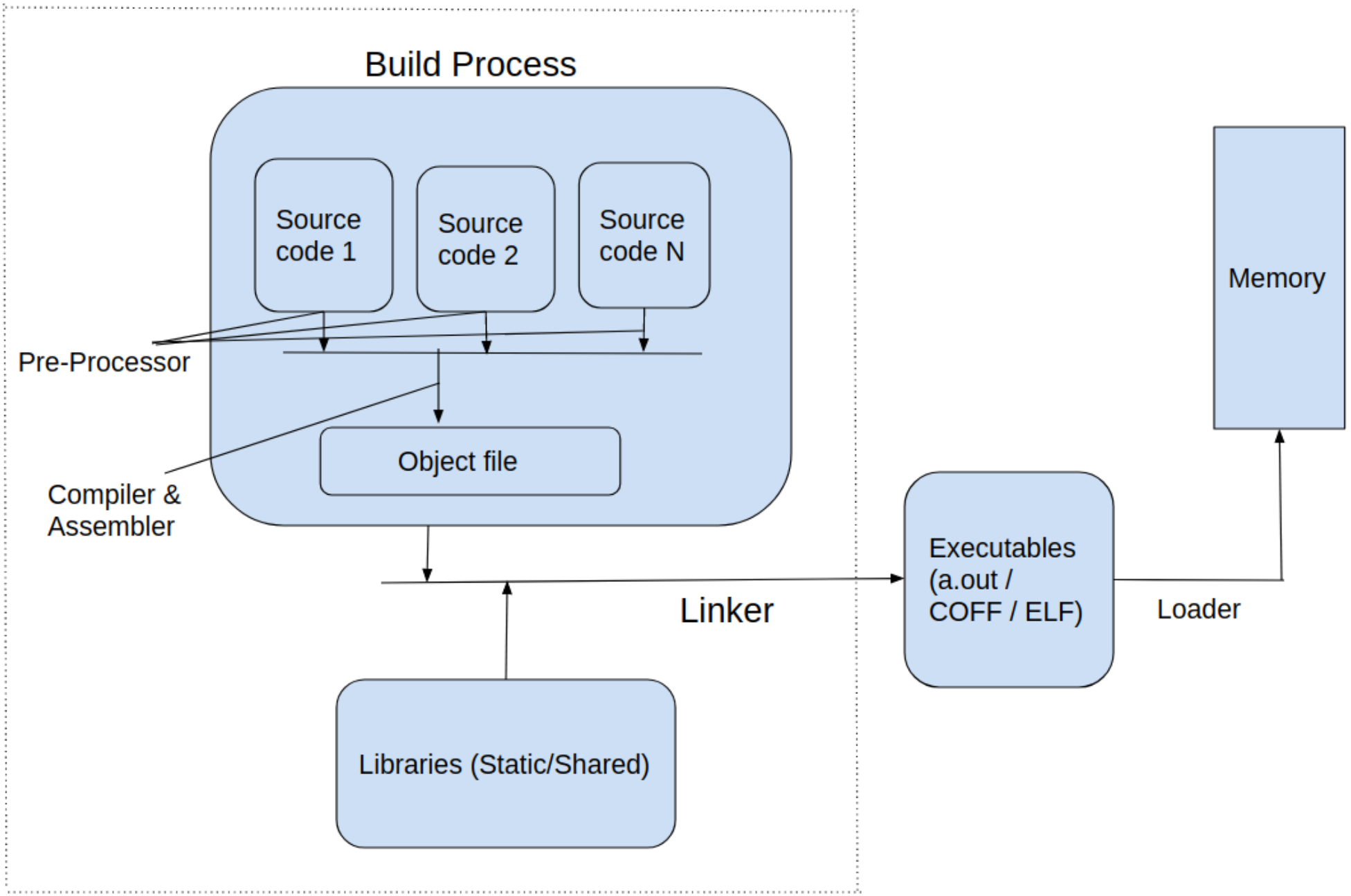
PROCESS

- Processes and Programs
- Process ID and Parent Process ID
- Memory Layout of a Process.
- Examples of different memory section of Process using C Code.

Program and Process



Processes and Programs

- A process is an instance of an executing program.
- A program is a file containing a range of information that describes how to construct a process at run time.

