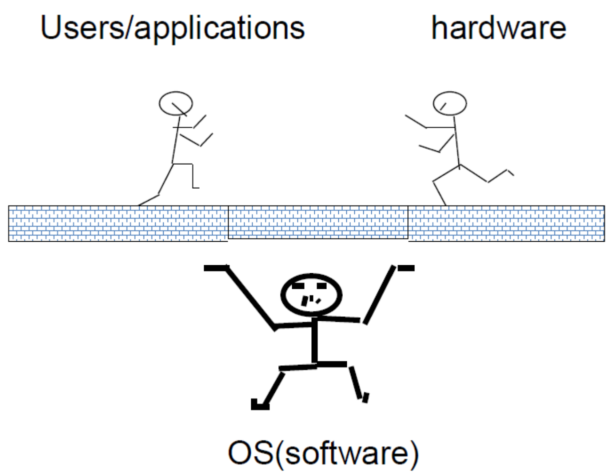
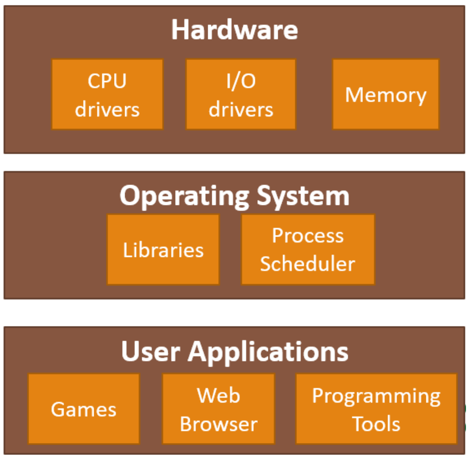
Background on Operating Systems

Operating Systems Bridge the Gap





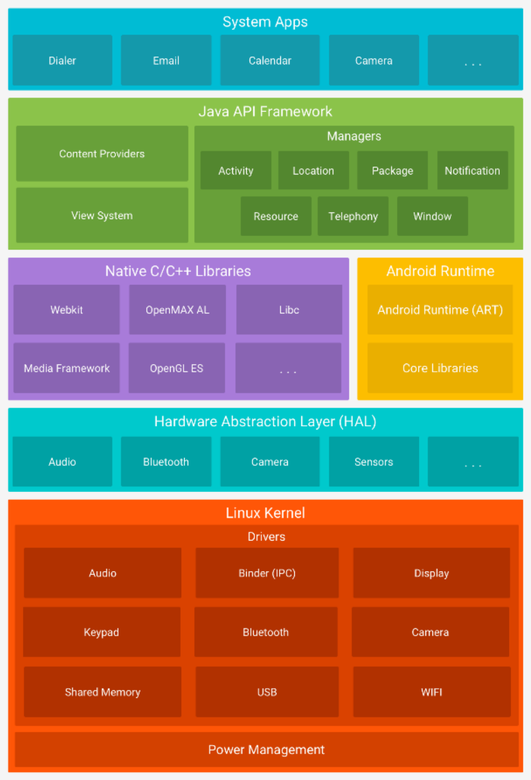
A computer system can be summarized into four components:

- Hardware components – physical components and drivers.
- OS – coordinates use of hardware with user applications
- Applications – typical applications used by a computer user

The primary tasks of an OS:

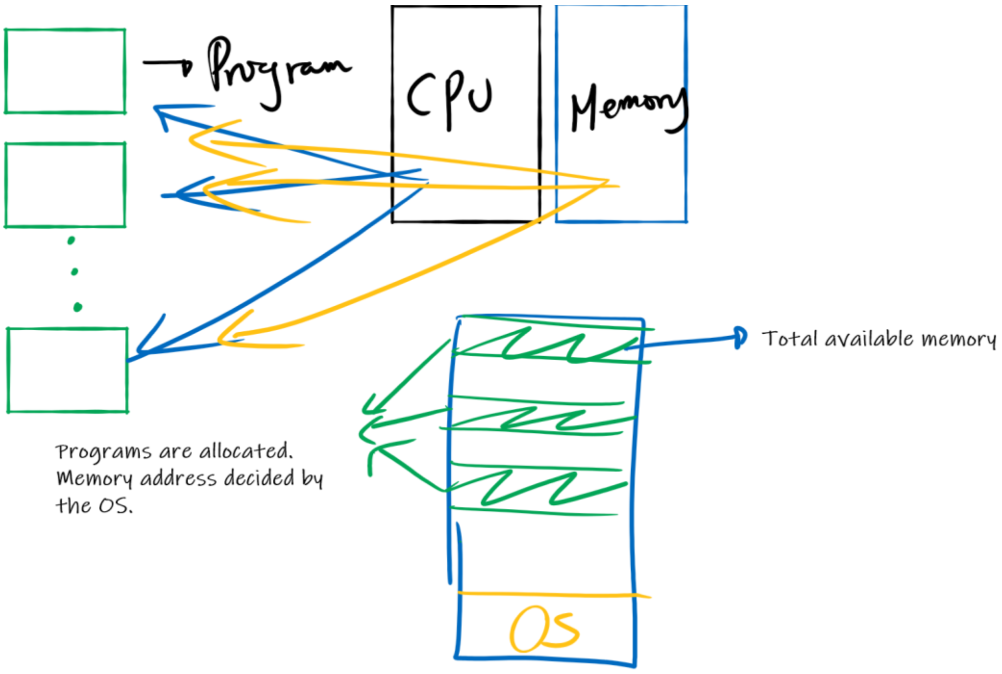
- Resource allocator – allocates resources such as CPU and memory to an application. Decides how to share resources among other user programs.
- Control program – controls behavior of user programs to avoid improper use of computer resources (CPU and memory hoggers, programs with infinite loops, etc).
- Security - Must protect application's data from one another (unless designed to share the data). Allocate resources fairly and efficiently. Settles conflicting requests for resources. Prevent errors and improper use of the hardware
- Improvisation – find a way to improvise when hardware resources are scarce. Provides an illusion of dedicated machine with infinite memory and processors.

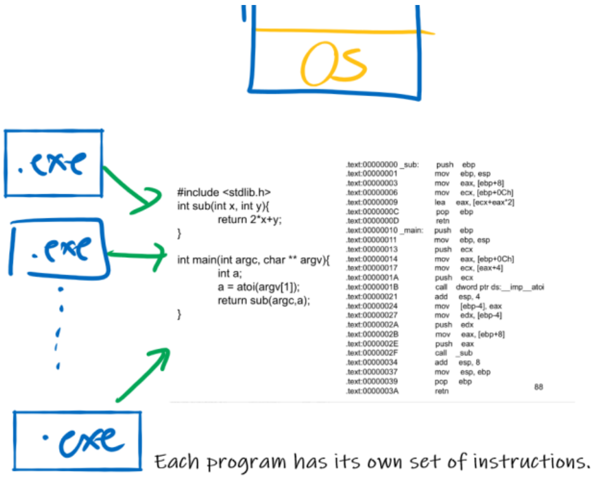
Architecture of android



An example of primary tasks in action:

Resource allocator



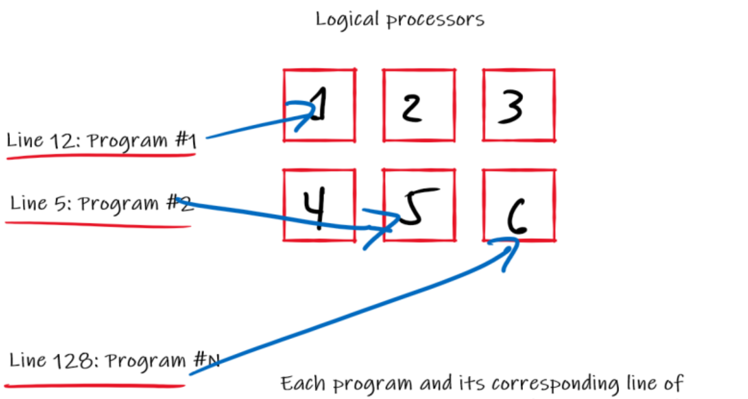


```
#include <stdlib.h>
int sub(int x, int y){
        return 2*x+y;
}

int main(int argc, char ** argv){
        int a;
        a = atoi(argv[1]);
        return sub(argc,a);
}
```

```
.text:00000000 _sub:     push  ebp
.text:00000001           mov   ebp, esp
.text:00000003           mov   eax, [ebp+8]
.text:00000006           mov   ecx, [ebp+0Ch]
.text:00000009           lea   eax, [ecx+eax*2]
.text:0000000C           pop   ebp
.text:0000000D           retn
.text:00000010 _main:    push  ebp
.text:00000011           mov   ebp, esp
.text:00000013           push  ecx
.text:00000014           mov   eax, [ebp+0Ch]
.text:00000017           mov   ecx, [eax+4]
.text:0000001A           push  ecx
.text:0000001B           call  dword ptr ds:__imp__atoi
.text:00000021           add   esp, 4
.text:00000024           mov   [ebp-4], eax
.text:00000027           mov   edx, [ebp-4]
.text:0000002A           push  edx
.text:0000002B           mov   eax, [ebp+8]
.text:0000002E           push  eax
.text:0000002F           call  _sub
.text:00000034           add   esp, 8
.text:00000037           mov   esp, ebp
.text:00000039           pop   ebp                     88
.text:0000003A           retn
```

