompress
- old compression utility
- adaptive Lempel-Ziv-Welch (LZW) compression algorithm (1985)
- default file extension .Z

gzip (GNU zip)
- Lempel-Ziv 77 (LZ77) coding algorithm (the same as in zip/pkzip "deflation" algorithm), better then LZW, year 1992
- consistency check (CRC32)

bzip2
- Borrows-Wheeler block sorting compression algorithm, better then conventional LZ77/LZ78 algorithms

other: pack – Huffman coding utility
compact – adaptive Huffman coding utility

Syscall groups:
- process and user management (getpid, getuid, fork, execve, exit, ...)
- filesystem (open, read, write, close, mount, access, mkdir, unlink,...)
- signal related (kill, signal, sigaction)
- time measure (time, nanosleep, settimeofday, ...)
- networking (socket, bind, listen, connect, accept, sendto, recvfrom,...)
- SystemV IPC (shared memory, semaphores, message queues)
- platform or OS dependent (poweroff, setiopl, sysinfo, ...)

In UNIX every I/O (files, device access, peripherals) is performed via special files (block/character devices, pipes and sockets). After opening kernel returns integer file descriptor (=opened socket, file, pipe or device).
If a number of FD is used at the same time, syscall select() enables a process to wait for the first finished operation on any FD.

dd – universal data copy utility
- copies data from stdin to stdout
- in or out can be redirected from/to file
- block size can be specified
- total number of blocks copied can be limited
- read offset and write offset can be specified
- data conversion may be specified (LF<->CR/LF, ASCII<->EBDIC, to lowercase, to uppercase, aligning to blocks, swapping byte order)

mt – magnetic tape manipulation utility
- does not work with data
- rewind tape, forward move to next record, backward move to previous record, eject tape, ...

Tape is represented by device /dev/rmt/0, each open/close create or read one record to/from tape.

Parts of kernel are not platform dependent (process table, signals, checking file access rights, filesystem drivers) while others are dependent (memory mapping, device drivers, CPU exceptions handling.

Kernel types:
monolitic kernel – consists of a single binary. Any part may access any data structure. Fast, but hard to maintain kernel source, hard to debug, hard to port to other platforms, kernel consumes more memory.
micro-kernel based kernel – is strictly divided into parts with well defined interfaces including internal access control between parts. Permission checks and interfaces takes overhead -> kernel is slow, but stable, easy to maintain, port or debug. Kernel parts occupy memory only if needed.
modular kernel is dynamically linked monolitic kernel. Fast, but with some interface between parts, each part (module) can be loaded separately.
Current kernels are mostly modular.

Kernel and modules

Structure of OS:
- kernel
- processes

Kernel functionality:
- kernel acts as a library (it offers services = functions = syscalls, has no own life (threads), any process may call any syscall)
- kernel differs from libraries – it uses privileged mode CPU
- kernel catches all interrupts (HW interrupts, SW interrupts, CPU exceptions, etc.)
- kernel lists and schedules processes

Kernel and its modules run in privileged mode of CPU
- kernel can use privileged instructions (to access I/O ports and map memory)
- no process (even process of privileged user) run in privileged mode
- privileged bit flag is set whenever process enters the kernel (special flag of entry point of each system call)
- privileged bit is set whenever it returns from kernel
- privileged bit is set also for each interrupt/exception routine

Kernel needs privileged mode
- to access HW devices (privileged instructions)
- to map memory (privileged instructions)
  - swapping, I/O data transfers, and inter process communication,
  - allocation/deallocation memory for kernel itself and processes
  - to protect some information against process (UID, priority, ulimit, ...)
- to handle CPU state (interrupt, changing CPU flags, start paging)

Memory protection:
- each process can access only memory pages which has been allocated and mapped to the process address space by the kernel
- access to any other (not mapped) address causes page-fault (a kind of CPU exception) handled by kernel. Kernel then decides whether:
  a) the missing page has been previously allocated by process but due to swapping it was swapped out => select a free RAM memory frame, swap in the old content, remap the frame to the missing page and continue the process
  b) the missing page has not been allocated => send signal SEGV to the process

Copy-on-write during fork()
fork() does not actually copy all process pages, but both parent and child share the same page with read-only flag set at CPU level. Whenever a write occurs, exception is triggered and kernel copy the page and continue.

