MABEL Technology

Tech The Best

Programming Linux

Liran Ben Haim
liran@mabel-tech.com

MABEL Technology
Tech The Best

Rights to Copy

creative commons
COMMONS DEED

- **Attribution – ShareAlike 2.0**
- **You are free**
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

Attribution. You must give the original author credit.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: http://creativecommons.org/licenses/by-sa/2.0/legalcode

- **This kit contains work by the following authors:**
- © Copyright 2004-2009
  Michael Opdenacker /Free Electrons
  michael@free-electrons.com
  http://www.free-electrons.com
- © Copyright 2003-2006
  Oron Peled
  oron@actcom.co.il
  http://www.actcom.co.il/~oron
- © Copyright 2004–2008
  Codefidence ltd.
  info@codefidence.com
  http://www.codefidence.com
- © Copyright 2009–2010
  Bina ltd.
  info@bna.co.il
  http://www.bna.co.il

MABEL Technology
Tech The Best

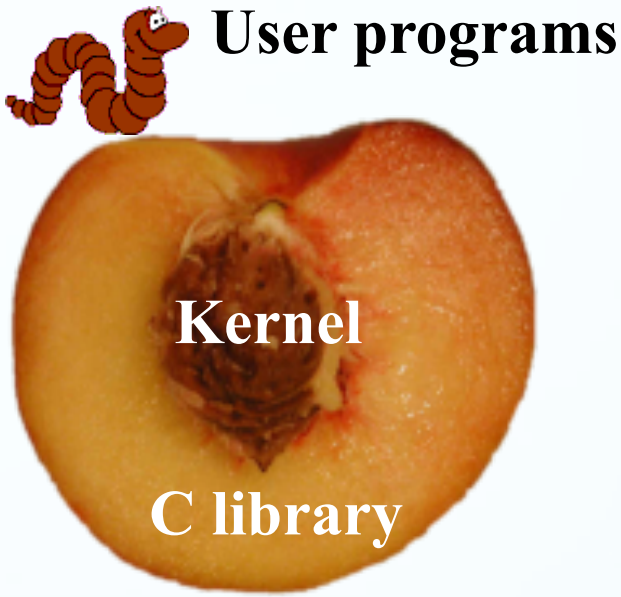
What is Linux?

- Linux is a kernel that implements the POSIX and Single Unix Specification standards which is developed as an open-source project.
  - Usually when one talks of “installing Linux”, one is referring to a Linux distribution.
- A distribution is a combination of Linux and other programs and library that form an operating system.
  - There exists many such distribution for various purposes, from high-end servers to embedded systems.
  - They all share the same interface, thanks to the LSB standard.
- Linux runs on 21 platforms and supports implementations ranging from ccNUMA super clusters to cellular phones and micro controllers.
- Linux is 18 years old, but is based on the 40 years old Unix design philosophy.

MABEL Technology
Tech The Best

Layers in a Linux System

- Kernel
- Kernel Modules
- C library
- System libraries
- Application libraries
- User programs



MABEL Technology
Tech The Best

Embedded Linux Driver Development

The Basics

Everything is a file

MABEL Technology
Tech The Best

Everything is a File

Almost everything in Unix is a file!

- Regular files
- Directories
  Directories are just files listing a set of files
- Symbolic links
  Files referring to the name of another file
- Devices and peripherals
  Read and write from devices as with regular files
- Pipes
  Used to cascade programs
  `cat *.log | grep error`
- Sockets
  Inter process communication

MABEL Technology
Tech The Best

Filenames

- File name features since the beginning of Unix:
- Case sensitive.
- No obvious length limit.
- Can contain any character (including whitespace, except `/`).
  File types stored in the file (“magic numbers”).
  File name extensions not needed and not interpreted. Just used for user convenience.
- File name examples:

```
README          .bashrc                 Windows
Buglist
index.htm       index.html          index.html.old
```

MABEL Technology
Tech The Best

File Paths

- A *path* is a sequence of nested directories with a file or directory at the end, separated by the `/` character.

- Relative path:
  `documents/fun/microsoft_jokes.html`
  Relative to the current directory

- Absolute path:
  `/home/bill/bugs/crash9402031614568`

- `/` : *root directory*.
  Start of absolute paths for all files on the system (even for files on removable devices or network shared).

MABEL Technology
Tech The Best

GNU/Linux Filesystem Structure (1)

Not imposed by the system. Can vary from one system to the other, even between two GNU/Linux installations!

`/`	Root directory
`/bin/`	Basic, essential system commands
`/boot/`	Kernel images, initrd and configuration files
`/dev/`	Files representing devices
	`/dev/hda`: first IDE hard disk
`/etc/`	System configuration files
`/home/`	User directories
`/lib/`	Basic system shared libraries

MABEL Technology
Tech The Best

GNU/Linux Filesystem Structure (2)

`/lost+found` Corrupt files the system tried to recover

`/media/` Mounted file systems (/mnt)

`/media/usbdisk/`, `/media/windows/` ...

`/opt/` Specific tools installed by the sysadmin

`/usr/local/` often used instead

`/proc/` Access to system information

`/proc/cpuinfo`, `/proc/version` ...

`/root/` root user home directory

`/sbin/` Administrator-only commands

`/sys/` System and device controls

(cpu frequency, device power, etc.)

MABEL Technology
Tech The Best

GNU/Linux Filesystem Structure (3)

`/tmp/` Temporary files

`/usr/` Regular user tools (not essential to the system)
`/usr/bin/`, `/usr/lib/`, `/usr/sbin`...

`/usr/local/` Specific software installed by the sysadmin (often preferred to `/opt/`)

`/var/` Data used by the system or system servers
`/var/log/`, `/var/spool/mail` (incoming mail), `/var/spool/lpd` (print jobs)...

MABEL Technology

Tech The Best

The Unix and GNU/Linux Command-Line

Shells and File Handling

MABEL Technology
Tech The Best

Command-Line Interpreters

- Shells: tools to execute user commands.
- Called “shells” because they hide the details on the underlying operating system under the shell's surface.
- Commands are entered using a text terminal: either a window in a graphical environment, or a text-only console.
- Results are also displayed on the terminal. No graphics are needed at all.
- Shells can be scripted: provide all the resources to write complex programs (variable, conditionals, iterations...)

MABEL Technology
Tech The Best

Command Help

- Some Unix commands and most GNU/Linux commands offer at least one help argument:

- `-h`
  (`-` is mostly used to introduce 1-character options)

- `--help`
  (`--` is always used to introduce the corresponding "long" option name, which makes scripts easier to understand)

You also often get a short summary of options when you input an invalid argument.

MABEL Technology
Tech The Best

Manual Pages

- `man [section] <keyword>`
  - Displays one or several manual pages for `<keyword>` from optional `[section]`.
    - `man fork`
      - Man page of the *fork()* system call
    - `man fstab`
      - Man page of the fstab configuration file
    - `man printf`
      - Man of *printf()* shell command
    - `man 3 printf`
      - Man of *printf()* library function
    - `man -k [keyword]`
      - `Search keyword in all man pages`

MABEL Technology
Tech The Best

ls Command

Lists the files in the current directory, in alphanumeric order, except files starting with the "`.`" character.

- `ls -a` (all)
  Lists all the files (including `.*` files)
- `ls -l` (long)
  Long listing (type, date, size, owner, permissions)
- `ls -t` (time)
  Lists the most recent files first
- `ls -S` (size)
  Lists the biggest files first
- `ls -r` (reverse)
  Reverses the sort order
- `ls -ltr` (options can be combined)
  Long listing, most recent files at the end

MABEL Technology
Tech The Best

Filename Pattern Substitutions

- Better introduced by examples:

- `ls *txt`
  The shell first replaces `*txt` by all the file and directory names ending by `txt` (including `.txt`), except those starting with `.`, and then executes the `ls` command line.

- `ls -d .*`
  Lists all the files and directories starting with `.`
  `-d` tells `ls` not to display the contents of directories.

- `ls ?.log`
  Lists all the files which names start by 1 character and end by `.log`

MABEL Technology
Tech The Best

Special Directories

`./`

- The current directory. Useful for commands taking a directory argument. Also sometimes useful to run commands in the current directory (see later).
- So `./readme.txt` and `readme.txt` are equivalent.

`../`

- The parent (enclosing) directory. Always belongs to the `.` directory (see `ls -a`). Only reference to the parent directory.
- Typical usage:
  `cd ..`

MABEL Technology
Tech The Best

The cd and pwd Commands

- `cd <dir>`
  Change the current directory to `<dir>`.

- `pwd`
  Displays the current directory ("working directory").

MABEL Technology
Tech The Best

The cp Command

- `cp <source_file> <target_file>`
  Copies the source file to the target.
- `cp file1 file2 file3 ... dir`
  Copies the files to the target directory (last argument).
- `cp -i` (interactive)
  Asks for user confirmation if the target file already exists
- `cp -r <source_dir> <target_dir>` (recursive)
  Copies the whole directory.

MABEL Technology
Tech The Best

The mv and rm Commands

- `mv <old_name> <new_name>` (move)
  Renames the given file or directory.
- `mv -i` (interactive)
  If the new file already exits, asks for user confirm
- `rm file1 file2 file3 ...` (remove)
  Removes the given files.
- `rm -i` (interactive)
  Always ask for user confirm.
- `rm -r dir1 dir2 dir3` (recursive)
  Removes the given directories with all their contents.

MABEL Technology
Tech The Best

Creating and Removing Directories

- `mkdir dir1 dir2 dir3 ...` (make dir)
  Creates directories with the given names.

- `rmdir dir1 dir2 dir3 ...` (remove dir)
  Removes the given directories
  Safe: only works when directories and empty.
  Alternative: `rm -r` (doesn't need empty directories).

MABEL Technology
Tech The Best

Displaying File Contents

- Several ways of displaying the contents of files.

- `cat file1 file2 file3 ...` (concatenate)
  Concatenates and outputs the contents of the given files.

- `more file1 file2 file3 ...`
  After each page, asks the user to hit a key to continue.
  Can also jump to the first occurrence of a keyword
  (`/` command).

- `less file1 file2 file3 ...`
  Does more than `more` with less.
  Doesn't read the whole file before starting.
  Supports backward movement in the file (`?` command).

MABEL Technology

Tech The Best

Task Control

MABEL Technology
Tech The Best

Full Control Over Tasks

- Since the beginning, Unix supports true preemptive multitasking.
- Ability to run many tasks in parallel, and abort them even if they corrupt their own state and data.
- Ability to choose which programs you run.
- Ability to choose which input your programs takes, and where their output goes.

MABEL Technology
Tech The Best

Processes

- "Everything in Unix is a file
Everything in Unix that is not a file is a process"

- Processes

  - Instances of a running programs

  - Several instances of the same program can run at the same time

  - Data associated to processes:
Open files, allocated memory, process id, parent, priority, state...

MABEL Technology
Tech The Best

Running Jobs in Background

- Same usage throughout all the shells.
- Useful:
  - For command line jobs which output can be examined later, especially for time consuming ones.
  - To start graphical applications from the command line and then continue with the mouse.
- Starting a task: add `&` at the end of your line:

```
find_prince_charming --cute --clever --rich
  &
```

MABEL Technology
Tech The Best

Background Job Control

- `jobs`
  Returns the list of background jobs from the same shell

- `fg`
  `fg %<n>`
  Puts the last / nth background job in foreground mode

- Moving the current task in background mode:
  `[Ctrl] Z`
  `bg`

- `kill %<n>`
  Aborts the nth job.

MABEL Technology
Tech The Best

Listing All Processes

- … whatever shell, script or process they are started from

- `ps aux (procps version)` OR `ps (Busybox version)`
  Lists all the processes running on the system

- ```
  ps aux
  USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
  bart      3039  0.0  0.2   5916 1380 pts/2    S    14:35   0:00 /bin/bash
  bart      3134  0.0  0.2   5388 1380 pts/3    S    14:36   0:00 /bin/bash
  bart      3190  0.0  0.2   6368 1360 pts/4    S    14:37   0:00 /bin/bash
  bart      3416  0.0  0.0      0    0 pts/2    R    15:07   0:00 [bash] ...
  ```

- PID: Process id
  VSZ: Virtual process size (code + data + stack)
  RSS: Process resident size: number of KB currently in RAM
  TTY: Terminal
  STAT: Status: R (Runnable), S (Sleep), D (Uninterrupted sleep), Z (Zombie), T(Traced)
  ```

MABEL Technology
Tech The Best

Live Process Activity

- `top` – Displays most important processes, sorted by cpu percentage

```
top - 15:44:33 up  1:11,  5 users,  load average: 0.98, 0.61, 0.59
   Tasks:  81 total,   5 running,  76 sleeping,   0 stopped,   0 zombie
   Cpu(s): 92.7% us,  5.3% sy,  0.0% ni,  0.0% id,  1.7% wa,  0.3% hi,  0.0% si
   Mem:    515344k total,   512384k used,     2960k free,    20464k buffers
   Swap:  1044184k total,        0k used,  1044184k free,   277660k cached

     PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    3809 jdoe      25   0  6256 3932 1312 R 93.8  0.8   0:21.49 bunzip2
    2769 root      16   0  157m  80m  90m R  2.7 16.0   5:21.01 X
    3006 jdoe      15   0 30928  15m  27m S  0.3  3.0   0:22.40 kdeinit
    3008 jdoe      16   0  5624  892 4468 S  0.3  0.2   0:06.59 autorun
    3034 jdoe      15   0 26764  12m  24m S  0.3  2.5   0:12.68 kscd
    3810 jdoe      16   0  2892  916 1620 R  0.3  0.2   0:00.06 top
```

- You can change the sorting order by typing `M`: Memory usage, `P`: %CPU, `T`: Time.
- You can kill a task by typing `k` and the process id.

MABEL Technology
Tech The Best

Killing Processes (1)

- `kill <pids>`
  Sends a termination signal (SIGTERM) to the given processes. Lets processes save data and exit by themselves. Should be used first. Example:
  `kill 3039 3134 3190 3416`

- `kill -9 <pids>`
  Sends an immediate termination signal (SIGKILL). The system itself terminates the processes. Useful when a process is really stuck

- `killall [-<signal>] <command>`
  Kills all the jobs running `<command>`. Example:
  `killall bash`

MABEL Technology
Tech The Best

Environment Variables

- Shells let the user define *variables*.
  They can be reused in shell commands.
  Convention: lower case names

- You can also define *environment variables*:
  variables that are also visible within scripts or executables called from the shell.
  Convention: upper case names.

- `env`
  Lists all defined environment variables and their value.

MABEL Technology
Tech The Best

Shell Variables Examples

- Shell variables (bash)

```
projdir=/home/marshall/coolstuff
ls -la $projdir; cd $projdir
```

Environment variables (bash)

```
cd $HOME
```

```
export DEBUG=1
./find_extraterrestrial_life
```

(displays debug information if `DEBUG` is set)

MABEL Technology
Tech The Best

File Ownership

- `chown -R sco /home/linux/src` (`-R`: recursive)
  Makes user `sco` the new owner of all the files in `/home/linux/src`.

- `chgrp -R empire /home/askywalker`
  Makes `empire` the new group of everything in `/home/askywalker`.

- `chown -R borg:aliens usss_entreprise/`
  `chown` can be used to change the owner and group at the same time.

MABEL Technology
Tech The Best

File Access Rights

Use `ls -l` to check file access rights

- 3 types of access rights
  - Read access (`r`)
  - Write access (`w`)
  - Execute rights (`x`)
- 3 types of access levels
  - User (`u`): for the owner of the file
  - Group (`g`): each file also has a "group" attribute, corresponding to a given list of users
  - Others (`o`): for all other users

MABEL Technology
Tech The Best

Access Right Constraints

- `x` without `r` is legal but is useless...
  You have to be able to read a file to execute it.
- Both `r` and `x` permissions needed for directories:
  `x` to enter, `r` to list its contents.
- You can't rename, remove, copy files in a directory if you don't have `w` access to this directory.
- If you have `w` access to a directory, you CAN remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without `w` access to it.

MABEL Technology
Tech The Best

Access Rights Examples

- `-rw-r--r--`
  Readable and writable for file owner, only readable for others

- `-rw-r-----`
  Readable and writable for file owner, only readable for users belonging to the file group.

- `drwx------`
  Directory only accessible by its owner.

- `-------r-x`
  File executable by others but neither by your friends nor by yourself. Nice protections for a trap...

MABEL Technology
Tech The Best

chmod: Changing Permissions

- `chmod <permissions> <files>`
  2 formats for permissions:

- Symbolic format. Easy to understand by examples:
  `chmod go+r`: add read permissions to group and others.
  `chmod u-w`: remove write permissions from user.
  `chmod a-x`: (`a`: all) remove execute permission from all.

- Or octal format (abc):
  `a,b,c = r*4+w*2+x` (`r`, `w`, `x`: booleans)
  Example: `chmod 644 <file>`
  (`rw` for `u`, `r` for `g` and `o`)

MABEL Technology
Tech The Best

Standard Output

- More about command output.
- All the commands outputting text on your terminal do it by writing to their *standard output*.
- Standard output can be written (redirected) to a file using the > symbol
- Standard output can be appended to an existing file using the >> symbol

MABEL Technology
Tech The Best

Standard Output Redirection

- `ls ~saddam/* > ~gwb/weapons_mass_destruction.txt`
- `cat obiwan_kenobi.txt > starwars_biographies.txt`
  `cat han_solo.txt >> starwars_biographies.txt`
- `echo "README: No such file or directory" > README`
  Useful way of creating a file without a text editor.
  Nice Unix joke too in this case.

MABEL Technology
Tech The Best

Standard Input

- More about command input:

- Lots of commands, when not given input arguments, can take their input from *standard input*.

- ```
  sort
  windows
  linux
  [Ctrl][D]
  linux
  windows
  ```

  `sort` takes its input from the standard input: in this case, what you type in the terminal (ended by `[Ctrl][D]`)

- `sort < participants.txt`
  The standard input of sort is taken from the given file.
  ```

MABEL Technology
Tech The Best

Pipes

- Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.
- Examples
  - `ls *.txt | less`
  - `cat *.log | grep -i error | sort`
  - `grep -ri error . | grep -v “ignored” | sort -u \ > serious_errors.log`
  - `cat /home/*/homework.txt | sort | more`
- This one of the most powerful features in Unix shells!
- Grep searches file, sort sorts them. More info at the appendixes.

MABEL Technology
Tech The Best

Standard Error

- Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- Standard error can be redirected through `2>` or `2>>`
- Example:
```
cat f1 f2 nofile > newfile 2> errfile
```
- Note: `1` is the descriptor for standard output, so `1>` is equivalent to `>`.
- Can redirect both standard output and standard error to the same file using `&>` :
```
cat f1 f2 nofile &> wholefile
```

MABEL Technology
Tech The Best

Special Devices (1)

- Device files with a special behavior or contents:

- `/dev/null`
  The data sink! Discards all data written to this file.
  Useful to get rid of unwanted output, typically log information:
  ```
  mplayer black_adder_4th.avi &> /dev/null
  ```

- `/dev/zero`
  Reads from this file always return `\0` characters
  Useful to create a file filled with zeros:

- ```
  dd if=/dev/zero of=disk.img bs=1k count=2048
  mkfs.ext3 ./disk.img
  mount -t ext3 -o loop ./disk1.img /mnt/image
  ```

See `man null` or `man zero` for details.
  ```

MABEL Technology
Tech The Best

Special Devices (2)

- `/dev/random`
  Returns random bytes when read. Mainly used by cryptographic programs. Uses interrupts from some device drivers as sources of true randomness (“entropy”).
  Reads can be blocked until enough entropy is gathered.

- `/dev/urandom`
  For programs for which pseudo random numbers are fine.
  Always generates random bytes, even if not enough entropy is available (in which case it is possible, though still difficult, to predict future byte sequences from past ones).

See `man random` for details.

MABEL Technology
Tech The Best

The Unix and GNU/Linux Command-Line

System Administration Basics

MABEL Technology
Tech The Best

Mounting Devices (1)

- To make filesystems on any device (internal or external storage) visible on your system, you have to *mount* them.
- The first time, create a mount point in your system: `mkdir /mnt/usbdisk` (example)
- Now, mount it:
  `mount -t vfat /dev/sda1 /mnt/usbdisk`
  `/dev/sda1`: physical device
  `-t`: specifies the filesystem (format) type
  (`ext2`, `ext3`, `vfat`, `reiserfs`, `iso9660`...)
- Mount options for each device can be stored in the `/etc/fstab` file.

MABEL Technology
Tech The Best

Listing Mounted Filesystems

- Just use the `mount` command with no argument:

```
/dev/hda6 on / type ext3 (rw,noatime)
  none on /proc type proc (rw,noatime)
  none on /sys type sysfs (rw)
  none on /dev/pts type devpts (rw,gid=5,mode=620)
  usbfs on /proc/bus/usb type usbfs (rw)
  /dev/hda4 on /data type ext3 (rw,noatime)
  none on /dev/shm type tmpfs (rw)
  /dev/hda1 on /win type vfat (rw,uid=501,gid=501)
  none on /proc/sys/fs/binfmt_misc type binfmt_misc
  (rw)
```

- Or display the `/etc/mtab` file
  (same result, updated by `mount` and `umount` each time they are run)

MABEL Technology
Tech The Best

Unmounting Devices

- `umount /mnt/usbdisk`
  Commits all pending writes and unmounts the given device, which can then be removed in a safe way.

- To be able to unmount a device, you have to close all the open files in it:
  - Close applications opening data in the mounted partition
  - Make sure that none of your shells have a working directory in this mount point.
  - You can run the `lsof` command (list open files) to view which processes still have open files in the mounted partition.

MABEL Technology
Tech The Best

Network Setup (1)

- `ifconfig -a`
  Prints details about all the network interfaces available on your system.

- `ifconfig eth0`
  Lists details about the `eth0` interface

- `ifconfig eth0 192.168.0.100`
  Assigns the `192.168.0.100` IP address to `eth0` (1 IP address per interface).

- `ifconfig eth0 down`
  Shuts down the `eth0` interface (frees its IP address).

MABEL Technology
Tech The Best

Network Setup (2)

- `route add default gw 192.168.0.1`
  Sets the default route for packets outside the local network. The gateway (here `192.168.0.1`) is responsible for sending them to the next gateway, etc., until the final destination.

- `route`
  Lists the existing routes

- `route del default`
  `route del <IP>`
  Deletes the given route
  Useful to redefine a new route.

MABEL Technology
Tech The Best

Name Resolution

- Your programs need to know what IP address corresponds to a given host name (such as `kernel.org`)
- Domain Name Servers (DNS) take care of this.
- You just have to specify the IP address of 1 or more DNS servers in your `/etc/resolv.conf` file:

```
nameserver 217.19.192.132
nameserver 212.27.32.177
```

- The changes takes effect immediately!

MABEL Technology
Tech The Best

Network Testing

- `ping freshmeat.net`
  `ping 192.168.1.1`
  Tries to send packets to the given machine and get acknowledgment packets in return.

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=0 ttl=150 time=2.51 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=150 time=3.16 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=150 time=2.71 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=150 time=2.67 ms
```

- When you can ping your gateway, your network interface works fine.
- When you can ping an external IP address, your network settings are correct!

MABEL Technology

Tech The Best

Proc File System Interface

- /proc is a virtual filesystem that exports kernel internal structures to user-space
  - `/proc/cpuinfo`: processor information
  - `/proc/meminfo`: memory status
  - `/proc/version`: version and build information
  - `/proc/cmdline`: kernel command line
  - `/proc/<pid>/fd`: process used file descriptors
  - `/proc/<pid>/cmdline`: process command line
  - `/proc/sys/kernel/panic`: time in second until reboot in case of fatal error
  - …

MABEL Technology
Tech The Best

Using a proc File

- Once the module is loaded, you can access the registered proc file:
  - From the shell:
    - **Read** `cat /proc/driver/my_proc_file`
    - **Write** `echo "123" > /proc/driver/my_proc_file`
  - Programatically, using `open(2), read(2) write(2)` and related functions.
- You can't delete, move or rename a proc file.
- proc files usually don't have reported size.

MABEL Technology
Tech The Best

Sysfs

- Sysfs is a virtual file system that represents the hardware and drivers in the system:
  - Devices existing in the system: their power state, the bus they are attached to, and the driver responsible for them.
  - The system bus structure: which bus is connected to which bus (e.g. USB bus controller on the PCI bus), existing devices and devices potentially accepted (with their drivers)
  - Available device drivers: which devices they can support, and which bus type they know about.
  - The various kinds ("classes") of devices: `input`, `net`, `sound`... Existing devices for each class.

