# AI224: Operating Systems — Lecture 3

Dr. Karim Lounis

Jan - May 2025

The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

# Process Management

# Processes

## Definition

**Process.** Is an instance of a computer program that is being executed.

- A process consists of:

- **An address space.** Memory locations where the program's code, variables, and function frames are stored.
- **The CPU state.** The values of CPU registers (e.g., PC, SP, etc) during the execution of the process (and while it is waiting/blocked).
- **A set of OS resources.** Represents the logical and physical resources the process holds on, e.g., open files, network connections, ...
- **A lifetime.** Since its creation, a process goes through different phases: created-executed-[interrupted-resumed]-terminated.

A program is a passive entity, whereas a process is its active form.

During its lifetime, a process is identified by a unique [temporary] number called PID (Process IDentifier) (e.g., in GNU/Linux ps -aux or top).

# Processes (top)

Output of `$ top` command run on MacOS Monterey (Apple M1):

```
Processes: 492 total, 2 running, 490 sleeping, 2315 threads                                                                              11:27:07
Load Avg: 1.50, 1.48, 1.57  CPU usage: 1.64% user, 3.88% sys, 94.47% idle  SharedLibs: 328M resident, 75M data, 20M linkedit. MemRegions: 325030 total, 2302M resident, 189M private, 1021M shared.
PhysMem: 7349M used (1287M wired), 299M unused. VM: 215T vsize, 3823M framework vsize, 31094(0) swapins, 33552(0) swapouts. Networks: packets: 236609/234M in, 182407/34M out.
Disks: 648222/27G read, 220893/4050M written.
```

