Computer architectures

User interfaces

Általános
INFORMATIKAI
Tanszék

User (operators) interfaces,

- User Interface (UI), Command Language (CL), etc.
- Their purpose:
  - We can “handle" the system with these,
  - We can manipulate devices and files
  - We can start programs (create processes)
  - We can provide the input data and display the results
- Their primary languages are:
  - Command language - response language

(Other applications can also have special interfaces ...)

Usually they have two classes

- (Alphanumeric) Command Language Interface (CLI)
  - Older
  - More effective
  - Requires less resources
- Graphical User Interface (GUI)
  - This is the newer one
  - Comfortable, user-friendly
  - Greater resource demand

Shell interfaces

- The model:
  - Given an (alphanumeric) terminal (console and its driver)
  - Given a command interpreter (shell) process
    - Provides a ready signal (prompt) to the console,
    - Reads, interprets, transforms and executes a command (pipe, or command list).
  - Commands are requests, that are serviced by the service routines of the OS core (kernel), or independent processes

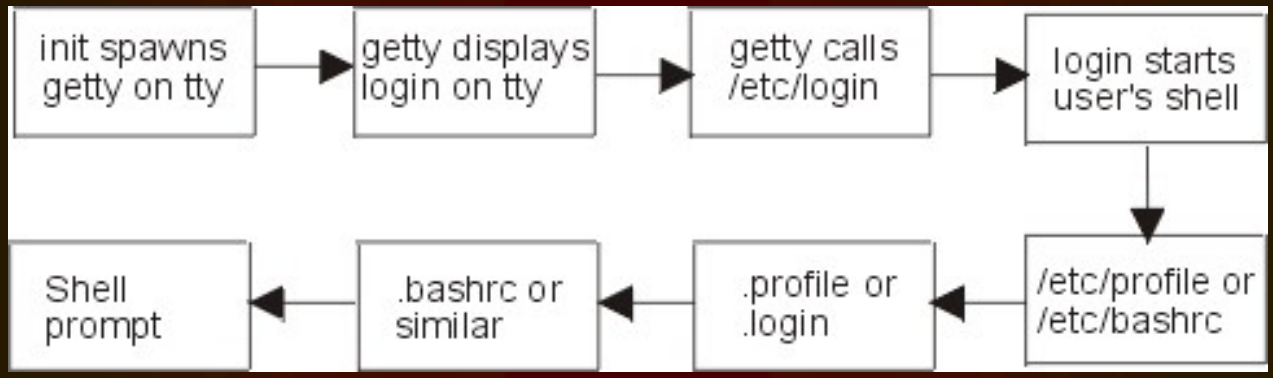
A command language, the Bourne shell

- The command language of the Unix OS
  - Unix (like) OSes are very common (many books, reviews)
  - Their command languages (especially the Bourne shell) are simple
- Why Bourne is the name of the shell? Stephen Bourne 1977

Name	Program	Usual prompt
Bourne shell	/bin/sh	$
Korn shell	/bin/ksh	$
C shell	/bin/csh	%
Bourne again shell	/bin/bash	$ Brian Fox1987

The starting of the Shell process

- Init starts the getty process
- getty process initiates the login prompt on the terminal
- login command check user credentials from the /etc/passwd
- getty starts the user shell process
- shell reads the system wide files /etc/profile, /etc/bashrc
- Shell reads user specific files .profile, .login
- Now it reads shell specific configuration file .bashrc
- Shell displays the default prompt



The shell process

- An independent entity, its identifier is the pid (process identification number)
- It runs the /bin/sh (or /bin/bash) program
- It has 3 open streams
  - The stdin (standard input) with descriptor 0, from which it reads commands, pipes and command lists.
  - The stdout (standard output) with descriptor 1, where the results are written.
  - The stderr (standard error output) with descriptor 2, where the error messages are written.
- The “open streams” are connected to devices “as usual”.