MABEL Technology
Tech The Best

Using SysFS

- For example:
  - View all PCI devices:

```
ls /sys/devices/pci0000\:00/

0000:00:00.0  0000:00:1a.1  0000:00:1c.2  0000:00:1d.2  0000:00:1f.3 ...
```

  - Check which interrupt is used by a certain device:

```
cat /sys/devices/pci0000\:00/0000:00:1c.4/irq

23
```

- Sysfs documentation:
  - Documentation/driver-model/
  - Documentation/filesystems/sysfs.txt

MABEL Technology
Tech The Best

System Administration Basics

Boot Sequence

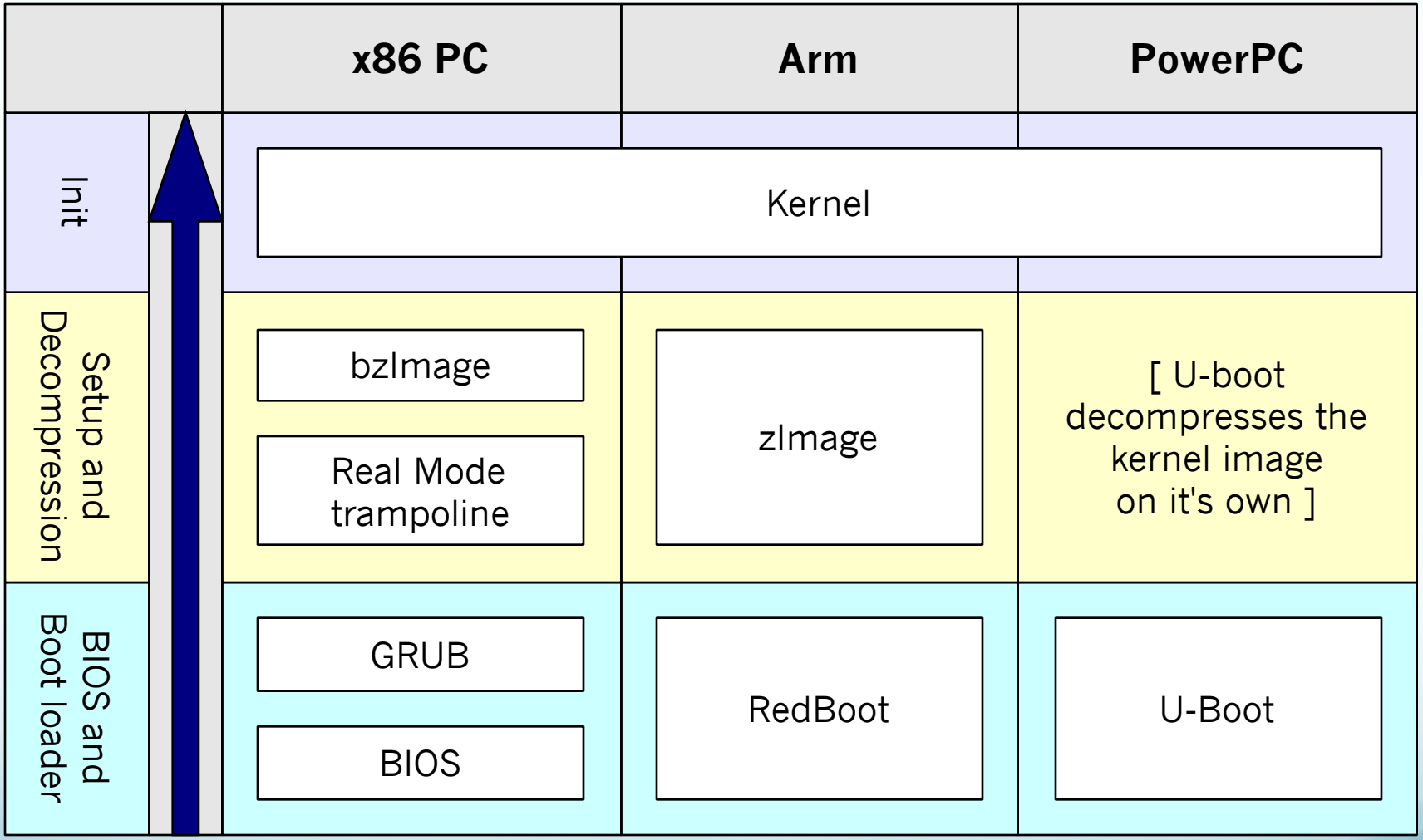
MABEL Technology
Tech The Best

Linux Boot Process

- BIOS and/or boot loader initializes hardware.
- Boot loader loads kernel image into memory.
  - Boot loader can get kernel image from flash, HD, network.
  - Possibly also loads a file system to RAM.
- Boot loader or kernel decompress compressed kernel.
- Kernel performs internal (hash table, lists etc.) and hardware (device driver) setup.
- Kernel finds and mounts the root file system.
- Kernel executes the “/sbin/init” application.

MABEL Technology
Tech The Best

Boot Sequences



MABEL Technology
Tech The Best

Root File System Options

- Many storage devices available:
  - On flash (NAND / NOR)
  - On CompactFlash / SDD
  - On Disk (SCSI / PATA / SATA)
  - In RAM (INITRamDisk or INITial RAM FileSystem)
  - From network (NFS)
- Type to use either hard coded in kernel config or passed to kernel by boot loader in the kernel command line string.

MABEL Technology

Tech The Best

Coding Embedded Linux Applications

Writing Applications

MABEL Technology
Tech The Best

System Calls and Library Functions

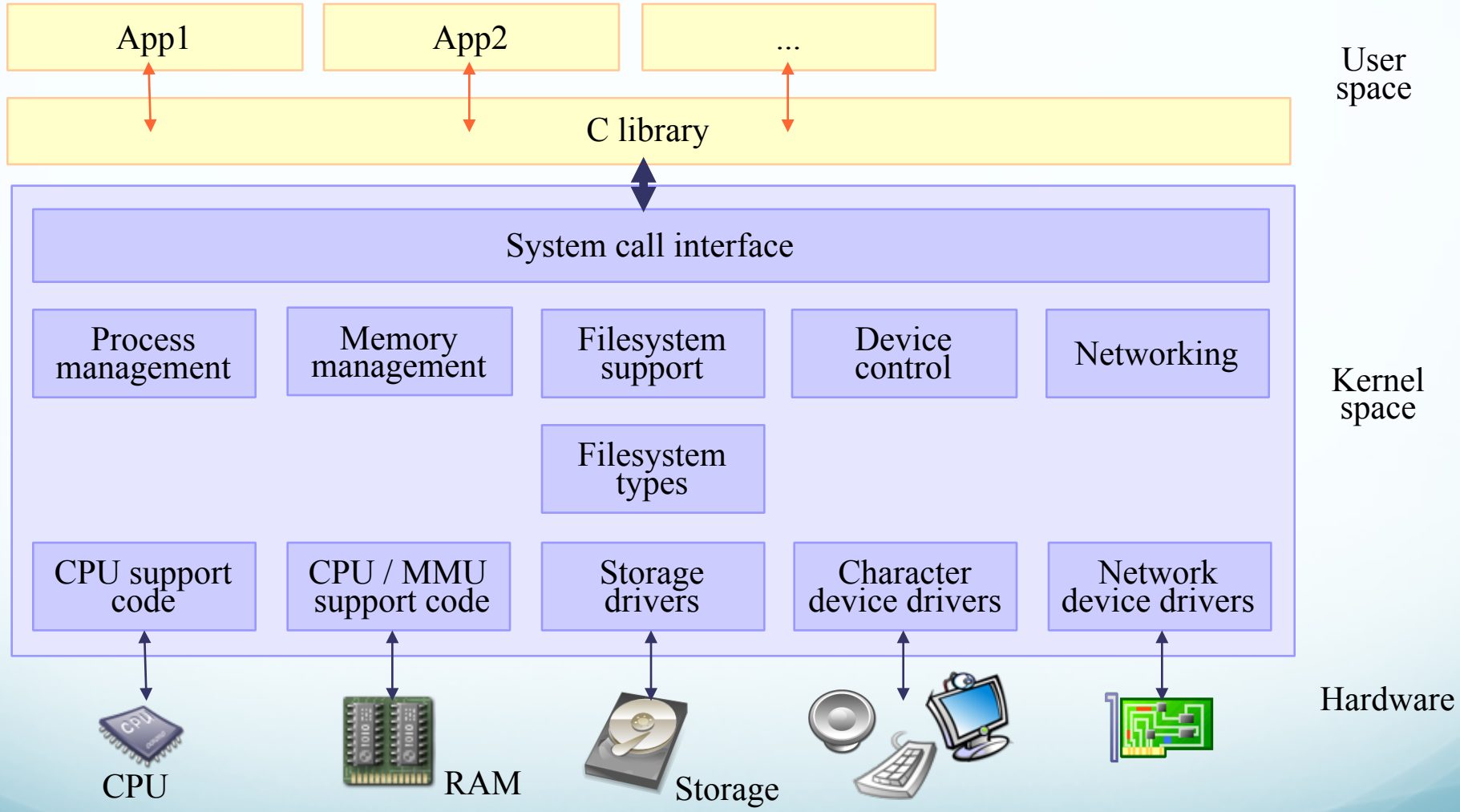
- System calls are requests to the kernel
  - appearance to programmers is of a C function call
    - read(...)
    - getpid()
    - fork()
  - trap instruction causes kernel to be activated
  - errors cause return value of -1, error code in variable `errno`

- Library functions are pre-written units of code
  - execute at user level
  - no kernel involvement

```
$ man 2 getuid
$ man 3 getpwuid
```

MABEL Technology
Tech The Best

Kernel Architecture



MABEL Technology
Tech The Best

Kernel-Mode vs. User-Mode

- All modern CPUs support a dual mode of operation:
  - User-mode, for regular tasks.
  - Supervisor (or privileged) mode, for the kernel.
- The mode the CPU is in determines which instructions the CPU is willing to execute:
  - “Sensitive” instructions will not be executed when the CPU is in user mode.
- The CPU mode is determined by one of the CPU registers, which stores the current “Ring Level”
  - 0 for supervisor mode, 3 for user mode, 1-2 unused by Linux.

MABEL Technology
Tech The Best

The System Call Interface

- When a user-space tasks needs to use a kernel service, it will make a “System Call”.
- The C library places parameters and number of system call in registers and then issues a special trap instruction.
- The trap atomically changes the ring level to supervisor mode and the sets the instruction pointer to the kernel.
- The kernel will find the required system called via the system call table and execute it.
- Returning from the system call does not require a special instruction, since in supervisor mode the ring level can be changed directly.

MABEL Technology
Tech The Best

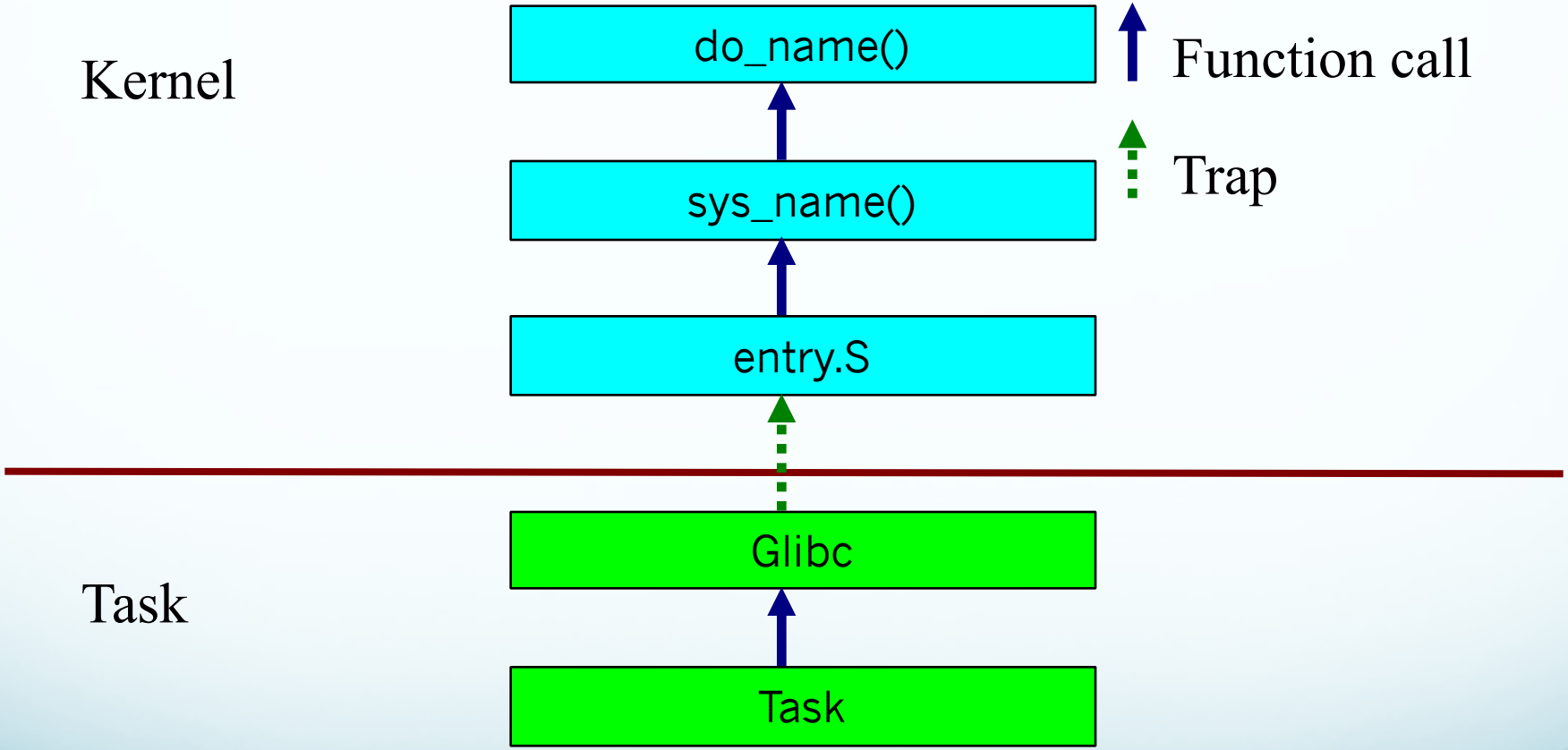
Make a call

- Using software interrupt instruction (int, syscall, swi etc.)
- Input and output parameters using hardware registers
- Example (MIPS)

```
int mygetpid()
{
    asm volatile(
                "li $v0,4020\n\t"
                "syscall\n\t"
        );
}
```

MABEL Technology
Tech The Best

Linux System Call Path



MABEL Technology
Tech The Best

Error Handling

- Always output error messages on stderr
- Always return an error code to the OS
- Always check for errors from function calls
  - Most calls return -1 or a NULL pointer on error
  - Global variable `errno` contains error number from last failed system call
  - `errno` is not cleared by a successful system call
- Error handling routines

```
#include <errno.h>
   perror(const char *msg)          prints errno message and msg to stderr
#include <stdlib.h>
   exit (int n)                     exits from program (see also atexit)
   abort (int n)                    aborts from program (see also atexit)
#include <string.h>
   const char *strerror(int errno)  returns pointer to error message
```

MABEL Technology
Tech The Best

Error Codes

#include <errno.h>

- Some useful error codes set in errno when a system call returns -1

```
1     EPERM      Not super-user
2     ENOENT     No such file or directory
3     ESRCH      No such process, LWP, or thread
4     EINTR      Interrupted system call
5     EIO        I/O error
6     ENXIO      No such device or address
7     E2BIG      Arg list too long
8     ENOEXEC    Exec format error
9     EBADF      Bad file number
10    ECHILD     No child processes
```

MABEL Technology
Tech The Best

Example - Error Handling

```
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>

int main()
{
    int NewSocket;

    if ((NewSocket = socket(AF_INET,SOCK_STREAM,0)) == -1)
    {
            perror (”Failed to create socket");
            exit (errno);
    }

    /* Continue with the program */


    return 0;                    /* or exit(0) */
}
```

MABEL Technology
Tech The Best

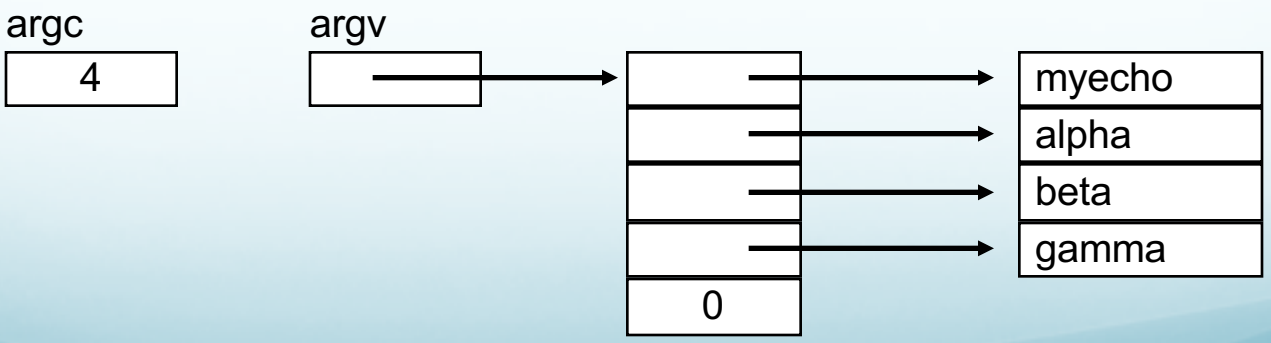
Review - Command Line Arguments

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i;
    for (i=1; i<argc; i++)
        printf ("%2d: %s\n", i, argv[i]);
    return 0;
}
```

argument vector

```
$ myecho alpha beta gamma
```



MABEL Technology
Tech The Best

Option Processing

- "Standard" command-line syntax:
  command [*options...*] [*arguments...*]

#include <unistd.h>

- Use `getopt()` to process command line arguments
  int getopt (int argc, char *argv[], const char *optstring)

argc	*No. of arguments - usually a copy of argc from main()*
argv	*List of arguments - usually a copy of argv from main()*
optstring	*String describing which options are expected and which take additional arguments*
returns	*EOF at end of option list*

  - Supporting variables

extern char *optarg;	*Pointer to option argument, when appropriate*
extern int optind;	*Index of the next element in argv to be processed*
extern int opterr;	*Set this to 0 to suppress error-message printing*
extern int optopt;	*Character that caused error condition*

MABEL Technology
Tech The Best

Example - Options and Arguments

- This program accepts options a, l & d (argument)

```
int main (int argc, char *argv[]) {
    int c, i;
    int all=FALSE, longl=FALSE;
    char *dir=NULL;

    opterr = 0;
    while ( (c=getopt(argc, argv, "ad:l")) != EOF ) {
      switch ( c ) {
        case 'a': all=TRUE;         break;
        case 'd': dir=optarg;       break;
        case 'l': longl=TRUE;       break;
        case '?': fprintf(stderr,"Unknown option %c\n",
                                  optopt);
                  exit(1);
      }
    }
    for (i=optind; i<argc; i++) {
        printf ("Argument: %s\n", argv[i]);
        ...
```

MABEL Technology
Tech The Best

The Environment

- Exported shell variables
- Null terminated list of strings of the form
  variable=value
- Read the entire environment with
  extern char **environ;
- Access individual variables with
  const char *getenv(const char *string)
- Update environment with
  int putenv(const char *string)
  - Parameter must be of the format *variable=value*

#include <stdlib.h>

```
const char *home=getenv("HOME");
if (home)
  printf ("HOME=%s\n");
else
  fprintf (stderr, "HOME variable not defined");
```

```
if (putenv ("TERM=ansi") == -1)
  perror ("Cannot set TERM");
```

MABEL Technology
Tech The Best

Limits

- Define properties of Unix implementation being used
- Compile time:
  - defined in include file <limits.h>
  - max/min values of data types
  - minimum values of run-time limits
- Run time:
  - related to files and directories
    - pathconf()
  - other limits
    - sysconf()

MABEL Technology
Tech The Best

sysconf()

- Determines current value of a configurable system limit or option

  long sysconf (int name)

  name — *name of the limit or variable to check, as defined in <limits.h> or <unistd.h>*

- Returns
  - value of limit/variable if available
  - -1, errno set to EINVAL if name is invalid
  - -1, errno clear if name represents unsupported functionality

```
if ( (max_open = sysconf(_SC_OPEN_MAX)) < 0 ) {
   if ( errno == 0 )
      unable to determine value
   else
      perror("sysconf: _SC_OPEN_MAX");
}
```

MABEL Technology
Tech The Best

pathconf()

- Determines the value of a configurable option or limit associated with files or directories

  long pathconf (const char *path, int name)

  long fpathconf (int fildes, int name)

path	*pathname of file or directory*
fildes	*file descriptor representing an open file*
name	*name of the limit or option to check (see <limits.h>)*

- Return values as for sysconf()

```
if ( (max_name_len = pathconf(".", _PC_NAME_MAX)) < 0 ) {
   if ( errno == 0 )
      unable to determine value
   else
      perror("pathconf: _PC_NAME_MAX");
}
```

MABEL Technology
Tech The Best

Dates and Times

- Unix counts time in seconds from the *Epoch*
  - 00:00:00, January 1, 1970 UTC
    - universal coordinated time
- Use time() to return current date and time
  - returns current value or -1

```
#include <time.h>
time_t time(time_t *tptr);
```

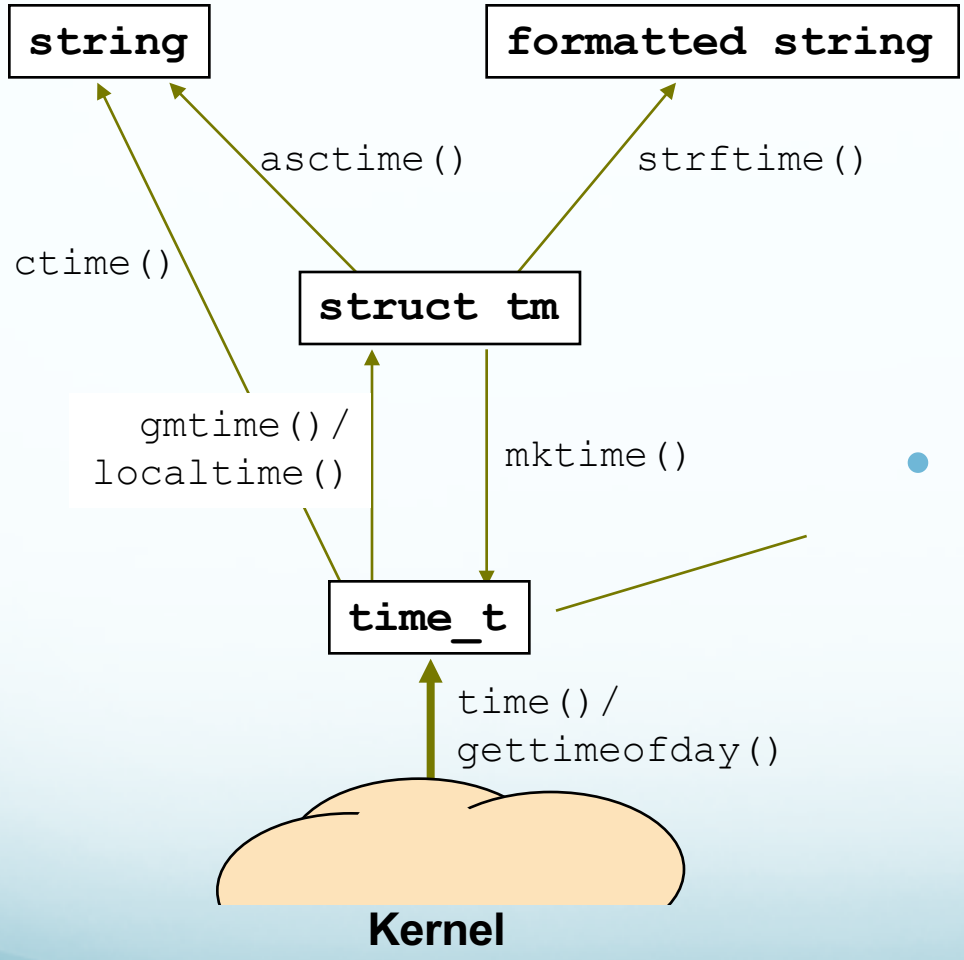
- Some systems support gettimeofday()
  - higher resolution

```
#include <sys/time.h>
int gettimeofday(struct timeval *,
                 void * );

struct timeval {
   …
   long tv_sec;  /* seconds since Epoch */
   long tv_usec; /* and microseconds */
};
```

MABEL Technology
Tech The Best

Displaying Dates and Times



- Use `tv_sec` field from `timeval` structure if time is obtained using `gettimeofday()`

MABEL Technology
Tech The Best

"Broken Down Time"

- Useful for obtaining partial date/time information
- Generated using `gmtime()` or `localtime()`

```
struct tm {
   int tm_sec;       /* Seconds after minute [0-61] */
   int tm_min;       /* Minutes after hour [0-59] */
   int tm_hour;  /* Hours [0-23] */
   int tm_mday;  /* Day of the month [0-31] */
   int tm_mon;       /* Month of the year [0-11] */
   int tm_year;  /* Years since 1900 */
   int tm_wday;  /* Days since Sunday [0-6] */
   int tm_yday;      /* Days since 1 Jan [0-265] */
   int tm_isdst;/* In Daylight Saving Time */
};
```

- Print using `asctime()` or `strftime()`
- Convert to `time_t` using `mktime()`

MABEL Technology
Tech The Best

Manual Pages

- `man [section] <keyword>`
  - Displays one or several manual pages for `<keyword>` from optional `[section]`.
    - `man fork`
      - Man page of the *fork()* system call
    - `man fstab`
      - Man page of the fstab configuration file
    - `man printf`
      - Man of *printf()* shell command
    - `man 3 printf`
      - Man of *printf()* library function
    - `man -k [keyword]`
      - `Search keyword in all man pages`

MABEL Technology
Tech The Best

A Simple Makefile

```
CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc

CFLAGS=-g

LDFLAGS=-lpthread

.PHONY: clean all

all: testapp

test1.o: test1.c test1.h

        $(CC) $(CFLAGS) -c test1.c


test2.o: test2.c test2.h

        $(CC) $(CFLAGS) -c test2.c


testapp: test1.o test2.o

        $(CC)  $(LDFLAGS) test1.o test2.o -o testapp


clean:

    @rm -f *.o *~ test
```

MABEL Technology
Tech The Best

Opening Files

- To work with a file we need to open it and get a file descriptor:

```
int fd;

fd = open("/dev/drv_ctl", O_RDWR);
```

- open() returns the file descriptor or -1 for failure.
  - In case of failure the special variable errno can be used to get the error code.

