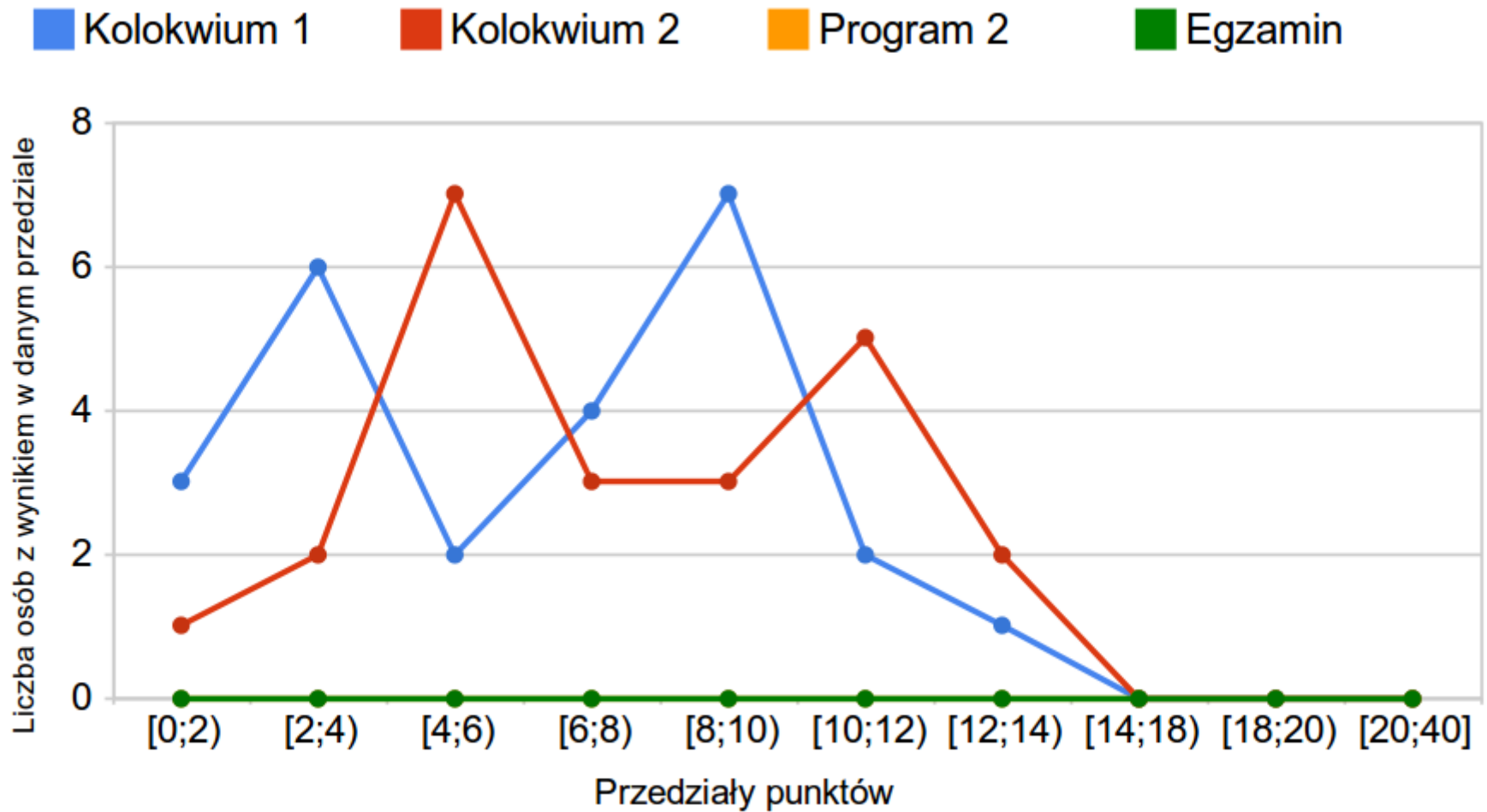


# Wstęp do Informatyki dla bioinformatyków