How the shell process works

- It writes the prompt signal (prompt) to the stdout, indicating that it is waiting for a command, a pipe, or a commands list.
- On the stdin it reads the command, pipe, or command list
  - analyzes, interprets and
  - transforms, then executes, or make it to be executed.
- The result of the execution is sent to stdout, or written to the stderr channel. Finally it produces a return value.

The return value

- Can be **normal** (**0**),
- or it could be **not be normal** (not 0), in case of several reasons e.g.
  - there is something wrong
  - there is no error, but there is a semantic problem.

  (E.g. grep filter does not find a pattern match, or test command “test” is not true (logical true/false).)
- We will use the return value in the program control.

The concept of command

- **A sequence of words delimited by white characters**
  - **the first word is the name of the command,**
  - **the subsequenting words are arguments.**
- **The sh reads, interprets, transforms, and executes**
  - **himself (internal command),**
  - **or in a child in process (external command)**

In both cases there is a return value!

There are standard data streams!

Whitespace is any character or series of whitespace characters that represent horizontal or vertical space in typography .
HT, Horizontal Tab , LF, Line feed , VT, Vertical Tab , FF, Form feed , CR, Carriage return , Separator, space , NEL, Next line

An example command

> find . -name a.c -print

where the numbering of words are:

0 1 2 3 4

That is, the above command consists of 5 words.

Note that the shell prompt is not part of the command! It does not require input, its output (including possible error messages) goes to the screen. What is the return value? And here?

> find . -name a.c -print >myfile.txt

You have to learn commands...

- The most important document is the on-line manual, the man

> man [-options] [section] tab

- Unfortunately, there are no "jokers", the names of the important commands must be known exactly!
- It is worth making a "command card" with the names of the important commands and their brief descriptions.

Commands: editors

- ed row oriented
- vi (vim) screen oriented
- mcedit screen oriented
- pico, nano simple, in many places (vt100 required)
- joe
- etc.

Commands: printouts

- cat concatenates, to stdout
- pr prints to stdout
- head first lines of a file to stdout
- tail last lines of a file to stdout
- more, less filter that folds into pages
- od octal dump

Commands: related to lists

- ls directory content list (instead of dir)
- mkdir creating a directory
- rmdir delete a directory
- cd change default directory
- pwd query default directory
- chmod file protection mask change (Not only for dir)
- chown file owner change (Not only for directories)
- file query the file type (Not only for directories)

Commands: copies, moves

- cp copy
- mv move (also instead of rename!)
- ln (link) "links"
- rm (unlink) deletes "link", remove: file deletion

- find searches for a file in a tree (complicated, but very useful!)

Commands: status queries

- ps list of processes
- file, ls, pwd see previous slides
- date dquery the date and time
- who, w, rwho, rusers who is logged in?
- rup which systems are live?
- top resource usage peaks
- osview, vmstat resource usage
- last last logins
- uptime How long has the system running?

Commands: status queries 2

- finger — who's who
- passwd — password setting
- chsh — startup shell setup
- chfn — name, etc. setting

(change finger information /etc/passwd)

- ldapsearch — LDAP database query
- xhost — allow X11 work
- set — query the environment
- du, df, quota — disk, file usage

Commands: process start, control

- sh, csh, ksh, tcsh, bash — launch the shell
- exec — start process
- kill — "killing" a process, sending a signal
- anesthetize the sleep process
- wait process waiting
- start the at process at a given time
- nohup don't kill him on exit
- body expression testing

Commands: process start, control 2

- expr — evaluate expression
- if, case, for, do while — control structures
- break, continue — control structures
- echo — rewriting arguments
  (something surprisingly useful)
- mplayer — video player
- xmms, aumix — audio playback

Commands: communication

- ssh, telnet, rlogin, rsh — connection establishment
- rwho, rusers, finger — status queries
- write — message to consoles
- talk, xtalk — interactive "conversation"
- mail, mutt, pine, mozilla-thunderbird — e-mail
- ftp, scp — file transfer
- lynx, w3m, firefox, netscape — WWW browser

