# Process

## Overview

* A process is a program in execution
* A process is more than the program code, which is sometimes known as the text section
* A process also includes the current activity, as represented by the value of the program counter and the contents of the processor’s registers
* A process includes:
* Stack: contains temporary data such as function parameters, return addresses, and local variables
* Heap: memory that is dynamically allocated during process run time.
* Data: contains global and static variables.
* Text: includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

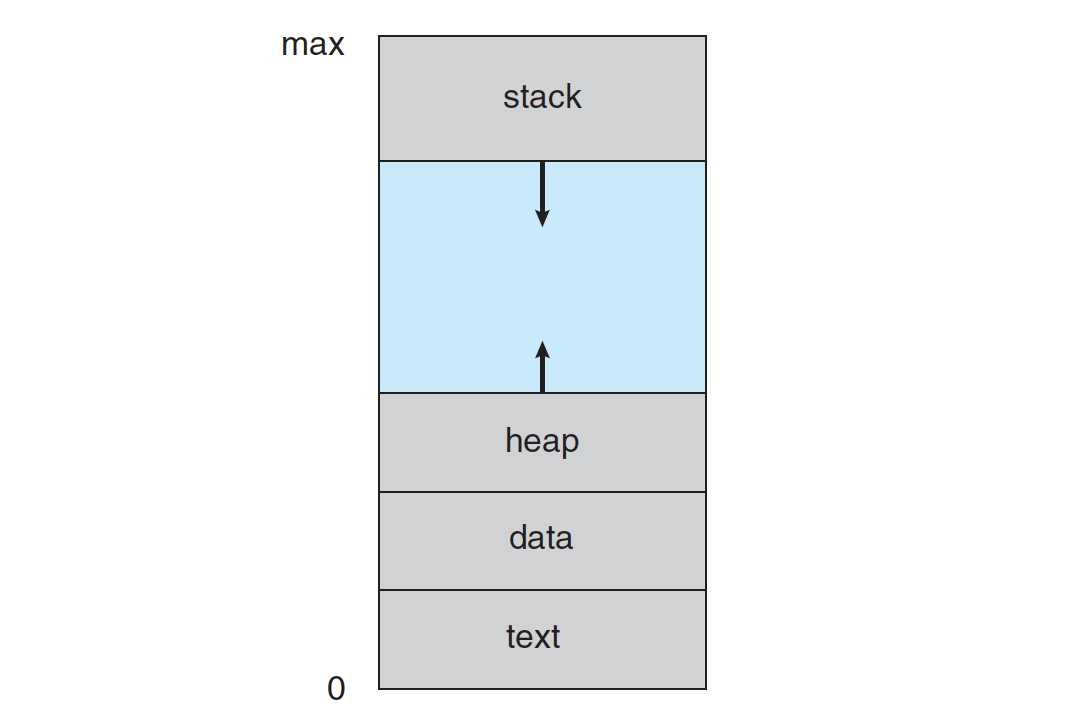


Figure 1. Process in memory

* A program is a ‘passive’ entity (executable file), process is a ‘active’ entity
* A program becomes a process when an executable file is loaded into memory

## Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

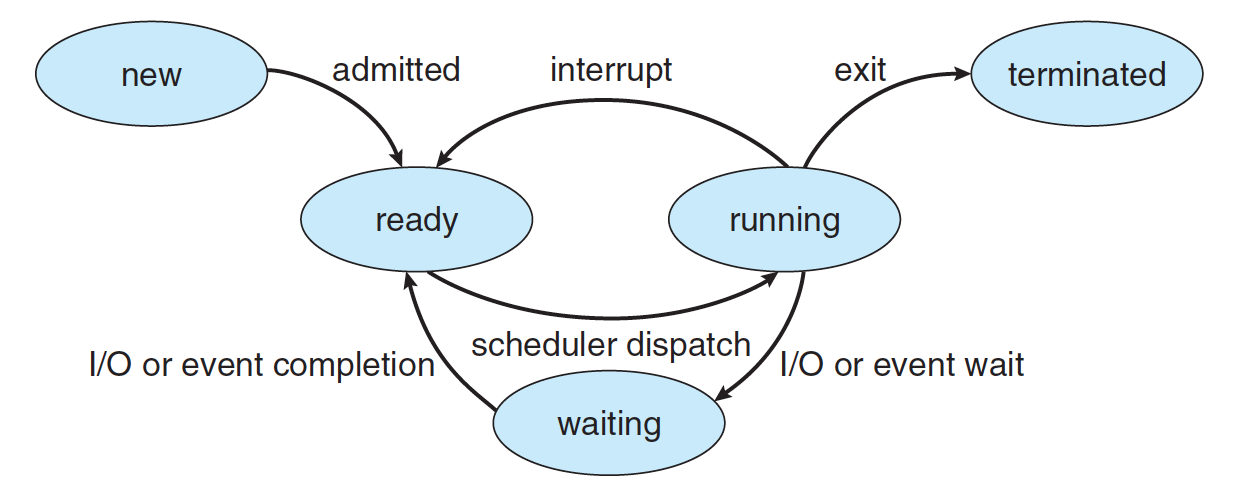


Figure 2. Diagram of process state

|  |  |
| --- | --- |
| **New** | The process is being created. |
| **Running** | Instructions are being executed. |
| **Waiting** | The process is waiting for some event to occur (such as an I/O completion or reception of signal). |
| **Ready** | The process is waiting to be assigned to a processor. |
| **Terminated** | The process has finished execution. |

## Process Control Block

Each process is represented in the operating system by a process control block (PCB).