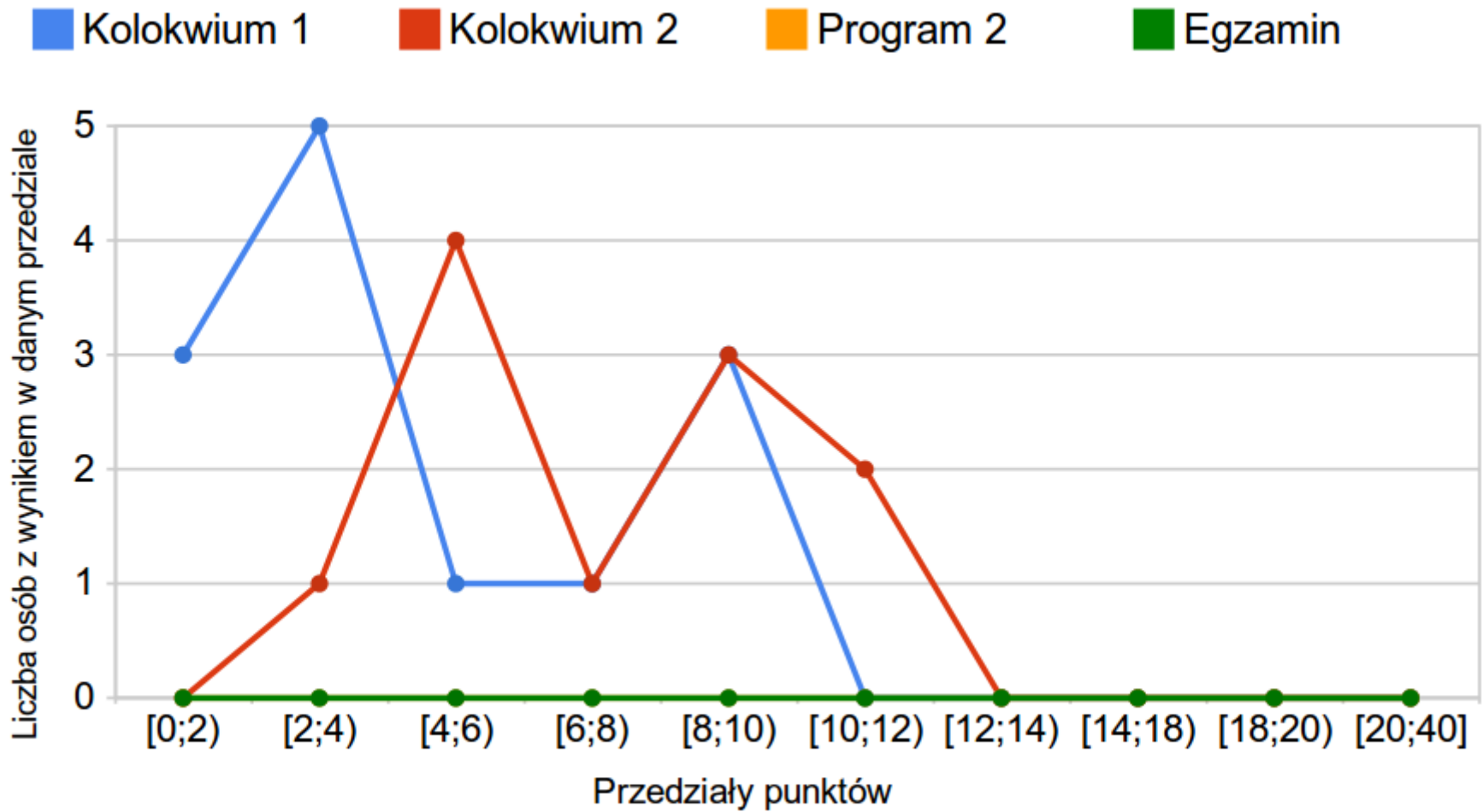
Powłoka systemowa 1  
O systemie operacyjnym,  
plikach, procesach i strumieniach

