Linux Processes

> A process is a program being executed at a moment in time.

Or more precisely:

> Important
>
> A process the unit of processing managed by the operating system

As an example:

> Example
>
> If you open the Python console, then you are starting a program (Python), creating a process linked to it. If you launch two Python consoles, then you created two processes for the same program. These processes are independent between them, continuing with this example, changing making changes in one console (e.g. creating Python variables), does not affect the other process (the other Python console). It is pretty clear this you can open several times one program, creating independent instances of it, for example, opening several times Visual Studio Code or Google Chrome.

A process is made of:

- Program code: instructions that the program has to execute, an executable file (like the one that is created when compiling C or C++)
- Program data: variables creating during the execution of the program, like the variables that are added to the Python console
- Data associated with program execution:
    - CPU registers
    - State of the process (Ready, Executing or Blocked)
    - Input/Output information
    - Priority

All of this "stuff" need to be stored in memory somehow. Each process has a quantity of data allocated to it in the RAM (Random Access Memory). All of this memory between two places:

- **Process Control Block** (PCB, not to be confused with a **P**rinted **C**ircuit **B**oard!). Each process has its own PCB, stored in the Kernel Space on the RAM. All of the PCBs of all the processes are stored in what is called the Operating System Tables, inside a particular table called the Processes Table.

    > The PCB contains the registers (state of the process), the identification of the process and the control state of it. There are used to store the state of the process when it is not executing.

- **Memory Image**: Each process has its own image on the RAM.

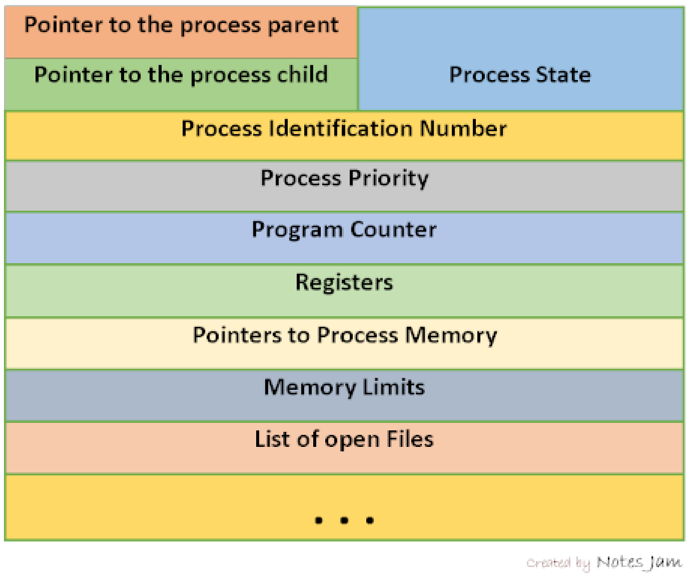
    > The image contains the program code and the program variables.

All of this makes sense, there needs to be a separation of the information related to a process. Since the registers are sensible information that should be accesses carefully, there need to be in the Kernel Space in the RAM.

PCB (Process Control Block)

Understand once it is executing all of register information changes, therefore, the PCB is outdated when the process is executing. Once it is stopped the PCB needs to be updated with the latest execution information. This needs to happen each time the process is stopped. If the process is deleted, then the PCB of it is deleted too.

Information contained in the PCB, for illustrative purposes.

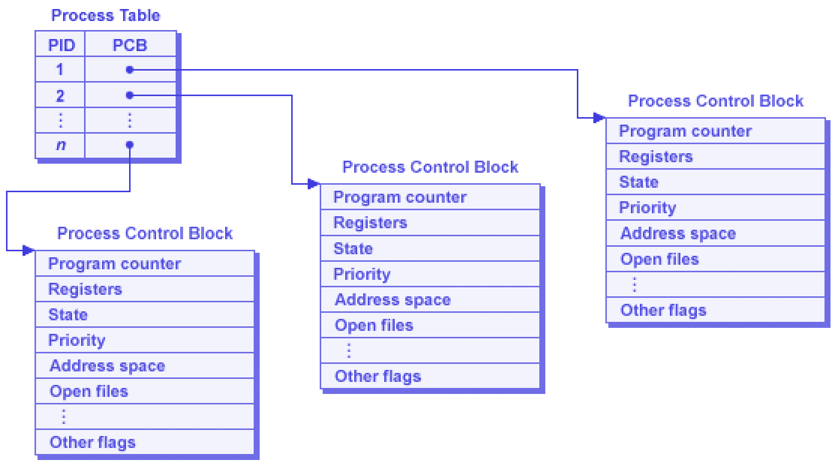


Pointer to the process parent
Pointer to the process child
Process State
Process Identification Number
Process Priority
Program Counter
Registers
Pointers to Process Memory
Memory Limits
List of open Files
• • •
Created by Notes Jam

