APPS@UCU

Linux course

Provesses

Morhunenko Mykola

APPLIED SCIENCES FACULTY

Contents

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. Nowedays 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)

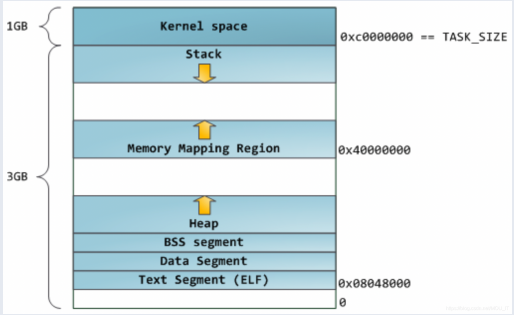
- For each process there is some amount of allocated memory
- Usually it reffere to a segments
- There are such segments as:
  - Text segmend - read only (so the process can not modify it's own code) shareble (multiple processes can share same executable code) segment with machine code instructions to 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 - dynamicly growing area for local variables storage, divided on stack frames. Each stack frame is allocated for each currently called function and stores it's 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)

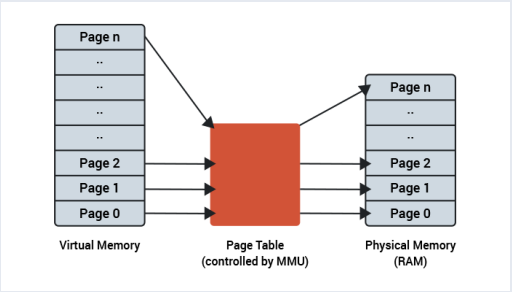
- All previocly discussed layout is a layout in a virtual memory
- As all modern OS's nowedays, Linux kernel do a virtual memory management
- The aim of this apprach - to increase efficiency of RAM-CPU communication
- Page - the smallest fixed-size unit of a virtual memory. In most cases on modern OS's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

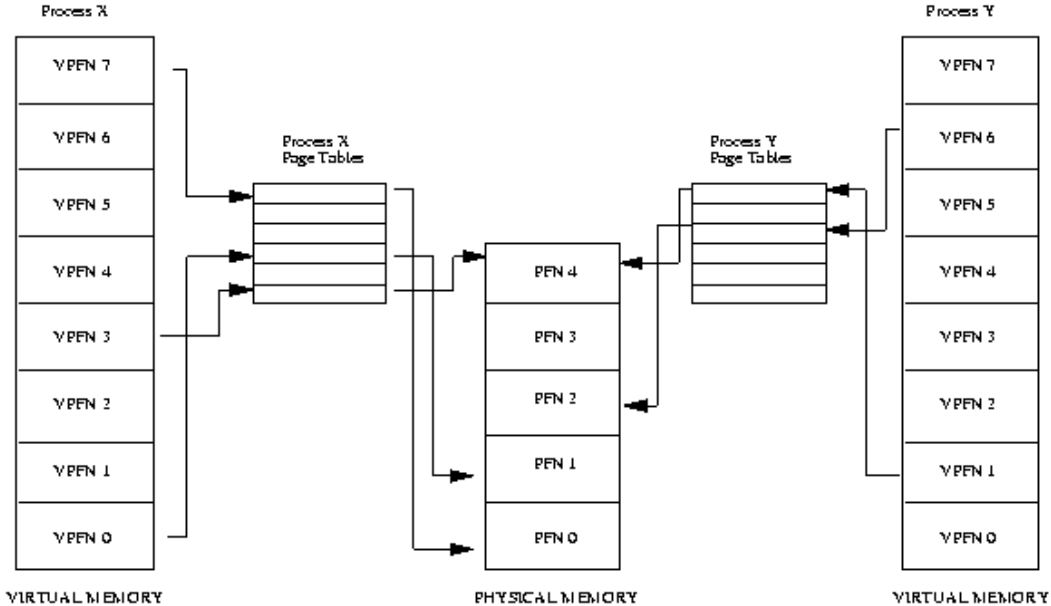
Virtual memory scheme for one process on Linux x86-32 is shown on the slide