Bartek Wilczyński 11.01.2016

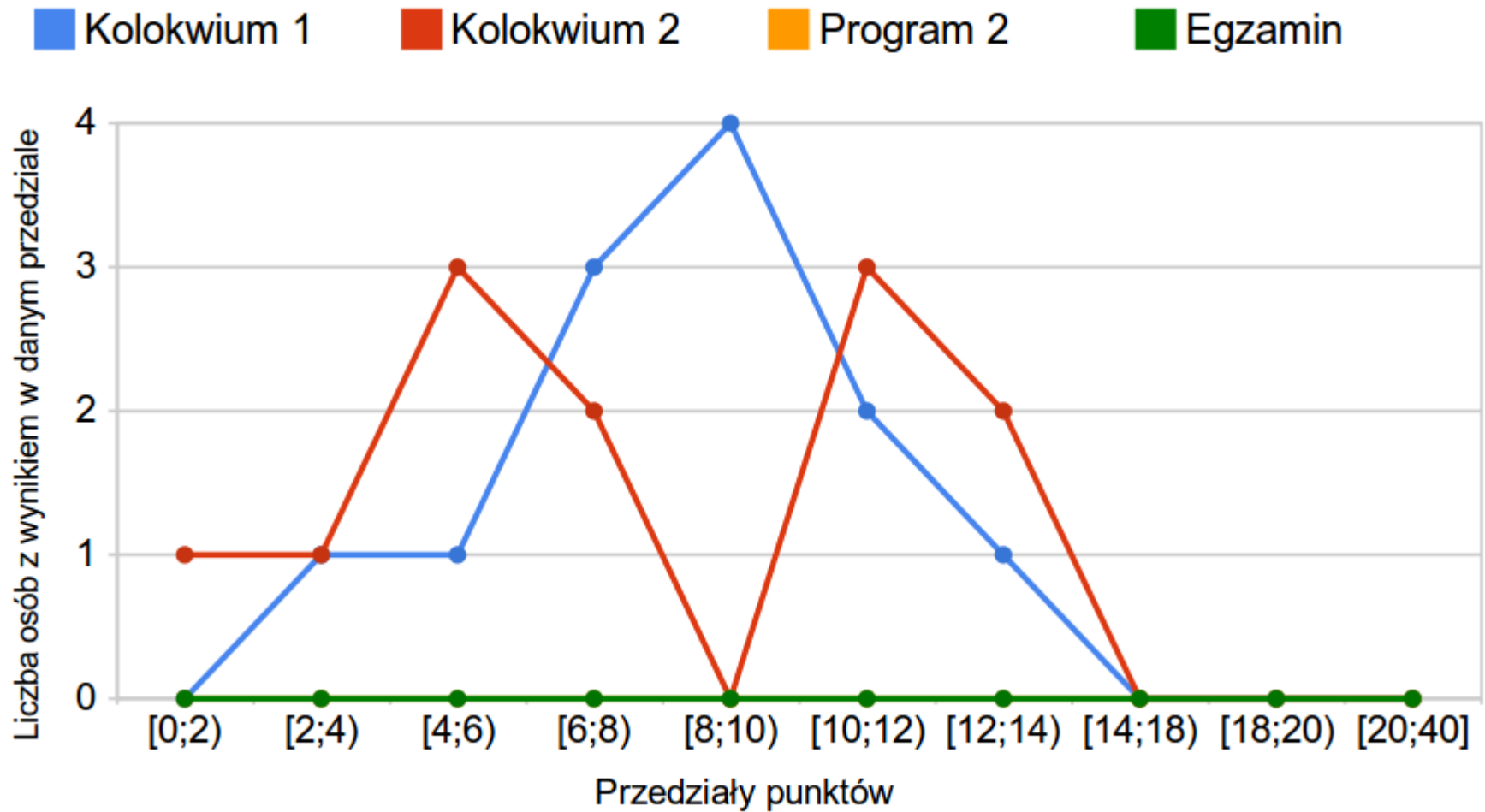
# Wyniki kolokwium 2



# Grupa I



# Grupa II

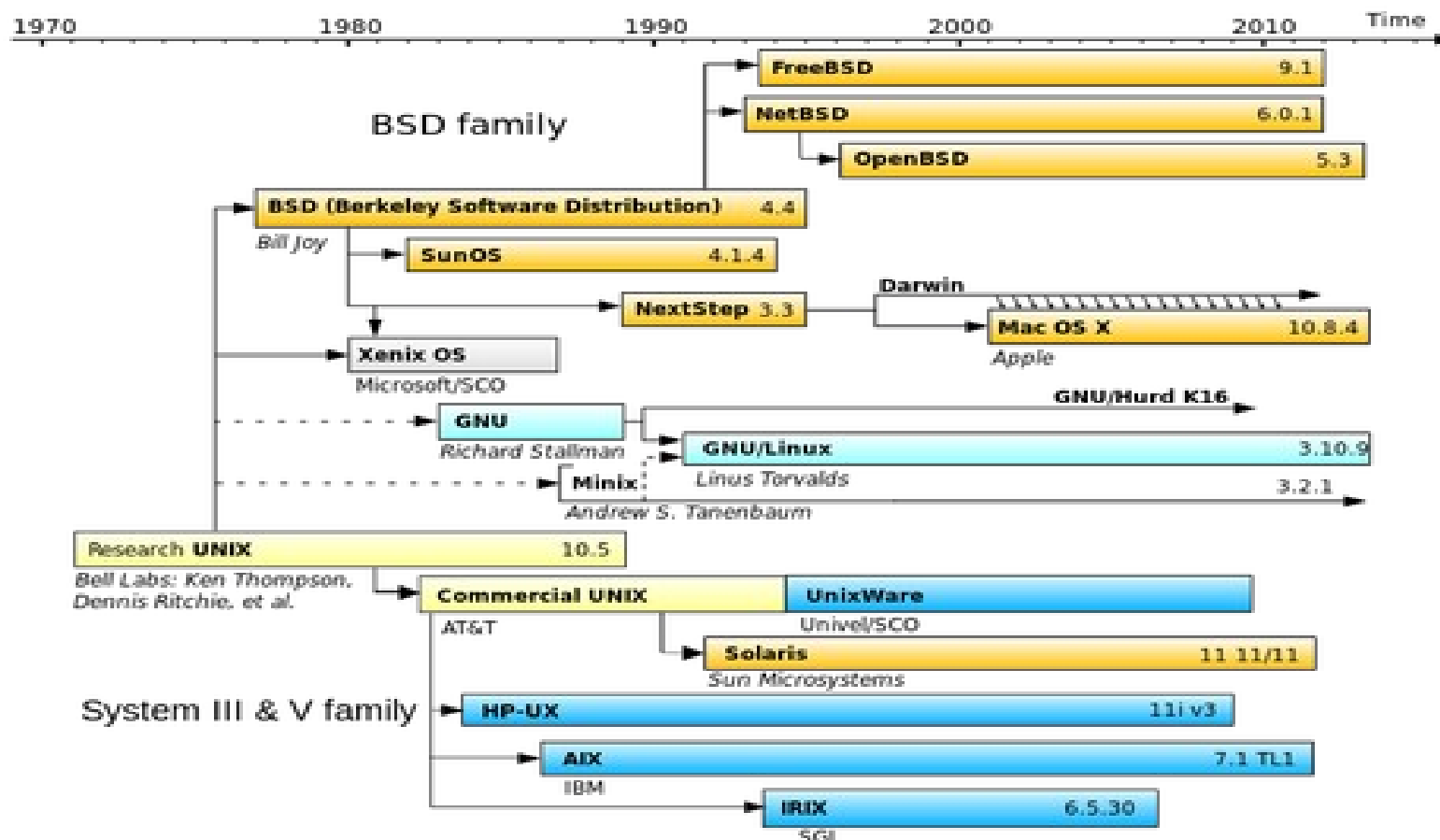


# Tematy na dziś

- Rola systemu operacyjnego
- Systemy Unix i podobne
- System plików Unix i prawa dostępu do plików
- Procesy i zarządzanie nimi
- Komunikacja między procesami
- Wejście/wyjście i strumienie

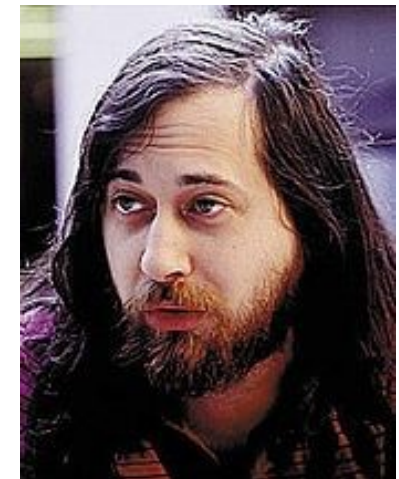
# System operacyjny UNIX

- Rodzina systemów Unix'owych



# System GNU/Linux

- Pierwsze jądro napisał Linus Torvalds w oparciu o jądro Minix
- Obecnie wspiera bardzo wiele architektur procesorów, od telefonów komórkowych (android) przez routery, aparaty fotograficzne, komputery osobiste aż do klastrów obliczeniowych
- My będziemy używać go w systemie GNU zapoczątkowanym przez Richarda Stallmana