Child and Parent Process

- In Linux Processes have a Parent-child relation.
- A Process is created by its parent
- In Linux startup sequence

  Process ‘0’ is first process – called ‘swapper’ process

  Process ‘1’ is ‘Init’ Process – Init process plays an important role. It creates and monitors set of other process.
- Init process becomes the parent of any ‘Orphan’ Process.
- Man 8 init – for manual page on ‘init’ process.

Process ID

#include <unistd.h>

- pid_t getpid(void);

  Always successfully returns process ID of caller
- The Linux kernel limits process IDs to being less than or equal to 32,767(default on 32 bit).
- /proc/sys/kernel/pid_max – contains 1 greter than actual max number of process allowed in system.
- ‘Ps -ef’ – get running process and PID

Parent Process ID

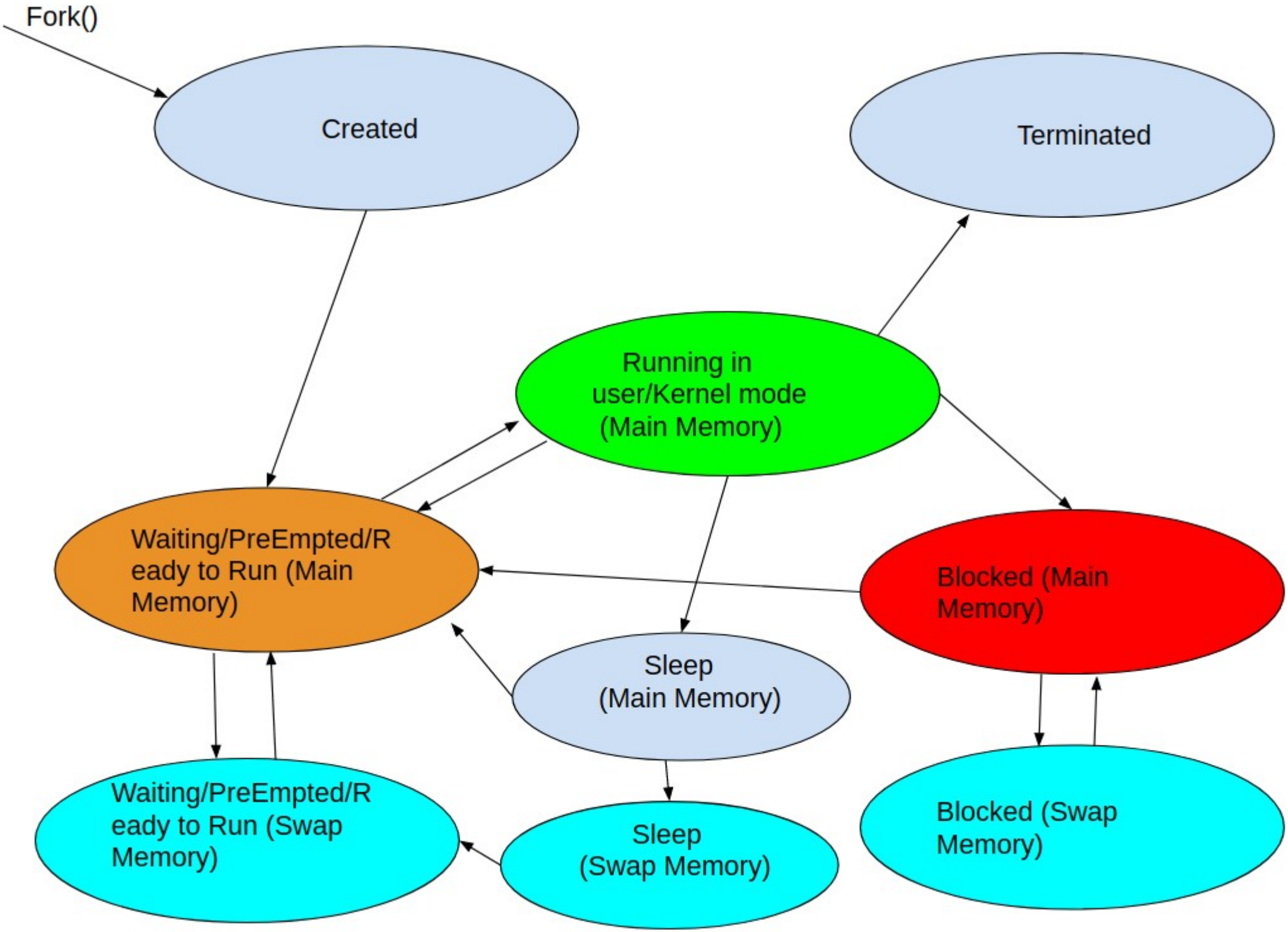
#include <unistd.h>

pid_t getppid(void);

Always successfully returns process ID of parent of caller process.

Process States

- A process is “created” state using fork() system call.
- ‘Running’ state – Process is running in main memory.
- ‘Ready to Run’ in ‘Main Memory’
- ‘Ready to Run’ in ‘Swap Memory’
- ‘Sleep’ state in ‘Main Memory’
- ‘Sleep’ state in ‘Swap memory’
- ‘Blocked’ state in ‘Main Memory’
- ‘Blocked’ state in ‘Swap Memory’
- ‘Terminated’ state


Fork()
Created
Terminated
Running in user/Kernel mode (Main Memory)
Waiting/PreEmpted/Ready to Run (Main Memory)
Blocked (Main Memory)
Sleep (Main Memory)
Waiting/PreEmpted/Ready to Run (Swap Memory)
Sleep (Swap Memory)
Blocked (Swap Memory)

Memory Layout of a Process

- The different segments of a process are..

  1. Text segment – The code resides here

  2. Data Segment – for data variables during compile time.