```
if(-1 == fd)

fprintf(stderr, "Error %s", strerror(errno));
```

MABEL Technology
Tech The Best

File Flags

- Various file status flags can be passed to open()
  - **O_NONBLOCK** - open file in non-blocking mode.
  - **O_SYNC** – disable OS caching of writes to the file.
  - **O_ASYNC** – send notification when data is available.
  - File creation permission can also be set in open. See man open(2) for more flags.
- File status flags can also be changed after open(2) using fcntl(2):

```
fcntl(fd, F_SETFL, O_NONBLOCK);
```

MABEL Technology
Tech The Best

File Operations

- **read, write**: read or write A buffer of data to a file
- **pread, pwrite**: read or write from specified file offset.
- **readv, writev**: like read/write, but with an array of buffers
- **select, poll, epoll**: block for event on one or more files
- **ioctl**: I/O control, exchange a buffer with device driver
- **fsync, fdatasync**: flush kernel cache of driver to hardware
- **mmap**: map resource to memory (will be descried later).

MABEL Technology
Tech The Best

Read Operations

- `ssize_t read(int fd, void *buf, size_t count);`
- **count**: maximal number of bytes to read.
- **buf**: pointer to buf to copy/transfer data to.
- **fd**: file descriptor
- return value is how many bytes were transferred (can be less then requested) or -1 for error (errno has error code).
- Position in file starts from 0 and advances with read/writes.
  - You can change position with lseek(2) or use pread(2).

MABEL Technology
Tech The Best

Example - Copying a file

```
int main (int argc, char *argv[])
{
  int rfd, wfd;
  size_t n;
  char buffer[MAXBUF];

  if (argc != 3)
    { fprintf(stderr,"Usage: mycopy source dest\n"); exit (1); }
  if ( (rfd = open(argv[1], O_RDONLY, 0)) == -1 )
    { perror (argv[1]); exit (2); }
  if ( (wfd = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, 0666)) == -1 )
    { perror (argv[2]); exit (2); }

  while ((n=read(rfd, buffer, MAXBUF)) > 0)
    if (write(wfd, buffer, n) != n)
      { perror ("Write"); exit(2); }
  if (n == -1)
    { perror("Read"); exit(2); }

  close(rfd);  close(wfd);    exit(0);
}
```

MABEL Technology
Tech The Best

Duplicating File Descriptors

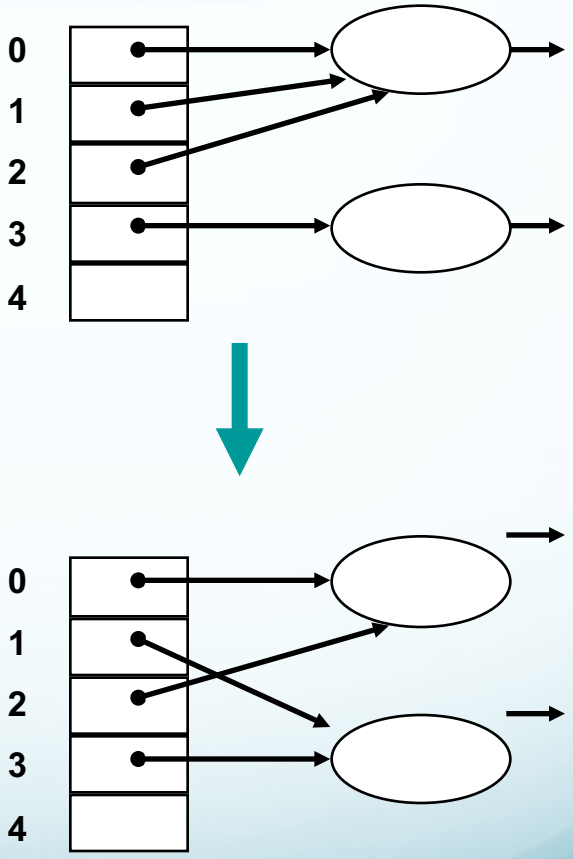
- Open file descriptors can be duplicated
  - int dup (int oldfd)
    - oldfd *file descriptor to duplicate*
    - returns *new file descriptor or -1 on error*
  - int dup2 (int oldfd, int newfd)
    - oldfd *file descriptor to duplicate*
    - newfd *file descriptor to duplicate onto (existing file will be closed)*
    - returns *new file descriptor or -1 on error*

```
int fd = open("mydata", O_RDWR, 0);

dup2 (fd, 1);

/* What is the net effect of this? */
```

#include <unistd.h>



MABEL Technology
Tech The Best

File Status - reading the inode

```
#include <sys/types.h>
#include <sys/stat.h>
```

- Retrieve file information with

  int stat(const char *path, struct stat *statbuf)

  int fstat(int fd, struct stat *statbuf)

  path — *pathname of file to access*

  fd — *open file descriptor*

  statbuf — *pointer to buffer to receive data*

  returns — *0 or -1 on error*

```
struct stat statbuf;

if ( stat ("mycopy.c", &statbuf) < 0 ) {
       perror("stat mycopy.c");
       …
}
```

MABEL Technology
Tech The Best

File Status Information

```
#include <sys/types.h>
#include <sys/stat.h>
```

- Status information structure

```
struct stat {
   ushort  st_mode;     file mode
   ino_t           st_ino;      inode number
   short           st_nlink;    number of links
   ushort  st_uid;      user id
   ushort  st_gid;      group id
   off_t           st_size;     file size in bytes
   time_t  st_atime;    time of last access
   time_t  st_mtime;    time of last modification
   time_t  st_ctime;    time of last status change
   ...
};
```

- Interrogate st_mode word using built in macros

MABEL Technology
Tech The Best

Example, using stat()

mystat.c

```
int main (int argc, char *argv[])
{
  struct stat buffer;
  int i;

  for (i=1; i<argc; i++)
  {
    char permissions[] = "---";
    int rwx;

    if (stat(argv[i],&buffer) == -1)
        { perror(argv[i]); continue; }

    printf ("%s: size=%d, owner=%d, ",
            argv[i], buffer.st_size, buffer.st_uid);

/* Continued over… */
```

```
    if      (S_ISREG (buffer.st_mode)) printf("file");
    else if (S_ISDIR (buffer.st_mode)) printf("directory");
    else if (S_ISFIFO(buffer.st_mode)) printf("fifo");
    else if (S_ISLNK (buffer.st_mode)) printf("symblic link");
    else if (S_ISCHR (buffer.st_mode)) printf("char dev");
    else if (S_ISBLK (buffer.st_mode)) printf("block dev");

    rwx = buffer.st_mode & (S_IRWXU|S_IRWXG|S_IRWXO);

    if (rwx & S_IRUSR) permissions[0] = 'r';
    if (rwx & S_IWUSR) permissions[1] = 'w';
    if (rwx & S_IXUSR) permissions[2] = 'x';

    printf(", user permissions: %s \n", permissions);
  }
}
```

MABEL Technology
Tech The Best

Creating files - the umask

```
#include <sys/stat.h>
```

- Unix file creation mode is against the umask

  mode_t umask(mode_t mask)

  mask new umask to install

  returns old umask value or -1 on error

  - Bits set in umask are cleared from the creation mode
  - Mask specified in octal for readability (eg. 022)

- The umask can be at the command using

umask	*plain text files (open)*	*directories (mkdir)*
	0666 rw-rw-rw-	0777 rwxrwxrwx
000	0666 rw-rw-rw-	0777 rwxrwxrwx
022	0644 rw-r--r--	0755 rwxr-xr-x
027	0640 rw-r-----	0750 rwxr-x---
077	0600 rw-------	0700 rwx------

MABEL Technology
Tech The Best

Random Access

- Read/write pointer can be positioned using
  off_t lseek(int fd, off_t offset, int origin)

  #include <fcntl.h>

  fd — *open file descriptor*

  offset — *byte offset from origin*

  origin — *one of SEEK_SET, SEEK_CUR or SEEK_END*

  returns — *new byte offset or -1 on error*

- It is not an error to seek beyond the end of file

```
int get_prev_record (int fd, struct mydata *record)
{
  int recsize = sizeof(struct mydata);
  off_t pos = lseek(fd, 0L, SEEK_CUR);  /* Get current position */

  if (pos>=recsize && lseek(fd,pos-recsize),SEEK_SET) != -1)
      return (read (fd, (char *)record, recsize);
  return -2;
}
```

Stream I/O and Unix I/O

```
#include <stdio.h>
FILE *fp = fopen("foo", "r+");
```

```
/* From stdio.h */

typedef struct {
   …
   unsigned char _file; /* file descriptor */
} FILE;
```

- Use `fileno(fp)` to retrieve descriptor from FILE handle
- Use `fdopen(fdesc, mode)` to build FILE handle from descriptor
  - descriptor obtained from open() or dup()

MABEL Technology
Tech The Best

Ioctl operations

- ioctl operations send command to driver and optionally sends or gets a long or pointer to buffer.

```
int ioctl(int fd, int request, unsigned long param);
```

  - **request**: integer defined by driver.
  - **param**: long or pointer to buffer (usually a structure).
  - can be either in, our or in/out operation.

- Ioctls are for when nothing else fits - don't use ioctl(2) if other file operations is better suitable (read, write ...)

MABEL Technology
Tech The Best

Blocking for events

- You can use select(2) to block for events
- ```
  int select(int nfds, fd_set *readfds, fd_set
  *writefds, fd_set *exceptfds, struct timeval
  *timeout);
  ```
  - **nfds**: number of highest file descriptor + 1
  - **fd_sets**: sets of file descriptors to block for events on, one for read, one for write, one for exceptions.
  - **timeout**: NULL or structure describing how long to block.
  - Linux updates the timeout structure according to how much time was left to the time out.
  ```

MABEL Technology
Tech The Best

File Descriptor Sets

- fd_set is a group of file descriptor for actions or reports:

```
void FD_CLR(int fd, fd_set *set);
```

  - Remove fd from this set.

```
int  FD_ISSET(int fd, fd_set *set);
```

  - Is fd in the set?

```
void FD_SET(int fd, fd_set *set);
```

  - Add fd to this set.

```
void FD_ZERO(fd_set *set);
```

  - Zero the set.

MABEL Technology
Tech The Best

Example

```
int fd1, fd2;

fd_set fds_set;

int ret;

fd1 = open("/dev/drv_ctl0", O_RDWR);

fd2 = open("/dev/drv_ctl1", O_RDWR);

FD_ZERO(&fds_set);

FD_SET(fd1, &fds_set);

FD_SET(fd2, &fds_set);

do {

ret = select(fd1 + fd2 + 1,
&fds_set, NULL, NULL, NULL);


} while(errno == EINTR);
```

- Open two file descriptors fd1 and fd2.
- Create an fd set that holds them.
- Block for events on them.
  - We try again if we interrupted by a signal.

```
if(ret == -1) {

perror("Select failed.");

exit(1);

}


if(FD_ISSET(fd1, &fds_set)) {

printf("event at FD 1... “);

ioctl(fd1, IOCTL_CLR, 1);

printf("clear\n);

}
```

- If we have an error bail out.
- Check if fd1 one has a pending read even.
- If so, use ioctl(2) to notify driver to clear status

MABEL Technology
Tech The Best

System calls for Directories

#include <unistd.h>

- Make a new directory
  int mkdir (const char *path, int mode)
    mode *initial access permissions, modified by umask*
- Remove a directory
  int rmdir (const char *path)
    path *directory to remove - must be empty except for . and ..*
- Change working directory or root directory
  int chdir (const char *path)
  int chroot (const char *path)
    path *directory to change to - if chroot(), this becomes root of filesystem*
- Get current working directory
  char *getcwd (char *buffer, size_t bufsize)
    buffer *area of memory in which to store pathname of current directory, must be bufsize bytes*
    returns *pointer to pathname*

MABEL Technology
Tech The Best

System calls for Directory Entries

```
#include <unistd.h>
```

- Create a new entry (hard link)
  int link (const char *oldpath, const char *newpath)
  - Old pathname must exist
  - Only root can link to directories
- Remove an existing link
  int unlink (const char *path)
  - Don't use to unlink directories
  - File is deleted when last link removed
- Delete a file
  int remove (const char *pathname)
  - ANSI C standard function - use in preference to unlink
- Rename or move a file
  int rename (const char *oldpath, const char *newpath)
  - Can rename from one directory to another (c.f. mv command)

MABEL Technology
Tech The Best

System calls for Symbolic Links

#include <unistd.h>

- Create a new symbolic link
  int symlink (const char *oldpath, const char *newpath)
  - Old pathname does not have to exist
- Read symbolic link inode
  int lstat (const char *path, struct stat *statbuf)
  - Behaves like **stat()** except when accessing a symbolic link
  - Cannot be accessed via file descriptor as links are "transparent"
- Read link pathname
  int readlink (const char *path, char *buffer, int bufsize)

buffer	*buffer for link pathname which is NOT null terminated*
bufsize	*size of buffer*
returns	*number of bytes written to buffer or -1 on error*

MABEL Technology
Tech The Best

Example - Reading Symbolic Links

linkinfo.c

```
int main (int argc, char *argv[])
{
  struct stat statbuf;
  char buffer[FILENAME_MAX+1];
  int i, n;

  for (i=1; i<argc; i++) {
    if (lstat(argv[i],&statbuf) == -1)
        { perror(argv[i]); continue; }
    printf ("%s", argv[i]);
    if (S_ISLNK(statbuf.st_mode))
    {
        if ((n=readlink(argv[i],buffer, FILENAME_MAX)) == -1 )
            { perror(argv[i]); exit(1); }
        buffer[n] = '\0';
        printf ("->%s", buffer);
    }
    putchar ('\n');
  }
}
```

MABEL Technology
Tech The Best

Altering Access Permissions

#include <unistd.h>

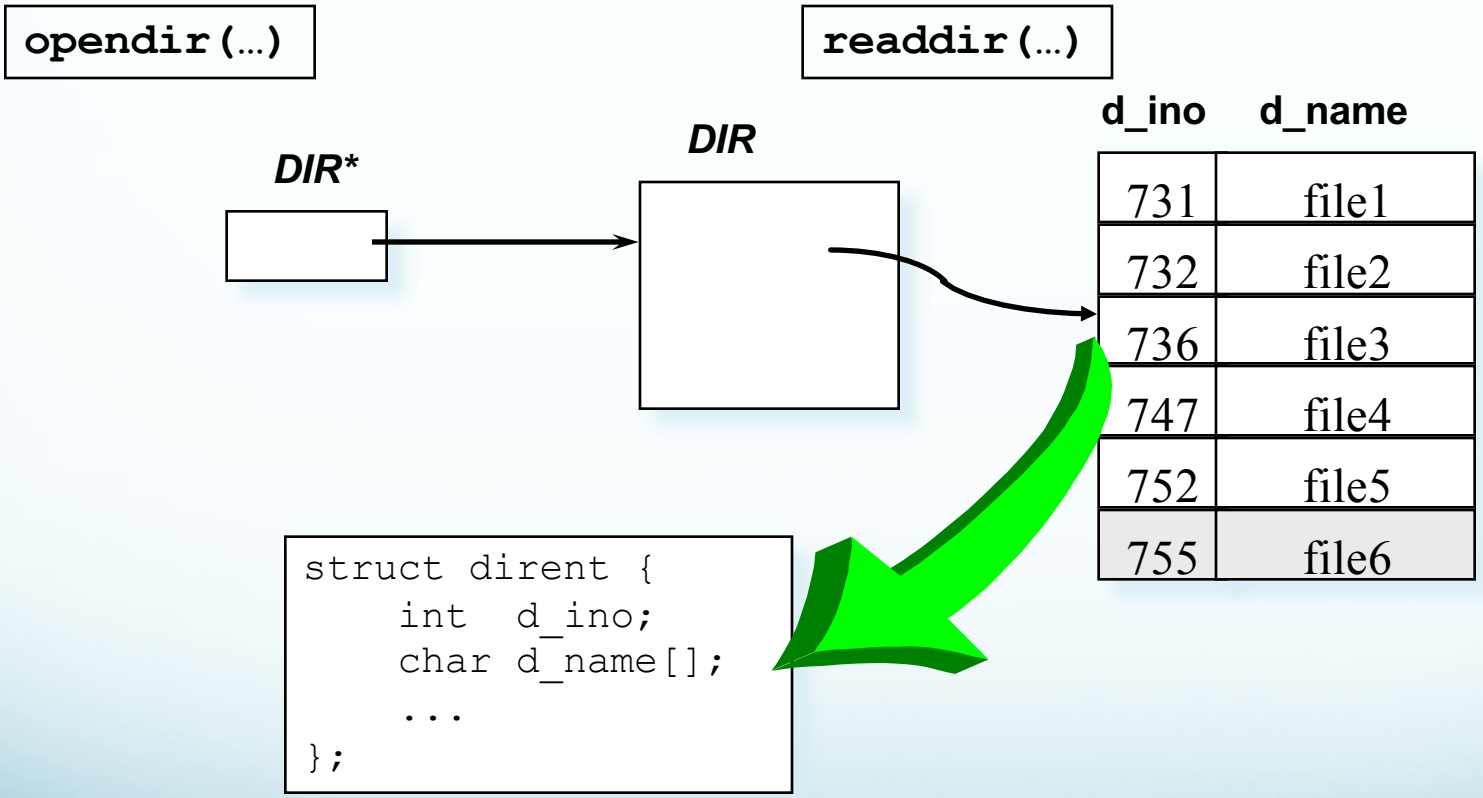
- Change UID/GID
  int chown (const char *path, uid_t owner, gid_t group)
  int fchown (int fd, uid_t owner, gid_t group)
  - Specify owner or group as -1 to leave retain existing value
- Change permissions
  int chmod (const char *path, mode_t mode)
  int fchmod (int fd, mode_t mode)
  - Usual to read old mode first and set/clear required bits

```
struct stat statbuf;

if (stat("myscript",&statbuf) == -1)
  { perror("stat failed"); exit(1); }
statbuf.st_mode |= (IS_IXUSR|IS_IXGRP|IS_IXOTH);
statbuf.st_mode &= ~(IS_IWGRP|IS_IWOTH);
if (chmod("myscript",statbuf.st_mode) == -1)
  { perror("chmod failed"); exit(1); }
```

MABEL Technology
Tech The Best

Accessing Directory Contents



MABEL Technology
Tech The Best

Reading Directories

- Open directory for reading
  DIR *opendir (const char *path)
  - returns NULL if directory cannot be opened

#include <dirent.h>

- Close directory
  void closedir (DIR *dirp)
  - releases memory dynamically allocated by opendir()
- Read next file entry
  struct dirent *readdir(DIR *dirp)
  - returns pointer to the next entry, data will overwritten by each subsequent call to the function
  - returns NULL at end of the directory

```
struct dirent *dentp;
DIR *dirp;

dirp = opendir(".");
while (dentp=readdir(dirp)) {
  printf ("%s\n", dentp->d_name);
}
closedir(dp);
```

MABEL Technology
Tech The Best

Positioning in Directories

- Positioning in a directory

```
#include <dirent.h>
```

  off_t telldir(DIR *dirp)
    - *Return current directory position*

  void seekdir(DIR *dirp, off_t offset)
    - Position to previous telldir location (don't assume 0 is start)

  void rewindir(DIR *dirp)
    - *Rewind back to start of directory*

```
off_t profile = 0;
dirp = opendir(".");
while (dentp=readdir(dirp)) {
  if (!strcmp(dentp->d_name,".profile"))
    profile = telldir(dirp);
  }
if (profile)
  seekdir(dirp,profile);
closedir(dp);
```

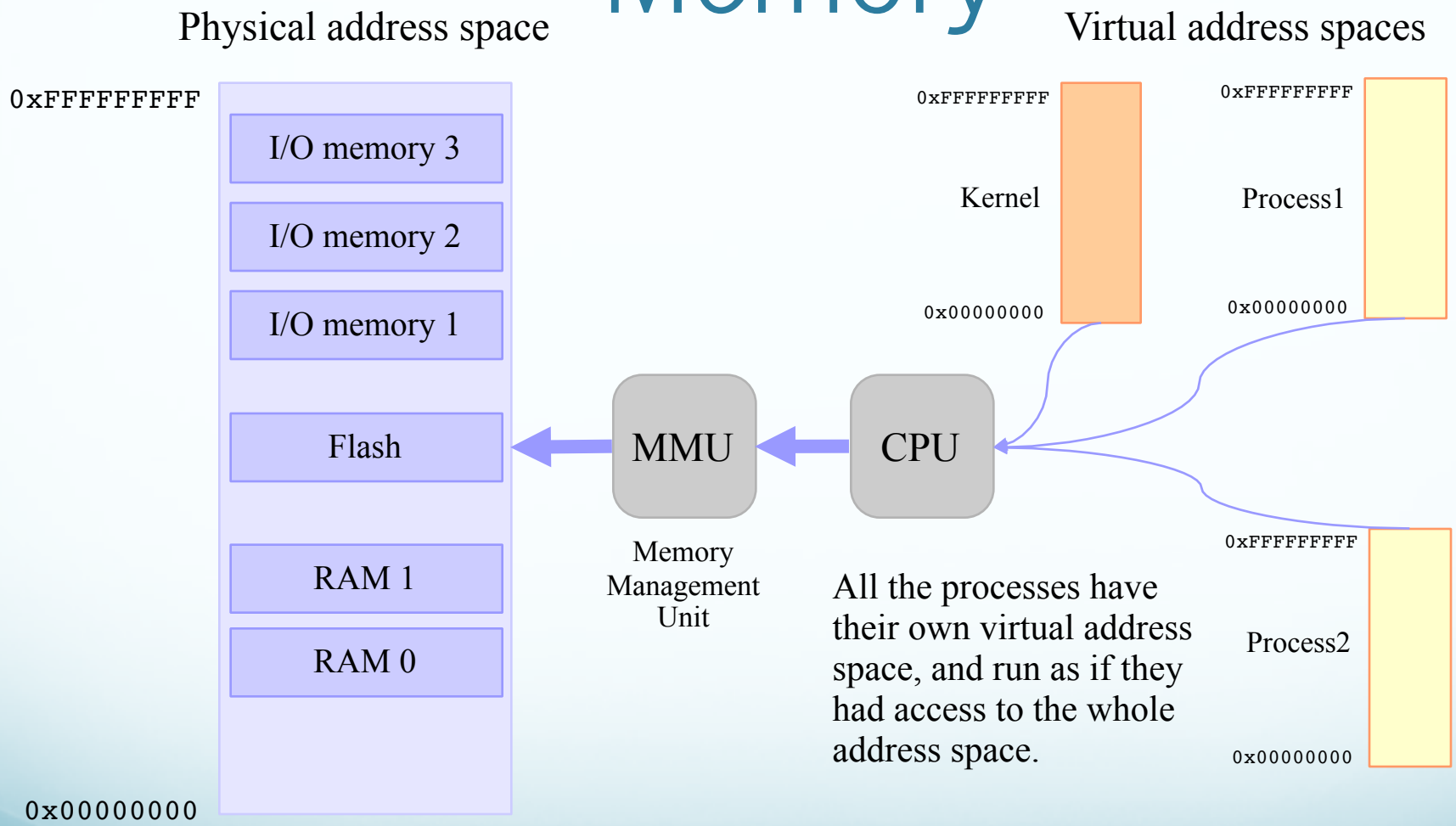
MABEL Technology
Tech The Best

Working with processes

- A processes is a container which holds a running program.
- Each container is a sand box which has its own set of file descriptors and virtual memory space.
- When a process ends all the resources it uses (memory, files etc.) are automatically collected by the system.
- You first create a new process, then set it's properties.
- When a new process is created it starts to run automatically.
- You create new processes using the fork(2) call.

MABEL Technology
Tech The Best

Physical and Virtual Memory

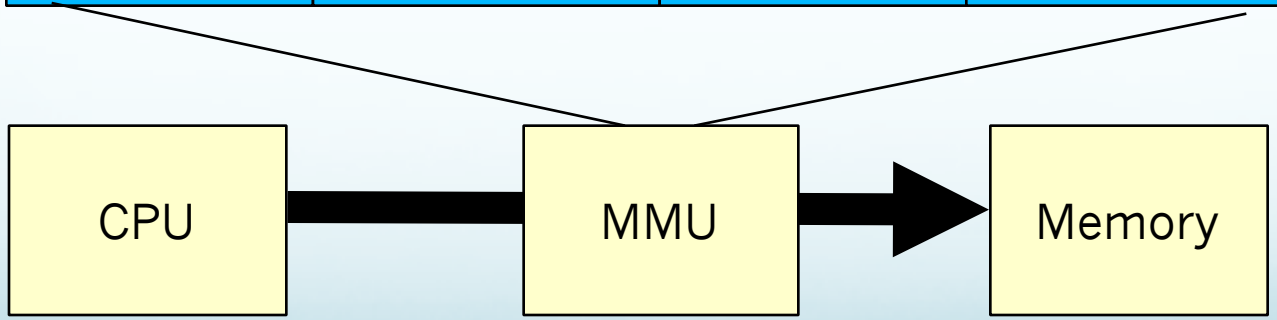


MABEL Technology
Tech The Best

The Memory Management Unit

- The tables that describes the virtual to physical translations are called page tables. They reside in system memory.
- Each entry describes a slice of memory called a page.