Just remember that in the PCB there is:

- CPU Registers
- ID Information: Process ID, User ID
- Control Info: State, Memory assigned (pointers), Open Files

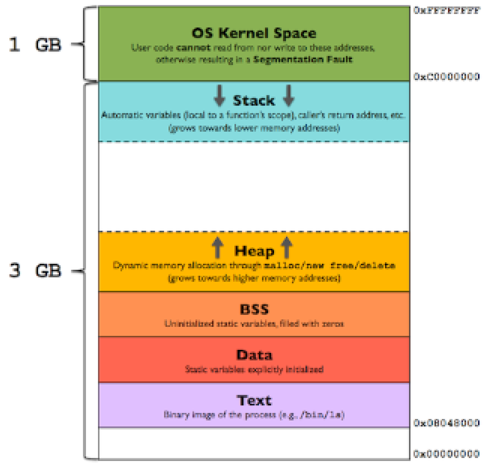
Also remember that for each of the processes running or waiting, there is a PCB stored in the Operating System Table:



Memory Image

All of the other memory that the process can use in on the memory image, accessing memory out of it required additional permissions. It is adjusted dynamically, depending of the memory needed. Just remember this.

If you want to know more then, refer to this image:

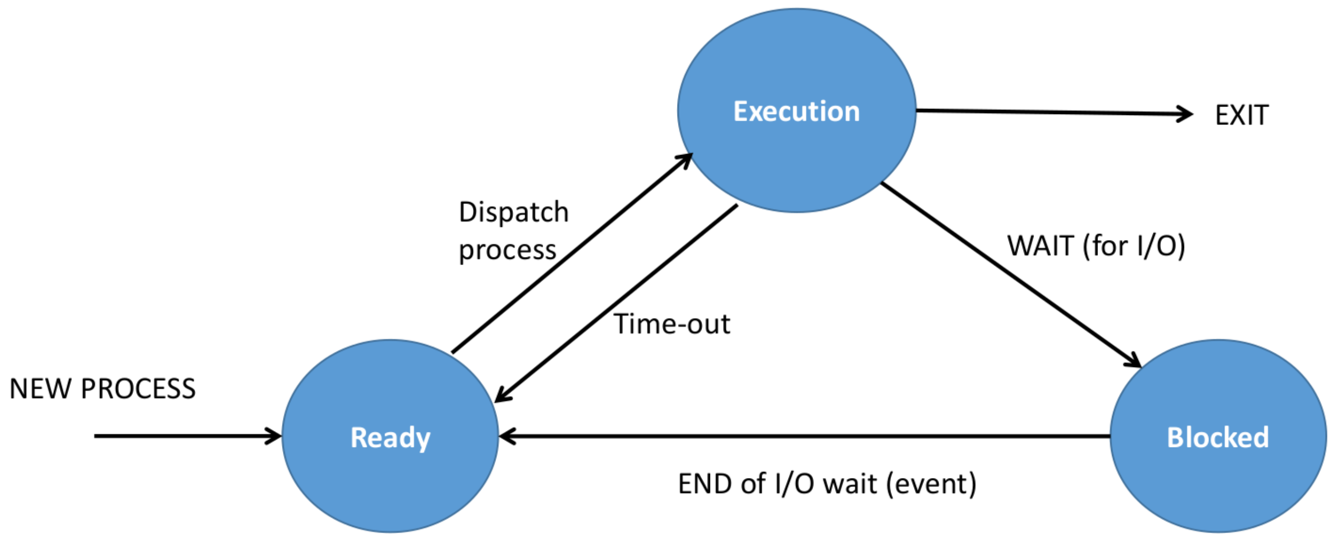


1 GB
3 GB
0xFFFFFFFF
OS Kernel Space
User code cannot read from nor write to these addresses, otherwise resulting in a Segmentation Fault
0xC0000000
Stack
Automatic variables (local to a function's scope), caller's return address, etc. (grows towards lower memory addresses)
Heap
Dynamic memory allocation through malloc/new free/delete (grows towards higher memory addresses)
BSS
Uninitialized static variables, filled with zeros
Data
Static variables explicitly initialized
Text
Binary image of the process (e.g. /bin/ls)
0x08048000
0x00000000

The memory image is the 3GB that are not in the OS kernel space. Remember that the PCB are located in the OS kernel space. And both of this things are in the RAM.