     A. Initialised data segment

     B. Un-Initialised data segment (BSS)

  3. Stack segment – for local variables

  4. Heap segment – for dynamic memory data’s.

Text Segment of a Process

- The ‘text segment’ contains the code of the program that is ran. This segment of memory cannot be written, so that the code is not altered by any pointers. This means ‘Text segment’ is Read only.

‘Initialized data segment’ of a Process

- The ‘initialized data segment’ contains global and static variables that are ‘explicitly initialized’ with value in the code.

Memory Layout of a Process

- The uninitialized data segment contains global and static variables that are not explicitly initialized in the code. The system initializes all value’s of this segment to ‘0’.

‘Stack’ segment of a Process

- The stack is a dynamically growing and shrinking segment containing local data’s(stack frames) of functions. One stack frame is allocated for each function that are called. A frame stores the function’s local variables, function arguments.

‘Heap’ segment of a Process

- The heap is an area from which memory (for variables) can be dynamically allocated at run time.

Virtual Memory Management

- Similar to other modern kernels, Linux has a technique known as virtual memory management. The aim of this technique is to make efficient use of both the CPU and RAM (physical memory/Main Memory)
- Each Process has its own memory space(4GB) on a 32 bit system.
- Each process has a private user space memory. This means one process cannot access memory of other process directly.
- Each process has below seperate Memory segments(the size of each memory segemnt is configurable)