| PID | COMMAND | %CPU | TIME | #TH | #WQ | #PORT | MEM | PURG | CMPRS | PGRP | PPID | STATE | BOOSTS | %CPU_ME | %CPU_OTHRS | UID | FAULTS | COW | MSGSENT | MSGRECV | SYSBSD | SYSMACH | CSW | PAGEIN | IDLEW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | kernel_task | 15.6 | 04:01.15 | 503/8 | 0 | 0 | 10M | 0B | 0B | 0 | 0 | running | 0[0] | 0.00000 | 0.00000 | 0 | 29394 | 0 | 5076943+ | 3593973+ | 0 | 0 | 18437867+ | 0 | 1035923+ |
| 375 | WindowServer | 8.1 | 04:03.30 | 21 | 5 | 1990 | 397M+ | 13M+ | 61M | 375 | 1 | sleeping | *0[1] | 0.06698 | 0.33306 | 88 | 895055+ | 8651 | 2666252+ | 1812699+ | 5205241+ | 9465263+ | 2036761+ | 2170 | 88570+ |
| 3772 | top | 5.4 | 00:01.62 | 1/1 | 0 | 28 | 5937K | 0B | 0B | 3772 | 632 | running | *0[1] | 0.00000 | 0.00000 | 0 | 12968+ | 70 | 981329+ | 490659+ | 50418+ | 586515+ | 318+ | 15 | 1 |
| 394 | nsurlsession | 3.4 | 00:08.15 | 7 | 6 | 109 | 5857K | 0B | 1472K | 394 | 1 | sleeping | 0[63] | 0.00000 | 0.78562 | 242 | 17676+ | 132 | 7974+ | 4410+ | 495898+ | 17660+ | 110953+ | 289 | 464+ |
| 732 | Google Chrom | 1.7 | 00:51.06 | 12 | 1 | 146 | 43M+ | 0B | 15M | 563 | 563 | sleeping | *0[3] | 0.00000 | 0.00000 | 501 | 43445+ | 660 | 498875+ | 367936+ | 1489859+ | 855665+ | 399259+ | 265 | 18034+ |
| 533 | distnoted | 1.0 | 00:33.43 | 2 | 1 | 402 | 2417K | 0B | 336K | 533 | 1 | sleeping | *0[1] | 0.00000 | 0.98354 | 501 | 745 | 46 | 21707+ | 42674+ | 335965+ | 107221+ | 45964+ | 1 | 0 |
| 3715 | com.apple.St | 0.9 | 00:02.27 | 2 | 1 | 35 | 5569K | 0B | 0B | 3715 | 1 | sleeping | 0[0] | 0.00000 | 0.83900 | 242 | 7588+ | 60 | 4607+ | 3097+ | 14035+ | 9331+ | 3061+ | 11 | 0 |
| 594 | Terminal | 0.8 | 00:01.70 | 9 | 4 | 274 | 92M | 21M | 6208K | 594 | 1 | sleeping | *0[164] | 0.05470 | 0.00000 | 501 | 15926 | 305 | 16180+ | 5184+ | 25437+ | 32628+ | 10855+ | 907 | 199+ |
| 563 | Google Chrom | 0.8 | 02:05.26 | 39 | 1 | 929 | 276M | 0B | 115M | 563 | 1 | sleeping | *0[390] | 0.00000 | 0.00000 | 501 | 325224+ | 2455 | 2168914+ | 1142557+ | 2930409+ | 7213120+ | 1338609+ | 4190 | 16699+ |
| 593 | AdobeReader | 0.8 | 00:31.37 | 17 | 4 | 233 | 103M | 0B | 73M | 593 | 1 | sleeping | 0[153] | 1.74644 | 0.00000 | 501 | 15525 | 745 | 37210+ | 5879 | 204898+ | 3643055+ | 489734+ | 266 | 119801+ |
| 355 | distnoted | 0.5 | 00:16.09 | 2 | 1 | 194 | 1505K | 0B | 416K | 355 | 1 | sleeping | *0[1] | 0.00000 | 0.49056 | 241 | 593 | 46 | 223959+ | 33842+ | 190839+ | 291596+ | 22607+ | 6 | 0 |
| 359 | AdskLicensin | 0.4 | 00:15.96 | 12 | 0 | 47 | 12M | 0B | 6568K | 359 | 1 | sleeping | *0[1] | 0.00000 | 0.00000 | 499 | 7887 | 69 | 4468 | 1335 | 975288+ | 4344 | 831820+ | 1039 | 223846+ |
| 491 | mobileassetd | 0.3 | 00:19.95 | 6 | 4 | 158 | 11M | 0B | 4864K | 491 | 1 | sleeping | 0[39] | 0.20708 | 0.25060 | 0 | 17615 | 637 | 2357+ | 2997+ | 93109+ | 8562+ | 29709+ | 111 | 252 |
| 1558 | com.apple.Dr | 0.2 | 00:03.12 | 3 | 2 | 5106 | 6177K | 0B | 1872K | 1558 | 1 | sleeping | 0[1] | 0.00000 | 0.00000 | 270 | 488 | 32 | 43831+ | 33130+ | 67521+ | 22899+ | 45636+ | 51 | 3344+ |
| 3773 | screencaptur | 0.2 | 00:00.45 | 4 | 2 | 145 | 8434K | 752K | 0B | 601 | 601 | sleeping | *0[329+] | 0.00894 | 0.00000 | 501 | 6072+ | 773+ | 3671+ | 1063+ | 2489+ | 8690+ | 2508+ | 65 | 2 |
| 1005 | Google Chrom | 0.2 | 00:18.54 | 19 | 2 | 282+ | 96M+ | 0B | 33M | 563 | 563 | sleeping | *0[6] | 0.00000 | 0.00000 | 501 | 59968+ | 691 | 57111+ | 27064+ | 175374+ | 160350+ | 101587+ | 0 | 3672+ |
| 1801 | com.apple.Am | 0.2 | 00:01.22 | 4 | 2 | 72 | 3585K | 0B | 1792K | 1801 | 1 | sleeping | *1[1] | 0.13531 | 0.00000 | 0 | 877 | 89 | 6781+ | 3136+ | 21681+ | 15819+ | 11970+ | 67 | 3 |
| 829 | RdrCEF Helpe | 0.2 | 00:10.14 | 10 | 2 | 154 | 43M | 0B | 28M | 768 | 768 | sleeping | *1[5] | 0.00000 | 0.00000 | 501 | 7431 | 744 | 2146 | 227193+ | 200648+ | 538644+ | 335092+ | 374 | 41612+ |