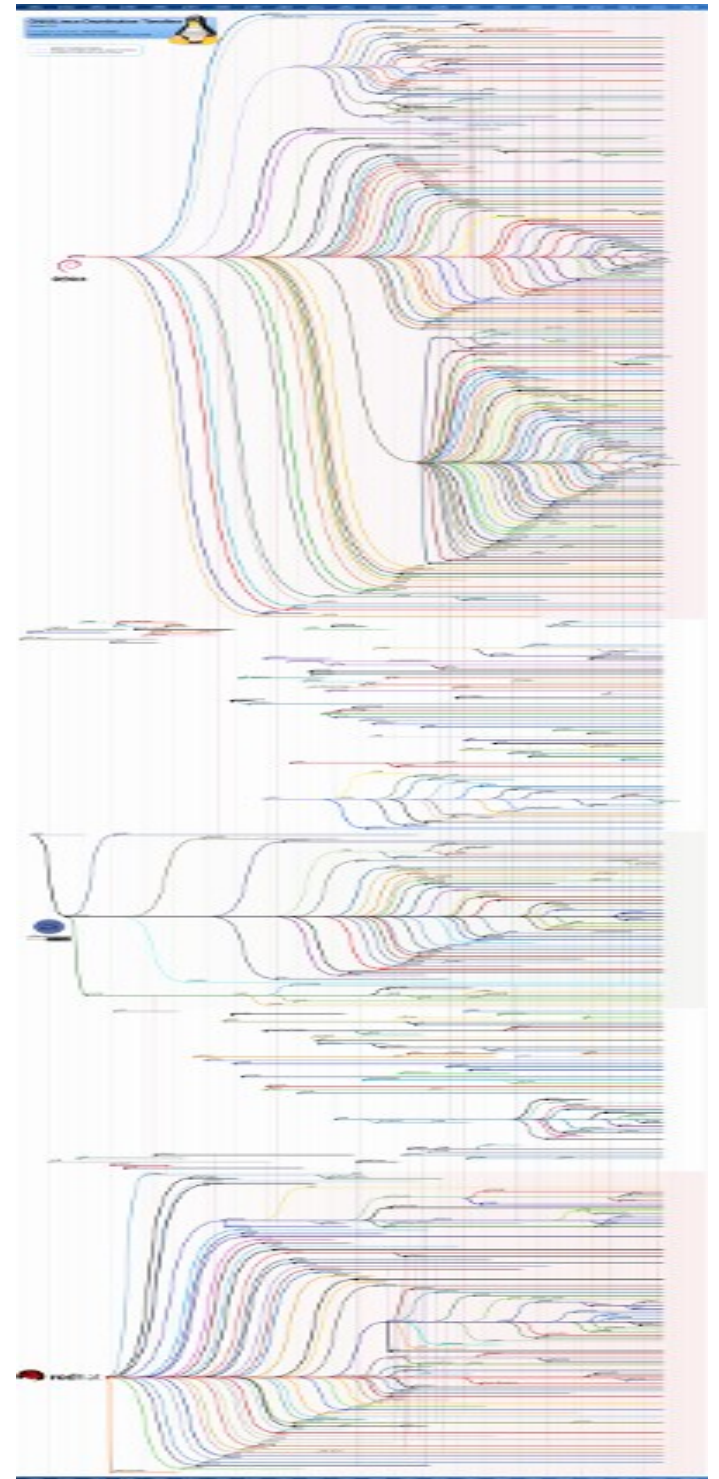
# Typowe warstwy komunikacji

- Użytkownik
- Interfejs sprzętowy (klawiatura, mysz, ekran)
- Powłoka graficzna (X.org)
- Program terminala (gnome-terminal)
- **Powłoka systemowa (BASH)**
- **Interpreter (Python)**
- System operacyjny (linux)
- Zasoby systemowe (procesor, pamięć, dysk)