Each program has its own set of instructions.



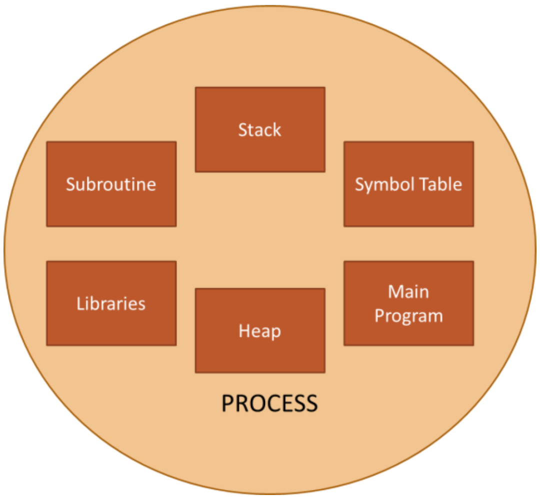
Each program and its corresponding line of instruction can be assigned to be executed on a logical processor. This heavily depends on the OS scheduling mechanism and CPU hardware design.

Control program

The OS ensures that all applications receive a fair number of resources. Each program has components where allocations are already pre-defined.

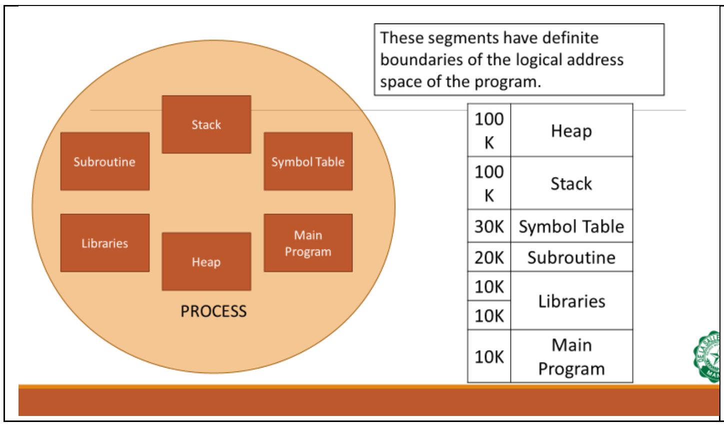
Components of a Program

- Main Program – points to the main function
- Subroutine – contains all functions declared in code
- Libraries – external functions or dependencies required
- Symbol table – holds variables
- Stack – holds temporary variables for function calls. Also stores the return address of a given function call.
- Heap – memory for dynamically allocated objects



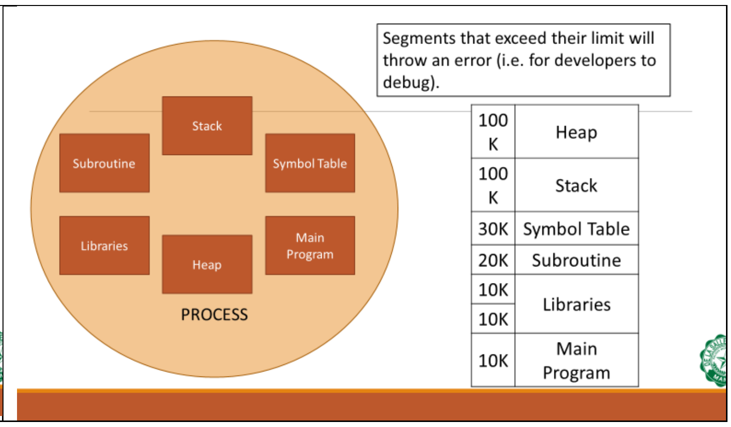
ACTIVITY:

1. Provide a scenario that causes stack to overflow.
2. Provide a scenario that causes the heap to overflow.



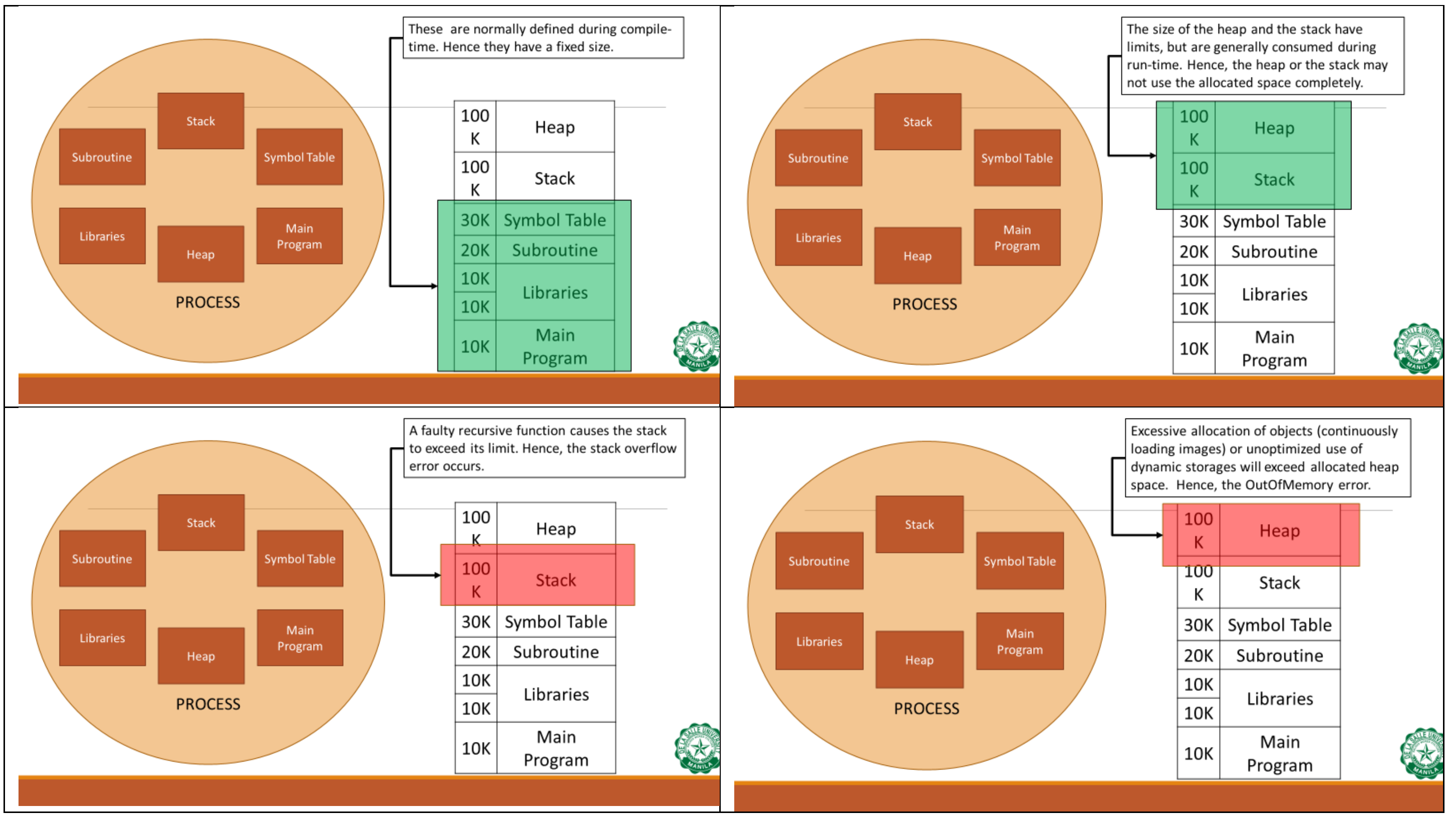
These segments have definite boundaries of the logical address space of the program.

100 K	Heap
100 K	Stack
30K	Symbol Table
20K	Subroutine
10K 10K	Libraries
10K	Main Program



Segments that exceed their limit will throw an error (i.e. for developers to debug).

100 K	Heap
100 K	Stack
30K	Symbol Table
20K	Subroutine
10K 10K	Libraries
10K	Main Program


These are normally defined during compile-time. Hence they have a fixed size.