Advanced security:
- mandatory access control (e.g. Security Enhanced Linux)
Each file belongs to a category. Each process belongs to another category. There is a set of strict rules which category may do some action with which category. (web servers can run CGI scripts but not /bin/* utilities, etc.) These rules are kernel-mandatory, i.e. Process is labeled must satisfy rules and can not escape.

- role based access control RBAC (Solaris RBAC)
A collection of rules define which user can run which process as UID=0, or which functionality inside some process is allowed to which user. Partially implemented by SetUID bit and special shell, partially by patching binaries. No support in kernel needed.

Kernel building (Linux only)
1) configure kernel (select which functionality (drivers) should be present)
2) compile static part of kernel (results in file bzImage or vmlinuz)
3) compile modules (lots of kernel object files *.ko)
4) setup new kernel for use (copy modules to correct location, instruct boot loader to use new static part and reboot the new static part)

Developing new modules
- each module should satisfy predefined interface with kernel
- there is only very limited version of standard C library in kernel
- developer must take care about reentrance, parallel processing on multiple CPUs, asynchronous events (HW interrupts, DMA), data structure locking
- module can change behavior or add a new kernel functionality

Advanced security: ***(cont.)***
- privileges (e.g. Solaris 10 privileges)
Each kernel action (mount, reboot, open socket, change file ownership, fork, exec, change UID,...) has an associated bit in privilege vector. Each process has associated permitted, inheritable, effective and limit privilege vectors. These vectors are calculated by kernel during fork and exec. Only few bit operations are allowed. Process can control all privileges passed to his children. This concept fully replaces concept of universal user UID=0. If enabled for all processes, UID=0 is no longer privileged user, but all privileges are determined from privileged vectors of each process (e.g. Secure Solaris OS).

Advanced kernel features:
- virtual servers: processes may be split into isolated groups. Each group looks like separate OS with its own filesystem, users, processes, login, etc. They are safe, managed by different administrators, but share the same resources (memory, CPUs, network cards, etc.). This saves total server costs and improves security. (E.g. Solaris zones, BSD jails.)

- kernel debugging:
  1) developer can run kernel under debugger
  2) some kernels can securely trace themselves without additional overhead, enabling administrators to detect which process produces some packets, which user forced kernel to intensively swap, catch a race condition, profile any kernel function depending on arguments, etc. (E.g. Solaris DTrace)

UNIX network administration

Review:
- protocols, encapsulation:
  Ethernet, PPP, SLIP, PLIP
  IP, IPX, ARP, RARP
  TCP, UDP
  RPC, SSL
  TELNET, FTP, NFS, SSH, HTTP, DNS
- addresses: IP, MAC, port
- socket
- host name, domain name, resolver
- name services

Advanced kernel features: ***(cont.)***
- clustering: a number of servers may create a cluster. Cluster provides high availability (HA) for applications. Whenever a device fails, hot-spare device is used (network card, disk) or if application cannot work (= monitoring process does not response working state of service) then the application is restarted on another node, disk access is rerouted to the node and IP address of service associated with the application is moved to the new node. (e.g. Sun Cluster, BSD clusters)
Clusters may also enable running same applications on multiple nodes splitting load by parallel processing. In such case, application must support it. (e.g. Oracle Parallel Server on Sun Cluster).