Commands: useful filters

- grep — pattern finder
- awk, nawk — pattern search processor
- wc — line, word, character counter
- sed — stream editor
- cut — field cutter
- tail, head, more — a way for write out
- sort — sorter

Commands: let's learn

- man — on-line manual page query
- apropos — keyword in manual (if any)
- whereis — where is a command
- whatis — man page description
- xman — X11 manual (graphic)

Repeat: the definition of command

- A string of words delimited by white characters
  - first word is the name of the command,
  - other words are arguments.
- sh reads, interprets, transforms, executes
  - in himself (internal command),
  - in a child in process (external command)

In both cases there is a return value!

There are standard data streams!

The concept of pipeline

- **The pipe is a line of commands connected with | operator:**
- **left command | right command**
- **Semantics: the left command is executed then, its standard output is expressed in a pipe, then the command right is executed, which's standard input of is gained from this pipe.**
- **The return value of the pipe: is the return value of the command right.**
- **The command is a degenerated pipe. Example:**

> ypcat password | grep kovacs

The command list

- **A series of pipelines connected by a list operator:**

left pipe op right pipe

List operators :

```
&&    ||              # higher precedence but lower than |
&     ;       \n      # lower precedence
```

The semantics :

; \n serial execution of pipes

& asynchronous execution (left pipe in the background)

&& continues the list if the left pipe has a normal return value

|| continues the list if the pipe has no normal return value

Command lists

- The return value of the list is the return value of the last pipe.
- The return value of a pipe running in the background can be specially handled.
- The pipe is a degenerate list (from now if we write a list, we can write a pipe and even a command!)
- && and || in operator lists, we can see first time the meaning of the return value! They influence the process of control!

Examples

> cd here && rm junk # delete only if ...

> ls here || cp something here # if here is not exists, creates it

Let's explain this!

> (mv a tmp && mv ba) && mv tmp b

Examples

- Tun it "in the background".

```
> myprog  1  2  &
125                          <- PID!
>
```

- What's the difference?

```
> echo something echo something
something echo something
```

```
> echo something; echo something
something
something
```

Redirection of data streams

- Before the list/command is executed, sh looks for **redirection operator** > >> < in words (before the words). (The << is special!)
- If it finds such, separate process(es) are created, the data streams **are mapped to (from) files**, and then the list/command is executed on them. (The pipe is also a separate process.)
- It passes the "leftover" arguments to the separate process(es) .

The redirect operators

```
< file            # let the file be the stdin
> file            # let the file be the stdout , rewrite
>> file           # let the file be the stdout , append
<< [-] so far     # here document , embedded input
```

- Example:

```
> mypr < from here > to there   first   second
    0                             1       2

> exec > outfile 2> errorfile            # in script if necessary
  # redirections are effective for the current shell
  # if the redirection is OK, the return value is 0 (normal)
  # if there is a redirect error, the return value is 1
```

The redirect operators

<< [-] so far # here document , embedded input

Example: (! indicates the end of data lines)

a.script

```
grep this <<!
the first line has this
line 2, not in this line
line 3
!
echo 'what's up?'
```

This is how we can start it, and the following is the result:

```
$ a.script
the first line has this
what's up?
$
```

Filename exposition (Substitution)

- **Metacharacters used in arguments (among them the wildcards: * ? [ ]) are handled specially before executing the list/command by the shell.**
- **It considers wildcards *as* a pattern .**
- **The patterns are substituted into an alphabetically ordered list of filenames that match the directory namespace.**
- **Only then the command/list is executed.**

The matching

- A "regular" character matches themselves...
- The ? matches any single character.
- The * matches any number of characters .
- [...] matches a single, enclosed character.
- The [! . ..] matches single, any, except the character after the !
- etc., check it out!