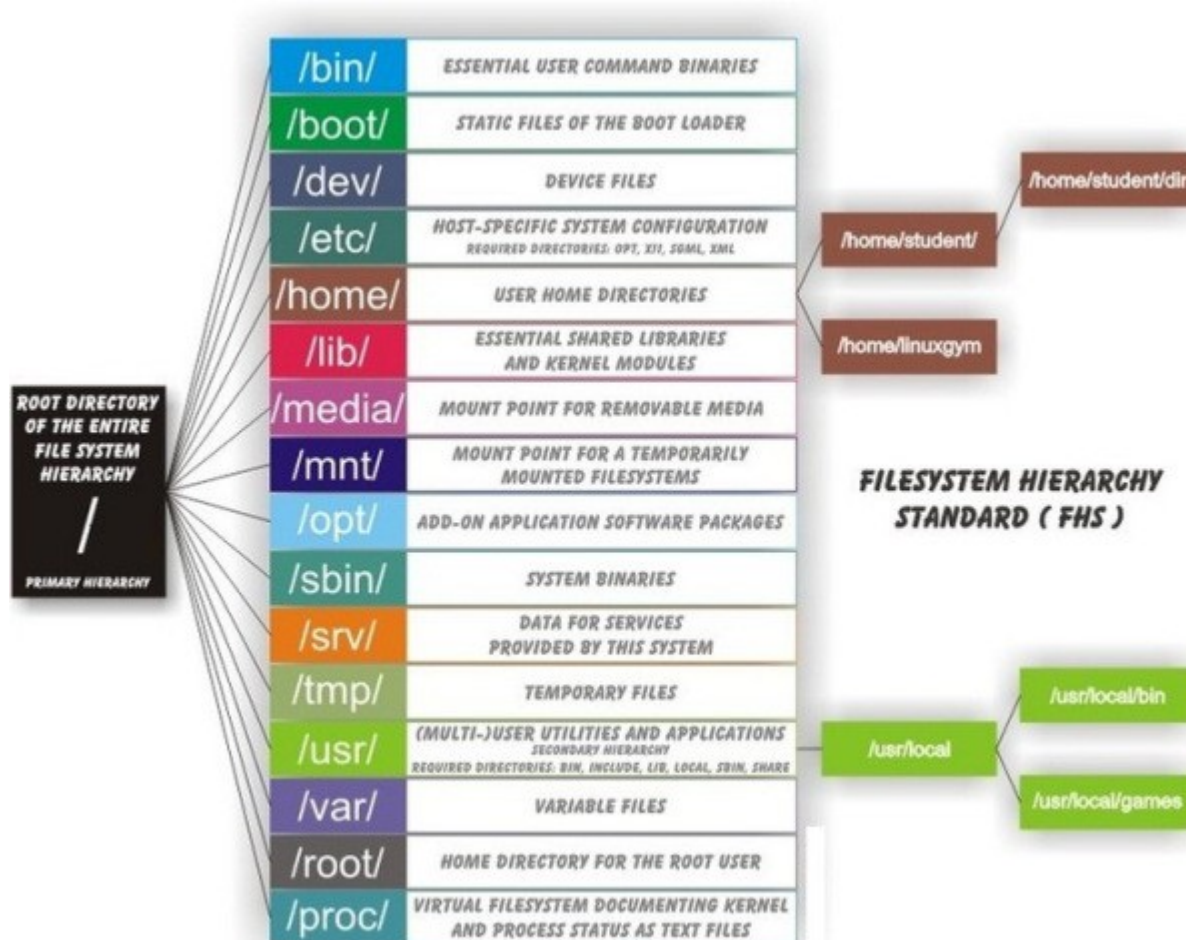
# Dystrybucje GNU/Linux

- Obecnie dostępne są setki dystrybucji GNU/Linux
- Do najbardziej popularnych obecnie należą pochodne dystrybucji
  - Debian (Ubuntu, Mint, itp.)
  - Redhat (fedora, SUSE itp.)
- Dla uporządkowania powstał projekt Linux Standard Base (obecnie w wersji 5)