Context	Virtual	Physical	Permission
12	0x8000	0x5340	RWX
15	0x8000	0x3390	RX



MABEL Technology
Tech The Best

Process IDs

- To find out a process' id:

  pid_t getpid(void)

  returns *process id or -1 on error*

- To find a process' parent id:

  pid_t getppid(void)

  returns *process id of parent process or -1*

- To find a process' process group:

  pid_t getpgid (int pid)

  pid *process whose group is to be returned (0 for this process)*

  returns *process group id or -1 on error*

- To find a process' session id:

  pid_t getsid (void)

  returns *session id of calling process or -1 on error*

#include <unistd.h>

```
pid_t myPid;

myPid = getpid();
myPpid = getppid();
printf("I am process %d\n", myPid);
printf("My parent is %d\n", myPpid);
```

IDs for access permissions

- For access checking, a process runs on behalf of a user
  - Process is also a member of (at least one) group
- There are different IDs represented:
- Real user/group id
  - Who we really are
- Effective user/group id
  - Used to check file access permissions
  - May have been changed (eg. setuid programs)
- Saved set-user/group-id
  - Saved during exec function (later)

MABEL Technology
Tech The Best

Changing user and group ids

- Effective user/group id automatically set when setuid/setgid program is executed
- To change a process user (or group) id:

int setuid (uid_t uid)
uid *the user id to change to*

int setgid (gid_t gid)
gid *the group id to change to*

  - if caller has root privileges, change real, effective and saved user-ids
    - can change to any user id
  - if caller has normal privileges, change effective id only
    - can only change to real or saved set-user-id

int seteuid (uid_t uid)

int setegid (gid_t gid)

  - as above but changes *only* effective user or group id

MABEL Technology
Tech The Best

Process Resource Limits

- Various resource limits apply to processes
- To query current limit:

  int getrlimit (int resource, struct rlimit *rlim)

  resource the resource being queried
  rlim pointer to structure to contain information
- To change a resource limit

  int setrlimit (int resource, const struct rlimit *rlim)

  resource the resource to be changed
  rlim the new value of the resource limit

```
struct rlimit {
    rlim_t  rlim_cur;    Soft limit - the current limit
    rlim_t  rlim_max;    Hard limit - maximum value for current limit
}
```

MABEL Technology
Tech The Best

Resources with limits

- `RLIMIT_CORE`
  - maximum size of core file
    - set to 0 to disable core file production
- `RLIMIT_CPU`
  - maximum amount of CPU time
- `RLIMIT_NOFILE`
  - maximum number of open files
- `RLIMIT_STACK`
  - maximum size of program stack segment
- `RLIMIT_DATA`
  - maximum size of program data segment
- `RLIMIT_FSIZE`
  - maximum size of a file this process may create

```
#include <sys/types.h>
#include <sys/resource.h>
```

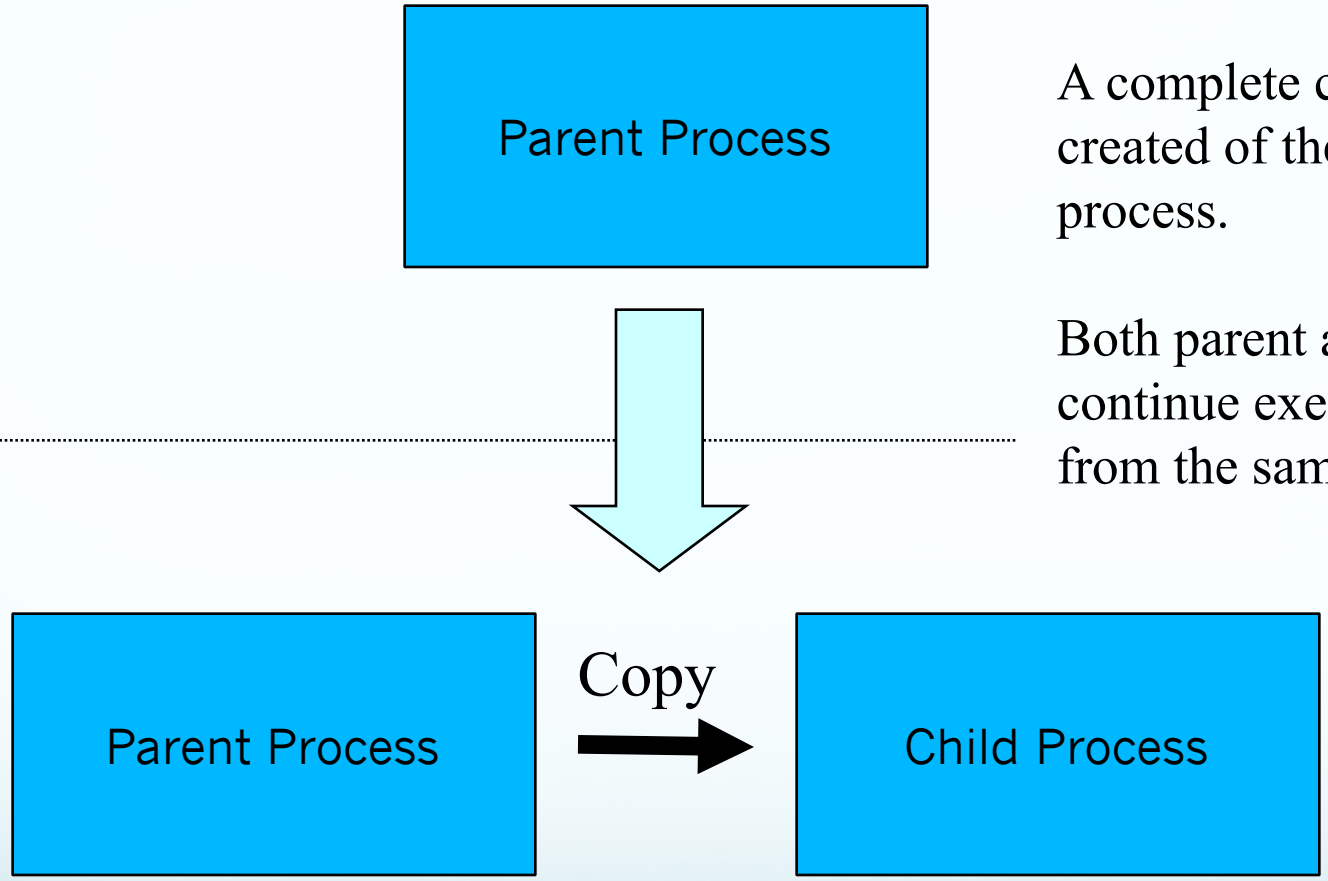
MABEL Technology
Tech The Best

Creating Processes Using fork()

```
pid_t pid;
if (pid = fork()) {
        int status;
        printf("I'm the parent!\n");
        wait(&status);
        if (WIFEXITED(status)) {
                printf("Child exist with status of %d\n", WEXITSTATUS(status));
        } else {
                printf("I'm the child!\n");
                exit(0);
        }
}
```

MABEL Technology
Tech The Best

How fork() Seems to Work

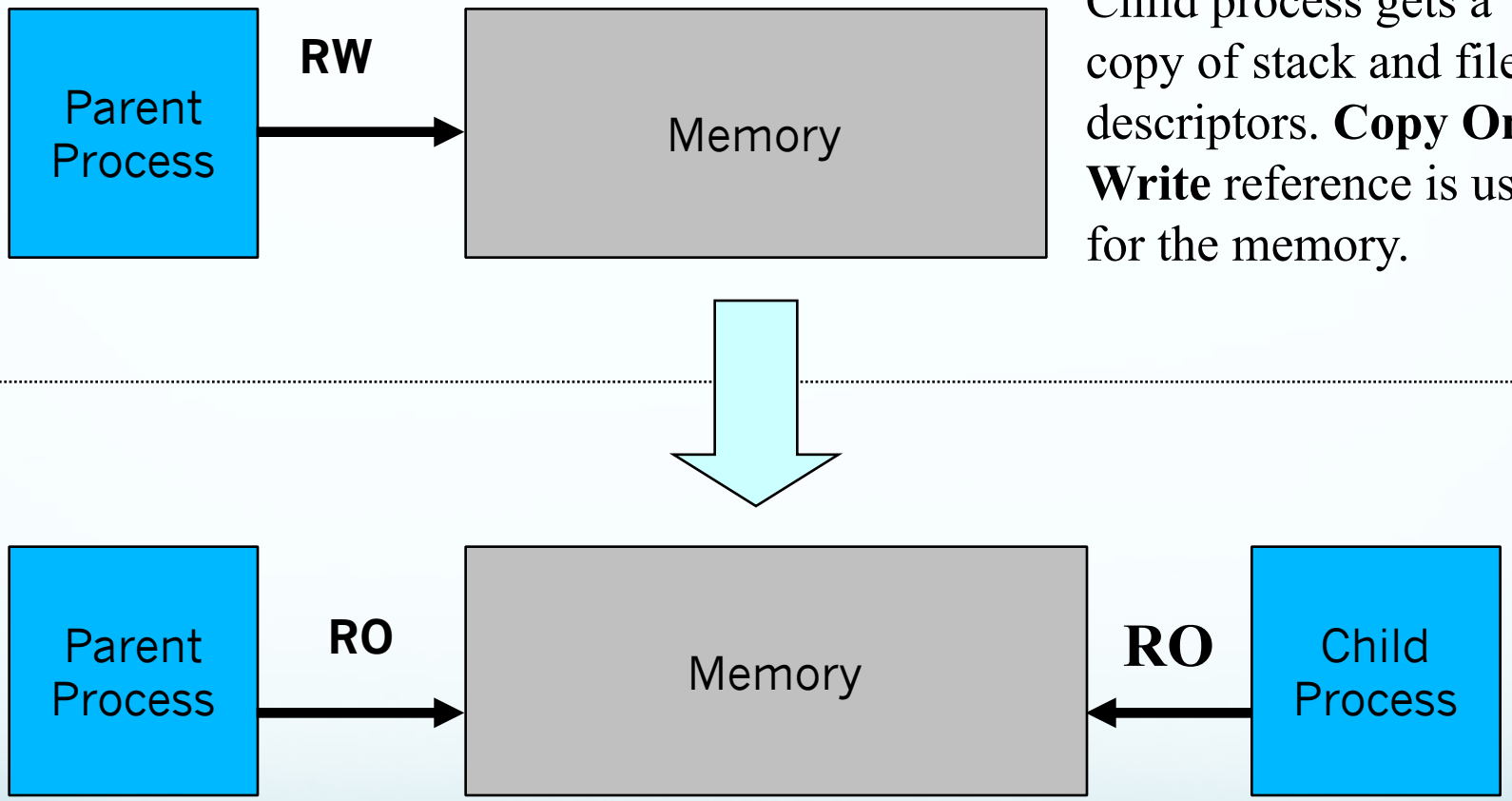


A complete copy is created of the parent process.

Both parent and child continue execution from the same spot.

MABEL Technology
Tech The Best

How fork() Really Works

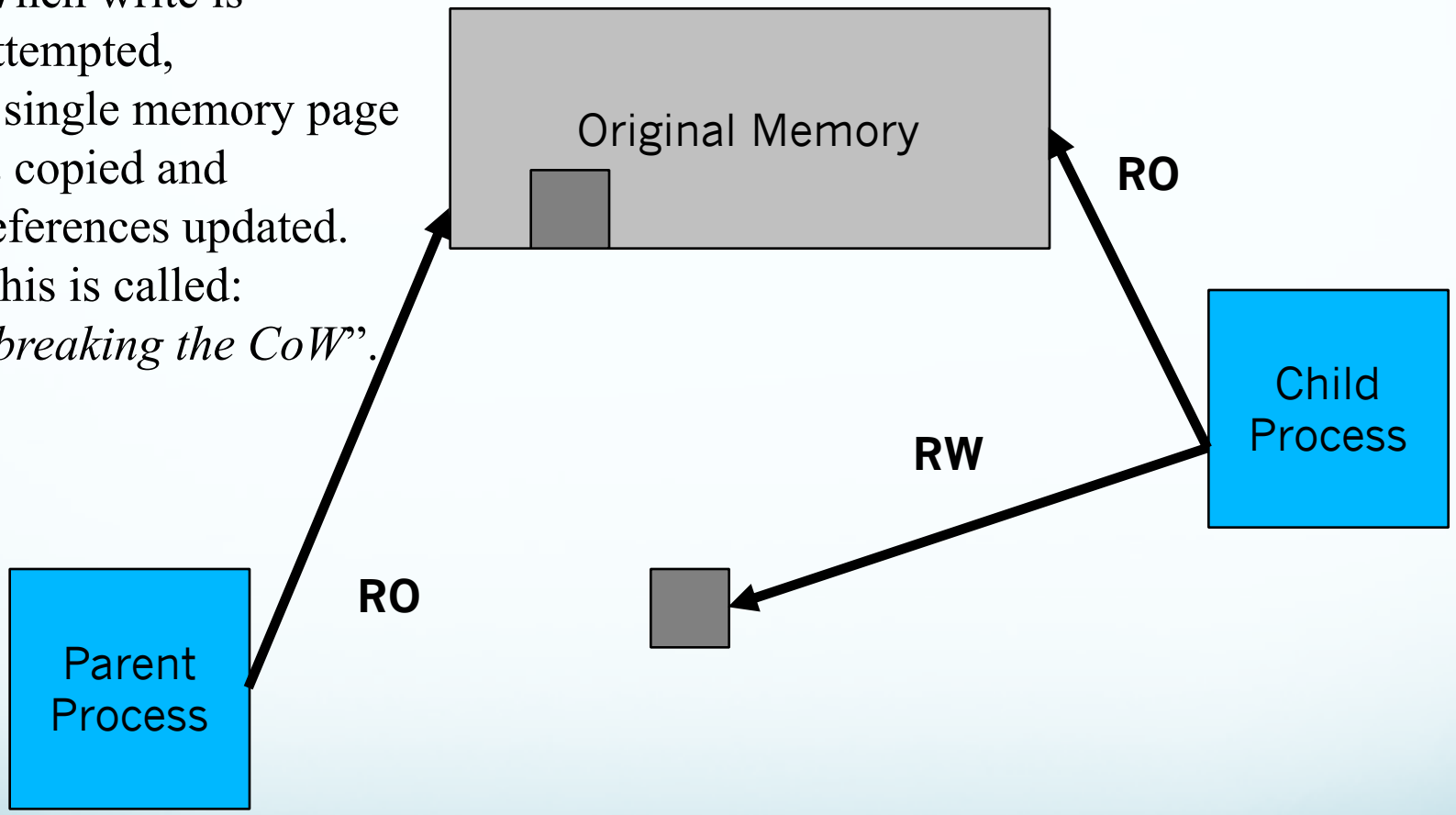


Child process gets a copy of stack and file descriptors. **Copy On Write** reference is used for the memory.

MABEL Technology
Tech The Best

What Happens During Write?

When write is attempted, a single memory page is copied and references updated. This is called: "*breaking the CoW*".



fork() - Inherited Attributes

- Important attributes inherited by child process
  - Real and effective user and group ids
  - Current working directory
  - Open files
  - Signal handlers and mask
  - File-mode creation mask (umask)
  - environment

- Differences between child and parent process
  - Child has its own process id and parent process id
  - Alarm signal timers reset to 0 in child
  - file locks are not inherited

MABEL Technology
Tech The Best

Overlaying a Process

#include <unistd.h>

- To overlay current process

int execve (const char *path, const char **argv, const char **envp)

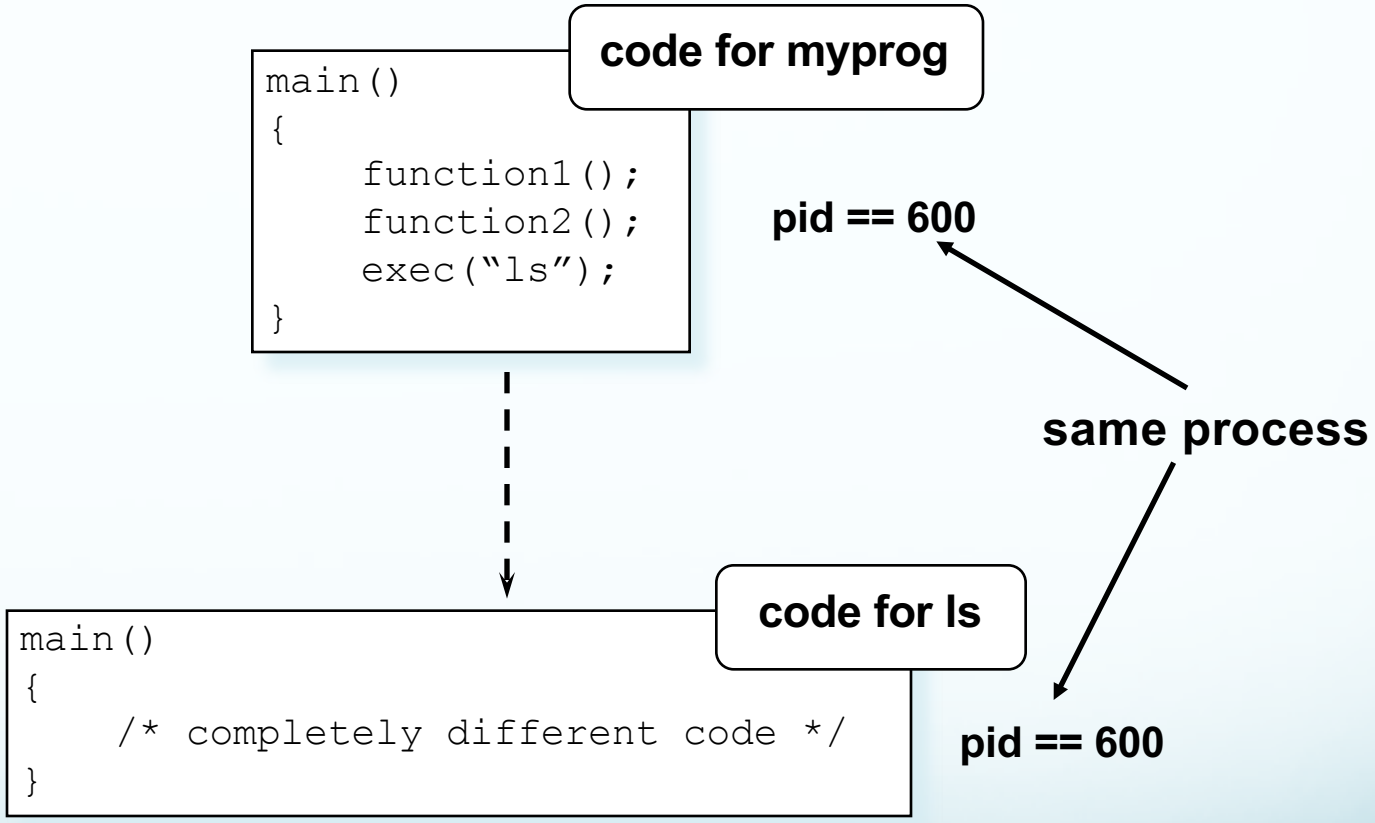
path	pathname of program to execute
argv	argument vector for new program (command line)
envp	environment pointer for new program (usually **environ**)
returns	never returns unless error occurs

- Usual to use more convenient library functions.

```
char *newenv[] = {"TERM=vt220", "PATH=/usr/bin",NULL};
execlp ("grep",      "grep","abc","file1","file2",NULL);
execl  ("/bin/grep", "grep","abc","file1","file2",NULL);
execle ("/bin/grep", "grep","abc","file1","file2",NULL, newenv);
```

```
char *args[] = {"grep","abc","file1","file2",NULL};
execvp ("grep",      args);
execv  ("/bin/grep", args);
execve ("/bin/grep", args, newenv);
```

What happens at exec()



exec() - Retained Attributes

- Important attributes retained across a call to the `exec()` family
  - Process id and parent-process id
  - Real user and group ids
  - Current working directory
  - Open files (except those marked "close-on-exec")
  - File-mode creation mask (umask)
  - Any file locks
  - Alarm signal timers
- Differences for new program
  - Possibility of new effective user and/or group ids
  - Signal handlers set to functions are cleared (defaulted and ignored signals unaffected)

Example- fork() & exec()

- Parent executes "ps -f", child executes "ls -l"

forkexec.c

```
int main ()
{
  pid_t pid;

  switch (pid=fork())
  {
    case -1: perror("fork failed"); exit(1);

    case  0: execlp ("ls", "ls", "-l", NULL);
             perror ("child exec failed"); exit(2);

    default: execlp ( "ps", "ps", "-f", NULL);
             perror ("exec failed"); exit(2);
  }
}
```

MABEL Technology
Tech The Best

Synchronising Parent and Child

```
#include <sys/wait.h>
```

- Parent should synchronise with child processes
  - Allows kernel to tidy up child user area

pid_t wait (int *status)

status	*variable to hold child exit status*
returns	*pid of the child whose state changed*

pid_t waitpid (pid_t pid, int *status, int options)

status	*word to hold child exit status*
pid	*process(es) to wait for:*
	-1 (any child) *0 (any child in process group)*
	>0 (specific child) *<0 (any child in group abs(pid))*
options	*options(eg non-blocking wait)*
returns	*pid of the child whose state changed*

MABEL Technology
Tech The Best

Interpreting the exit status

- Contains exit status and other information

```
#include <sys/wait.h>
```

- Access using the following macros

WIFEXITED(status)	*true if child called exit()*
WEXITSTATUS(status)	*returns parameter to child exit() call*
WIFSIGNALED(status)	*true if child terminated by signal*
WTERMSIG(status)	*signal number of terminating signal*
WCOREDUMP(status)	*true if a core file was written*

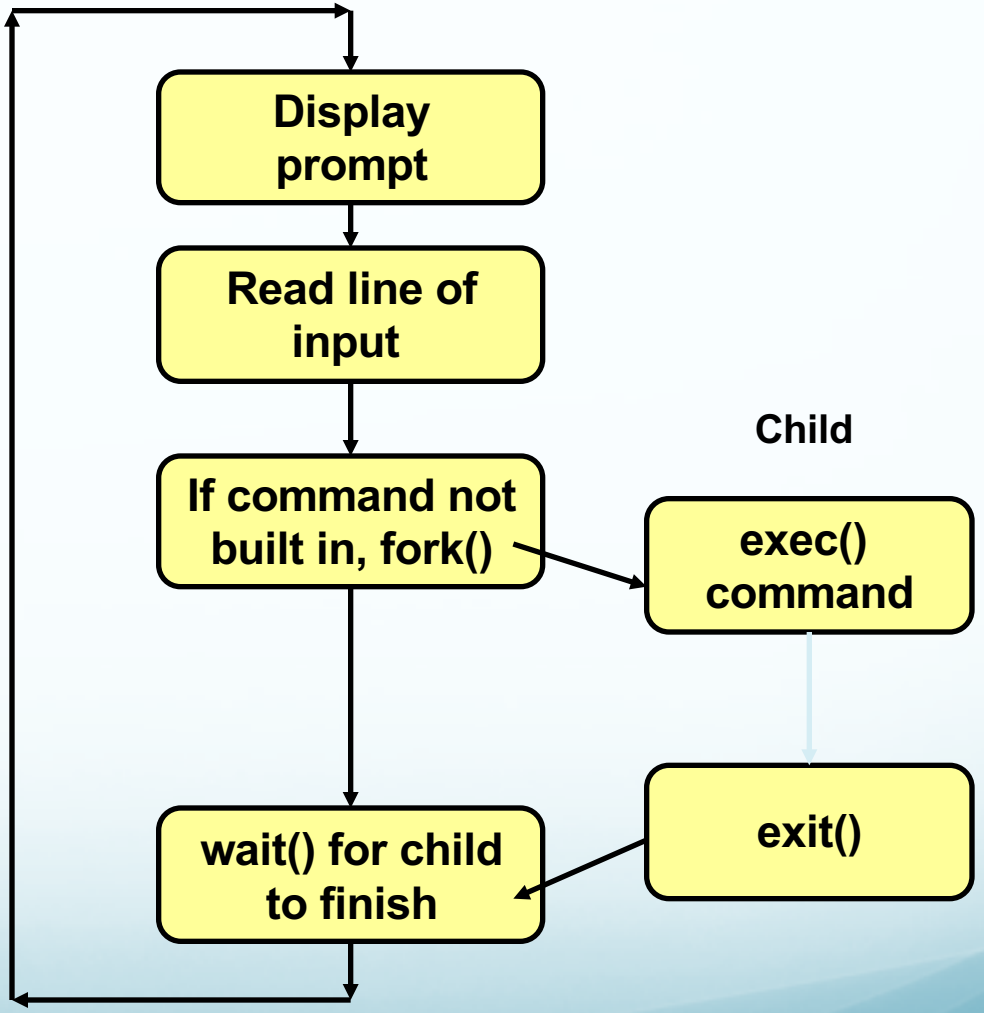
```
int main ()
{
  pid_t pid;
  int status;
  switch (pid=fork()) {
    case -1: perror("parent fork failed"); exit(1);
    case  0: execlp ("ls", "ls", "-al", NULL);
             perror ("child exec failed"); exit(2);
    default:
      if (wait(&status) > 0 && WIFEXITED(status))
         printf ("Child exited with status %d\n", WEXITSTATUS(status));
}
```

MABEL Technology
Tech The Best

The Shell

- Basic control flow within a Unix shell uses all process lifetime related system calls
- How could background commands be handled?

```
$ command &
```

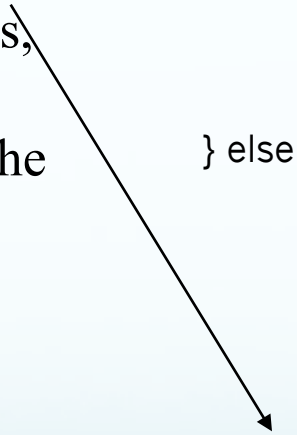


MABEL Technology
Tech The Best

Creating Processes (2)

This call will replace the program memory (code and data) with the image from storage at the specified path.