Examples

- Let's say there are 4 files in the current directory:

  a abc abc.d xyz

> ls * # ⇒ ls abc abc.d xyz

> ls a* # ⇒ ls abc abc.d

> ls [a]?? # ⇒ ls abc

> ls [!a]?? # ⇒ ls xyz

Note that the substitution takes place first, only then the ls command is executed!

Compare: > rm * and DOS> DEL * Be careful: > echo *

abc abc.d xyz

Variable definition and substitution

- There are variables defined for the shell (it has also constants)
- We can also define variables (and use them within their scope)
- **The definition:** variable = text string
- **The substitution:** $ variable
- E.g.

```
var= cigar
echo 'I lighted my ' $var
```

Deactivation of metacharacters

- Metacharacters (operators, separators, wildcards, redirectors, etc.) can be quotated if necessary
- **Deactivate a single character** : \ spec _character
- **Deactivate multiple characters** :

  'text chain'

  "text chain" # the variable substitution remains
- E.g.

  echo "I lighted my $ var"

Deactivation of metacharacters

- E.g.

```
# a - sh variable, $a - explanation
> a=abc                 # gets a value
> echo '$a'             # the $ operator is deactivated
$a
> echo "$a"             # the $ operator is effective
abc
> echo "$\a"            # \a means tha a character
$\a
>
```

The shell

- We only used one of the two meanings of the term shell: the shell is a command interpreter process
- Another meaning (the shell is also a programming language) will studied later.
- The deactivation of metacharacters, additional "substitutions", shell data structures, etc. will discussed later.
- Comment: The command-line user interface is powerful, but inconvenient.

The motivation to move forward

- How could the handling (contact with the computer system) be made more friendly?
  - Users with heterogeneous knowledge, different needs
    - The needs of novice users
      - few, simple commands,
      - be safe (can't cause much damage),
      - detailed and contextual help,
      - even for the expense of performance.
  - There were alphanumeric menu interfaces...
    - it's easier to choose from text menus...
  - Instead of hard-to-remember commands
    - selecting icons, menu items,
    - graphics!