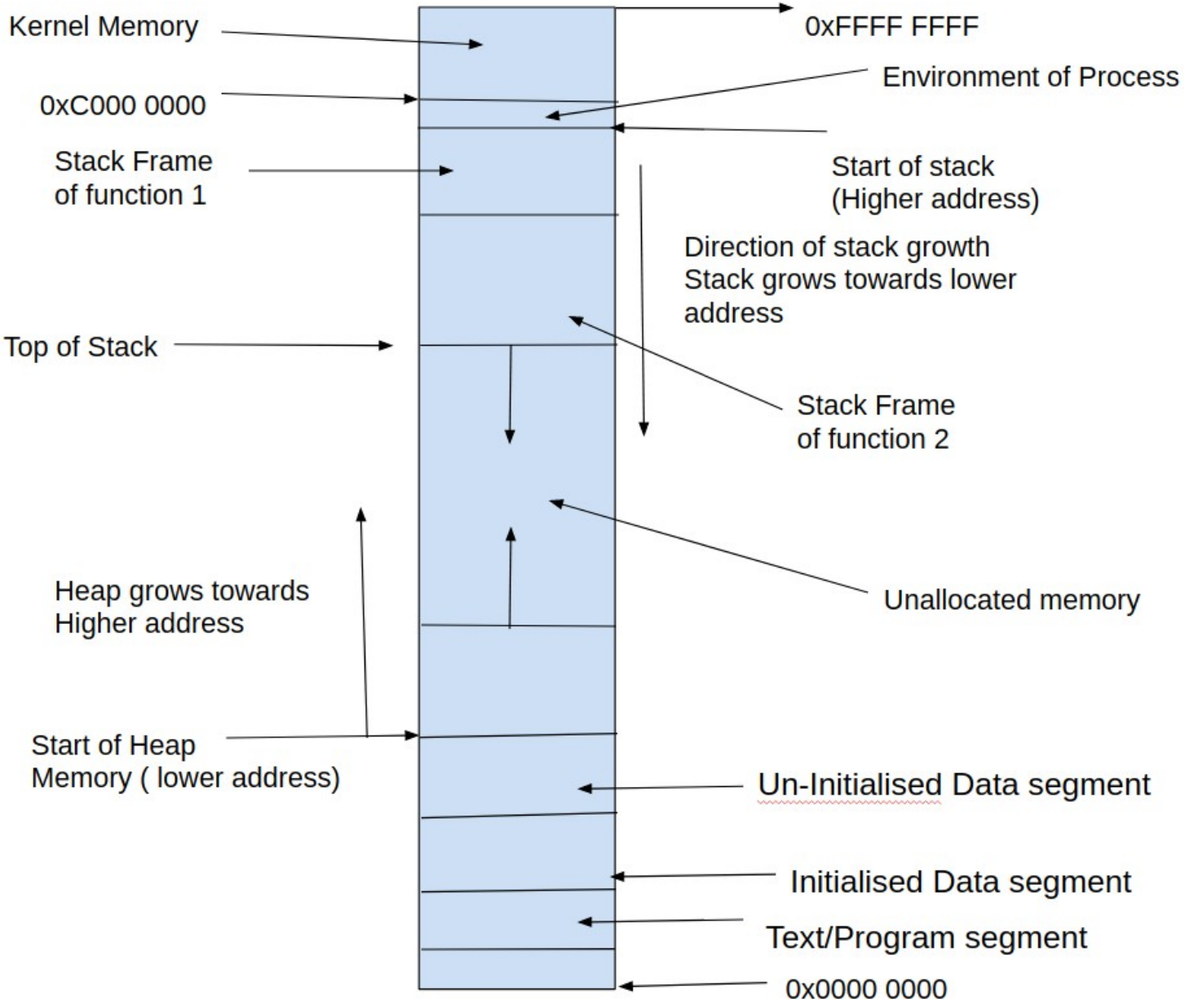
  a. Text segment

  b. Data segment(Initialised and uninitialised)

  c. Stack segment

  d. Heap segment
- The Process virtual memory has a user space, and a Kernel space. This memory region is configurable, but it is usually 3GB user space and 1GB Kernel space.