Open file descriptors, priority and other properties remains the same.



```
pid_t pid;

if (pid = fork()) {
        int status;

        printf("I'm the parent!\n");

        wait(&status);

} else {

        printf("I'm the child!\n");

        execve("/bin/ls", argv, envp);

}
```

MABEL Technology
Tech The Best

Invoking other commands

- ANSI C provides a simple interface for invoking arbitrary OS commands

  int system (const char *command)

  command *any valid shell command including I/O redirection*

  returns *exit status of child or -1 on error*

#include <stdlib.h>

- Blocks until requested command completes
- Command string is interpreted using /bin/sh

```
int main ()
{
  int status = system("ls -al");
  if (status == -1)
    perror("system call failed");
  else
    printf ("Command exit status %d\n", status);
}
```

MABEL Technology
Tech The Best

Exiting Processes

- A process exists when either of the following occurs:
  - Return from the main function.
  - Calling the *exit()* function.
  - Exception (more on those later).
- A process should return an exit status to it's parent process
- Upon exit:
  - All exit handlers are called (use *atexit()* to register them)
  - All memory, file descriptors and other resources are released by the system.

MABEL Technology
Tech The Best

Locking Memory

- `int mlock(const void *addr, size_t len);`
  - mlock disables paging for the memory in the range starting at addr with length len bytes.

- `int mlockall(int flags);`
  - *mlockall()* disables paging for all pages mapped into the address space of the calling process.
  - **MCL_CURRENT** locks all pages which are currently mapped into the address space of the process.
  - **MCL_FUTURE** locks all pages which will become mapped into the address space of the process in the future. These could be for instance new pages required by a growing heap and stack as well as new memory mapped files or shared memory regions.

MABEL Technology
Tech The Best

Protecting Memory

- `int mprotect(const void *addr, size_t len, int prot);`
- The function mprotect() specifies the desired protection for the memory page(s) containing part or all of the address range.
- If an access is disallowed by the protection given it, the program receives a SIGSEGV signal.
- prot is a bitwise-or of the following values:
  - **PROT_NON**
  - **PROT_READ**
  - **PROT_WRITE**
  - **PROT_EXEC**
- Useful to find bugs (accessing a wrong memory region, stack etc.)

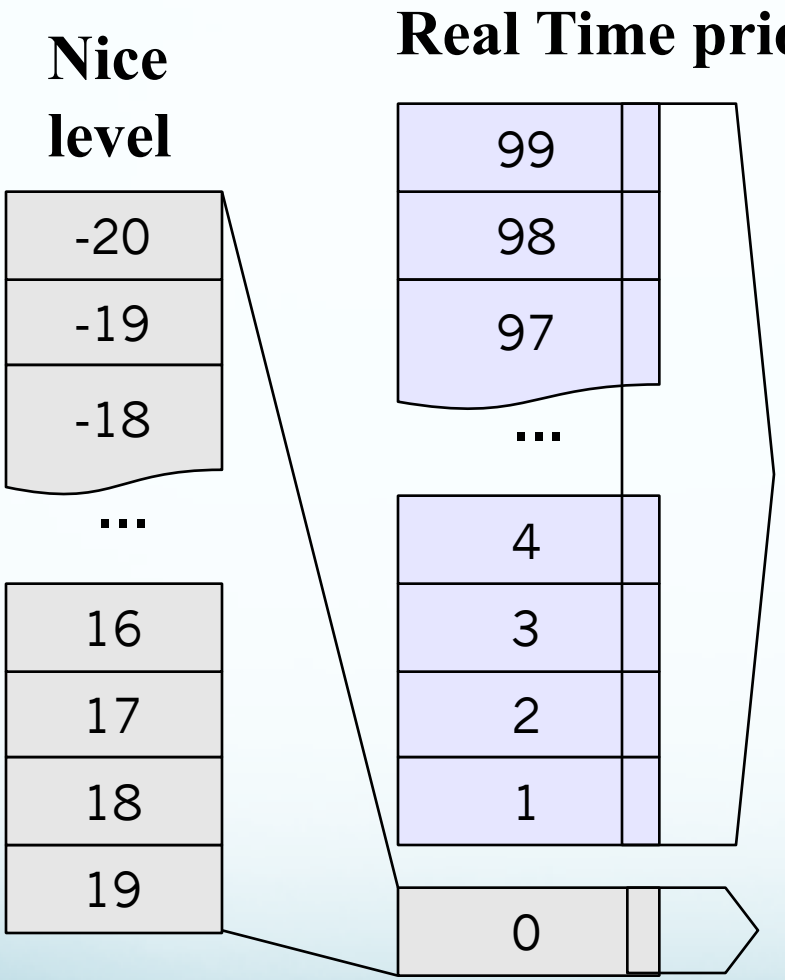
MABEL Technology
Tech The Best

Linux Process Stack

- Linux process stack is auto expanding.
- A default stack (8Mb) is allocated at process creation.
- By default, use of additional stack will automatically trigger allocation of more stack space.
- Not for threads!!!
- This behavior can be limited by setting a resource limit on the stack size.
  - See *setrlimit()*
  - *ulimit -s [size]/unlimited*

MABEL Technology
Tech The Best

Linux Priorities



Real time processes
SCHED_FIFO
SCHED_RR

Non real-time processes
SCHED_OTHER

MABEL Technology
Tech The Best

POSIX Priorities (2)

- **SCHED_OTHER**: Default Linux time-sharing scheduling
  - Priority 0 is reserved for it.
  - Fair, no starvation.
  - Use this for any non time critical tasks.
- **SCHED_FIFO**: First In-First Out scheduling
  - Priorities 1 – 99.
  - Preemptive.
- **SCHED_RR**: Round Robin scheduling
  - Like SCHED_FIFO + time slice

MABEL Technology
Tech The Best

Changing Real Time Priorities

```
int sched_setscheduler(pid_t pid, int policy, const struct  sched_param *p);

struct sched_param {

int sched_priority


};
```

- *sched_setscheduler()* sets both the scheduling policy and the associated parameters for the process identified by pid. If pid equals zero, the scheduler of the calling process will be set. The interpretation of the parameter p depends on the selected policy. Currently, the following three scheduling policies are supported under Linux: SCHED_FIFO, SCHED_RR, and SCHED_OTHER;

- There is also a *sched_getscheduler()*.

MABEL Technology
Tech The Best

Guarantee Real Time Response

- To get deterministic real time response from a Linux process, make sure to:
  - Put the process in a real time scheduling domain using sched_setscheduler()
  - *Use mlockall()* to lock all process memory, both current and future.
  - Pre-fault stack pages
    - To do this call a dummy function that allocates on stack an automatic variable big enough for your entire future stack usage and writes to it.

MABEL Technology
Tech The Best

Real Time Example

```
#define MY_PRIORITY (49)

#define MAX_SAFE_STACK (8*1024)


void stack_prefault(void) {

  unsigned char dummy[MAX_SAFE_STACK];

  memset(&dummy, 0, MAX_SAFE_STACK);

}


int main(int argc, char* argv[])

{

    struct sched_param param;

param.sched_priority =  MY_PRIORITY;

// Continue on next slide... ==>
```

- Some defines.
- This make sure the stack is allocated.
- We fill this structure with out requested priority

MABEL Technology
Tech The Best

Real Time Example

```
if(sched_setscheduler(0, SCHED_FIFO,
&param) == -1) {

       exit(1);

  }


 if(mlockall(MCL_CURRENT|MCL_FUTURE) == -
1) {

    exit(2);

  }


  stack_prefault();


 do_interesting_stuff();

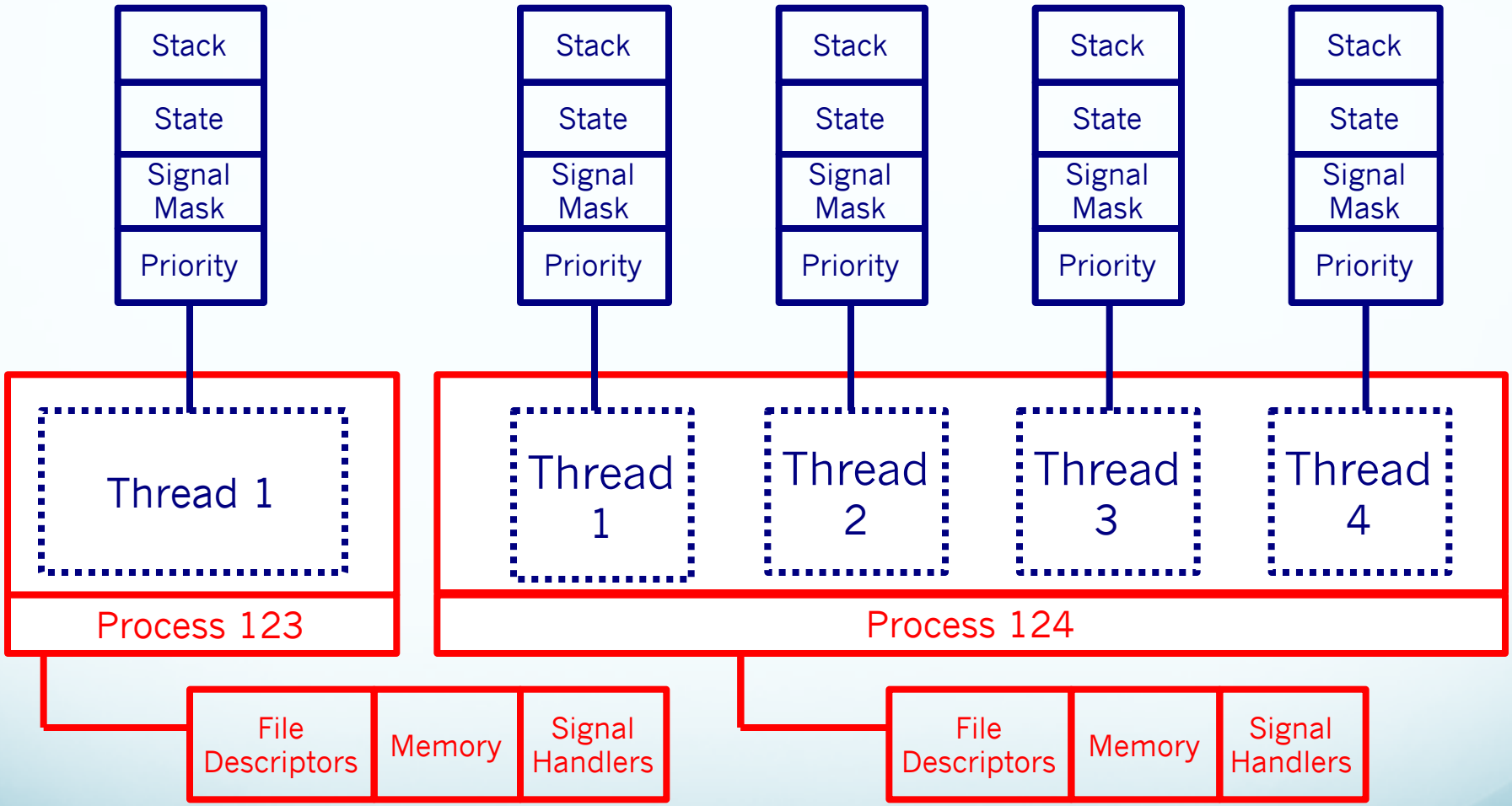
}
```

- Set ourselves as a real time task with FIFO scheduling
- Lock all memory allocation in physical memory, current and future.
- Pre-fault the stack.
- It's now OK to do real time stuff.

MABEL Technology

Tech The Best

Threads and Processes



MABEL Technology
Tech The Best

POSIX Threads

- Linux uses the POSIX Threads threading mechanism.
- Linux threads are Light Weight Processes – each thread is a task scheduled by the kernel's scheduler.
- Process creation time is roughly double than that of a thread's.
- But the context switch time are the same.
- To use threads in your code:
  - `#include <pthread.h>`
- In your Makefile:
  - Add `-lpthread` to `LDFLAGS`

MABEL Technology
Tech The Best

Creating Threads

- Function: **pthread_create()**

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
void *(*start_routine)(void *), void * arg);
```

- The *pthread_create()* routine creates a new thread within a process. The new thread starts in the start routine *start_routine* which has a start argument *arg*. The new thread has attributes specified with *attr*, or default attributes if *attr* is NULL.
- If the *pthread_create()* routine succeeds it will return 0 and put the new thread ID into *thread*, otherwise an error number shall be returned indicating the error.

MABEL Technology
Tech The Best

Creating Thread Attributes

- Function: **pthread_attr_init()**
- int pthread_attr_init(pthread_attr_t *attr);

- Setting attributes for threads is achieved by filling a thread attribute object *attr* of type *pthread_attr_t*, then passing it as a second argument to *pthread_create(3)*. Passing NULL is equivalent to passing a thread attribute object with all attributes set to their default values.
- *pthread_attr_init()* initializes the thread attribute object *attr* and fills it with default values for the attributes.
- Each attribute *attrname* can be individually set using the function *pthread_attr_setattrname()* and retrieved using the function *pthread_attr_getattrname()*.

pthread attribute objects

- Created as a separate object
  - type `pthread_attr_t`
  - Initialise using `pthread_attr_init ()`
  - Set using `pthread_attr_set`*`attribute-name`* `()`
  - Query using `pthread_attr_get`*`attribute-name`* `()`
  - Delete using `pthread_attr_destroy ()`
- Passed as second parameter to pthread_create
  - same attribute object can be used for different threads
  - often defaulted to NULL

```
pthread_t thread_one;

pthread_attr_t myattr;
pthread_attr_init (&myattr);
pthread_attr_setdetachstate (&myattr, PTHREAD_CREATE_DETACHED);

pthread_create (&thread_one, &myattr, mythread_func, NULL);
```

MABEL Technology
Tech The Best

Destroying Thread Attributes

- Function: **pthread_attr_destroy()**
- int pthread_attr_destroy(pthread_attr_t *attr);

- *pthread_attr_destroy()* destroys a thread attribute object, which must not be reused until it is reinitialized. *pthread_attr_destroy()* does nothing in the LinuxThreads implementation.
- Attribute objects are consulted only when creating a new thread. The same attribute object can be used for creating several threads. Modifying an attribute object after a call to *pthread_create()* does not change the attributes of the thread previously created.

MABEL Technology
Tech The Best

Detach State

- Thread Attribute: **detachstate**

- Control whether the thread is created in the joinable state or in the detached state. The default is joinable state.

- In the joinable state, another thread can synchronize on the thread termination and recover its termination code using *pthread_join(3)*, but some of the thread resources are kept allocated after the thread terminates, and reclaimed only when another thread performs *pthread_join(3)* on that thread.

- In the detached state, the thread resources are immediately freed when it terminates, but *pthread_join(3)* cannot be used to synchronize on the thread termination.

- A thread created in the joinable state can later be put in the detached thread using *pthread_detach(3)*.

MABEL Technology
Tech The Best

Sched Policy

- Thread Attribute: **schedpolicy**

- Select the scheduling policy for the thread: one of SCHED_OTHER (regular, non-realtime scheduling), SCHED_RR (realtime, round-robin) or SCHED_FIFO (realtime, first-in first-out).

- Default value: SCHED_OTHER.

- The realtime scheduling policies SCHED_RR and SCHED_FIFO are available only to processes with superuser privileges.

- The scheduling policy of a thread can be changed after creation with *pthread_setschedparam(3)*.

MABEL Technology
Tech The Best

Sched Param

- Thread Attribute: **schedparam**
- Contain the scheduling parameters (essentially, the scheduling priority) for the thread.
- Default value: priority is 0.
- This attribute is not significant if the scheduling policy is SCHED_OTHER; it only matters for the realtime policies SCHED_RR and SCHED_FIFO.
- The scheduling priority of a thread can be changed after creation with *pthread_setschedparam(3)*.
- For SCHED_OTHER policy use setpriority(2)

MABEL Technology
Tech The Best

Inherit Sched

- Thread Attribute: **inheritsched**
- Indicate whether the scheduling policy and scheduling parameters for the newly created thread are determined by the values of the schedpolicy and schedparam attributes (PTHREAD_EXPLICIT_SCHED) or are inherited from the parent thread (value PTHREAD_INHERIT_SCHED).
- Default value: PTHREAD_EXPLICIT_SCHED.

MABEL Technology
Tech The Best

Destroying Threads

- Function: **pthread_exit()**
- void pthread_exit(void *status);
- The *pthread_exit()* routine terminates the currently running thread and makes *status* available to the thread that successfully joins, *pthread_join()*, with the terminating thread. In addition, *pthread_exit()* executes any remaining cleanup handlers in the reverse order they were pushed, *pthread_cleanup_push()*, after which all appropriate thread specific destructors are called.
- An implicit call to *pthread_exit()* is made if any thread, other than the thread in which main() was first called, returns from the start routine specified in *pthread_create()*.

MABEL Technology
Tech The Best

Thread and Process Termination

Action	*Main Thread*	*Other Thread*
pthread_exit()	Thread terminates	Thread terminates
exit()	All threads terminate	All threads terminate
Thread function returns	**All threads terminate**	**Thread terminates**

MABEL Technology
Tech The Best

Thread Cancellation

- Any thread may send a cancellation request to another
  - call `pthread_cancel()`
  - cancelled thread
    - terminates at the next *cancellation point*
    - returns `PTHREAD_CANCELED`
- Threads may *ignore* cancellation
  - `pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);`
- Threads may *defer* cancellation
  - `pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);`
  - defers cancellation request until the *next* cancellation point
- Cleanup routines tidy resources on cancellation

```
pthread_cleanup_push (mycleanup, NULL);
...
pthread_cleanup_pop (1);    /* 1 executes cleanup */
```

MABEL Technology
Tech The Best

Waiting For a Thread to Finish

- Function: **pthread_join()**
- int pthread_join(pthread_t thread, void **status);

- If the target thread thread is not detached and there are no other threads joined with the specified thread then the *pthread_join()* function suspends execution of the current thread and waits for the target thread thread to terminate. Otherwise the results are undefined.
- On a successful call *pthread_join()* will return 0, and if status is non NULL then status will point to the status argument of *pthread_exit()*. On failure *pthread_join()* will return an error number indicating the error.
- Also exists *pthread_tryjoin_np()* and *pthread_timedjoin_np()*.

MABEL Technology
Tech The Best

Threads Considered Harmful

- Threads were created to allow for a light-wight process on OS were process creation time is very high but Linux processes are light weight, thanks to CoW.
- Threads share too much and using them nulls the advantage of having protection between tasks. Partial sharing can easily be done using shared memory between processes.
- There is nothing you can do with threads you can't do with processes.

Thread Safety

- Any off-stack data is a potential trap
  - that's globals, statics, and anything on the heap
  - more than one thread may update the same item
  - a 'test then set' action is not atomic

- Is the C run-time library safe?
  - errno is OK, but do you need -D_REENTRANT ?
  - are there reentrant versions (`_r`)?
  - If you don't know, play safe

MABEL Technology

Tech The Best

Thread Safety continued …

- Is this thread safe?

```
#include <pthread.h>
#include <stdio.h>
#include <string.h>

void * mythread_func (void *pId)
{
   char sztr[]= "A B C D E F G H I J K";
   char *pstr;
   int *id = (int *)pId;

   pstr = strtok (sztr, " ");

   do
   {
      printf ("Thread %d <%c>\n", *id, *pstr);
       getchar ();

   } while (pstr = strtok (NULL, " "));

}  /* mythread_func */
```

MABEL Technology
Tech The Best

Thread Specific Data

- Allows different threads to have different thread-specific values for the same logical data item
- Thread 1 responsible for:
  - pthread_key_create( )/pthread_key_delete( )
- Any thread:
  - pthread_setspecific( )
  - pthread_getspecific( )

MABEL Technology

Tech The Best

Thread-safe `strtok_r` - main

```
void * mythread_func (void *pId);

pthread_key_t g_Key;

int main (int argc, char *argv[])
{
   pthread_t thread_one, thread_two;
   int one = 1;
   int two = 2;

   pthread_key_create (&g_Key, NULL);

   pthread_create (&thread_one, NULL, mythread_func, &one);
   pthread_create (&thread_two, NULL, mythread_func, &two);

   pthread_join (thread_one, NULL);
   pthread_join (thread_two, NULL);

   pthread_key_delete (g_Key);

}  /* main */
```

MABEL Technology
Tech The Best

Thread-safe strtok_r - function

```
void * mythread_func (void *pId)
{
   char sztr[]= "A B C D E F G H I J K";
   char *pstr;
   int *id = (int *)pId;

   char *pchBuffer = (char *)malloc (strlen(sztr)+1);
   pthread_setspecific (g_Key, pchBuffer);

   pstr = strtok_r (sztr, " ", pthread_getspecific (g_Key));

   do
   {
      printf ("Thread %d <%c>: Press any key get next item...\n",
              *id, *pstr);
      getchar ();
   } while (pstr = strtok_r (NULL, " ", pthread_getspecific (g_Key)));

   free (pchBuffer);

}  /* mythread_func */
```

MABEL Technology
Tech The Best

Process VMA

- You can view a user application mapping from the proc file system:

- `> cat /proc/1/maps` (init process)

```
start            end              perm  offset   major:minor inode   mapped file name
00771000-0077f000 r-xp 00000000 03:05 1165839    /lib/libselinux.so.1
0077f000-00781000 rw-p 0000d000 03:05 1165839    /lib/libselinux.so.1
0097d000-00992000 r-xp 00000000 03:05 1158767    /lib/ld-2.3.3.so
00992000-00993000 r--p 00014000 03:05 1158767    /lib/ld-2.3.3.so
00993000-00994000 rw-p 00015000 03:05 1158767    /lib/ld-2.3.3.so
00996000-00aac000 r-xp 00000000 03:05 1158770    /lib/tls/libc-2.3.3.so
00aac000-00aad000 r--p 00116000 03:05 1158770    /lib/tls/libc-2.3.3.so
00aad000-00ab0000 rw-p 00117000 03:05 1158770    /lib/tls/libc-2.3.3.so
00ab0000-00ab2000 rw-p 00ab0000 00:00 0
08048000-08050000 r-xp 00000000 03:05 571452     /sbin/init (text)
08050000-08051000 rw-p 00008000 03:05 571452     /sbin/init (data, stack)
08b43000-08b64000 rw-p 08b43000 00:00 0
f6fdf000-f6fe0000 rw-p f6fdf000 00:00 0
fefd4000-ff000000 rw-p fefd4000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```

MABEL Technology
Tech The Best

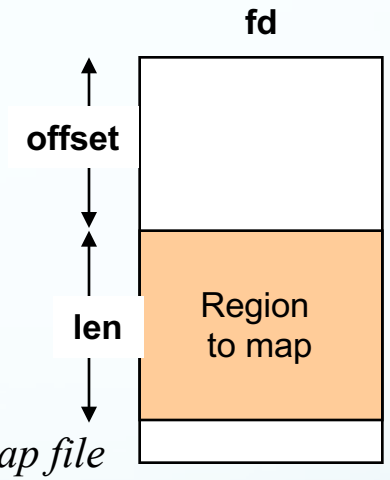
Mapping Files

```
#include <sys/mman.h>
```

- To map a file into a process' address space:

  caddr_t mmap(caddr_t addr, size_t len, int prot, int flags, int fd, off_t offset)

addr	*address to map file at (0 to let kernel decide)*	
len	*length of mapped area in bytes*	
prot	*PROT_READ*	*attach read only*
	PROT_WRITE	*read/write (file must be open for write)*
flags	*MAP_SHARED*	*shared data backed up to file*
	MAP_PRIVATE	*private copy backed up to swap file*
	MAP_FIXED	*forces kernel to use specified address*
fd	*open file descriptor of file to map*	
offset	*offset into file for start of mapping*	
returns	*mapped address or -1 on error*	



MABEL Technology
Tech The Best

Example - File Copy

mmapcp.c

```
int main(int argc, char *argv[])
{
  int target, source, len;
  char *src, *dest;
  struct stat statbuf;

  if (argc != 3)
    {fprintf(stderr,"Usage: %s source dest\n",argv[0]);exit(1);}

  source = open (argv[1], O_RDONLY);
  fstat (source, &statbuf);
  len = statbuf.st_size;

  target = open(argv[2],O_RDWR|O_CREAT,statbuf.st_mode&PERMS);
  ftruncate (target, len);

  src = mmap (0,len,PROT_READ,MAP_PRIVATE,source,0);
  dest = mmap (0,len,PROT_READ|PROT_WRITE,MAP_SHARED,target,0);
  if (src == NOMAP) { perror("can't map source file"); exit(1); }
  if (dest == NOMAP){ perror("can't map target file"); exit(1); }
  close (source); close (target);

  memcpy (dest, src, len);
}
```

```
#define NOMAP ((caddr_t) -1)
#define PERMS (S_IRWXU|S_IRWXG|S_IRWXO)
```

MABEL Technology

Tech The Best

Example - A file of records

- Imagine file contains a series of equal sized records

```
int main(int argc, char *argv[])
{
  int dataFile, len, recIndex;
  struct rec *records;
  struct stat statbuf;

  dataFile = open ("data_file", O_RDWR);
  fstat (dataFile, &statbuf);
  len = statbuf.st_size;
  records = (struct rec *) mmap (
              0,len,PROT_READ|PROT_WRITE,MAP_SHARED,dataFile,0);
  if (records == NOMAP){ perror("can't map data file"); exit(1); }
  close (dataFile);

  while ( …not finished… ) {
    /* Get record index number from some key value */
    records[recIndex].val1 = new value;
    records[recIndex].val2 = new value;
   }
}
```

```
struct rec {
  int val1;
  int val2;
}
```

MABEL Technology
Tech The Best

Operating on mapped files

- To unmap a part of an address space:
  int munmap (caddr_t addr, int len)
  addr start of area to unmap
  len amount of data to unmap

- To synchronise memory with disk file:
  int msync (caddr_t addr, int len, int flags)
  addr *start of area to sync*
  len *amount of data to sync*
  flags *MS_SYNC wait (don't return) until disk I/O is complete*
  MS_ASYNC don't wait for disk I/O to complete
  MS_INVALIDATE force next read to read from disk copy

- To alter (reduce) access on memory area
  int mprotect (caddr_t addr, int len, int prot)
  addr *start of area to change*
  len *amount of data to change*
  prot *PROT_READ, PROT_WRITE, PROT_EXECUTE*

Operating on mapped files

- To lock a portion of a process address space in memory and prevent paging (superuser only):
  int mlock (caddr_t addr, int len)
  - addr *start of area to lock*
  - len *amount of data to lock*

- To alter read-ahead behaviour on a region of memory:
  int madvise (caddr_t addr, int len, int advice)
  - addr *start of region to advise on*
  - len *length of region to advise on*
  - advice *pattern of access on specified region:*
    - *MADV_SEQUENTIAL pages will be accessed sequentially*
    - *MADV_RANDOM pages will be accessed randomly*
    - *MADV_WILLNEED request to make bring region into memory*
    - *MADV_DONTNEED finished with specified region*

`madvise()` is not on all platforms

MABEL Technology
Tech The Best

Memory Allocation

- malloc(3)
- alloca(3)
- obstacks

MABEL Technology
Tech The Best

Linux IPC

- **I**nter **P**rocess (and thread) **C**ommunication
  - Mutexes
  - Condition Variables
  - Semaphores
  - Shared memory
  - Message Queues
  - Pipes
  - Unix Domain Sockets
  - Signals

MABEL Technology
Tech The Best

Creating a Mutex

- Function: **pthread_mutex_init()**

- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

- pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

- The *pthread_mutex_init()* routine creates a new mutex, with attributes specified with *attr*, or default attributes if *attr* is NULL.

