

PROCESS

Program

Program contains information that describes how to construct a process, including:

- Binary format identification:
 - Metainformation describing the format of the executable file.
 - Enables the kernel to interpret the remain information in file.
 - UNIX format: executable format & linking format
- Machine-language instructions:
 - These encode the algorithm of the program.
- Program entry-point address:
 - The location of the instruction at which execution of the program start.
- Data
 - Values used to initialize variables, constants, ...
- Symbol and relocation tables:
 - Describe <location, name> of functions, variables within program.
- Shared library and dynamic linking information:
 - List the shared libraries needed to use at runtime
- Other information

Process Memory Layout

Process is an instance of an executing program.

Process memory contains many parts called segments, including:

- Text segment:
 - Machine-language instructions of the program
 - This is made read-only to avoid accidentally modify by its own instructions via bad pointer value.
- Initialized data segment :
 - Contain global and static variables that are explicitly initialized
 - Read from the executable file when program is loaded into memory
- Uninitialized data segment or block started by symbol (bss):
 - Contain global and static variables that are not explicitly initialized
- Stack:
 - Dynamically growing and shrinking segment containing stack frames.
 - Stack frame is allocated each time a function is called. A frame stores the function's local variables, arguments, return values.
- Heap:
 - Can be dynamically allocated at runtime.

Practise

```
#include <stdlib.h>

static int gs_n = 1;      //initialized data segment
static char gs_c;         //uninitialized data segment

int g_n = 1;              //initialized data segment
```

```

char c;                //uninitialized data segment

int sum(int a, int b) {    //allocated in frame for sum()
    int s;                //allocated in frame for sum()
    s = a + b;
    return s;              //return value passed via register
}

int main(int argc, char* argv[]) { //allocated in frame for main()
    static int s_number = 1;    //initialized data segment
    static int s_character;     //uninitialized data segment

    int count = 1;              //allocated in frame for main()
    char character;              //allocated in frame for main()

    int* p = malloc(sizeof(int)); //point to memroy in heap segment

    return 0;
}

```

Show size of program on terminal

```

invistd@server:~/share$ gcc ./hello.c -o hello -g
invistd@server:~/share$
invistd@server:~/share$
invistd@server:~/share$ size ./hello
   text    data    bss    dec    hex filename
   1576     612     20    2208    8a0 ./hello
invistd@server:~/share$

```

Output:

text: size of text segment
data: size of initialized data segment
bss: size of unitialized data segment
dec, hex: total size of program in decima, hexima

Virtual Address Spaces

Virtual Address

When a process reads or writes to a memory location, it uses a virtual address.

The virtual memory manager will translate the virtual address to a physical address.

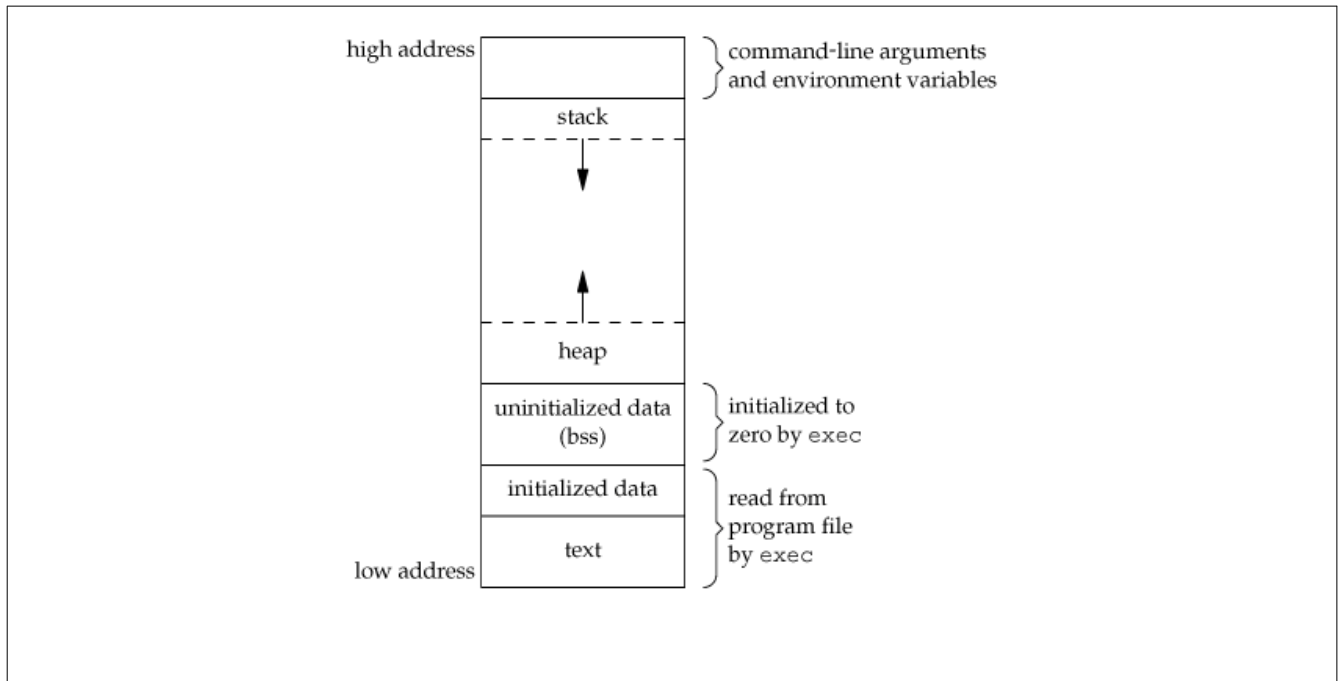
Advantages:

- Program can use a contiguous range of virtual addresses, although in physic memory they are not continugous.
- Virtual address of one process is isolated from other processes. Because, a physic memory address is only used by one process at a time.

Virtual Address Space

Virtual address space (VAS) is the set of ranges of virtual addresses that operating system makes available to a process.

VAS Structure



Practise

```
#include <stdlib.h>

static int global_static_i = 1;
int global_i = 1;

static int global_static_u;
int global_u;

int sum(int a, int b) {
    int s = 0;
    s = a + b;
    return s;
}

int main(int argc, char* argv[]) {
    static int local_static_i = 0;
    static int local_static_u;

    int local_i = 0;
    int local_u;

    sum(1, 2);
    sum(3, 4);

    int* p = malloc(sizeof(int));
```

```

    return 0;
}

```

Virtual address space of the program

<i>argc, argv, environ</i>		
stack <i>(non-static local variable)</i>	stack frame for main()	local_i local_u
	stack frame for sum(1, 2)	int a int b int s return value
	stack frame for sum(3, 4)	int a int b int s return value
unallocated memory		
heap <i>(allocated by malloc, calloc)</i> <i>(allocated by new)</i>	p = malloc(sizeof(int)	
uninitialized data (bss) <i>(global static variable, uninitialized)</i> <i>(local static variable, uninitialized)</i> <i>(global variable)</i>	global_static_u local_static_u global_u	
initialized data <i>(global static variable, initalized)</i> <i>(local static variable, initialized)</i> <i>(global variable, initialized)</i>	global_static_i global_i local_static_i	
text	program code	