| 309 | fseventsd | 0.1 | 00:08.81 | 10 | 1 | 273 | 4545K | 0B | 736K | 309 | 1 | sleeping | *0[1] | 0.00000 | 0.00000 | 0 | 18016+ | 62 | 31146+ | 12061+ | 327421+ | 54778+ | 61551+ | 18 | 5703+ |
| 368 | corebrightne | 0.1 | 00:05.75 | 4 | 3 | 138 | 4545K | 0B | 2048K | 368 | 1 | sleeping | *0[1] | 0.14601 | 0.00000 | 0 | 13367+ | 97 | 19514+ | 26739+ | 117145+ | 42183+ | 63886+ | 411 | 1069 |
| 305 | logd | 0.1 | 00:16.26 | 4 | 3 | 1535 | 20M | 0B | 17M | 305 | 1 | sleeping | *0[1] | 0.00000 | 0.01535 | 0 | 62027 | 69 | 33618+ | 38741+ | 292117+ | 49405+ | 113687+ | 46104 | 143 |
| 752 | Google Chrom | 0.1 | 00:06.80 | 13 | 1 | 122 | 25M+ | 0B | 18M | 563 | 563 | sleeping | *0[3] | 0.00000 | 0.00000 | 501 | 13370+ | 661 | 104867+ | 56497+ | 125905+ | 237759+ | 92602+ | 8 | 1929+ |
| 500 | softwareupda | 0.1 | 00:20.67 | 5 | 3 | 158 | 20M | 0B | 2448K | 500 | 1 | sleeping | 0[32] | 0.00000 | 0.15176 | 200 | 16088 | 168 | 152840+ | 80700+ | 148668+ | 122827+ | 45766+ | 1139 | 179 |
| 316 | powerd | 0.1 | 00:04.00 | 4 | 3 | 136 | 4065K | 0B | 992K | 316 | 1 | sleeping | *0[1] | 0.00000 | 0.06698 | 0 | 14462+ | 106 | 50301+ | 28669+ | 85401+ | 96675+ | 35748+ | 119 | 188 |
| 727 | Google Chrom | 0.1 | 01:06.08 | 22 | 5 | 347 | 405M | 2048K | 56M | 563 | 563 | sleeping | *1[5] | 0.00000 | 0.00000 | 501 | 151513 | 730 | 856575 | 867430+ | 849890+ | 2097089+ | 783266+ | 657 | 11812 |
| 1562 | PerfPowerSer | 0.0 | 00:07.17 | 5 | 3 | 805 | 10M | 256K | 2288K | 1562 | 1 | sleeping | 0[29] | 0.00000 | 0.00000 | 0 | 6766 | 156 | 454083 | 234604+ | 57698+ | 290009 | 60896+ | 182 | 44 |
| 602 | Finder | 0.0 | 00:20.84 | 5 | 2 | 543+ | 81M+ | 0B | 33M | 602 | 1 | sleeping | *0[282] | 0.00000 | 0.00000 | 501 | 109015+ | 465 | 169982 | 88904+ | 426802+ | 605160+ | 126730+ | 3266 | 376 |
| 481 | TouchBarServ | 0.0 | 00:17.24 | 4 | 2 | 289 | 17M | 704K | 6848K | 481 | 1 | sleeping | *0[1] | 0.00000 | 0.00297 | 0 | 5557 | 147 | 121337+ | 70233+ | 430546+ | 628517+ | 277238+ | 246 | 4278 |
| 771 | Adobe Crash | 0.0 | 00:02.70 | 4 | 3 | 54 | 4897K | 0B | 2176K | 771 | 593 | sleeping | *0[2] | 0.02027 | 0.00000 | 501 | 4990+ | 60 | 24061+ | 4079+ | 37107+ | 64375+ | 37090+ | 0 | 2491+ |
| 733 | Google Chrom | 0.0 | 00:01.04 | 9 | 1 | 110 | 21M | 0B | 16M | 563 | 563 | sleeping | *0[3] | 0.00000 | 0.00000 | 501 | 11458 | 646 | 8079 | 5135 | 14345 | 18540+ | 9962+ | 21 | 733+ |
| 1219 | Google Chrom | 0.0 | 00:10.50 | 16 | 1 | 271- | 104M- | 0B | 63M | 563 | 563 | sleeping | *0[6] | 0.00000 | 0.00000 | 501 | 56005+ | 695 | 51668+ | 32066 | 116927+ | 143223+ | 67154+ | 44 | 740+ |
| 395 | distnoted | 0.0 | 00:02.05 | 2 | 1 | 32 | 1105K | 0B | 384K | 395 | 1 | sleeping | *0[1] | 0.00000 | 0.03326 | 0 | 488 | 46 | 88 | 16281+ | 97785+ | 32772+ | 17387+ | 0 | 0 |
| 479 | distnoted | 0.0 | 00:01.83 | 2 | 1 | 32 | 1121K | 0B | 448K | 479 | 1 | sleeping | *0[1] | 0.00000 | 0.02931 | 205 | 498 | 46 | 92 | 16279+ | 97475+ | 32797+ | 17343+ | 0 | 0 |
| 405 | distnoted | 0.0 | 00:01.63 | 2 | 1 | 29 | 1137K | 0B | 464K | 405 | 1 | sleeping | *0[1] | 0.00000 | 0.02291 | 202 | 460 | 46 | 89 | 16277+ | 81583+ | 32790+ | 17276+ | 0 | 0 |
| 323 | AdskAccessSe | 0.0 | 00:02.34 | 6 | 1 | 77 | 8878K | 0B | 6820K | 323 | 1 | sleeping | *0[1] | 0.00000 | 0.00000 | 0 | 10472 | 116 | 9520+ | 4734+ | 68007+ | 8118+ | 41205+ | 2996 | 10676+ |
| 466 | distnoted | 0.0 | 00:01.62 | 2 | 1 | 29 | 1105K | 0B | 448K | 466 | 1 | sleeping | *0[1] | 0.00000 | 0.02156 | 88 | 470 | 46 | 90 | 16276+ | 81578+ | 32790+ | 17195+ | 0 | 0 |
| 330 | mds | 0.0 | 00:16.49 | 8 | 5 | 322 | 37M | 0B | 8960K | 330 | 1 | sleeping | *0[1] | 0.00000 | 0.00000 | 0 | 58350 | 143 | 24430 | 18966 | 518974+ | 74993 | 169111+ | 284 | 4396+ |

