PROCESS

Process & Program

Process is an instance of an executing program.

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

- Binary format identification:
    - Metainformation describing the format of the executable file.
    - Enables the kernal 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 memory contains many parts called segments, involves:

- 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, heximal

Virtual Address Spaces

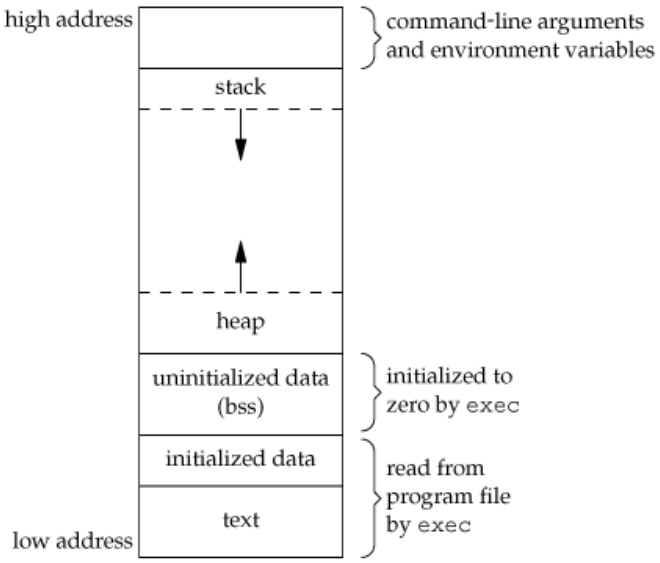
Virtual Address

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 contigous 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 avaiable 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

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