

APPS@UCU

Linux course

Processes

Morhunenko Mykola



Contents

- 1 Processes. Introduction. General knowlage
- 2 Process states
- 3 Processes Identifiers
- 4 Scedualing
- 5 Real time processes
- 6 Sources

Processes. Introduction

Process

- What is the difference between the program and the process?
- **Program** - a file, containing some information that describes how to construct a process in a runtime
- It includes:
 - Binary format identification - some metainformation about the format of executable file. Nowadays UNIX executable files called **Executable and Linking format (ELF)**
 - Machine-language instructions - main algorithm of the program
 - Program entry-point address
 - Data
 - Symbol and relocation tables
 - Some other information, more about that on the [Operating systems course](#)
- In our case - **process** - a program loaded to a memory for execution, with all it's allocated resources

Memory layout of the process (recall PoCO)

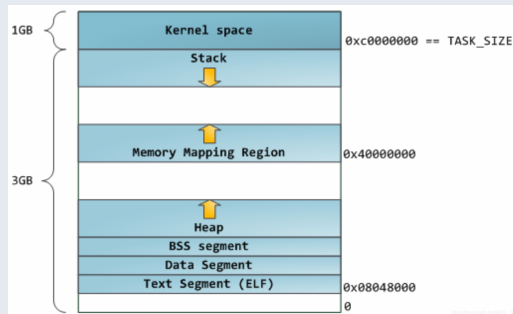
- For each process there is some amount of allocated memory
- Usually it refers to segments
- There are such segments as:
 - **Text segment** - read only (so the process can not modify its own code) shareable (multiple processes can share same executable code) segment with machine code instructions to the program executing
 - **Initialised data segment** - contains initialised global and static variables
 - **Uninitialised data segment** - **bss** - contains uninitialised global and static variables. When the program is stored on a disk, space for this segment is not initialised, this space is allocated by the program loader at the beginning of the runtime
 - **Heap** - memory area from which memory for variables can be allocated during the runtime
 - **Stack** - dynamically growing area for local variables storage, divided on stack frames. Each stack frame is allocated for each currently called function and stores its arguments, local variables, return address and return value

```
username $ size /bin/ls # check size of a binary file segments
      text      data      bss      dec      hex filename
132378      4840      4824   142042    22ada /bin/ls
```

Virtual memory (recall PoCO)

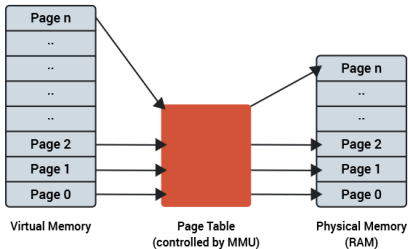
- All previously discussed layout is a layout in a **virtual memory**
- As all modern OS's nowadays, Linux kernel do a virtual memory management
- The aim of this approach - to increase efficiency of RAM-CPU communication
- **Page** - the smallest fixed-size unit of a virtual memory. In most cases on modern OS's it's size is **4KiB**
- **Page table** - one more abstraction close-related to processes. Each process has it's own page table of contents

Virtual memory scheme for one process on Linux x86-32 is shown on the slide



Virtual memory (recall PoCO)

- Just to clear things up:
- **MMU - memory management unit** - hardware component to manage all cpu-memory communication and caching
- **OS memory management** - ensures the availability of memory resources for each process continuously

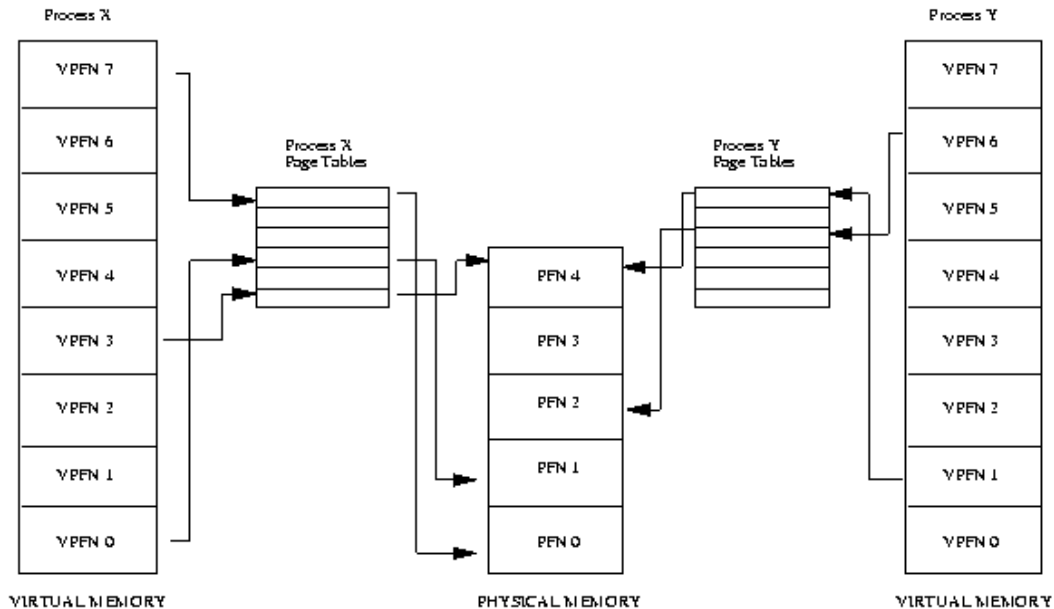


Why do we need virtual memory

Virtual memory is used for:

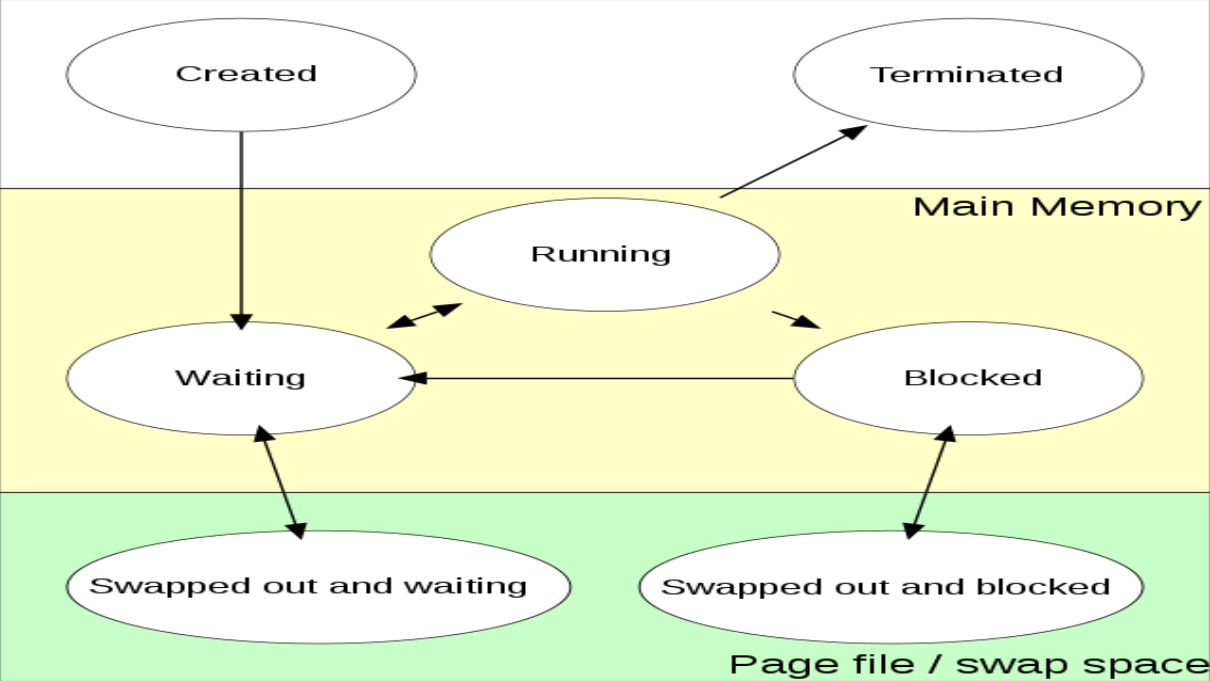
- **Abstraction** - to be free from physical memory addresses/limits
- **Processes isolation** - every process in a separate address space
- **Sharing** - a single mapping can serve multiple needs
- **Flexibility** - assign a virtual address to a file





Process states

- As a process executes it changes state according to its circumstances
- There are following states in Linux:
 - **CREATED**
 - **RUNNING & RUNNABLE** - running or ready to run
 - **Waiting** - The process is waiting for an event or for a resource
 - **INTERRUPTABLE_SLEEP** - waiting but can be interrupted by signals
 - **UNINTERRUPTABLE_SLEEP** - waiting and can not be interrupted under any circumstances
 - **STOPED** - stopped, usually by receiving a signal
 - **ZOMBIE** - process that should be terminated, but for some reason (usually some bug) it can not be terminated properly
- It is more important for scheduling





Process Identifiers

- **top** - program for system monitoring that is part of every Linux system
- **htop** , **gtop** , **ytop** , **utop** and othe system monitors may be easier to understand for begginers, but all of them are based on **top** , or inspired by it
- Here you can see almost all necessary information about your system and (what is more important) processes
- As all other high-performance software, entry bound for this application is too high
- Let's see, what is so important information about process can be found in **top** output

PID

- The very first thing, that is associated with any process, it's **PID** - process id
- It's a positive integer, and system works with processes by their PID's - names (commands) are for humans
- There are no fixed ID's for any process, with exception of **init** (more about that in the next topic). PID for **init** equals 1
- Maximum PID number for your OS can be found using the following command:

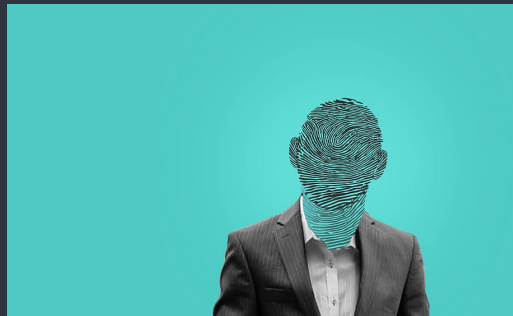
```
username $ cat /proc/sys/kernel/pid_max
```

- Also one more important PID for all processes - parent PID or PPID
- If parent of any process "died" - the child become "adopted" by the **init** process
- Parent of any process can be found like:

```
username $ cat /proc/PID/status | grep PPid
```

Other ID's

- **UID** , **GID** - also ID's that are related to the process, with **RUID** (Real User Id) , **SUID** (Saved User Id) and **EUID** (Effective User Id)
- In terms of this course it is not important, it is more related to the Linux administration



Linux Memory Types

- There are three main "memory types" in a processes administration
- The **physical memory** , limited resource
- **Virtual memory** , nearly unlimited resource
- **Swap** (optional)
- If there is no **swap** on the system, everything will be Ok, up to some point in time (but it is better to have it, and as partition, not as a file)

Sources

Sources

- "The Linux Kernel book", David A Rusling
- "Linux programming interfaces", M. Kerrisk
- Linux process states and signals, Medium(Cloud Chef)
- top manual