Here the first process is `kernel-task` with PID=0. Also, the process related to the `top` command, which displayed this list of processes, has PID=3772.

# Processes (top)

Output of `$ top` command run on GNU/Linux Debian:

```
top - 06:31:08 up 1 min,  1 user,  load average: 0.30, 0.11, 0.04
Tasks: 229 total,   1 running, 228 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3925.8 total,   2333.6 free,    783.2 used,    809.1 buff/cache
MiB Swap:    976.0 total,    976.0 free,      0.0 used.   2878.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
     14 root      20   0       0      0      0 I   0.3   0.0   0:00.27 kworker/0:1-events
   1301 jeffery   20   0 4587260 197060 107568 S   0.3   4.9   0:04.56 gnome-shell
   1794 jeffery   20   0  402080  49320  39200 S   0.3   1.2   0:00.75 gnome-terminal-
   2118 jeffery   20   0   10100   3352   2616 R   0.3   0.1   0:00.15 top
      1 root      20   0  164824   9956   7260 S   0.0   0.2   0:01.98 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.03 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0-events
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
      7 root      20   0       0      0      0 I   0.0   0.0   0:00.01 kworker/u16:0-flush-254:0
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
      9 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.01 ksoftirqd/0
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.25 rcu_sched
     13 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/0
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     16 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     17 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/1
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.01 ksoftirqd/1
     19 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0-events
     20 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-events_highpri
```

Here the first process is `systemd` (PID=1), which used to be `init` (process responsible for starting all other processes). Also, the process related to the `top` command, which displayed this list of processes, has PID=2118.

# Types of Processes

Two types of processes: system processes that execute system codes, and user processes that execute user codes (but treated in the same way).

System processes are executed in kernel mode (master mode — system mode) whereas user processes are executed in user mode (slave mode).

E.g., Google Chrome, Texteditor, IDE, etc processes are user processes, whereas system update, demons, event logger, etc are system processes.

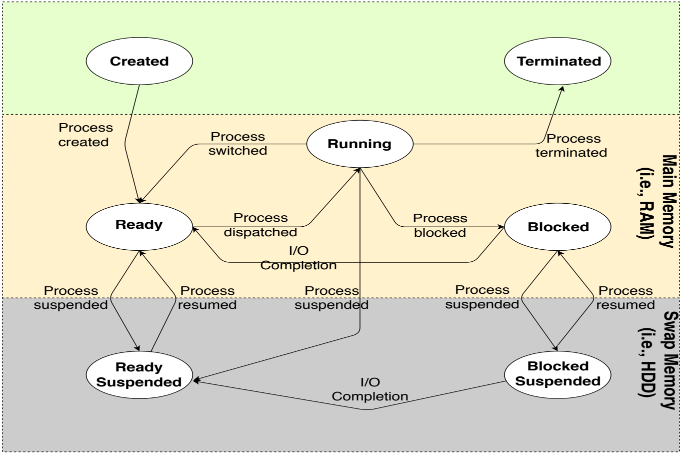
Various terms are used (interchangably): process, job, task (batch mode).