The size of the heap and the stack have limits, but are generally consumed during run-time. Hence, the heap or the stack may not use the allocated space completely.

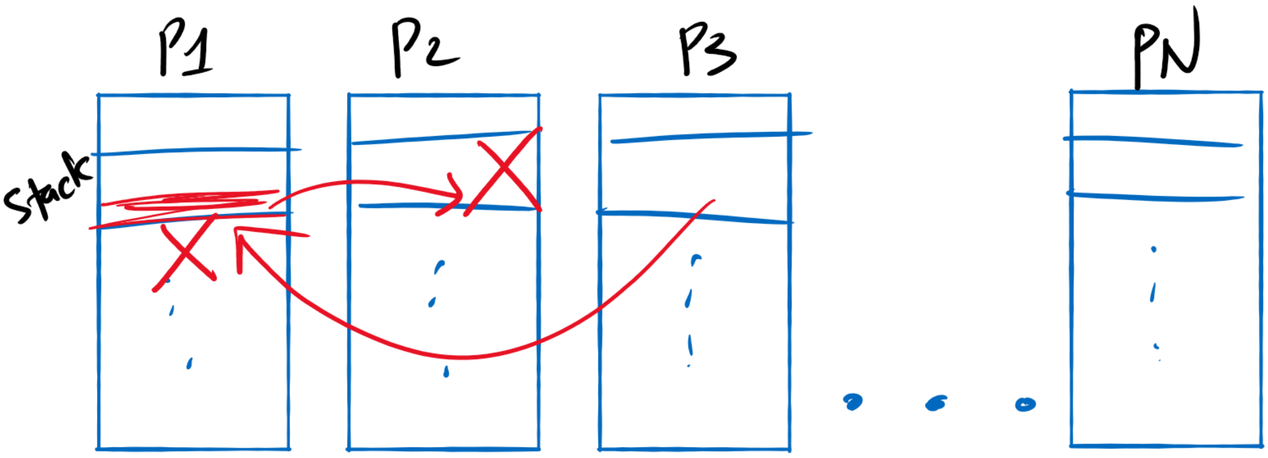
A faulty recursive function causes the stack to exceed its limit. Hence, the stack overflow error occurs.

Excessive allocation of objects (continuously loading images) or unoptimized use of dynamic storages will exceed allocated heap space. Hence, the OutOfMemory error.

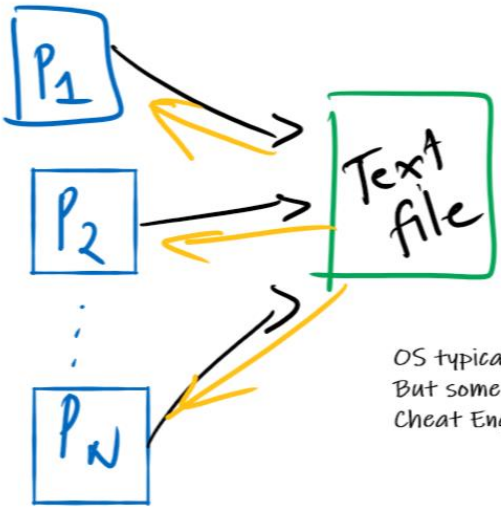
100 K	Heap
100 K	Stack
30K	Symbol Table
20K	Subroutine
10K 10K	Libraries
10K	Main Program

The OS pre-allocates memory on each application. Each application has a certain boundary on their components. Overstepping these boundaries throw errors.

Security

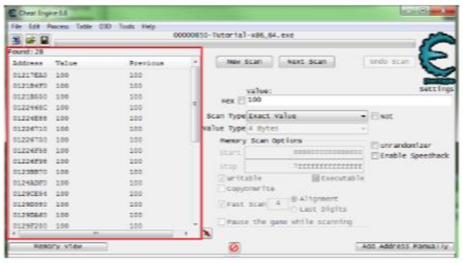


Each process has its own allocation and cannot be accessed by any other process.



If processes need to share information, each process must have instructions on writing/ reading a shared file.

OS typically deny any malicious memory accesses. But some tools exist to circumvent this (e.g. Cheat Engine).