- If the *pthread_mutex_init()* routine succeeds it will return 0 and put the new mutex ID into *mutex*, otherwise an error number shall be returned indicating the error.

MABEL Technology
Tech The Best

Mutex Attributes

- int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
- int pthread_mutexattr_init(pthread_mutexattr_t *attr);
  - Init and destroy attribute structure.
- int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int pshared);
  - Permit a mutex to be operated upon by any task that has access to the memory where the mutex is allocated.
- More ahead…

MABEL Technology
Tech The Best

Destroying a Mutex

- Function: **pthread_mutex_destroy()**
- int pthread_mutex_destroy(pthread_mutex_t *mutex);
- The *pthread_mutex_destroy()* routine destroys the mutex specified by *mutex*.
- If the *pthread_mutex_destroy()* routine succeeds it will return 0, otherwise an error number shall be returned indicating the error.

MABEL Technology
Tech The Best

Locking Mutexes

- Function: **pthread_mutex_lock()**
- int pthread_mutex_lock(pthread_mutex_t *mutex);
- The *pthread_mutex_lock()* routine shall lock the mutex specified by *mutex*. If the mutex is already locked, the calling thread blocks until the mutex becomes available.
- If the *pthread_mutex_lock()* routine succeeds it will return 0, otherwise an error number shall be returned indicating the error.

Function: **pthread_mutex_trylock()**

int pthread_mutex_trylock(pthread_mutex_t *mutex);

- The *pthread_mutex_trylock()* routine shall lock the mutex specified by *mutex* and return 0, otherwise an error number shall be returned indicating the error. In all cases the *pthread_mutex_trylock()* routine will not block the current running thread.

MABEL Technology
Tech The Best

Locking Mutexes

- Function: **pthread_mutex_timedlock()**
- int pthread_mutex_timedlock(pthread_mutex_t *mutex, struct timespec *restrict abs_timeout);
- The *pthread_mutex_timedlock()* function shall lock the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the mutex becomes available as in the *pthread_mutex_lock()* function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.
- The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds abs_timeout), or if the absolute time specified by *abs_timeout* has already been passed at the time of the call.

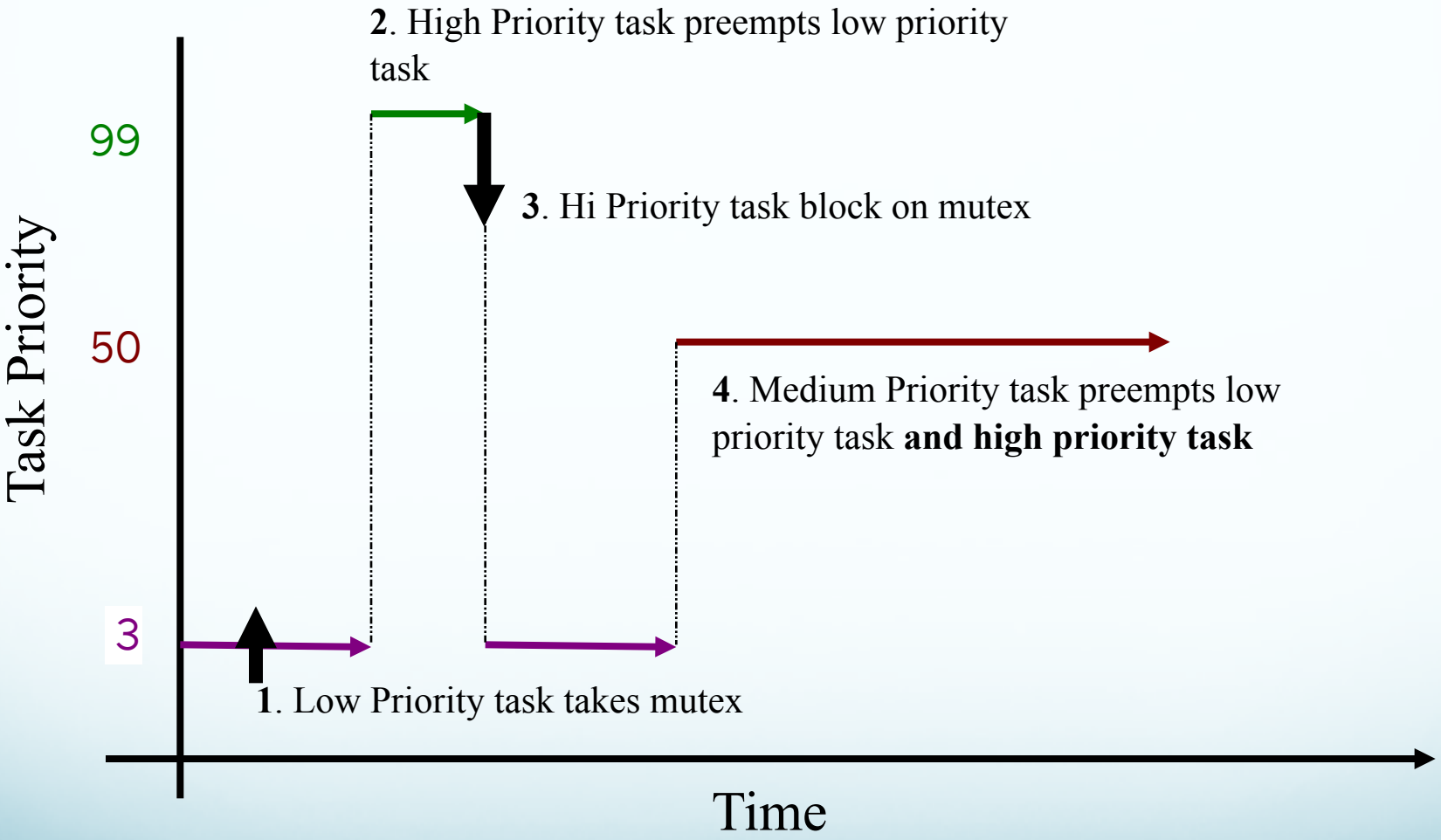
MABEL Technology
Tech The Best

Unlocking Mutexes

- Function: **pthread_mutex_unlock()**
- int pthread_mutex_unlock(pthread_mutex_t *mutex);
- If the current thread is the owner of the mutex specifed by *mutex*, then the *pthread_mutex_unlock()* routine shall unlock the mutex. If there are any threads blocked waiting for the mutex, the scheduler will determine which thread obtains the lock on the mutex, otherwise the mutex is available to the next thread that calls the routine *pthread_mutex_lock()*, or *pthread_mutex_trylock()*.
- If the *pthread_mutex_unlock()* routine succeeds it will return 0, otherwise an error number shall be returned indicating the error.

MABEL Technology
Tech The Best

Priority Inversion



MABEL Technology
Tech The Best

Priority Inheritance and Ceilings

- Priority inheritance and ceilings are methods to protect against priority inversions.
- Linux only got support for them in 2.6.18.
- Patches to add support for older version exists.
  - Embedded Linux vendors usually provide patched kernels.
- If the kernel version you're using is not patched, make sure to protect against this scenario in design
  - One possible way: raise the priority of each tasks trying to grab a mutex to the maximum priority of all possible contenders.

MABEL Technology
Tech The Best

Setting Mutex Priority Protocol

- `int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int protocol);`
  - Set priority protocol: PTHREAD_PRIO_NONE, PTHREAD_PRIO_INHERIT PTHREAD_PRIO_PROTECT

`int pthread_mutexattr_setprioceiling(pthread_mutex_t *restrict mutex, int prioceiling, int * oldceiling);`

- **prioceiling**: which priority ceiling to use.

- Can also set in run time with pthread_mutex_setprotocol(3) and pthread_mutex_setprioceiling(3) calls.

MABEL Technology
Tech The Best

Condition Variables

- Condition variables are queues where tasks can wait blocking until an a certain condition becomes true.
  - A condition is a predicate, like in: if(x==0) ...
- Condition variables use mutexes to protect the testing and setting of the condition.
- The condition variable API allows a task to **safely** test if the condition is true, blocking if it isn't, without race conditions.
- Use of condition variables can help avoid many subtle bugs.

MABEL Technology
Tech The Best

Condition Variables API

- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);`
- `int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);`
- `int pthread_cond_signal(pthread_cond_t *cond);`
- `int pthread_cond_broadcast(pthread_cond_t *cond);`

- The first two function calls are used for waiting on the condition var.
- The latter two function calls are used to wake up waiting tasks:
  - *pthread_cond_signal()* wakes up one single task that is waiting for the condition variable.
  - *pthread_cond_broadcast()* wakes up all waiting tasks.

MABEL Technology
Tech The Best

Condition Variable Usage Example

```
void my_wait_for_event(pthread_mutex_t *lock, pthread_cond_t *cond) {
            pthread_mutex_lock(lock);
            while (flag == 0)
                    pthread_cond_wait(cond, lock);
            flag = 0;
            pthread_mutex_unlock(lock);
}

void my_post_event(pthread_mutex_t *lock, pthread_cond_t *cond) {
            pthread_mutex_lock(lock);
            flag = 1;
            pthread_cond_signal(cond);
            pthread_mutex_unlock(lock);
}
```

The condition variable function calls are used within an area protected by the mutex that belong to the condition variable. The operating system releases the mutex every time it blocks a task on the condition variable; and it has locked the mutex again when it unblocks the calling task from the signaling call.

MABEL Technology
Tech The Best

POSIX Semaphores

- POSIX Semaphores also available:
- `#include <semaphore.h>`
- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
- `int sem_wait(sem_t *sem);`
- `int sem_trywait(sem_t *sem);`
- `int sem_post(sem_t *sem);`
- `int sem_getvalue(sem_t *sem, int *sval);`
- `int sem_destroy(sem_t *sem);`
- The *pshared* flag controls whether the semaphore can be used from different processes via shared memory.

MABEL Technology
Tech The Best

Even More Locks

- **POSIX spin locks**
  - pthread_spin_lock() ...
- **Reader Writer locks**
  - pthread_rwlock_rdlock()
  - pthread_rwlock_wrlock() ...

POSIX Shared Memory

- int shm_open(const char *name, int oflag, mode_t mode);

- *shm_open()* creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to *mmap(2)* the same region of shared memory. The *shm_unlink()* function performs the converse operation, removing an object previously created by *shm_open()*.

- The operation of *shm_open()* is analogous to that of *open(2)*. *name* specifies the shared memory object to be created or opened. For portable use, *name* should have an initial slash (/) and contain no embedded slashes.

- Tip: make sure to reserve the size of the shared memory object using *ftruncate()*!

MABEL Technology
Tech The Best

POSIX Message Queue

- POSIX message queues allow processes to exchange data in the form of messages.
- Message queues are created and opened using mq_open(3);
  - The returns a queue descriptor (mqd_t).
  - Each queue is identified by a name of the form /somename.
- Messages are transferred to and from a queue using mq_send(3) and mq_receive(3).
- Queue are closed using mq_close(3), and deleted using mq_unlink(3).

MABEL Technology
Tech The Best

Getting A Queue

- #include <mqueue.h>
- mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr);
  - **O_RDONLY, O_WRONLY, O_RDWR**.
  - **O_NONBLOCK**: Open the queue in non-blocking mode.
  - **O_CREAT**: Create the message queue if it does not exist.
  - **O_EXCL**: If a queue already exists, then fail.
- The mode argument provides permissions.

MABEL Technology
Tech The Best

Attributes

- mqd_t mq_getattr(mqd_t mqdes, struct mq_attr *attr);
- mqd_t mq_setattr(mqd_t mqdes, struct mq_attr *newattr, struct mq_attr *oldattr);
- Retrieve and modify attributes of the queue referred to by the descriptor mqdes.

```
struct mq_attr {
    long mq_flags;       /* Flags: 0 or O_NONBLOCK */
    long mq_maxmsg;      /* Max. # of messages on queue */
    long mq_msgsize;     /* Max. message size (bytes) */
    long mq_curmsgs;     /* # of messages currently in queue */
};
```

MABEL Technology
Tech The Best

Sending Messages

- mqd_t mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio);
- mqd_t mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio, const struct timespec *abs_timeout);
- mq_send() adds the message msg_ptr to the queue.
- msg_prio specifies the message priority.
  - FIFO is used for same priority messages.
- If queue is full send is blocked, unless O_NONBLOCK used.

MABEL Technology
Tech The Best

Receiving Messages

- mqd_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned *msg_prio);
- mqd_t mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t, msg_len, unsigned *msg_prio, const struct timespec *abs_timeout);
- mq_receive() removes the oldest message with the highest priority from the queue, and places it in the buffer.
- If prio is not NULL, then it is used to return the priority.
- If the queue is empty, mq_receive() blocks unless O_NONBLOCK was used.

MABEL Technology
Tech The Best

Message Notification

- mqd_t mq_notify(mqd_t mqdes, const struct sigevent *notification);
- mq_notify() allows the process get an asynchronous notification when a new message arrives.
- Use a NULL notification to un-register.
- Notification only occurs for new messages on empty queue.
- mq_receive takes precedence over notifications.
- Notification occurs once: after a notification is delivered, the notification registration is removed.

MABEL Technology
Tech The Best

Notifications

```
struct sigevent {
    int    sigev_notify;       /* Notification method */
    int    sigev_signo;        /* Notification signal */
    union  sigval sigev_value; /* Data passed with notification */
    void   (*sigev_notify_function) (union sigval);
                               /* Function for thread notification */
    void   *sigev_notify_attributes;
                               /* Thread function attributes */
};
```

- **Notification methods:**
  - **SIGEV_NONE**: no notifications
  - **SIGEV_SIGNAL**: send signal sigev_signo with value sigev_value.
  - **SIGEV_THREAD**: start thread with function sigev_notify_function and sigev_value parameter

MABEL Technology
Tech The Best

POSIX Pipes

- `int pipe(int filedes[2]);`

- *pipe(2)* creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by *filedes*.

- *filedes[0]* is for reading, *filedes[1]* is for writing.

```
int mkfifo(const char *pathname, mode_t mode);
```

- A fifo(7) is a special file that behaves like a pipe.

- Since fifo(7) exists in the file system, it can be opened by two non related processes.

MABEL Technology
Tech The Best

Signals

- Signals are asynchronous notifications sent to a process by the kernel or another process
- Signals interrupt whatever the process was doing at the time to handle the signal.
- Two default signal handlers exist:
  - **SIG_IGN**: Ignore the specified signal.
  - **SIG_DFL**: Go back to default bahviour.
- Or you can register a call back function that gets called when the process receives that signal.

MABEL Technology
Tech The Best

Regular Signals

SIGHUP	Hangup detected on controlling terminal
SIGINT	Interrupt from keyboard
SIGQUIT	Quit from keyboard
SIGILL	Illegal Instruction
SIGABRT	Abort signal from abort(3)
SIGFPE	Floating point exception
SIGKILL	Kill signal
SIGSEGV	Invalid memory reference
SIGPIPE	Broken pipe: write to pipe with no readers
SIGALRM	Timer signal from alarm(2)
SIGTERM	Termination signal
SIGUSR1	User-defined signal 1
SIGUSR2	User-defined signal 2
SIGCHLD	Child stopped or terminate
SIGCONT	**Continue if stopped**
SIGSTOP	**Stop process**
SIGTSTP	**Stop typed at tty**
SIGTTIN	**tty input for background process**
SIGTTOU	**tty output for background process**
SIGBUS	**Bus error (bad memory access)**
SIGPOLL	**Pollable event (Sys V). Synonym of SIGIO**
SIGPROF	**Profiling timer expired**
SIGSYS	**Bad argument to routine (SVID)**
SIGTRAP	**Trace/breakpoint trap**
SIGURG	**Urgent condition on socket (4.2 BSD)**
SIGIO	**I/O now possible (4.2 BSD)**

See *signal(7)* for default behaviors.

There are two signal handlers you cannot modify or ignore – SIGKILL and SIGSTOP.

MABEL Technology
Tech The Best

Real Time Signals

- Additional 32 signals from SIGRTMIN to SIGRTMAX
- No predefined meaning...
  - But LinuxThreads lib makes use of the first 3.
- Multiple instances of the same signals are queued.
- Value can be sent with the signal.
- Priority is guaranteed:
  - Lowest number real time signals are delivered first. Same signals are delivered according to order they were sent.
  - Regular signals have higher priority then real time signals.

MABEL Technology
Tech The Best

Signal Action

- int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
  - Register a signal handler.
    - *signum*: signal number.
    - *act*: pointer to new *struct sigaction*.
    - *oldact*: pointer to buffer to be filled with current *sigaction* (or NULL, if not interested).

MABEL Technology
Tech The Best

Signal Action cont.

- The *sigaction* structure is defined as something like:
  - struct sigaction {

```
        void (*sa_handler)(int);
        void (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t sa_mask;
        int sa_flags;
        ...
}
```

  - *sa_mask* gives a mask of signals which should be blocked during the execution of the signal handler.
    - The signal which triggered the handler will also be blocked, unless the SA_NODEFER or SA_NOMASK flags are used.

MABEL Technology
Tech The Best

Real Time Signals Flags

- *sa_flags* can be used to pass flags to change behavior:
  - **SA_ONESHOT:** Restore the signal action to the default state once the signal handler has been called.
  - **SA_RESTART:** Make blocking system calls restart automatically aftyer a signal is received.
  - **SA_NODEFER:** Do not prevent the signal from being received from within its own signal handler.
  - **SA_SIGINFO:** The signal handler takes 3 arguments, not one. In this case, *sa_sigaction* should be set instead of *sa_handler*.
    - For details about *siginfo_t* structure, see *sigaction(2)*.

MABEL Technology
Tech The Best

Sending Signals

- int sigqueue(pid_t pid, int sig, const union sigval value);
  - Queue signal to process.
  - *pid* is the process ID to send the signal to.
  - *sig* is the signal number.
  - *sigval* is:

```
union sigval {
            int   sival_int;
            void *sival_ptr;

};
```

  - The *sigval* is available to the handler via the *sig_value* field of *siginfo_t*.

MABEL Technology
Tech The Best

Signal Masking

- The *sigprocmask()* call is used to change the list of currently blocked signals.

```
int sigprocmask(int how, const sigset_t *set, sigset_t
    *oldset);
```

  - The behavior of the call is dependent on the value of *how*, as follows:
    - **SIG_BLOCK:** The set of blocked signals is the union of the current set and the set argument.
    - **SIG_UNBLOCK:** The signals in set are removed from the current set of blocked signals.
    - **SIG_SETMASK**: The set of blocked signals is set to the argument set.

MABEL Technology
Tech The Best

Signal Sets

- These functions allow the manipulation of POSIX signal sets:
  - int **sigemptyset**(sigset_t *set);
    - Initializes the signal set given by *set* to empty, with all signals excluded from the set.
  - int **sigfillset**(sigset_t *set);
    - Initializes set to full, including all signals.
  - int **sigaddset**(sigset_t *set, int signum);
  - int **sigdelset**(sigset_t *set, int signum);
    - Add and delete respectively signal *signum* from set.

MABEL Technology
Tech The Best

Example - Using Signal Masks

sigwait.c

```
int main ()
{
    sigset_t newset, pending;

    sigemptyset (&newset);
    sigaddset (&newset,SIGINT);
    sigaddset (&newset,SIGQUIT);
    sigprocmask (SIG_BLOCK,&newset,NULL);

    printf ("Press ^C or ^\\ in the next 5 seconds\n");
    sleep(5);

    sigpending (&pending);
    if (sigismember(&pending,SIGQUIT))
        printf ("SIGQUIT is pending");
    if (sigismember(&pending,SIGINT))
        printf ("SIGINT is pending");

    return 0;
}
```

MABEL Technology
Tech The Best

Example - Signal Handlers

sigusr.c

```
int main (int argc, char *argv[])
{
  struct sigaction new;
  sigset_t emptymask, procmask;

  printf("%d\n", getpid());
  sigemptyset (&emptymask);
  new.sa_handler = handler;
  new.sa_mask    = emptymask;
  new.sa_flags   = 0;
  sigaction(SIGUSR1,&new,NULL);
  sigaction(SIGUSR2,&new,NULL);
  sigemptyset (&procmask);
  sigaddset (&procmask, SIGINT);

  /* wait for signals */
  for (;;) {
    sigsuspend(&procmask);
    write(1, "..round again..\n", strlen("..round again..\n"));
  }
}
```

```
void handler (int id)
{
  if (id == SIGUSR1)
    write( 1, "SIGUSR1\n",
          strlen("SIGUSR1\n") );
  if (id == SIGUSR2)
    write( 1, "SIGUSR2\n",
          strlen("SIGUSR2\n") );
}
```

MABEL Technology
Tech The Best

Signals & Threads

- Signal masks are **per thread**.
- Signal handlers are **per process**.
- Exception signals (SIGSEGV, SIGBUS...) will be caught by **thread doing the exception**.
- Other signals will be caught by **any thread in the process** whose mask does not block the signal – use *pthread_sigmask()* to modify the thread's signal mask.
- **Tip**: Use a "signal handler" thread that does *sigwait(3)* to make thread catching less random!

MABEL Technology
Tech The Best

Async. Signal Safety

- Must be very careful to write signal handler only with async safe code.
  - For example, no printf(), malloc(), any function that takes lock or functions calling them.
- Can only use POSIX async-signal safe functions
  - See signal(7) for the list.
- Typical signal handler just sets a flag.

MABEL Technology
Tech The Best

POSIX Timers Overview

- #include <signal.h>
- #include <time.h>
- int timer_create(clockid_t clockid, struct sigevent *restrict evp, timer_t *restrict timerid);
- int timer_settime(timer_t timerid, int flags, const struct itimerspec *restrict value, struct itimerspec *restrict ovalue);
- int timer_gettime(timer_t timerid, struct itimerspec *value);
- int timer_getoverrun(timer_t timerid);
- int timer_delete(timer_t timerid);

MABEL Technology
Tech The Best

Timer Creation

- The timer_create() function creates a timer using with specified clock, clock_id, as the timing base.
- timer_create() returns, in the location referenced by timerid, a timer ID which identifies the timer in timer requests.
  - The timer is disarmed state upon return from timer_create().
- The evp argument is a sigevent structure which defines how the application is notified when the timer expires.
  - If evp is NULL, the process will get the default signal (SIGALRM).