- **Process**. Defined ealier (cf., previous slide).
- **Task**. Could be a process or a thread performing some computation (e.g., printing, downloading, updating, etc). When multiple tasks are executed, that is multitasking — multithreading or multiprocessing.

  In Java, when a thread is running, it is called a task.
- **Job**. A complete unit of work. It could be a set of tasks.

# Process States (Process State Graph)

During its lifetime, a process transits from one state to another as follows:



# Process States (Process State Graph)

The different states are defined as follows:

- **New.** Just been created.
- **Ready.** Having all needed resources and waiting (ready queue) for CPU to be allocated.
- **Running.** Being executed on the CPU.
- **Blocked.** Waiting (blocked queue) for another process to terminate (e.g., I/O routines) or an event to happen (e.g., signal reception).
- **Suspended.** Swapped out of RAM for some reason, e.g.,:
  - Swapped out of RAM to free some space for other process.
  - Intentionally suspended by superuser ([8am-6pm]) to save CPU cycles.
  - Swapped out of RAM after being longly waiting for an event to happen.
  - Swapped out by the parent process.
- **Terminated.** The process has finished its task and/or have been killed (e.g., in Unix-like operating systems the command kill -9 2730 is used to terminate the process which PID is 2730).

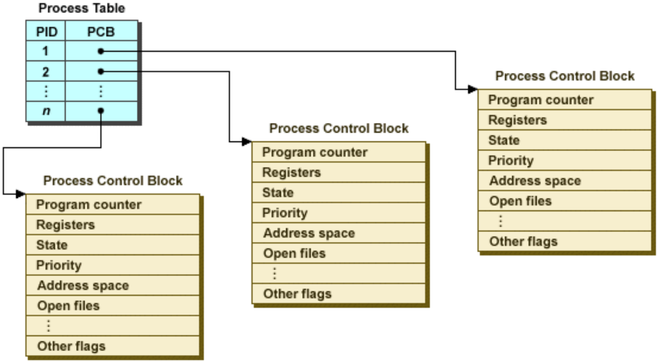
# Process States (Process State Graph)

On Unix-like operating systems, processes could be in:

- **Running — R.** Currently executing on the CPU.
- **Sleeping — S.** Waiting for a resource to be available.
- **Stopped — T.** Suspended and can be resumed.
- **Idle — I.** Idle kernel threads — waiting for a task to be assigned.
- **Zombie — Z.** Processes finshed, but father not.
- **Dead — X.** Process has terminated (not visible in the list).

# Process Control Block

Every process is represented by a data structure called PCB (Process Control Block) that stores information about the process last state:




Enough information to resume execution and track process.