1) IPv4 addresses
- IP address of network interface eth0:
```
ifconfig eth0 147.32.80.1 netmask 255.255.255.0 up
```
- other IP addresses of the same interface (IP-aliasing)
```
ifconfig eth0:1 10.0.0.5 netmask 255.255.0.0 up
```
- testing
```
ifconfig eth0
ping -n 147.32.80.2
```

2) ARP – no configuration needed
- list ARP cache:
```
arp -an
```
- test ARP responces (better then ping, cannot be blocked, but only on LAN)
```
arping -I eth0 147.32.80.42
```

3) routing
- add routing table rule
`route add -net 192.168.10.0/24 gw 147.32.80.1`
- delete route
`route del -net 192.168.10.0/24`
- show routing table
`route -n`

4) packet forwarding (act as router)
- enabling/disabling (1 enables, 0 disables)
Linux: `echo 1 > /proc/sys/net/ipv4/ip_forward`
Solaris: `ndd -set /dev/ip forwarding 1`
- state
Linux: `cat /proc/sys/net/ipv4/ip_forward`
Solaris: `ndd /dev/ip`

Portmapper (process `rpcbind`, `rpc.portmap`)
- server create a socket (syscall `socket()`)
- does not bind it to any port (formally server bind it to PORT_ANY), while kernel assigns any free port to that socket
- asks kernel to listen (syscall `listen()`)
- server gets port number from kernel by syscall `getsockname()`
- server creates another connection to local port 111, where portmapper is listening and ask it to register assigned port with service name and version
- when client has to connect, it connects first to remote port 111
- client asks portmapper to translate service name and version to port
- it closes connection to portmapper
- it creates a new connection to the remote port got from portmapper

- saves well-known ports, allows several versions of one service running
- can cooperate with inetd as well

5) name service resolving host names
- master NS switch is file /etc/nsswitch.conf
- hostname resolver is controlled by line starting with "hosts:" followed the names of real name services, e.g.
hosts: files dns nis ldap

DNS client configuration is in /etc/resolv.conf
```
domain felk.cvut.cz      # default DNS domain
nameserver 147.32.80.9 # IP of the first DNS name server
nameserver 147.32.16.1 # IP of alternative DNS name server
search felk.cvut.cz mojefirma.cz # domain search list = list of suffixes successively appended to any name which failed to translate by DNS
```

- testing:
`nslookup or dig`

Network statistics:
- commands `netstat`, `ifconfig`
- in Linux: directory `/proc/net`
- testing: arping, ping, traceroute, spray, dig, nslookup, ypcat, ...

Firewalls:
- stateless or statefull packet filters (PF)
- network address translation (NAT)
- application gateways (PROXY)

Firwall implementations:
- Linux iptables (PF+NAT)
- Darren Reed's opensource IPF (PF+NAT)
- Squid and Apache (HTTP+HTTPS+FTP PROXY)
- BIND named (DNS server and PROXY)

5) services
- service name to port mapping is in /etc/services (well-known ports only)
- depending on start method, network service may be provided either by
  - standalone process (daemon): the daemon is started by its start-script in /etc/init.d (SSH, HTTP, DNS, NFS, ...)
  - or by internet super-server inetd (telnet, rlogin, talk, rusers, spray...)

- depending on port allocation, network service may use
  - statically allocated port, i.e. well-known port (SSH, telnet, HTTP, DNS...)
  - dynamically assigned port by kernel, registered in portmapper – used by all RPC applications (NFS, NIS, spray, ...)

Terminal

- terminal is a device equipped by a display and keyboard which can display ASCII characters and interprets escape sequences (clear screen)
- historically, terminals were hardware devices connected by a serial line
- now terminals are mostly software emulated:
  - Linux kernel emulates a terminal on local graphic card and keyboard
  - Solaris kernel emulates a terminal on local screen and keyboard if no X-Window server is running
  - graphical application xterm emulates a terminal in X-Window
  - MS Windows native application putty or WinSSH emulates a terminal
- each such emulator does interpret its own escape sequences

Inetd
- reads its configuration /etc/inetd.conf (list of ports and associated programs)
- creates a number of sockets, binds them to that ports and listens on them
- whenever a new connection arrives, inetd writes a record to log file, forks and parent process closes incomming socket and waits for another connection, while child closes all passive sockets, dupplicates arrived socket to all stdin, stdout and stderr and executes program specified in inetd.conf
- execued program serves the request and terminates