Graphical user interfaces

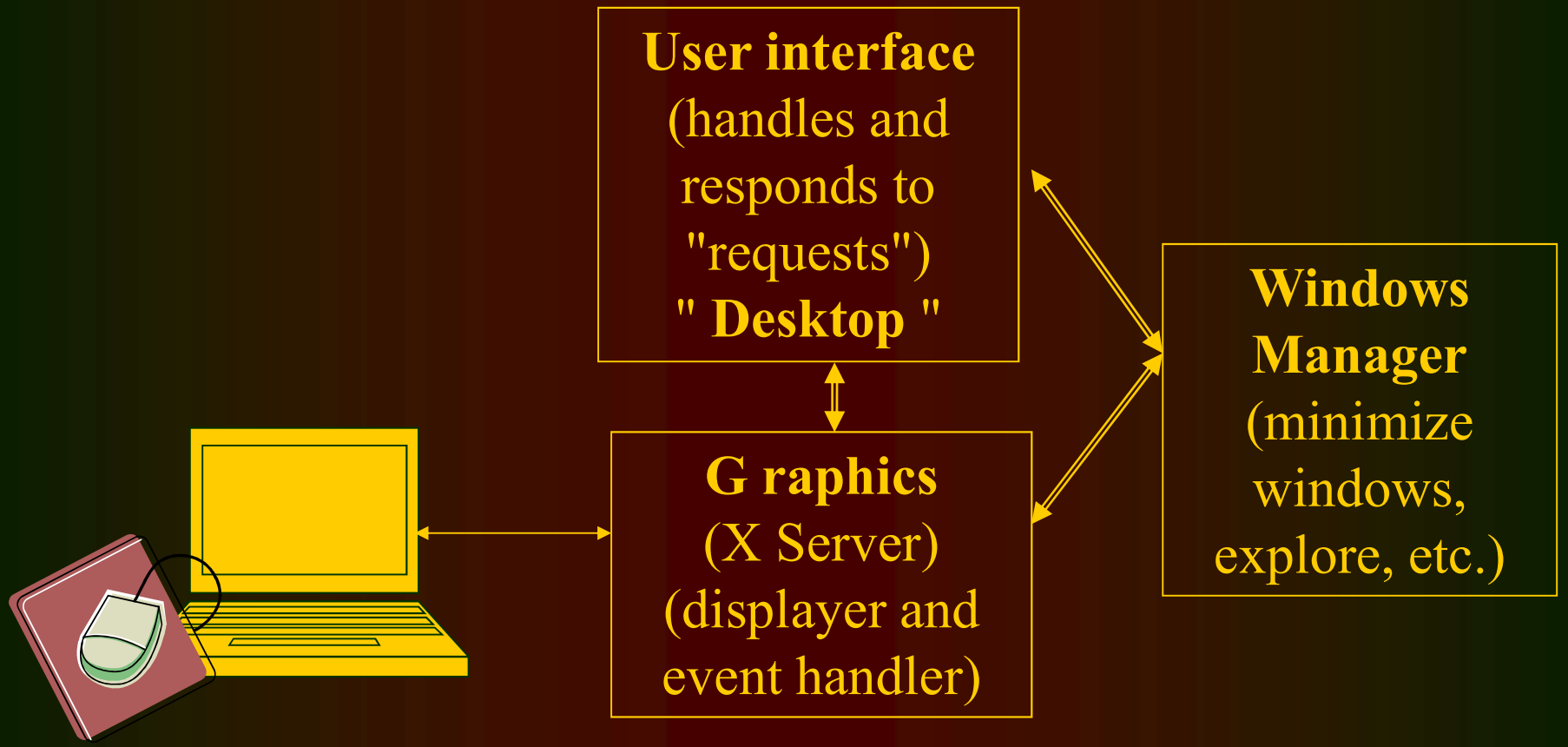
- **Basic idea : we have a *desktop***
- **on it we have:**
  - **tool shelf (drawer) - menus,**
  - **documents,**
  - **folders, documents in them, filing cabinets;**
  - **tools (trash bin, paper shredder), etc.**
- **It appears stylized.**
- **Command language: select, click, double-click, drag, type text, etc.**
- **Response language: icons, windows, menus, etc.**

Who was the first?

- Battle for first place: Xerox versus Apple
  - Xerox PARC : Smalltalk prototype , later ALTO STAR
    - These were not commercial products...
  - Apple : Macs Hi , Lisa
    - Apple employed Xerox PARC researchers,
    - they developed own GUI.
    - It was the first commercial GUI.
- Apple versus Microsoft
  - Initially, MS was Apple's app writer (licensed for Macs for UI use) and was a writer of IBM system software...
  - When he fell out with IBM, he released Windows 1.0
    - this is a GUI on top of MSDOS , with "stolen" elements... MS lost a lawsuit

The model

- **Usually multiple processes with different functions**
  Display/Session Manager – X Server-Login-WM-Desktop



CLI and GUI comparison

- What do you need for CLI?
  - Connection builder (ssh – init - tty)
  - Session creator (ssh – login)
  - CLI (ssh - bash)
- What do you need for the GUI?
  - Connection and session builder: Display Session Manager
  - Window Manager (for iconization, window movement)
  - GUI (Desktop and X server/workplace manager)

A typical X workstation

- **Display/Session Manager** starts at system startup.
- This also starts the **X server** on the machine.
- The Display/Session Manager provides the "login" windows (**login-password**) and helps establish a controlled session.
- When a successful session is established it starts
  - **Window Manager**,
  - **Desktop**. We “handle" the machine with it. (User Interface)

Your Windows Desktop

- **Objects: uniformly managed entities**
  - they have **properties** (they depend on the object),
  - **operations** can be performed on them.
- E.g. hard disk object,
  - property: capacity, free capacity, etc.
  - operations: formatting, error checking, etc.
- E.g. file object
  - property: name, size, type, date of creation, etc.
  - operations: copy, delete, rename, etc.