Virtual memory (recall PoCO)

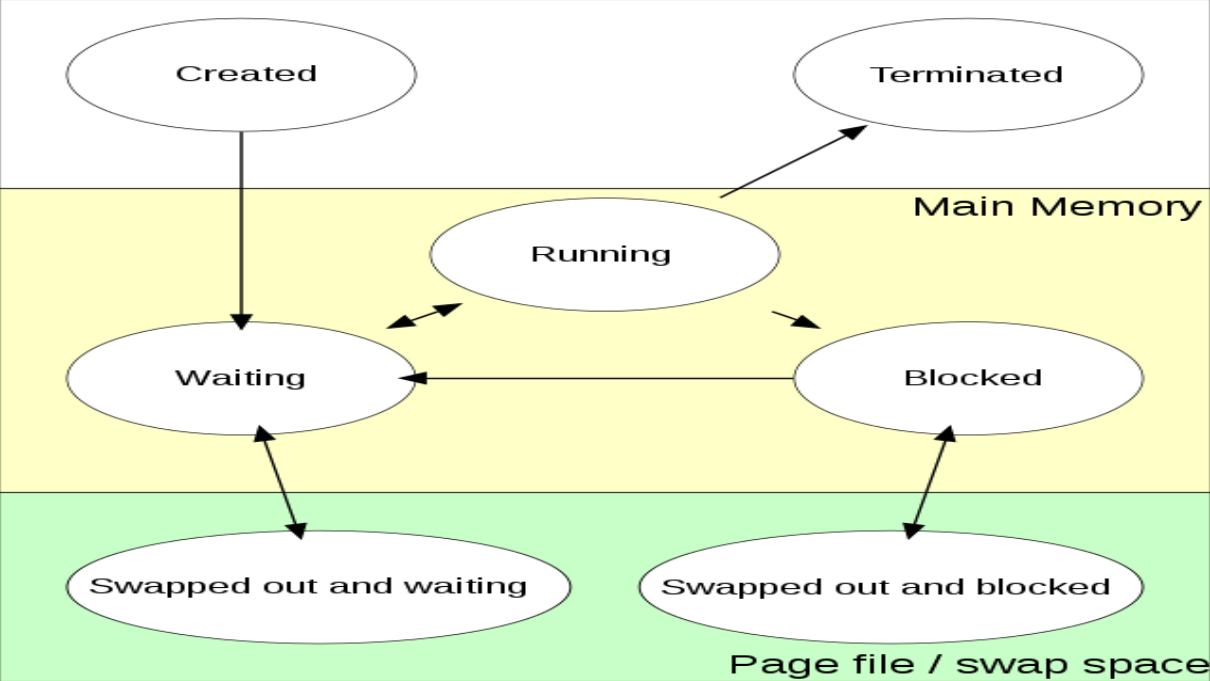
- Just to clear things up:
- MMU - memory management unit - hardware component to manage all cpu-memory communication and caching
- OS memory management - ensures the availability of memory resources for each process continuously




Process X
VPFN 7
VPFN 6
VPFN 5
VPFN 4
VPFN 3
VPFN 2
VPFN 1
VPFN 0
VIRTUAL MEMORY
Process X
Page Tables
PFN 4
PFN 3
PFN 2
PFN 1
PFN 0
PHYSICAL MEMORY
Process Y
Page Tables
Process Y
VPFN 7
VPFN 6
VPFN 5
VPFN 4
VPFN 3
VPFN 2
VPFN 1
VPFN 0
VIRTUAL MEMORY

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 scedualing


Created
Terminated
Main Memory
Running
Waiting
Blocked
Swapped out and waiting
Swapped out and blocked
Page file / swap space

Process Identifiers

Process. PID

- The very first thing, that is assosiated 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 - parrent PID or PPID
- If parrent 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
```

Sources

Sources

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