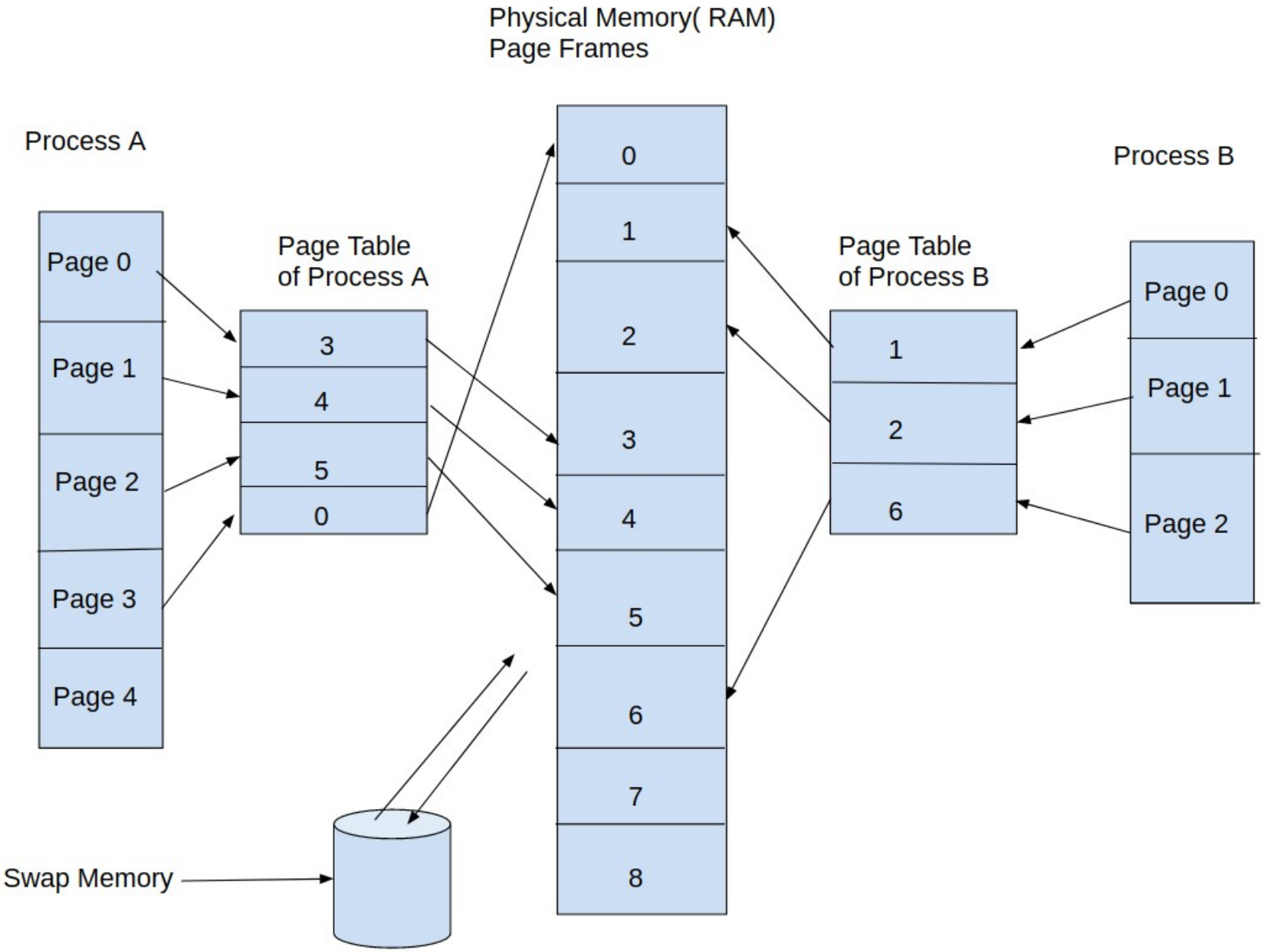
Virtual Memory of process



Page Frame

- A virtual memory of each process, is split into small, fixed size units called ‘pages’ (default page size usually is 4096 Bytes, but configurable to different size).
- Similarly, RAM(Physical memory) is divided into a series of page frames of the same size.
- At any given point of time, only some of the ‘pages’ of a process are present in physical memory (RAM), these pages are called ‘Resident Set’.
- Copies of the unused pages(whose entry is not present in page table) of a process are maintained in the swap area (usually disk space) and loaded into physical memory, when required.
- Frame size varies – generally 4K, 8K or 16K and is configurable.

Page Table


Physical Memory(RAM)
Page Frames
Process A
Page 0
Page 1
Page 2
Page 3
Page 4
Page Table
of Process A
3
4
5
0
0
1
2
3
4
5
6
7
8
Page Table
of Process B
1
2
6
Process B
Page 0
Page 1
Page 2
Swap Memory