The Windows Desktop

- **Hierarchy of object:**
- **if an object contains another object, it is a "folder".**
- **All directories are folders, but not all folders are directories!**
- **The main folder is the desktop itself. It is the whole screen.**
- **The main elements of the desktop:**

–start button, **–Taskbar (TaskBar),** **–(Command) icons**

Your Windows Desktop

- **Command launch can take various forms**
  - **double click on its icon in any folder,**
  - **double click on the file icon to which the management program is associated (e.g. Word starts on doc),**
  - **drag file icon to application icon (e.g. doc file to Word),**
  - **Start button by clicking from its submenus ,**
  - **Start button with Run menu,**
  - **From an application using OLE (Object Linking and Embedding) (e.g. by clicking on an Excel table embedded in a Word document)**

Data transfer between applications

- Using the OLE (possibly also as a link),
- Through the cutting board.

- Also by dragging within some applications (copy, move, possibly as a link)

The CDE

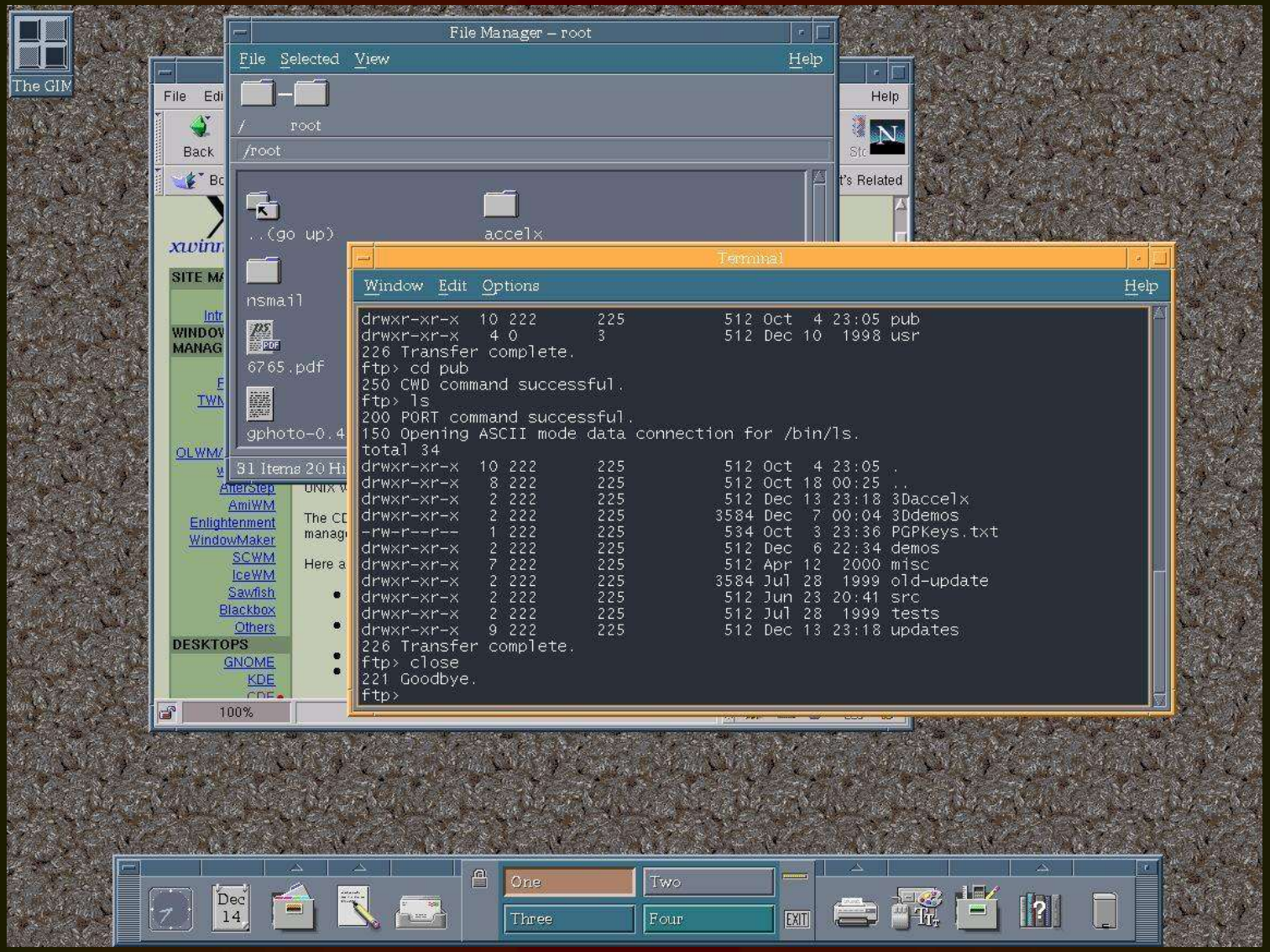
Free software on August 6, 2012, under the GNU Lesser General Public License