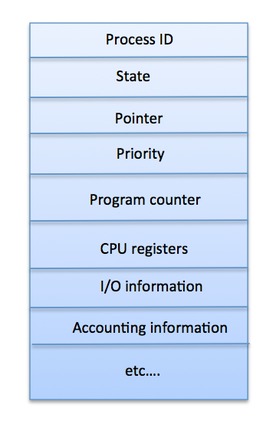


Figure 3. Process Control Block

|  |  |
| --- | --- |
| **Process State** | The current state of the process i.e., whether it is ready, running, waiting, or whatever |
| **Process privileges** | This is required to allow/disallow access to system resources. |
| **Process ID** | Unique identification for each of the process in the operating system. |
| **Pointer** | A pointer to parent process. |
| **Program Counter** | Program Counter is a pointer to the address of the next instruction to be executed for this process |
| **CPU registers** | Various CPU registers where process need to be stored for execution for running state. |
| **CPU Scheduling Information** | Process priority and other scheduling information which is required to schedule the process. |
| **Memory management information** | This includes the information of page table, memory limits, Segment table depending on memory used by the operating system |
| **Accounting information** | This includes the amount of CPU used for process execution, time limits, execution ID etc. |
| **IO status information** | This includes a list of I/O devices allocated to the process. |

## Process Scheduling

### What is Process Scheduling?

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

### Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

<dịch: HĐH duy trì tất cả PCB trong Hàng đợi lập lịch quy trình. HĐH duy trì một hàng đợi riêng cho từng trạng thái quy trình và PCB của tất cả các tiến trình trong cùng trạng thái thực thi được đặt trong cùng một hàng đợi. Khi trạng thái của một tiến trình được thay đổi, PCB của nó sẽ bị hủy liên kết khỏi hàng đợi hiện tại và được chuyển sang hàng đợi trạng thái mới>

* **Job Queue** – This queue keeps all the processes in the system.
* **Ready queue** – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
* **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.



Figure 4. Process Schedule Queue

The OS can use different policies to manage each queue:

* FIFO
* Round Robin
* Priority
* Etc.

### Context Swich

Context switch is the mechanism to store and restore the state or context of a CPU in Process Control Block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

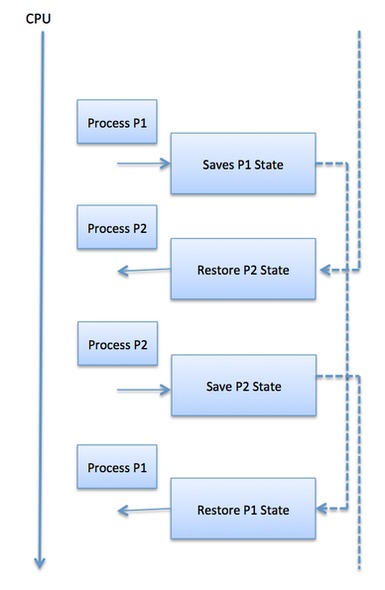


Figure 5. Illustration of Context Switch

Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use:

* Program Counter
* Scheduling information
* Base and limit register value
* Currently used register
* Changed State
* I/O State information
* Accounting information

### OS Scheduling Algorithms

* First-Come, First-Serve
* Shortest Job First
* Priority
* Round Robin

# Thread

## What is Thread?

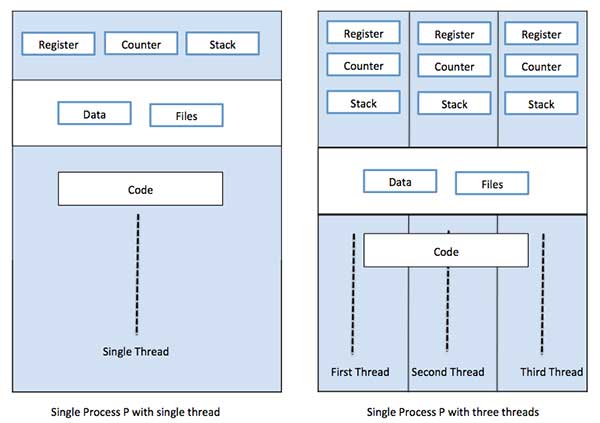


Figure 6. Single-thread vs Multi-thread process

References:

<https://medium.com/technofunnel/understanding-golang-and-goroutines-72ac3c9a014d>

<https://codeburst.io/why-goroutines-are-not-lightweight-threads-7c460c1f155f>

<https://medium.com/swlh/race-conditions-locks-semaphores-and-deadlocks-a4f783876529>

<https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

Code golang example for race condition

<https://medium.com/learning-the-go-programming-language/encoding-data-with-the-go-binary-package-42c7c0eb3e73> : BTL