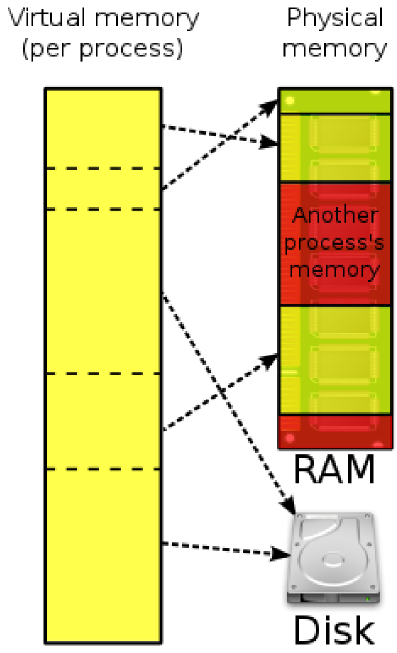
Found: 28

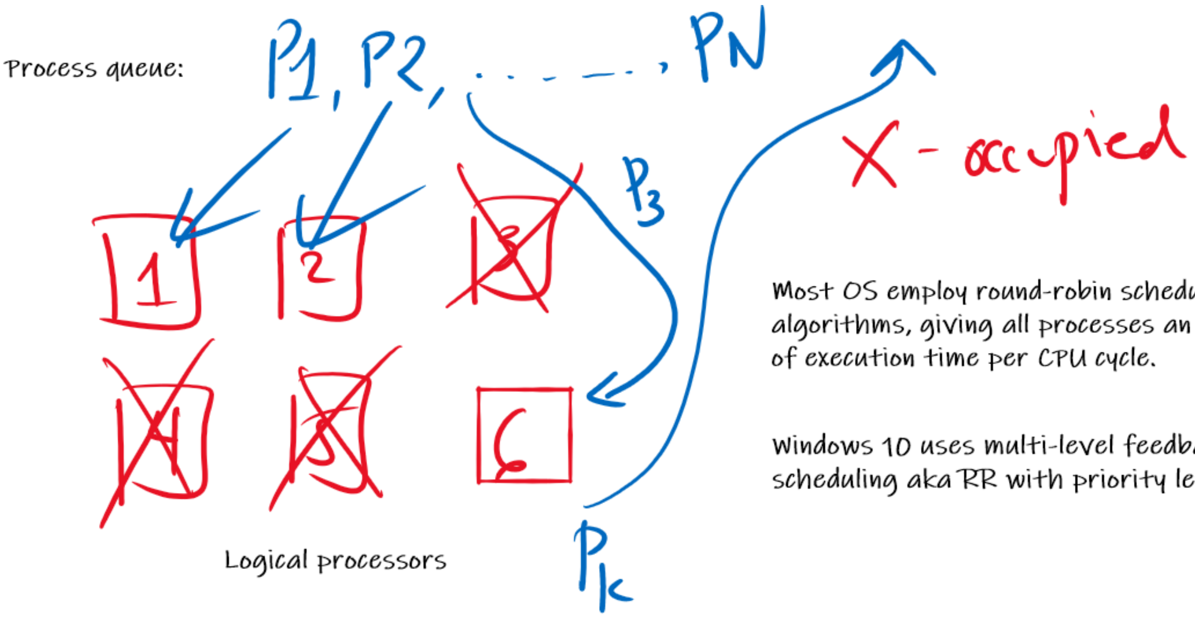
Address	Value	Previous
01217EA0	100	100
012184F0	100	100
01218030	100	100
0122468C	100	100
01224E88	100	100
01226710	100	100
01226790	100	100
01226F58	100	100
01226F98	100	100
012388B0	100	100
0124ADE0	100	100
0129CE84	100	100
01290D50	100	100
012908A0	100	100
0129F230	100	100

Value:
Hex □ 100
Scan Type Exact Value □ Not
Value Type 4 Bytes
Memory Scan Options
Start 0000000000000000
Stop 7fffffffffffffff
☑ Writable ☒ Executable
□ CopyOnWrite
☑ Fast Scan 4 ● Alignment ○ Last Digits
□ Pause the game while scanning
□ Unrandomizer
□ Enable Speedhack

Improvisation

In memory management, all OS perform a technique called **virtual memory**, where the OS uses secondary storages (such as HDD/SDD) as main memory whenever RAM is used up. This gives the illusion of a very large main memory.





Most OS employ round-robin scheduling algorithms, giving all processes an equal amount of execution time per CPU cycle.

Windows 10 uses multi-level feedback queue scheduling aka RR with priority levels.