# Hierarchia systemu plików

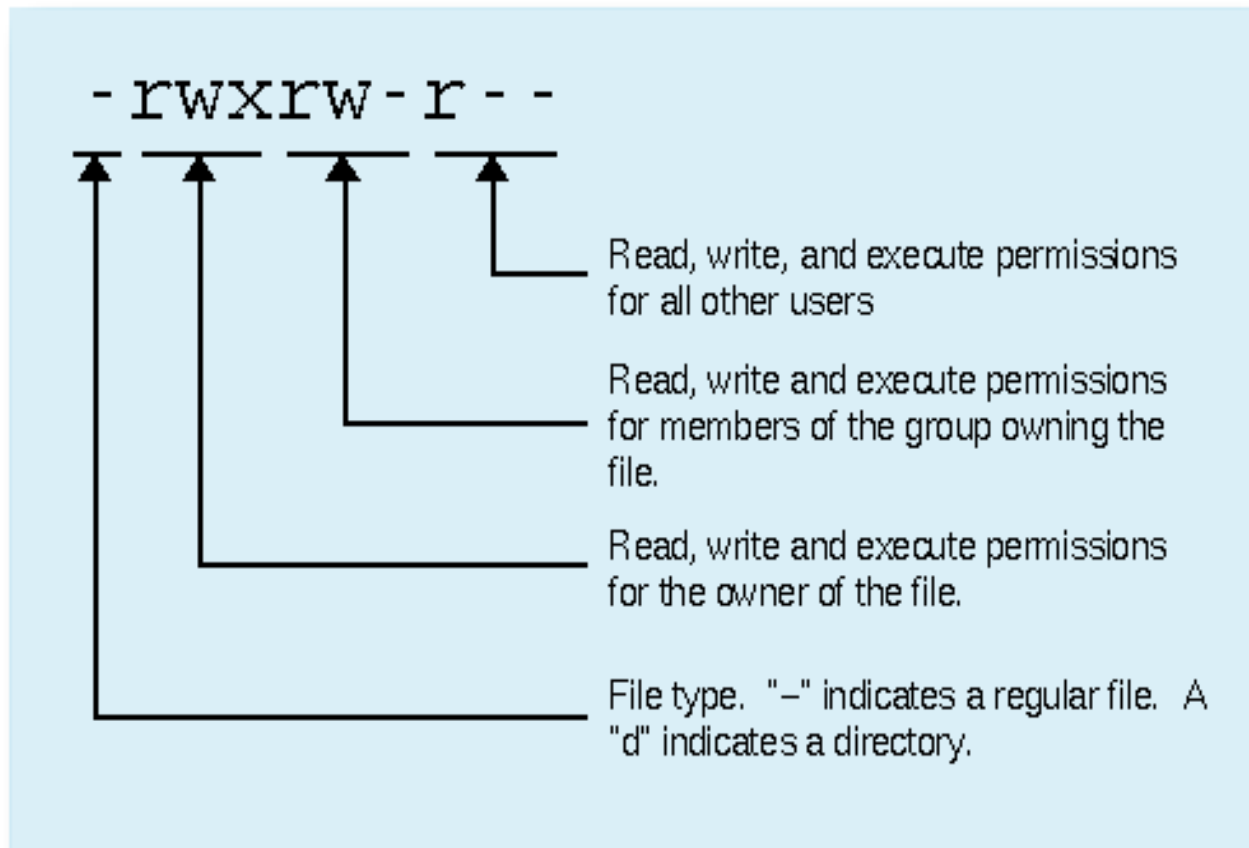
- Standaryzowana w LSB jako FHS



# Prawa dostępu do plików

- Systemy UNIX nadają prawa dostępu dla użytkowników i grup użytkowników
- Jeden użytkownik może należeć do wielu grup
- Każdy plik ma właściciela i grupę właścicielską
- Prawa dostępu są określone w postaci bitów określających prawa dla właściciela, grupy i pozostałych
- Do zmian służą polecenia **chmod** i **chown**

# Schemat praw dostępu do pliku/katalogu



# Procesy w systemie

- Każde działanie w systemie jest wynikiem jakiegoś procesu
- Jednocześnie na jednym komputerze wykonuje się wiele procesów
- Użytkownicy mogą uruchamiać procesy w systemie, mają one takie same uprawnienia jak ich “właściciel”, czyli ten kto je uruchomił
- Procesy mogą działać w tle (w bashu uruchamianie z & na końcu)
- Każdy proces ma swoje standardowe wejście/wyjście