MABEL Technology
Tech The Best

Clocks

- **CLOCK_REALTIME** System wide wall clock.
- **CLOCK_MONOTONIC** Monotonic time. Cannot be set.
- **CLOCK_PROCESS_CPUTIME_ID** Hi-res per-process timer.
- **CLOCK_THREAD_CPUTIME_ID** Thread-specific timer.
- **CLOCK_REALTIME_HR** Hi-res *CLOCK_REALTIME.*
- **CLOCK_MONOTONIC_HR** Hi-res *CLOCK_MONOTONIC*.

MABEL Technology
Tech The Best

Arming

- timer_gettime() reads the current timer value.
  - The value is returned as the interval until timer expiration, even if the timer was armed with absolute time
- timer_settime() sets the time until the next expiration of the timer.
- If the TIMER_ABSTIME flag is not set, the time will be relative, otherwise it will be absolute.
- The timer_getoverrun() function shall return the timer expiration overrun count for the specified timer.

MABEL Technology

Tech The Best

Time

```
struct itimerspec {

struct timespec it_interval;    /* timer period */

struct timespec it_value;       /* timer expiration */

};

struct timespec {

time_t  tv_sec;          /* seconds */

long    tv_nsec;         /* nanoseconds */

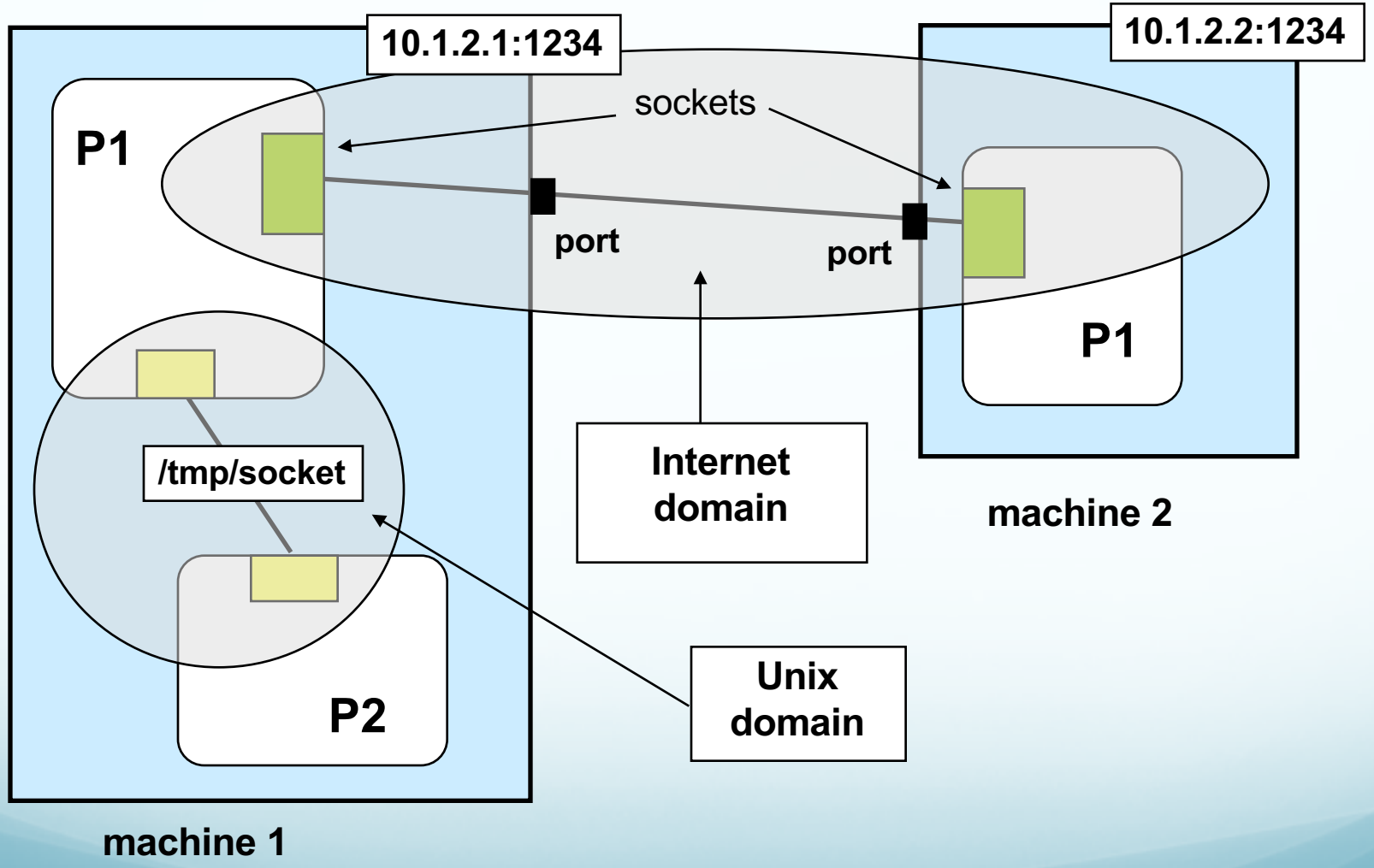
};
```

The timer is armed with it_value and re-armed when expires with it_interval.

For one shot timer leave interval zero.

MABEL Technology
Tech The Best

Sockets for IPC



MABEL Technology
Tech The Best

Domains & Socket Types

- Communication domain
  - "Address Families"
  - Defines socket naming convention and communications protocols

AF_UNIX	**Unix**
AF_INET	**Internet**

  - Other domains available
- Socket type
  - Defines how data is transferred through the socket

SOCK_DGRAM	**datagram (connectionless)** **like the sending a letter** **efficient but no error handling**
SOCK_STREAM	**virtual circuit (connection oriented)** **like making a telephone call** **reliable but often substantial overhead**

  - Other socket types may be available, dependent on domain

MABEL Technology
Tech The Best

Unix Domain

- Supports Stream and Datagram sockets
- SOCK_STREAM
  - Byte stream
  - Bi-directional
- SOCK_DGRAM
  - Message stream
  - Bi-directional
- Socket are identified by filename

```
struct sockaddr {
  short sa_family;
  char  sa_data[14];
}
```

```
struct sockaddr_un
{
    short   sun_family;
    char    sun_path[108];
}
```

```
struct sockaddr_un sock = { AF_UNIX, "/tmp/socket" };
```

MABEL Technology
Tech The Best

Internet Domain

- Supports SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- SOCK_STREAM
  - Uses TCP
- SOCK_DGRAM
  - Uses UDP
- SOCK_RAW
  - Access to IP/ICMP
  - Not usually for applications
- Socket names are 48 bit values
  - 32 bit Internet address
  - 16 bit port number

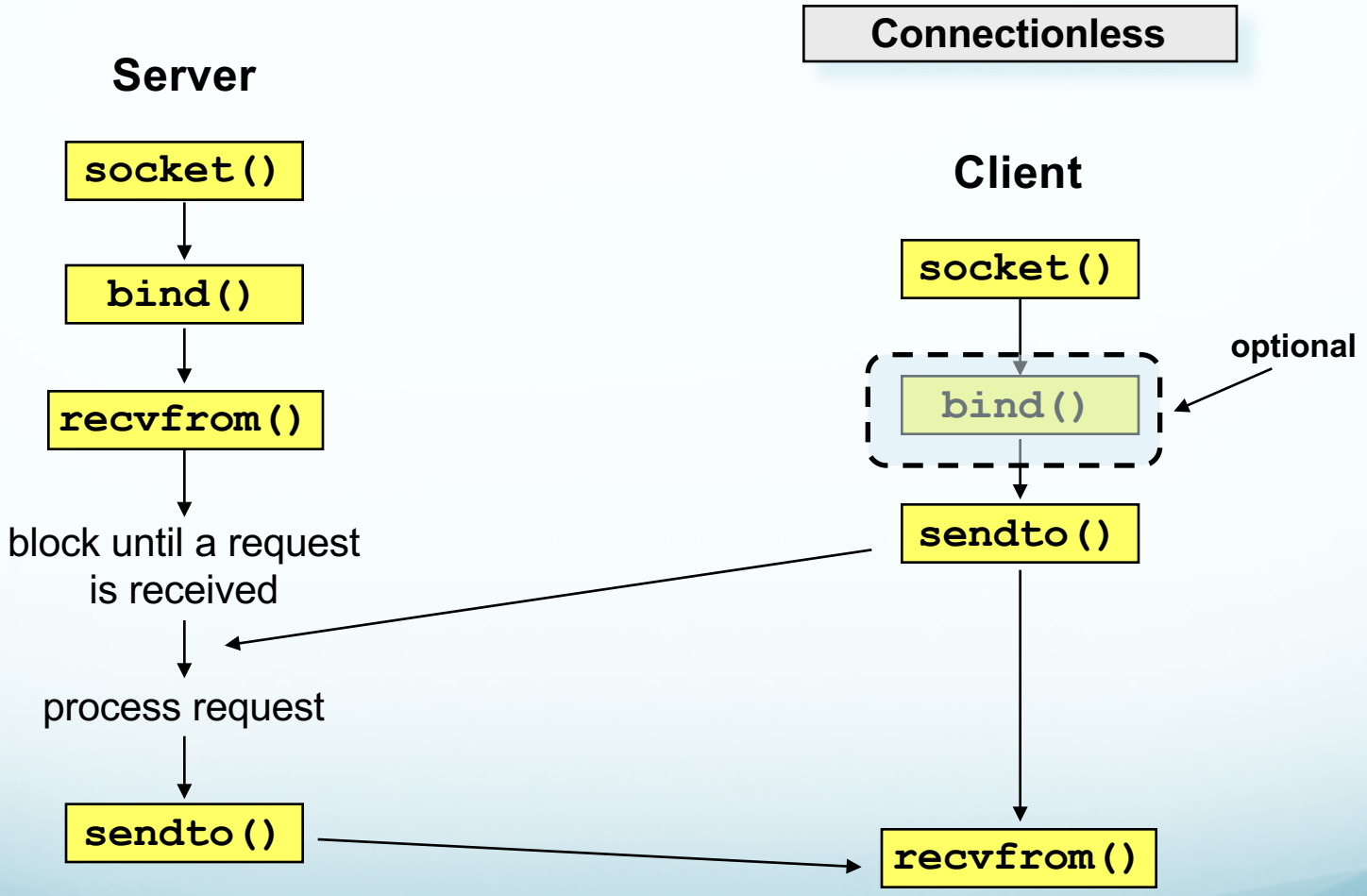
```
struct sockaddr {
  short sa_family;
  char sa_data[14];
}
```

```
struct sockaddr_in
{
    short  sin_family;
    ushort sin_port;
    struct in_addr
           sin_addr;
}
```

```
struct in_addr
{
  unsigned long int
    s_addr;
}
```

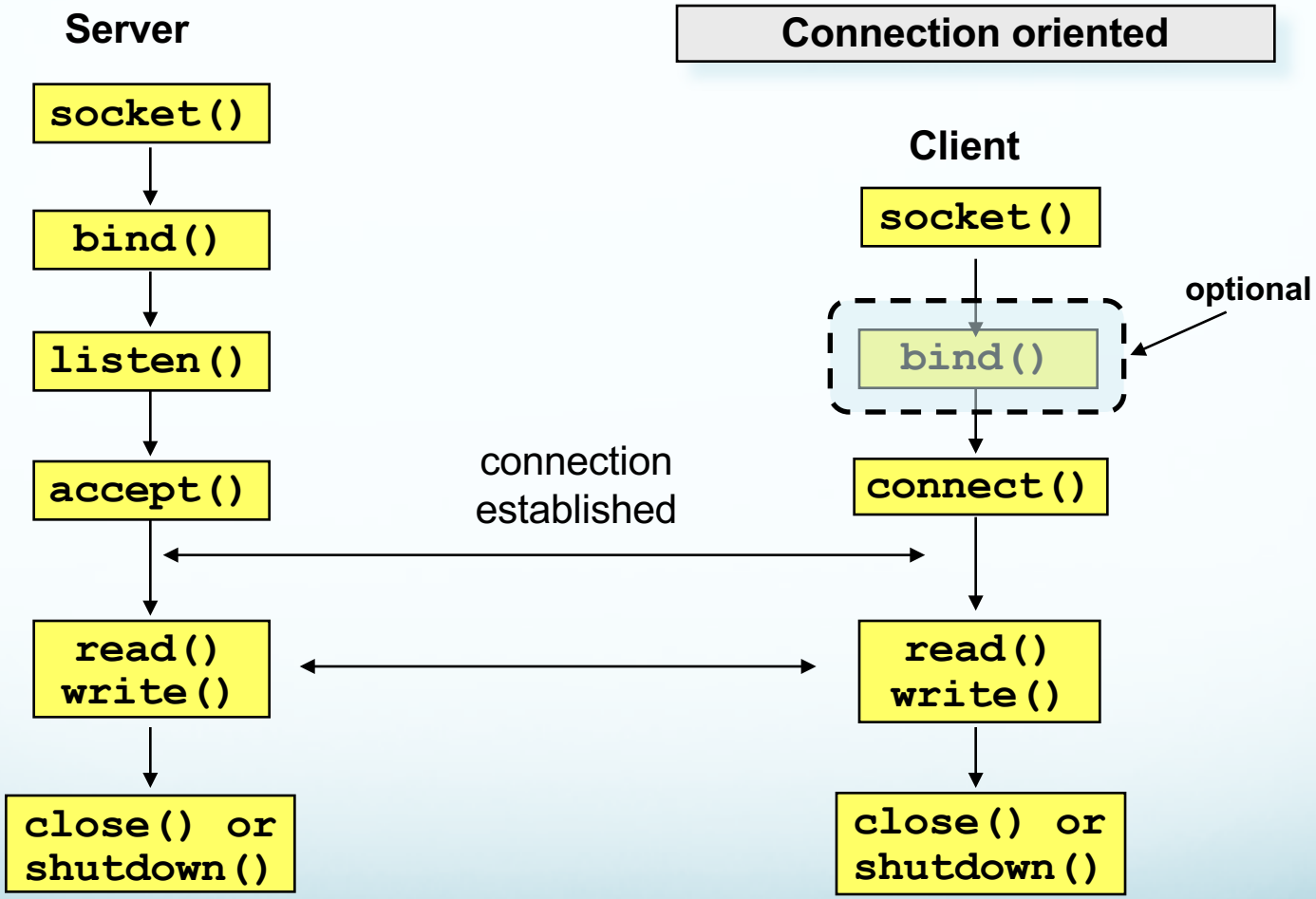
MABEL Technology
Tech The Best

Using Datagram Sockets



MABEL Technology
Tech The Best

Stream Sockets



MABEL Technology
Tech The Best

Using Unix Domain Sockets

```
#include <sys/socket.h>
```

- To create a Unix Domain socket

  int socket(int family, int type, int protocol)

  family *socket domain, AF_UNIX for Unix Domain*

  type *SOCK_DGRAM or SOCK_STREAM*

  protocol *always zero for Unix Domain*

  returns *socket descriptor or -1 on error*

```
int sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
```

- Socket cannot yet be used
  - no name associated with socket

MABEL Technology
Tech The Best

Naming Sockets in the Unix Domain

- To *bind* a name to a socket

  int bind(int sockfd, struct sockaddr *addr, int len)

  sockfd *socket file descriptor*

  addr *Unix domain socket address*

  len *length of socket address structure*

```
struct sockaddr_un {
    short   sun_family;
    char    sun_path[108];
}
```

```
struct sockaddr_un addr = {0};
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "sock1");
bind(sockfd, (struct sockaddr*)&addr, sizeof(addr));
```

MABEL Technology
Tech The Best

Sending Datagrams

- To send a datagram through a socket:

```
#include <sys/socket.h>
```

int sendto(int sockfd, char *buf, int len, int flags,
struct sockaddr *addr, size_t addrlen)

sockfd	*socket file descriptor*
buf	*data to send*
len	*amount of data in buf (bytes)*
flags	*usually 0 for Unix domain*
addr	*address of destination socket*
addrlen	*size of destination socket addres*
returns	*number of bytes sent or -1*

```
memset (&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "sock1");
sendto(sockfd,buffer,strlen(buffer),0,
        (struct sockaddr*)&addr,sizeof(addr));
```

MABEL Technology
Tech The Best

Receiving Datagrams

- To receive a datagram through a socke

```
#include <sys/socket.h>
```

int recvfrom(int sockfd, char *buf, int len, int flags,
struct sockaddr *addr, size_t *addrlen)

sockfd *socket file descriptor*

buf *where to store the message*

len *size of buffer for message*

flags *usually 0 for Unix domain*

addr *structure for sender's socket address*

addrlen *on call, set to max length of socket address structure*
on return, contains actual size of senders address

returns *number of bytes received or -1*

```
memset (&addr, 0, sizeof(addr));
addrlen = sizeof(addr);
recvfrom(sockfd,buffer,sizeof(buffer),0,
         (struct sockaddr*)&addr,&addrlen);
```

MABEL Technology
Tech The Best

Example - Datagram Receiver

udgram1.c

```
int main()
{
  char buffer[MAXBUF+1];
  struct sockaddr_un addr = {0};
  size_t addrlen, n;
  int sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
  addr.sun_family = AF_UNIX;
  strcpy(addr.sun_path, "sock1");
  bind(sockfd, (struct sockaddr*)&addr, sizeof(addr));
  memset(&addr, 0, sizeof (addr));
  addrlen = sizeof(addr);
  n = recvfrom(sockfd,buffer,MAXBUF,0,
                        (struct sockaddr*)&addr, &addrlen);
  buffer[n] = '\0';
  printf ("Data from client %s\n%s\n", addr.sun_path, buffer);
  unlink ("sock1");                /* delete socket */
  return 0;
}
```

MABEL Technology
Tech The Best

Example - Datagram Sender

udgram2.c

```
int main(int argc, char *argv[])
{
  char *msg = "Test message from process 2";
  struct sockaddr_un addr = {0};
  int n;
  int sockfd;
  sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);

  addr.sun_family = AF_UNIX;
  strcpy(addr.sun_path, "sock2");
  bind(sockfd,(struct sockaddr*)&addr,sizeof(addr));
  memset (&addr, '\0', sizeof(addr));
  addr.sun_family = AF_UNIX;
  strcpy(addr.sun_path, "sock1");
  n = sendto(sockfd, msg, strlen(msg), 0,
             (struct sockaddr*)&addr, sizeof(addr));
  unlink ("sock2");              /* delete socket */
  return 0;
}
```

MABEL Technology
Tech The Best

Stream Sockets

```
#include <sys/socket.h>
```

- Create and bind as before
- To mark socket as server socket

  int listen(int sockfd, int backlog)

  sockfd *socket file descriptor*

  backlog *max number of pending requests*
- Socket is now "listening socket"
  - used to accept incoming connections from clients
  - sometimes called "passive" socket

```
int sockfd = socket(AF_UNIX,SOCK_STREAM,0);
struct sockaddr_un addr;
memset (&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "sock1");
bind(sockfd,(struct sockaddr*)&addr,sizeof(addr));
listen (sockfd, 5);
```

MABEL Technology
Tech The Best

Accepting connections from clients

```
#include <sys/socket.h>
```

- To accept connections from clients:

int accept(int sockfd, struct sockaddr *addr, size_t *len)

sockfd	*socket file descriptor*
addr	*structure to hold incoming client address*
len	*on call, set to size of address structure,* *on return, contains actual size of client address*

```
int commfd;
struct sockaddr_un addr;
size_t len = sizeof(addr);

memset (&addr, 0, sizeof(addr));
commfd = accept(sockfd,(struct sockaddr*)&addr,&len);
```

MABEL Technology
Tech The Best

Connecting to a server

```
#include <sys/socket.h>
```

- To connect to a server:

  int connect(int sockfd, struct sockaddr *addr, int len)

  sockfd *socket file descriptor*

  addr *address of server socket address structure*

  len *length of server socket address structure*

  returns *0 or -1 on error*

```
struct sockaddr_un addr;
int sockfd;

sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
memset (&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "server");
connect(sockfd, (struct sockaddr*)&addr, sizeof(addr));
```

MABEL Technology
Tech The Best

Example - Stream Socket Client

unixclt.c

```
int main()
{
  char msg[MAXBUF];
  struct sockaddr_un addr;
  int n, sockfd;

  /* Create socket and connect to server */
  sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
  memset(&addr, 0, sizeof (addr));
  addr.sun_family = AF_UNIX;
  strcpy(addr.sun_path, "server");
  connect(sockfd,(struct sockaddr*)&addr,sizeof(addr));

  sprintf (msg, "Test message from client %d", getpid());
  n=write(sockfd, msg, strlen(msg));        /* Send message */
  n=read(sockfd, msg, MAXBUF);   /* Read reply */
  msg[n] = '\0';
  printf ("Answer: %s\n", msg);
  return 0;
}
```

MABEL Technology
Tech The Best

Example - Stream Socket Server (1)

unixsrv.c

```
int main()
{
  /* Note error handling not shown… */
  char buffer[MAXBUF];
  struct sockaddr_un addr;
  struct sockaddr_un client;
  int addrlen, n;
  int sockfd, commfd;
  struct sigaction act;
  pid_t pid;

  /* Set up signal handling for SIGCHLD,
     SIGINT and SIGTERM */
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  act.sa_handler = CHLDhandler;
  sigaction(SIGCHLD, &act, NULL);

  act.sa_handler = TERMhandler;
  sigaction(SIGINT, &act, NULL);
  sigaction(SIGTERM, &act, NULL);
```

```
void CHLDhandler (int sig)
{
  while (waitpid(0,NULL,WNOHANG) > 0)
    {}
}
```

```
void TERMhandler (int sig)
{
  unlink ("server");
  exit(0);
}
```

MABEL Technology
Tech The Best

Example - Stream Socket Server

```
/* …continued from over, create and bind socket, set it
     up as a server */

  sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
  memset(&addr, 0, sizeof (addr));
  addr.sun_family = AF_UNIX;
  strcpy(addr.sun_path, "server");
  bind(sockfd,(struct sockaddr*)&addr ,sizeof(addr));
  listen (sockfd, 5);

  for (;;) {
    /* Accept connection and deal with request */
    memset(&client, 0, sizeof (client));
    addrlen = sizeof(client);
    commfd = accept(sockfd,(struct sockaddr*)&client, &addrlen);
    if ( commfd < 0 ) {
      if ( errno == EINTR )          /* Probably a child dying… */
         continue;
      perror("accept"); exit(1);     /* A real problem */
    }
```

MABEL Technology
Tech The Best

Example - Stream Socket Server

unixsrv.c

```
    /* …continued from over, each accepted client
       connection is   handled in separate sub-process */

    switch ( pid = fork() ) {

      case -1:  /* Error, not shown here… */

      case 0:
        close (sockfd);          /* Don't need listening socket now
    */
        n=read(commfd, buffer, MAXBUF); buffer[n] = '\0';
        printf ("Message; %s\n", buffer);
        /* Could send a reply here if required… */
        exit(0);

      default:
        close (commfd);          /* Don't need connected socket */
  }

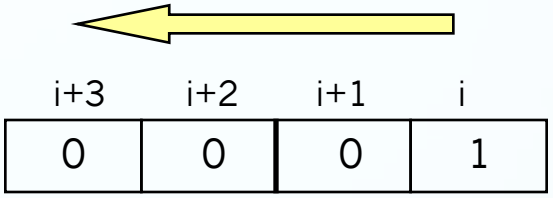
  return 0;
}
```

MABEL Technology

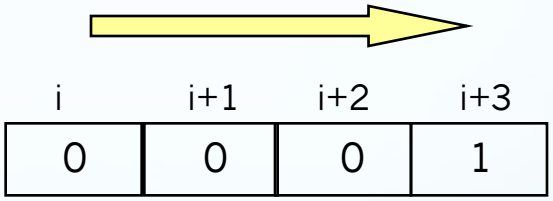
Tech The Best

Byte Ordering

- Machine representation of integer values



- Little Endian
  - stores low order byte at low address
  - e.g. Intel X86



- Big Endian
  - stores high order byte at low address
  - e.g. SPARC

- Important when transferring integer data
  - e.g. `write ( sockfd, &i, sizeof(int) );`
  - "send `sizeof(int)` bytes from address `i` through the socket"

MABEL Technology
Tech The Best

Host and Network Byte Order

- Network Byte Order
  - "system independent" byte ordering
  - sender converts to network byte order before sending
  - receiver converts from network byte order on receiving data

```
/* 32 bit values */
int htonl ( int i );

/* 16 bit values */
short htons ( short s );
```

```
...
int i = htonl(1);
...
/* Set up socket... */
write(sock, &i, sizeof(int));
...
```

```
/* 32 bit values */
int ntohl ( int i );

/* 16 bit values */
short ntohs ( short s );
```

```
int i;
...
/* Set up socket... */
read(sock, &i, sizeof(int));
printf("%d\n", ntohl(i));
...
```