Basic Process States

A process can be in 3 different basic states:



Memorize this diagram if you can. But understating it of course.

As all thinks in life, a process has a life:

1. Process is created
2. The process is executed:
    1. It can be executing
    2. It can be waiting
3. The process ends or dies

Understand that when a process is alive (executing) sometimes it has to wait for a user input for example. This is very important to keep in mind. The execution of the process does not happen continuously, *no se ejecuta todo del tirón*, because there can be interruptions or other processes that need to be executed.

If you use the `input` keyword in Python or `scanf` in C, the program has to wait for input of the user to be provided. In this state the program is "executing" in the sense that it is alive, but it not executing any instructions on the CPU, its just waiting for something to happen.

Executing State

The most simple state. The program code is executed by the CPU. Just keep in mind that the information stored on the PCB is now being used. This state can be changed by interruptions.

Blocked State

When a process is waiting for IO or other interruptions. Even in the case when a file in opened, milliseconds or even microseconds are wasted, in that time the core of the CPU or the CPU itself should not be idle (i.e. not doing any instruction). Thus, it makes sense to execute some other process in the CPU while the other one is "waiting".

If a state is interrupted and the execution needs to be followed later, we say that it is in a Block State.

It can be interrupted because:

- **Waiting for user input/output** (the important one)
- Waiting for a file to open
- Waiting for a file to close
- Hardware interrupt

In general, there is an interruption when a process needs to solicit something from the Operating System.

Ready State

Does a process always wait in the Blocked state? No.

If the OS gets the information that the process requested, then it set to **Ready**, in the sense that *it is ready* for execution. When Block the process is waiting for the OS to handle something, and when Ready it is waiting to be executed.

If the process is waiting to be executed, we say that is in a Ready State.

Usually this happens when a Blocked process has received the data that it needs and it want to be executed again. It cannot go directly to the Execute state it has to be set to Ready first.

Viewing the diagram it is clear. A process can go from Execute to both Ready and Blocked, but it cannot go from Blocked to Execute. This is because an executing state can be paused temporally just setting it to the Ready state.

As an example of this particular behavior:

1. A - Execute and B - Ready
2. A - Blocked (waiting for user input)
3. B - Execute
4. A - Ready (has received the data)
5. B - Ready (stopped the execution)
6. A - Execute (the execution of A resumes)

Context Switch

There is a context switch when we change the execution of a process to another one. In other words, **the sequence of events of switching the CPU from one process, task or thread to another**.

Changing process B to C

We want to go from:
- B executing
- C ready
To:
- B blocked
- C executing

The context switch consists of:

1. Save CPU state of B in PCB
2. Change B to block
3. Decide which process to execute now
4. Restore to state of C from the PCB to the CPU
5. Start the execution of C (change PC register)

Note that the Memory Image is always stored in the RAM. What it is changed on a Context Switch is the OS table containing the PCB, since we need to save and load the Process states to and from the CPU.

Scheduler

Part of the OS that decides what process is going to be executed next.

Dispatcher

Does the context switch itself. Loads and saves from and to the PCB.