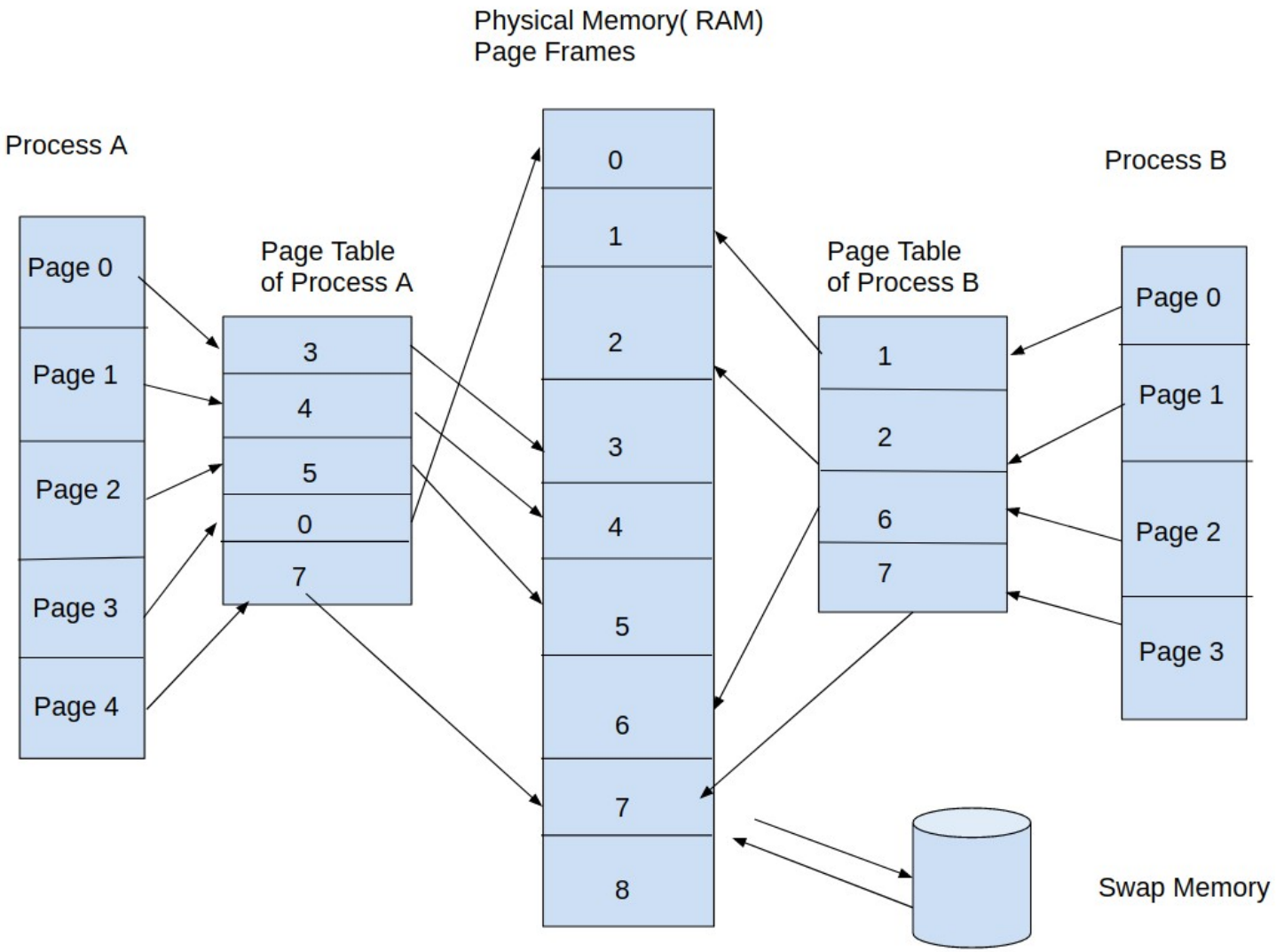
Page Table

- Two or more processes can share memory. This is possible by having page-table entries of different processes refer to the same pages of RAM. Memory sharing occurs below circumstances:

  a. Multiple processes executing the same program can share a single (read-only) copy of the program code.

  b. Processes can use the shared memory with other processes to exchange data’s.

Memory Sharing between Process

- 

Page Fault

- When a process tries to access a page that is not currently present in physical memory(and page table), a page fault occurs, at this point the kernel suspends execution of the process while the page is loaded from swap memory into Main memory(RAM).

Environment List

- Each process set of strings called the environment list, or simply the environment. Each of these strings is a definition of the form ‘name=value’.
- Thus, the environment represents a set of ‘name-value’ pairs that can be used to hold arbitrary information. The names in the list are referred to as environment variables.