# Oglądanie procesów

- ps – lista procesów (np. ps aux)
- pstree – drzewo procesów
- top – lista procesów, uaktualniana
- htop – high performance top

```
top - 11:36:27 up 19 days, 16:32, 7 users, load average: 0.38, 0.39, 0.35
Tasks: 262 total, 1 running, 261 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.4%us, 1.5%sy, 0.0%ni, 71.1%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 8143932k total, 8051184k used, 92748k free, 517532k buffers
Swap: 5787636k total, 185156k used, 5602480k free, 1774952k cached
```

| PID   | USER     | PR | NI | VIRT  | RES  | SHR  | S | %CPU | %MEM | TIME+     | COMMAND     |
|-------|----------|----|----|-------|------|------|---|------|------|-----------|-------------|
| 32516 | www-data | 20 | 0  | 536m  | 80m  | 21m  | S | 42   | 1.0  | 0:22.41   | apache2     |
| 14857 | mysql    | 20 | 0  | 756m  | 273m | 5364 | S | 14   | 3.4  | 184:31.95 | mysqld      |
| 25909 | root     | 20 | 0  | 19204 | 1524 | 1028 | R | 2    | 0.0  | 0:00.44   | top         |
| 1     | root     | 20 | 0  | 10428 | 684  | 648  | S | 0    | 0.0  | 0:09.91   | init        |
| 2     | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | kthreadd    |
| 3     | root     | RT | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:02.25   | migration/0 |
| 4     | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 2:53.85   | ksoftirqd/0 |
| 5     | root     | RT | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | watchdog/0  |
| 6     | root     | RT | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:02.46   | migration/1 |
| 7     | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:36.84   | ksoftirqd/1 |
| 8     | root     | RT | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | watchdog/1  |
| 9     | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:14.65   | events/0    |
| 10    | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:21.49   | events/1    |
| 11    | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | cpuset      |
| 12    | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | khelper     |
| 13    | root     | 20 | 0  | 0     | 0    | 0    | S | 0    | 0.0  | 0:00.00   | netns       |

# Komunikacja między procesowa

- Przede wszystkim standardowe wejście, wyjście i wyjście dla błędów
- Poza tym pliki (zapisywane przez jeden program i odczytywane przez inny)
- Łącza nienazwane (ang. Pipe)
- Sygnały, wysyłane poleceniem kill, lub z poziomu np. Htop
- Ważne sygnały: SIGTERM, SIGKILL, SIGSEGV, SIGHUP, itp.

```
paul@obsidian:~  
paul@obsidian:~$ kill -l  
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL  
5) SIGTRAP    6) SIGABRT    7) SIGBUS      8) SIGFPE  
9) SIGKILL    10) SIGUSR1   11) SIGSEGV    12) SIGUSR2  
13) SIGPIPE   14) SIGALRM   15) SIGTERM    17) SIGCHLD  
18) SIGCONT   19) SIGSTOP   20) SIGTSTP    21) SIGTTIN  
22) SIGTTOU   23) SIGURG    24) SIGXCPU   25) SIGXFSZ  
26) SIGVTALRM 27) SIGPROF   28) SIGMINCH   29) SIGIO  
30) SIGPWR    31) SIGSYS    35) SIGRTMIN   36) SIGRTMIN+1  
37) SIGRTMIN+2 38) SIGRTMIN+3 39) SIGRTMIN+4 40) SIGRTMIN+5  
41) SIGRTMIN+6 42) SIGRTMIN+7 43) SIGRTMIN+8 44) SIGRTMIN+9  
45) SIGRTMIN+10 46) SIGRTMIN+11 47) SIGRTMIN+12 48) SIGRTMIN+13  
49) SIGRTMIN+14 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12  
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8  
57) SIGRTMAX-7 58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4  
61) SIGRTMAX-3 62) SIGRTMAX-2 63) SIGRTMAX-1  
paul@obsidian:~$
```

# Przekierowanie wejścia/wyjścia

- Możemy przekierowywać wyjście programu  
np. `ls > pliki.txt`
- Takoz wejście  
np. `wc < pliki.txt`
- Oraz wyjście błędów  
np. `moj_program.py 2>&1`



# Łączy nienazwane i strumienie

- Możemy przekierowywać wyjście jednego programu na wejście innego, tworząc potok przy pomocy znaku |  
np. `ls | wc`
- Także razem z wyjściem błędów:  
`program 2>&1 | less`
- To tworzy plik nienazwany, do którego jednocześnie piszemy i czytamy.
- Jest to buforowane, więc zajmuje pamięć