# Process Control Block (GNU/Linux ps)

```
jeffery@pc-debian:~$> cat dummy.c
void main()
{
 while(1==1);
 return;
}
jeffery@pc-debian:~$> gcc dummy.c -o dummy.exe
jeffery@pc-debian:~$> ls -l dummy.c dummy.exe
-rw-r--r-- 1 jeffery jeffery   39 Feb  5 08:16 dummy.c
-rwxr-xr-x 1 jeffery jeffery 9288 Feb  5 08:21 dummy.exe
jeffery@pc-debian:~$>
```

```
jeffery@pc-debian:~$> ./dummy.exe &
[1] 2427
jeffery@pc-debian:~$> ps
    PID TTY          TIME CMD
   2267 pts/3    00:00:00 bash
   2427 pts/3    00:00:02 dummy.exe
   2429 pts/3    00:00:00 ps
jeffery@pc-debian:~$> kill -STOP 2427

[1]+  Stopped                 ./dummy.exe
jeffery@pc-debian:~$> kill -CONT 2427
jeffery@pc-debian:~$> kill -9 2427
jeffery@pc-debian:~$>
```

```
jeffery@pc-debian:~$> ps -o pid,ppid,comm,args,stat,user,pri,policy,nlwp,etime,rss,vsz,sz -p 2427
    PID    PPID COMMAND         COMMAND                     STAT USER      PRI POL NLWP     ELAPSED   RSS    VSZ    SZ
   2427    2267 dummy.exe       ./dummy.exe                 R    jeffery    19 TS      1       07:59   404   1796   449
jeffery@pc-debian:~$> ps -o pid,ppid,comm,args,stat,user,pri,policy,nlwp,etime,rss,vsz,sz -p 2427
    PID    PPID COMMAND         COMMAND                     STAT USER      PRI POL NLWP     ELAPSED   RSS    VSZ    SZ
   2427    2267 dummy.exe       ./dummy.exe                 T    jeffery    19 TS      1       09:24   404   1796   449
jeffery@pc-debian:~$> ps -o pid,ppid,comm,args,stat,user,pri,policy,nlwp,etime,rss,vsz,sz -p 2427
    PID    PPID COMMAND         COMMAND                     STAT USER      PRI POL NLWP     ELAPSED   RSS    VSZ    SZ
   2427    2267 dummy.exe       ./dummy.exe                 R    jeffery    19 TS      1       09:59   404   1796   449
jeffery@pc-debian:~$> ps -o pid,ppid,comm,args,stat,user,pri,policy,nlwp,etime,rss,vsz,sz -p 2427
    PID    PPID COMMAND         COMMAND                     STAT USER      PRI POL NLWP     ELAPSED   RSS    VSZ    SZ
jeffery@pc-debian:~$>
```

Here, I create a program that spins around forever (`dummy.c`). Then, I run it, suspend it, resume it, kill it, and observe certain of its PCB's fields.

# Process Creation and Termination (Unix-like Systems)

**Creation.** A process can create other processes (children).

- In Unix-like systems, a process executes the *fork*() system call.

  The OS duplicates the parent PCB and makes appropriate changes.
- The parent process can continue its execution or wait (call *wait*()).
- A process can pass data to another process (using pipes).
- The address space can be, either a duplication of the parent space (same data and code), or a newly loaded program (call *execlp*()).
- Resources are inherited (e.g., open files, variables, privileges, ...).

**Termination.** Occurs as follows:

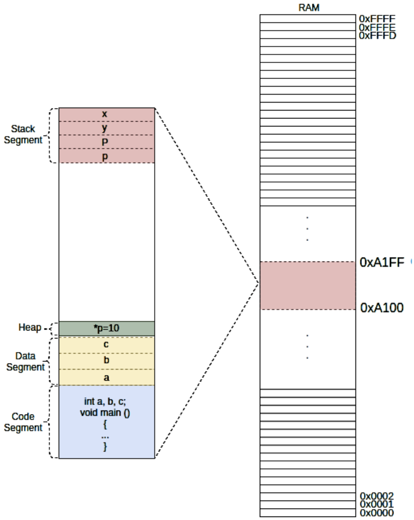
- A process executes the *exit*() system call.
- Running *exit*() returns an integer to the parent.
- A parent process can terminate a child using *kill*() system call.

# Process Creation and Termination (Unix-like Systems)

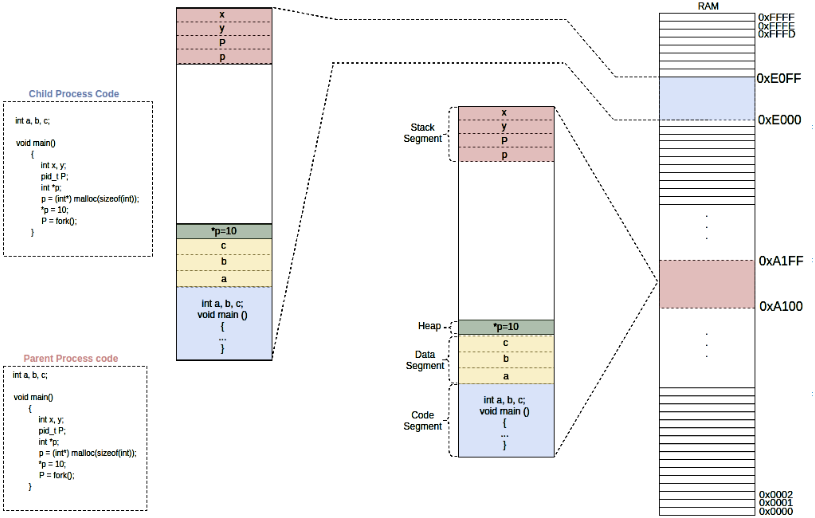
Parent Process code

```
int a, b, c;

void main()
    {
        int x, y;
        pid_t P;
        int *p;
        p = (int*) malloc(sizeof(int));
        *p = 10;
        P = fork();
    }
```