- Common Desktop Environment
  - Integrated, standardized, consistent, configurable,
  - platform-neutral operator/user interface in accordance with the open system principle.
  - We can manage it with the help of graphic desktops.
- Participants of the CDE project
  - submitted the best of their knowledge and technological ideas,
  - they created a functionality set (there was a huge debate about the set)
  - this has been implemented on all (many) platforms.
- X11, MOTIF

Elements of CDE

- The work table (desktop), which contains
  - The Front Panel
    - set of icons and menus for quick launches.
  - Standard file manager
    - direct manipulations on the device and file systems
  - Application manager
    - Can be started from the Front Panel, specifically for managing applications (installed executable program groups).
  - Multiple workspaces (Virtual Workspace)
    - They provide an environment for different work (e.g. normal office activity, work on a project, play, entertainment, etc.) within one window.
  - Useful "tools"
    - e.g. e-mail, calendar, calculator, editor, terminal emulator, helper, icon editor, style manager, etc.

Elements of CDE



CDE features

- Comfortable handling
  - OO characteristics (functionalities assigned to device/file type);
  - Various data entry options within and between applications
    - moving, copying, linking, creating, deleting, sharing
    - direct transfer (drag), or in several "steps"
    - indirect transfer (primary transfer: select+target+TRANSFER; quick transfer for input; clipboard transfer)
  - ToolTalk protocol
    - It is a messaging service between independent (independently developed) applications
    - "Packed" objects can be transferred

CDE features

- Sophisticated relationship manager
  - Its most important feature:
    - the running state of the programs can be preserved when exiting,
    - work can be continued upon entry.
    - You can also start with the default state.
- The set of tools is very rich
  - Mostly Sun and Novell additives, but everyone gave everything.
  - Almost everything an average user might need
    - mail, calc, text editor, file manager, printer manager, environment-sensitive help, style manager (setting background, colors, etc.);
    - e.g. calendars for each person, but they can be "projected to each other": to choose a common free time.

CDE features

- Application development components
  - For Motif-style application development, a graphic set from which we can build (primarily we compile the GUI), and
  - the C source code is generated (this is further edited, supplemented with the codes according to the application logic, and linked in reverse, the application is quickly ready)

Are you spreading? Is it not spreading?

- Each system vendor also has its own GUI
- In 1996, we could have thought it would be hugely popular.
- What was the reason for the slow spread?
- My personal opinion: WWW browsers are the reason!
  - They also provide a variety of functionalities, on a "unified" interface.
  - the WEB explosion stopped the spread of CDE.
- Today's suppliers ship with CDE
  - AIX, Solaris, HP-UX, Digital Unix, Linuxes

Computer architectures

User interfaces

End

Általános
INFORMATIKAI
Tanszék