- Prepares the process to be executed
- Restores the CPU of the process from its PCB
- Restarts the execution of the new process in the correct instructions (remember that the PC is taken from the PCB)

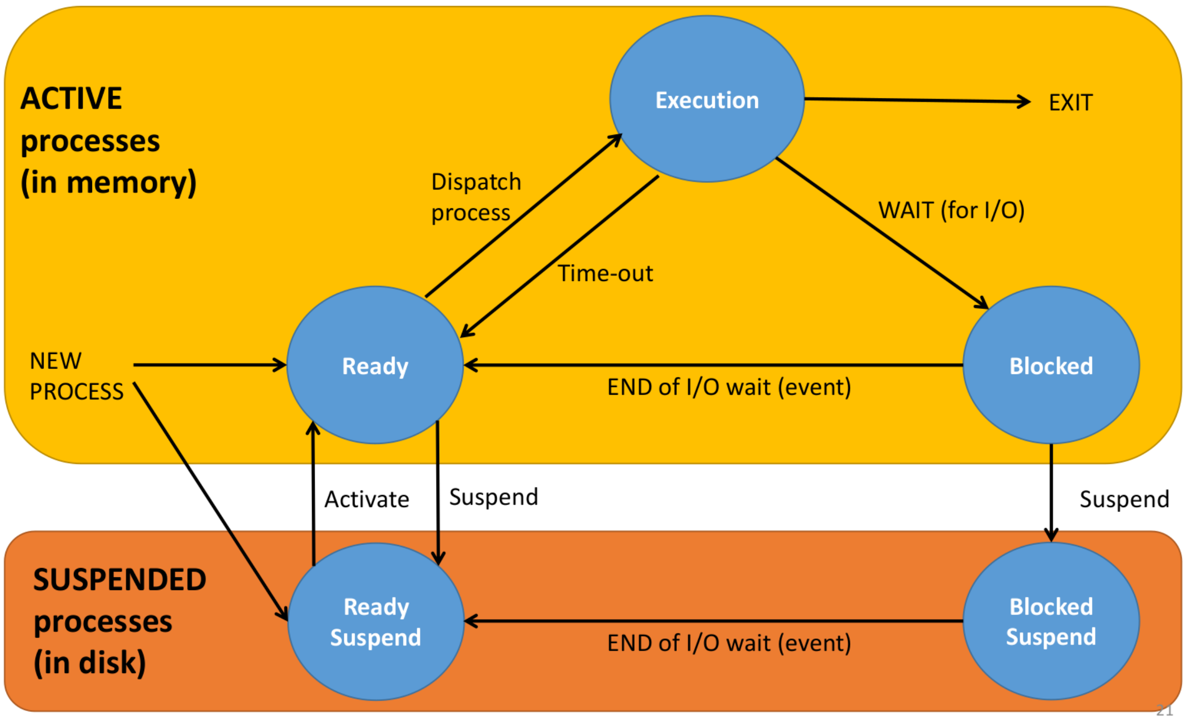
Suspend

A process is suspended when it resides in the Disk and not on the RAM of the computer, e.i. the Memory Image on the process is on the Disk.

Then introducing this concepts we also add two new states:

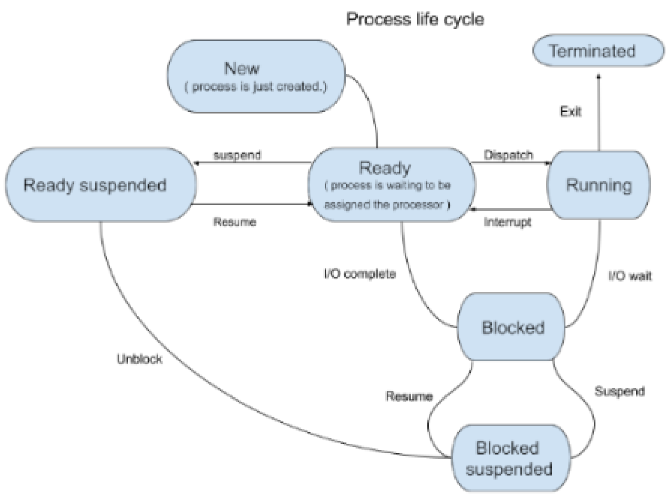
- Ready Suspend - Waiting to be executed
- Blocked Suspend - Waiting for IO or other data

Of course, they are the same but the process is stored in the disk, not in the RAM. We say that a process is Activated when going from a suspend state to one of the "normal" ones.



Complete States of a Process

This is not important, included for illustrative purposes.


Process life cycle
New
(process is just created.)
Terminated
Exit
Ready suspended
suspend
Resume
Ready
(process is waiting to be assigned the processor)
Dispatch
Interrupt
Running
I/O complete
I/O wait
Blocked
Unblock
Resume
Suspend
Blocked suspended