# Process Creation and Termination (Unix-like Systems)



# Process Creation and Termination (Unix-like Systems)

## Exercise

**Q1.** Given the following parent process code snippet, how many processes would be created in total?

```
...
for(unsigned int i=1; i<3; i++)
{
    fork();
    printf("Welcome to ENSIA");
}
...
```

**Q2.** How many times the message “Welcome to ENSIA” gets printed?

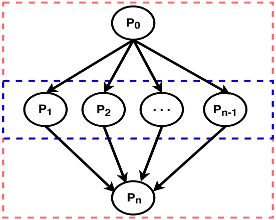
Try it at: https://www.programiz.com/c-programming/online-compiler/

# Concurrent Processes

Processes are said to be concurrent when:

- Multiple programs are loaded into the main memory, i.e., RAM.
- Their execution is happening in parallel or in fake-parallel.
- Their time durations or executions are overlapping or interleaving.

We can use PPG (Process Precedence Graph) to visualize execution order of concurrent processes. An execution can be sequential (serial) or parallel:




```
Begin
      P0
      ParBegin
            P1
            P2
            ...
            Pn-1
      ParEnd
      Pn
End
```

Figure: Processes Precedence Graph (Left) and its pseudocode (Right)

# Interprocess Communication (a glance)

Processes can cooperate with each other for several reasons:

- **Information Sharing.** Collaborating on the same file.
- **Computational Speedup.** Parallel execution of different parts of a program (multiprocessors systems only).
- **Modularity.** An operating system can be constructed in a modular fashion. E.g., modern operating systems.
- **Convenience.** An individual users may run different processes.

Operating systems provides two fundamental mechanisms:

1. **Shared memory** Need a shared memory region (localhost):
    - Files: Three processes concurrently read/write into a same file (Lab 4)
    - Signals: 2 processes cooperating for a sum using wait/exit (Lab 4)
    - Pipes: Writing into a file and reading from the other end (Lab 5)
    - Variables: Multithreading program to crack password (Lab 6)
2. **Message passing.** Need a physical/logical communication link:
    - Sockets: Processes communicating while running on different PCs.
    - RPCs: Processes invoking functions implemented on different PCs.

# Interrupts

Interrupts & Interrupt handlers

# Interrupts

### Definition

**Interrupt.** Is an event that alters the sequence of instructions executed by the CPU. A dedicated program called ISR (Interrupt Service Routine) or interrupt handler is executed as a consequence.

Each interrupt has it own interrupt handler (written in assembly language).

### Definition

**Interrupt handler.** A.k.a., ISR (Interrupt Service Routine), is a low-level program that is executed in kernel mode to service the cause of the interrupt.

Interrupt handlers are located by an IVT (Interrupt Vector Table)[1] stored in the memory. E.g., in x86 architectures, it is generally stored at address 0x0000-0x03FF. The number of interrupt types is limited by the architecture.

[1] Also check IDT (Interrupt Descriptor Table) used on modern x86 architectures

# Types of Interrupts

**Software-based Interrupts.** When the system is interrupted following the execution of an instruction in a program code, the interrupt is said to be software-based. There are two types:

1. **Exceptions.** An exception happens when the current instruction perform an illegal action such as: division by 0, arithmetic overflow, buffer overflow, page fault, access protected memory space, or attempt to execute a privileged instruction (process is terminated).

2. **System calls.** A.k.a., traps. User program are executed in user mode. In this mode, programs do not have direct access to peripheral. They have to issue a **system call** which causes an interrupt (INT instruction).

**Hardware-based Interrupts**: These asynchronous interrupts are trigged by a hardware unit such as the timer/clock, keyboard, mouse, power button, . . ., device controllers or DMA, to get the attention from the CPU. They can be maskable (MI) or non-maskable (NMI).