MABEL Technology
Tech The Best

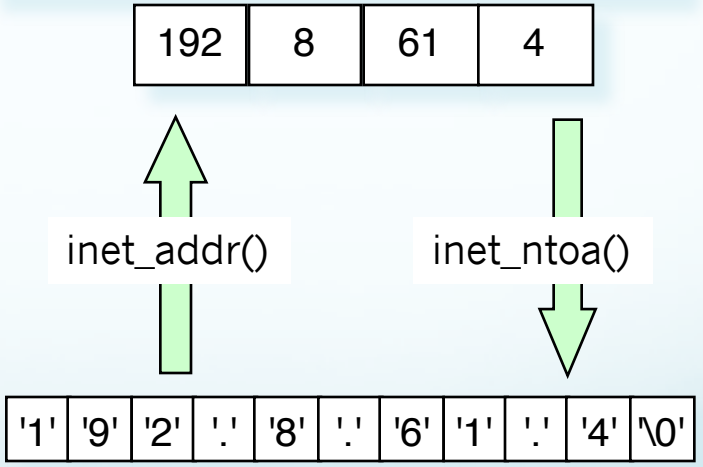
Internet Domain Socket Addresses

- IP address and port number
  - both must be in network byte order

```
struct sockaddr_in
{
  short           sin_family;
  ushort          sin_port;
  struct in_addr sin_addr;
}
```

```
struct sockaddr_in saddr;
...
saddr.sin_family = AF_INET;
saddr.sin_port = htons(80);
saddr.sin_addr.s_addr =
      inet_addr("192.8.61.4");
```

```
struct in_addr {
  unsigned long int s_addr;
}
```



MABEL Technology
Tech The Best

Getting Host Information

- To look up information about a host:

struct hostent *gethostbyname(const char *name)

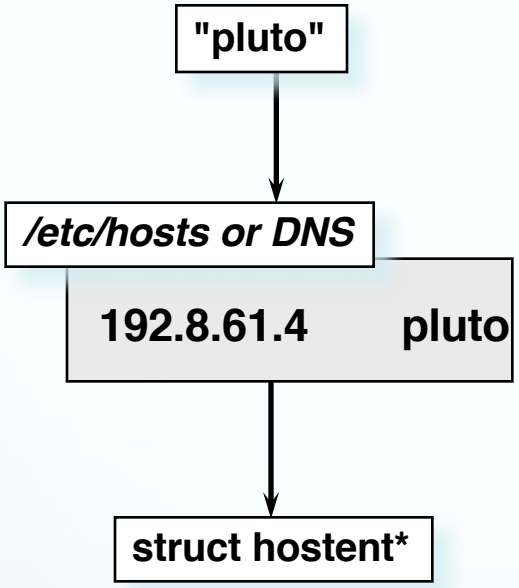
name	*host name*
returns	*host entry structure (NULL on error)*

struct hostent *gethostbyaddr(const char *addr, int len, int family)

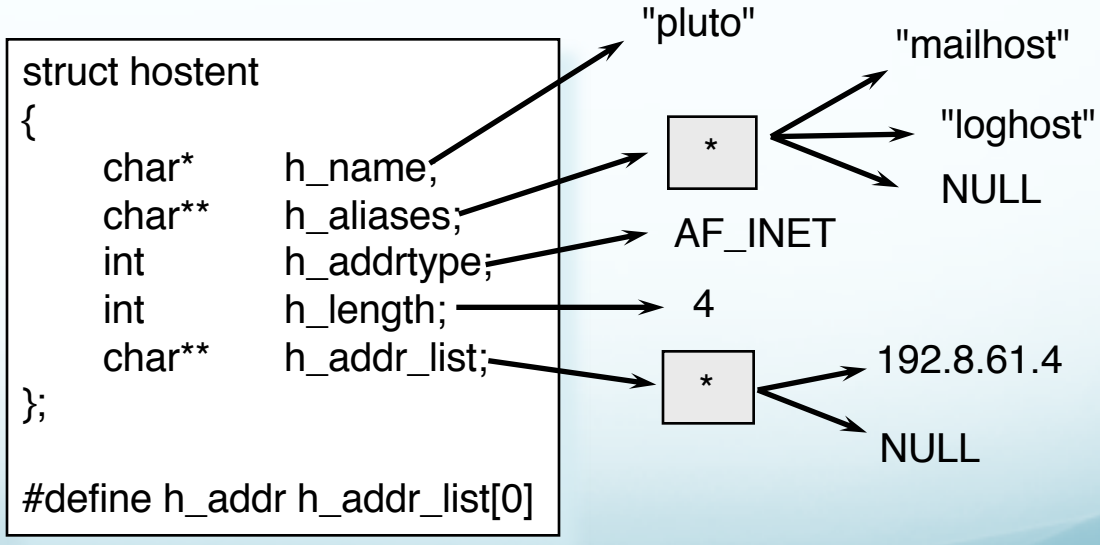
addr	*really a* ***struct in_addr**** *for address*
len	*size of address structure*
family	always AF_INET
returns	*host entry structure (NULL on error)*

MABEL Technology
Tech The Best

Using gethostbyname()




```
struct hostent* pHost;
pHost = gethostbyname("pluto");
…
memcpy( &saddr.sin_addr.s_addr,
          pHost->h_addr,
          pHost->h_length );
…
```



MABEL Technology
Tech The Best

Getting Service/Port Information

- To look up information about services and ports:

```
#include <netdb.h>
```

struct servent *getservbyname(const char *name, const char *protocol)

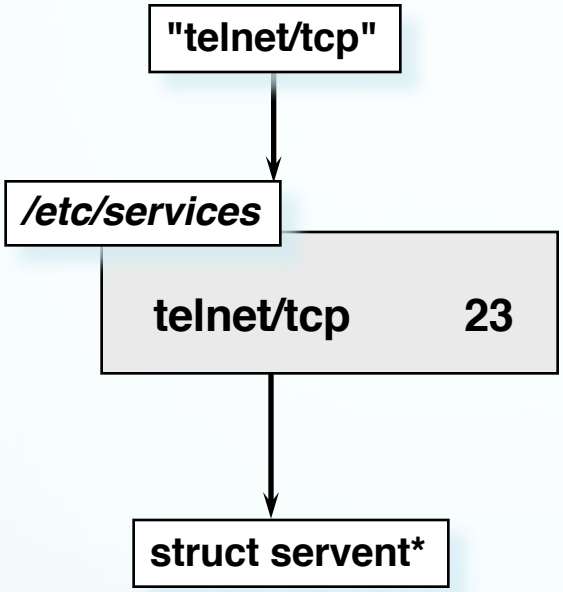
name	*service name*
protocol	*NULL or protocol name (eg. tcp or udp)*
returns	*service entry structure (NULL on error)*

struct servent *getservbyport(int port, const char *protocol)

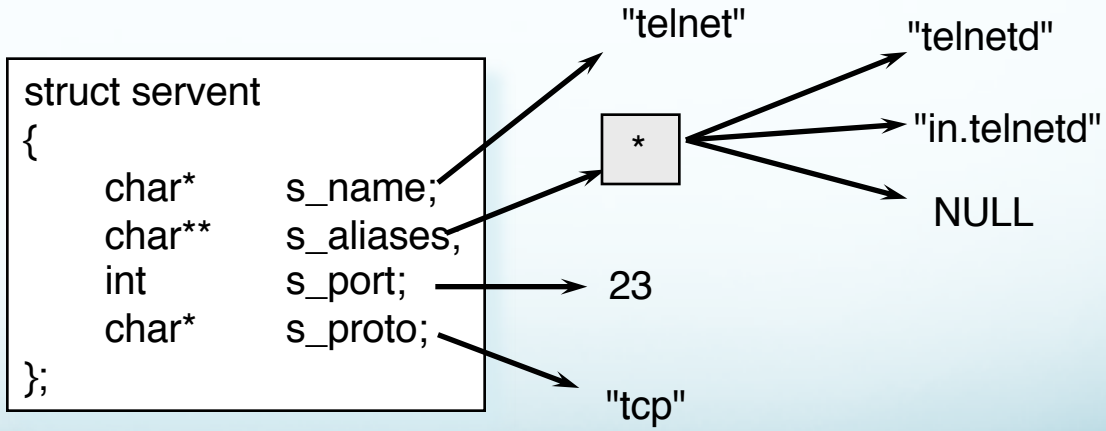
port	*port number*
protocol	*NULL or protocol name (eg. tcp or udp)*
returns	*host entry structure (NULL on error)*

MABEL Technology
Tech The Best

Using getservbyname()




```
struct servent *pService;
pService = getservbyname("telnet", "tcp");
…
saddr.sin_port = pService->s_port;
```



MABEL Technology
Tech The Best

Creating Internet Domain Sockets

- To create Internet Domain Sockets

int socket(int family, int type, int protocol)

family *AF_INET*

type *SOCK_DGRAM, SOCK_STREAM or (rarely) SOCK_RAW*

protocol *usually zero*

returns *socket descriptor or -1 on error*

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

MABEL Technology
Tech The Best

bind() for Internet Domain Sockets

- To bind an address to a socket:

  int bind(int sockfd, struct sockaddr *addr, int addrlen)

  sockfd — *socket descriptor*

  addr — *Internet domain socket address*

  addrlen — *length of socket address structure*

  - Special address values

    address == INADDR_ANY — *any valid IP address for this host*

    port no == 0 — *any unused port (usually client side only)*

```
struct sockaddr_in {
    short          sin_family;
    ushort         sin_port;
    struct in_addr sin_addr;
}
```

```
struct in_addr {
  unsigned long int s_addr;
}
```

MABEL Technology
Tech The Best

Example - Datagram Receiver

idgram1.c

```
int main()
{
  char buf[256];
  struct sockaddr_in addr;
  size_t addrlen, n;
  int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

  memset(&addr, 0, sizeof (addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(2000);
  addr.sin_addr.s_addr = INADDR_ANY;
  bind(sockfd, (struct sockaddr*)&addr, sizeof(addr));

  memset(&addr, 0, sizeof (addr));
  addrlen = sizeof(addr);
  n = recvfrom(sockfd, buffer, MAXBUF, 0,
                          (struct sockaddr*)&addr, &addrlen);
  buffer[n] = '\0';

  printf ("Data from client %s\n%s\n",
              inet_ntoa(addr.sin_addr.s_addr), buffer);
}
```

MABEL Technology
Tech The Best

Example - Datagram Sender

udgram2.c

```
int main()
{
  struct sockaddr_in addr;
  int n;
  char msg="Test message from process 2";
  int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

  /* The address structure contains the server address & port */
  memset(&addr, 0, sizeof (addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(2000);
  /* Assume the user gave us an address as the argument */
  addr.sin_addr.s_addr = inet_addr(argv[1]);

  /* We did not bind to a local address, a valid one will be
     automatically added to the datagram when it is sent. */
  n = sendto(sockfd, msg, strlen(msg), 0,
             (struct sockaddr*)&addr, sizeof(addr));

  return 0;
}
```

MABEL Technology
Tech The Best

Example - Stream Socket Client

inetclt.c

```
int main(int argc, char *argv[])
{
  char msg[MAXBUF];
  struct sockaddr_in addr = {0};
  int n, sockfd;

  /* Create socket and connect to server */
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  addr.sin_family = AF_INET;
  addr.sin_port = htons(2000);
  addr.sin_addr.s_addr = inet_addr(argv[1]);
  connect(sockfd, (struct sockaddr*)&addr, sizeof(addr));

  sprintf (msg, "Test message from client %d", getpid());
  n=write(sockfd, msg, strlen(msg));        /* Send message */
  n=read(sockfd, msg, MAXBUF);   /* Read reply */
  msg[n] = '\0';
  printf ("Answer: %s\n", msg);
  return 0;
}
```

MABEL Technology
Tech The Best

Example - Stream Socket Server

inetsrv.c

```
int main()
{
  /* Note error handling not shown… */
  char buffer[MAXBUF];
  struct sockaddr_in addr;
  struct sockaddr_in client;
  int addrlen, n;
  int sockfd, commfd;
  struct sigaction act;
  pid_t pid;

  /* Set up signal handling for SIGCHLD,
     SIGINT and SIGTERM */
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  act.sa_handler = CHLDhandler;
  sigaction(SIGCHLD, &act, NULL);

  act.sa_handler = TERMhandler;
  sigaction(SIGINT, &act, NULL);
  sigaction(SIGTERM, &act, NULL);
```

```
void CHLDhandler (int sig)
{
  while (waitpid(0,NULL,WNOHANG) > 0)
    {}
}
```

```
void TERMhandler (int sig)
{
  printf ("Shutting down\n");
  exit(0);
}
```

Example - Stream Socket Server

```
/* …continued from over, create and bind socket, set it
     up as a server */

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  memset(&addr, 0, sizeof (addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(2000);
  addr.sin_addr.s_addr = INADDR_ANY;
  bind(sockfd,(struct sockaddr*)&addr ,sizeof(addr));
  listen (sockfd, 5);

  for (;;) {
    /* Accept connection and deal with request */
    memset(&client, 0, sizeof (client));
    addrlen = sizeof(client);
    commfd = accept(sockfd,(struct sockaddr*)&client, &addrlen));
    if ( commfd < 0 ) {
      if ( errno == EINTR )          /* Probably a child dying… */
         continue;
      perror("accept"); exit(1);     /* A real problem */
    }
```

MABEL Technology
Tech The Best

Example - Stream Socket Server

unixsrv.c

```
    /* …continued from over, each accepted client
       connection is   handled in separate sub-process */

    switch ( pid = fork() ) {

      case -1:  /* Error, not shown here… */

      case 0:
        close (sockfd);          /* Don't need listening socket now
    */
        n=read(commfd, buffer, MAXBUF); buffer[n] = '\0';
        printf ("Message; %s\n", buffer);
        /* Could send a reply here if required… */
        exit(0);

      default:
        close (commfd);          /* Don't need connected socket */
  }

  return 0;
}
```

MABEL Technology
Tech The Best

Additional Socket Library Functions

- To retrieve the current hostname:

int gethostname(char *name, int namelen)

name	*buffer to hold null terminated host name*
namelen	*size of name buffer*

- To get the address of a local socket:

int getsockname(int sockfd, struct sockaddr *addr, size_t *len)

sockfd	*socket descriptor*
addr	*address data structure*
len	*on call, set to size of addr buffer, on return holds actual size of address*

- To get the address of a peer socket:

int getpeername(int sockfd, struct sockaddr *addr, size_t *len)

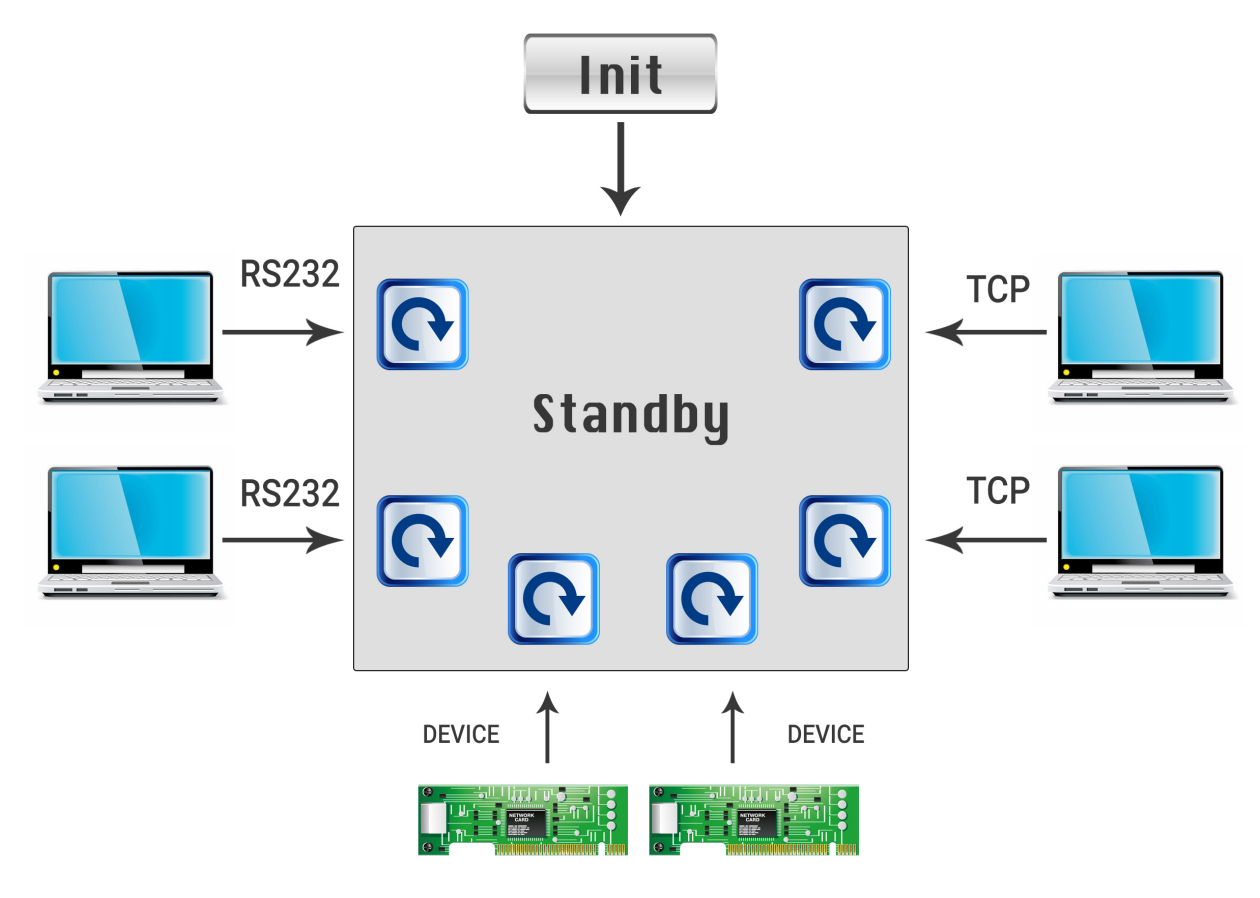
sockfd	*socket descriptor*
addr	*address data structure*
len	*on call, set to size of addr buffer, on return holds actual size of address*

Linux IO

MABEL Technology

Tech The Best

Typical System



IO Models

- Blocking I/O
- Non-blocking I/O
- I/O multiplexing (select/poll/...)
- Signal driven I/O (SIGIO)
- Asynchronous I/O
  - aio_* functions
  - io_* functions

Blocking IO

- Default mode
- Makes the calling thread to block in the kernel in case no data is available
- Can block forever
  - To block with timeout, use select

Non Blocking IO

- If there is no data, calling thread returns with EAGAIN or EWOULDBLOCK

```
flags = fcntl(fd, F_GETFL, 0);

fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

IO Multiplexing

- With **I/O multiplexing**, we call select/poll/epoll* and block in one of these system calls, instead of blocking in the actual I/O system call
- Disadvantage: using select requires at least two system calls (select and recvfrom) instead of one
- Advantage: we can wait for more than one descriptor to be ready

Signal driven I/O

- The **signal-driven I/O model** uses signals, telling the kernel to notify us with the SIGIO signal when the descriptor is ready

```
fd=open("name", O_NONBLOCK);

fcntl(fd, F_SETSIG, SIGRTMIN + 1);
```

- Its better to use RT signals (queued)

Asynchronous IO

- The POSIX asynchronous I/O interface
  - Allows applications to initiate one or more I/O operations that are performed asynchronously.
  - The application can select to be notified of completion of the I/O operation in a variety of ways:
    - Delivery of a signal
    - Instantiation of a thread
    - No notification at all
  - User space asynchronous implementation
    - Call the driver read/write callbacks

Kernel Async Support

- file_operations callbacks
  - aio_read, aio_write
    - Initialize the data processing
    - Create workqueue item/completion/timer/...
  - On completion
    - aio_complete
- Kernel 3.16
  - read_iter
  - write_iter
- To use from user space call io_submit

I/O Multiplexing

- select, pselect
- poll, ppoll
- epoll
  - epoll_create, epoll_create1
  - epoll_ctl
  - epoll_wait, epoll_pwait

MABEL Technology

Tech The Best

select(2) system call

- The **select( )** system call provides a mechanism for implementing synchronous multiplexing I/O
- A call to **select( )** will block until the given file descriptors are ready to perform I/O, or until an optionally specified timeout has elapsed
- The watched file descriptors are broken into three sets
  - File descriptors listed in the **readfds** set are watched to see if data is available for reading.
  - File descriptors listed in the **writefds** set are watched to see if a write operation will complete without blocking.
  - File descriptors in the **exceptfds** set are watched to see if an exception has occurred, or if out-of-band data is available (these states apply only to sockets).
- A given set may be **NULL**, in which case **select( )** does not watch for that event.
- On successful return, each set is modified such that it contains only the file descriptors that are ready for I/O of the type delineated by that set

MABEL Technology
Tech The Best

Blocking for events

- You can use select(2) to block for events
- ```
  int select(int nfds, fd_set *readfds, fd_set
  *writefds, fd_set *exceptfds, struct timeval
  *timeout);
  ```
  - **nfds**: number of highest file descriptor + 1
  - **fd_sets**: sets of file descriptors to block for events on, one for read, one for write, one for exceptions.
  - **timeout**: NULL or structure describing how long to block.
  - Linux updates the timeout structure according to how much time was left to the time out.
  ```

MABEL Technology
Tech The Best

File Descriptor Sets

- fd_set is a group of file descriptor for actions or reports:

```
void FD_CLR(int fd, fd_set *set);
```

  - Remove fd from this set.

```
int  FD_ISSET(int fd, fd_set *set);
```

  - Is fd in the set?

```
void FD_SET(int fd, fd_set *set);
```

  - Add fd to this set.

```
void FD_ZERO(fd_set *set);
```

  - Zero the set.

MABEL Technology

Tech The Best

Driver notification example

```
int fd1, fd2;

fd_set fds_set;

int ret;

fd1 = open("/dev/drv_ctl0", O_RDWR);

fd2 = open("/dev/drv_ctl1", O_RDWR);

FD_ZERO(&fds_set);

FD_SET(fd1, &fds_set);

FD_SET(fd2, &fds_set);

do {

ret = select(fd1 + fd2 + 1, &fds_set,
NULL, NULL, NULL);


} while(errno == EINTR);
```

- Open two file descriptors fd1 and fd2.
- Create an fd set that holds them.
- Block for events on them.
  - We try again if we interrupted by a signal.

MABEL Technology

Tech The Best

Driver notification example 2

```
if(ret == -1) {

perror("Select failed.");


exit(1);


}



if(FD_ISSET(fd1, &fds_set)) {

printf("event at FD 1... “);


ioctl(fd1, IOCTL_CLR, 1);


printf("clear\n);


}
```

- If we have an error bail out.
- Check if fd1 one has a pending read even.
- If so, use ioctl(2) to notify driver to clear status

MABEL Technology
Tech The Best

pselect

- POSIX implementation
- Does not modify the timeout parameter
- Can set signal mask before entering to sleep
- Uses the **timespec** structure (seconds, nanoseconds)

MABEL Technology
Tech The Best

Poll

- **int poll (struct pollfd *fds, unsigned int nfds, int timeout);**

- Unlike select(), with its inefficient three bitmask-based sets of file descriptors, poll( ) employs a single array of nfds pollfd structures

```
#include <sys/poll.h>

struct pollfd {
        int fd;
        short events;
        short revents;
};
```

- POSIX implementation – ppoll (with signal mask)

```
// The structure for two events
struct pollfd fds[2];

// Monitor sock1 for input
fds[0].fd = sock1;
fds[0].events = POLLIN;

// Monitor sock2 for output
fds[1].fd = sock2;
fds[1].events = POLLOUT;

// Wait 10 seconds
int ret = poll( &fds, 2, 10000 );

     // Check if poll actually succeed
     if ( ret == -1 )
         // report error and abort
     else if ( ret == 0 )
         // timeout; no event detected
     else
     {
         // If we detect the event, zero it out so we can reuse the structure
         if ( pfd[0].revents & POLLIN )
             pfd[0].revents = 0;
             // input event on sock1

         if ( pfd[1].revents & POLLOUT )
             pfd[1].revents = 0;
             // output event on sock2
     }
```

https://www.ulduzsoft.com/2014/01/select-poll-epoll-practical-difference-for-system-architects/

MABEL Technology
Tech The Best

Poll vs. Select

- poll( ) does not require that the user calculate the value of the highest-numbered file descriptor +1
- poll( ) is more efficient for large-valued file descriptors. Imagine watching a single file descriptor with the value 900 via select()—the kernel would have to check each bit of each passed-in set, up to the 900th bit.
- select( )'s file descriptor sets are statically sized.
- With select( ), the file descriptor sets are reconstructed on return, so each subsequent call must reinitialize them. The poll( ) system call separates the input (events field) from the output (revents field), allowing the array to be reused without change.
- The timeout parameter to select( ) is undefined on return. Portable code needs to reinitialize it. This is not an issue with pselect( )
- select( ) is more portable, as some Unix systems do not support poll( ).

MABEL Technology
Tech The Best

Event Poll

- Has state in the kernel
  - O(1) instead of O(n)
- epoll_create(2) – initializes epoll context
- epoll_ctl(2) – adds/removes file descriptors from the context
- epoll_wait(2) – performs the actual event wait
- Can behave as
  - Edge triggered
  - Level triggered

MABEL Technology
Tech The Best

Example

```
int epfd = epoll_create(0);
int client_sock = socket(…..);
static struct epoll_event ev;
ev.events = EPOLLIN | EPOLLOUT | EPOLLERR;
ev.data.fd = client_sock;
int res = epoll_ctl(epfd, EPOLL_CTL_ADD, client_sock, &ev);
struct epoll_event *events = malloc(SIZE);
while (1) {
        int nfds = epoll_wait(epfd, events, MAX_EVENTS, TIMEOUT);
        for(int i = 0; i < nfds; i++) {
                int fd = events[i].data.fd;
                handle_io_on_socket(fd);
         }
}
```