- inetd is efficient if there are low frequency of new connections and real process startup takes short time
- benefit of inetd is also central point of logging all network activities and possibility to control access based on client IP address.
- inetd cannot be used if service server must keep state (SSH, mountd) – statefull services

- each terminal has an associated terminal device in the kernel:
  1) directly connected HW terminals
     /dev/ttyS0, /dev/ttyS1, ... - connected to Linux by serial lines
     /dev/term/a, /dev/term/b, - connected to Solaris by serial lines
  2) directly emulated by the kernel
     /dev/tty1, /dev/tty2, ... - Linux local text-mode virtual terminals
     /dev/console - Solaris local keyboard+screen
  3) connected remotely through any network protocol – pseudo terminals are allocated dynamically whenever a new connection arrives (SSH, telnet)
     /dev/pts/1, /dev/pts/2, ... - common UNIX pseudo terminal device
     /dev/ttyp1, /dev/ttyp2, ... - old-style UNIX pseudo terminals
  4) application-emulated terminals (like xterm) uses the same pseudoterminals like remote connections

- any process running from interactive session has associated a controlling terminal device, moreover, the first process of the session (login shell) has standard input, output and error redirected to that terminal device
- current controlling terminal device can be displayed by command `tty`

Input from terminal (keyboard)

- HW terminal sends ASCII character of the key
- CTRL + <key> has ASCII value equal to ASCII <key> minus 0x40
- terminal driver in the kernel interprets some ASCII values as sending signal, sending end-of-file to the reading process, etc.
- correct interpretation is set by command `stty`

stty eof CHAR - CHAR will send an end-of-file to the process (^D)
stty erase CHAR - CHAR will erase the lasty character typed (=backspace)
stty intr CHAR - CHAR will send SIGINTR to all associated processes (^C)
stty quit CHAR - CHAR will send SIGQUIT (^\)
stty start CHAR - CHAR will enable output (^Q)
stty stop CHAR - CHAR will disable (temporary block) the output (^S)
stty susp CHAR - CHAR will send SIGSTOP (^Z)

- current settings can be displayed by command `stty -a`

Output to terminal (screen)

- HW terminals accepts ASCII characters
- special actions are invoked by ESC-sequences (clear screen, move cursor, change font to bold, change color, beep, blink, inverse video, ...)
- each terminal can interpret different sequences (they often share common part)
- the definition of sequences each terminal can interpret is in termcap database
- termcap database is a text file /etc/termcap (each line is a terminal definition)
- termcap is compiled into binary form terminfo by terminfo compiler `tic`
- each terminal is compiled into separate file /usr/share/terminfo/x/xx000-yy

x/ - directory named by the first character of terminal name
xx000-yy - common form of terminal name
xx - manufacturer
000 - model
yy - version (-am auto-margins, -rv reverse-video, -w wide, ...)

- termcap database is used by
  - fullscreen application (text editors)
  - ncurses library (various application that use colors, text-based pseudo windows, cursor moving, ...)
  - others (clear)
- application reads environment variable TERM, finds the appropriate ESC sequence in terminfo database and writes the sequence to the standard output

`tput` – print capability value from terminfo

tput cup 23 4 - move cursor to position 23,4
tput cols - display the number of characters per line
tput clear - clear screen
tput reset - init the terminal into power-on state
tput blik - turn on blinking
tput bold - switch to bold font
tput rev - turn on reverse video

other codes – see man terminfo

Serial line manipulation

- display status: setserial -a device
- change status: setserial device option

setserial /dev/ttys0 baud_base 9600
setserial /dev/ttyS0 port 0x3e8
setserial /dev/ttyS0 irq 3

minicom – connects current terminal session to serial line (typed characters on keyboard are sent to serial line, received characters are displayed)

Serial line discipline

- use serial line as a physical point-to-point network connection

slattach - changes line discipline and creates a network device
ifconfig - assigns IP address
dip - modem dialer
pppd - modem dialer + line discipline control + network dev. initialization