# Types of Interrupts

- **Maskable.** These are interrupts that the CPU can choose to ignore based on its current state and priority. They are typically used for non-critical events that can be handled later without causing significant issues. E.g., Keyboard input, network packets arriving, etc.
- **Non-Maskable.** These are interrupts that the CPU cannot ignore. They demand immediate attention, often for critical events that could jeopardize system stability or integrity. E.g., hardware errors, system watchdog timer expiring, etc..

# Types of Interrupts

Other types of interrupts[2]:

- **IPI (Inter-Processor Interrupt).** Used between processors to communicate in a multiprocessor system, e.g., one processor core requests another processor core to stop when the system is shutting down by the first processor (`$sutdown now`).
- **SI (Spurious Interrupt).** A.k.a., phantom or ghost interrupt, is an unwanted hardware interrupt, e.g., when an interrupt that has been signaled to the processor and it is no longer required (or interrupt source disappeared), or buggy hardware.

**Interrupt Storm.** Occurs whenever the operating system receives a large number of interrupts (from hardware) that consumes the majority of the processor's time spurious signals, interrupt handler execution took too long, . . ., faulty drivers. The system becomes non-responsive.

[2] Also, you may wanna check raster interrupts.

# System Calls

### Definition

**System Calls:** Constitutes an interface between the user (programs) and the services that are made available by an operating system.

- A systems call is a programmatic way in which a computer program requests a service from the operating system.
- Each operating system has its own system calls (e.g., GNU/Linux has 300, FreeBSD has 500, and Windows 7 has 700).
- System calls point to specific programs called routines that are written in C, C++, or assembly language.
- Operating systems provide an API (Application Programming Interface) as an interface between user programs and system calls.

E.g., `printf( )` or `cout` are APIs for calling functions from C libraries. These functions uses the `wirte()` system call to send data to the output.

# System Calls

System calls can be grouped into six categories:

1. **Process Control:** End, abort, load, execute, create process, terminate process, get/set process attributes, wait for a time, wait for an event, signal an event, allocate and free memory.

   e.g., *exec(), fork(), wait(), exit(), getuid(), getgid(), getpid(), getppid(), signal(), kill(), acquire_lock(), release_lock().*

2. **File Management:** Create, delete, open, and close files, read, write on files, get/set files attributes.

   e.g., *create(), open(), close(), read(), write(), lseek(), dup(), link(), unlink(), stat(), fstat(), access(), chmod(), chown, umask, ioctl().*

# System Calls

3. **Device Manipulation:** Request device, release device, read, write on device, get/set devices attributes.

   e.g., *request(), release(), ioctl(), read(), write(), open(), close().*

4. **Information Maintenance:** get/set time or data, get/set system data, get/set resource attributes.

   e.g., *time(), date(), alarm().*

5. **Communication:** Create, delete a network connection, send and receive messages, transfer status information.

   e.g., *pipe(), mmap(), shm_open(), gethostid().*

6. **Protection:** Set ownership and access rights on a resource.

   e.g., *chmod(), umask(), chown().*

# Hardware Interrupt mechanism

How Hardware events (keyboard presses, incoming network packets, mouse movement, ...) are escalated to running programs?

There exists two ways:

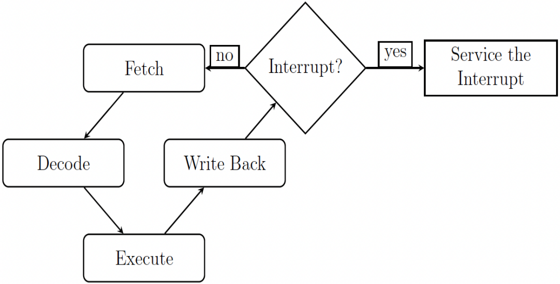
• Polling: When OS periodically queries each device to see if new information is available (simple but high latency and CPU cycle wasting).

• Interrupts: Device sends a signal to **interrupt controller**[3] to request attention, CPU preempts running process to handle device request.

After each (user mode) instruction is executed, the CPU checks whether the interrupt controller has any interrupt pending. If an interrupt is there, the current process is "interrupted", and the appropriate handler is executed.

[3] PIC & APIC, generally, integrated circuits within the southbridge.

# Hardware Interrupts and Interrupt Service Routines




Hardware interrupts.

- End