THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 3
x86 Registers
C-Programs, memory layout, and memory protection mechanism

Week 3 Assignments

- **Learning Materials**

  1- Week 3 Presentation

  2- Read Pages 45-65 & 77-88: Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux,

- **Assignment**

  1- Complete Lab 4 by coming Sunday 11:59 PM.

Agenda

- **General Purpose Registers**
- **Special Registers**
- **FLAGS**
- **Loading and Executing a Program**
- **Understand memory Layout**
- **Understand C-program layout**
- **Program Process Concept**
- **Display memory layout in Linux OS**
- **Protecting memory from buffer overflow attacks**

What are the Registers?

- A register is a small amount of data storage available to the CPU, whose contents can be accessed more quickly than the RAM
- Most common x86 registers, which fall into four categories:

1- General registers

- Used by the CPU during execution
- Examples: EAX, EBX, ECX, EDX, EBP, ESP, ESI

2- Segment registers

- Used to track sections of memory
- Examples: CS, SS, DS, ES, FS, GS

3- Status flags

- Used to make decisions
- Example: EFLAGS

4- Instruction pointer

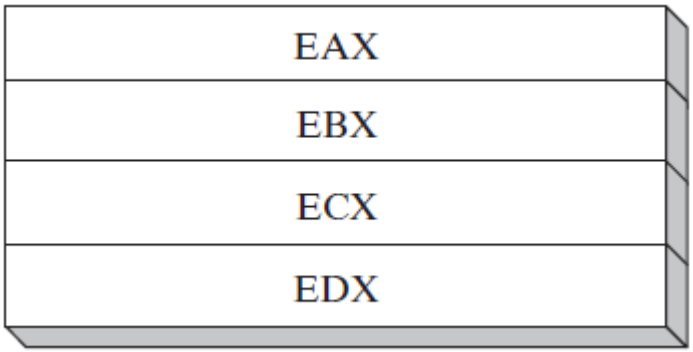
- Address of next instruction to execute
- Example: EIP

General and Specialized Registers Tasks

- General-Purpose Registers
  - EAX – accumulator (e.g. store or load a variable's value)
  - EAX is automatically used by multiplication and division instructions.
  - ECX – loop counter
  - EBX
  - ESP – Extend stack pointer: Points to the top of the function's stack (memory structure)
  - EBP -Extended frame pointer (stack)
  - ESI- Extended Source Index Register
  - EDI- Extended Destination Index Register
- EIP – instruction pointer
- EFLAGS
  - status and control flags
  - each flag is a single binary bit
- Segment Registers
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments

General Registers

- EAX Extended Accumulator Register (32 bits)
  (EAX, AX, AH, AL)
- EBX Extended Base Register (32 bits)
  (EBX, BX, BH, BL)
- ECX: Extended Counter Register (32 bits)
  (ECX, CX, CH,CL)
- EDX: Extended Data Register (32 bits)
  (EDX, DX, DH, DL)
- Can be used for several tasks



Accessing Parts of Registers

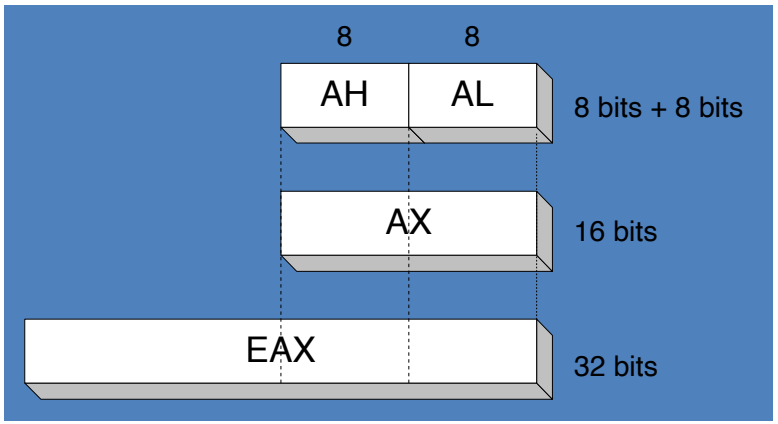
EAX Extend Accumulator Register (32 bits) (EAX, AX, AH, AL)

- Apply same concept to EAX, EBX, ECX, and EDX
- EAX, EBX, ECX, and EDX provide four 32-bits data registers, they can be used as:

  Four 32-bits registers (EAX, EBX, ECX, EDX)

  Four 16-bits registers (AX, BX, CX, DX)

  Eight 8-bits registers (AL, AH, BL, BH, CL, CH, DL, DH)

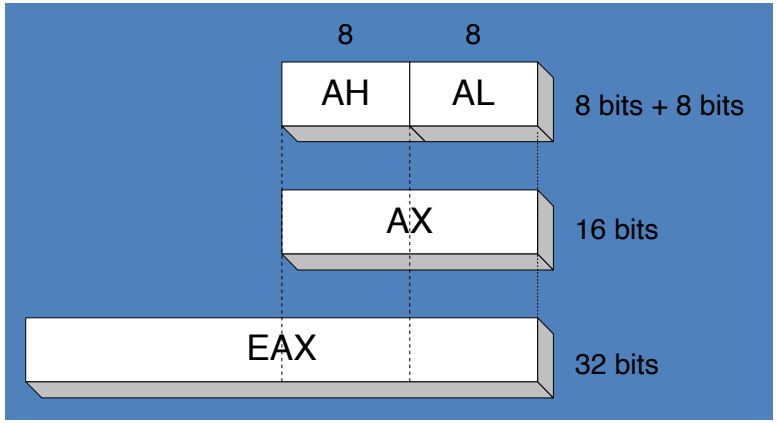


32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

Accessing Parts of Registers

MOV EAX, 0x12 34 56 78 ;
EAX = 0x12345678
AX = 0x5678,
AH = 0x56,
AL = 0x78

MOV AL,0x01 ;
AL = 0x01
AH = 0x56,
AX = 0x5601,
EAX = 0x12345601,



General Registers Tasks

- EAX Extend Accumulator Register
  (EAX, AX, AH, AL) (Called Accumulator Register)
  - Stores function return values
    If you see the EAX register just after a function call, EAX contains its the return value
  - Used for hardware access (e.g. screen and keyboard)
  - Used by multiplication and division instructions.

Examples:

```
mov eax, 1  // store 1 in "eax" register
mov ah,1    // store 1 in "ah" register
add eax, 2  // increase the value stored in eax by 2

mov eax, 4  // call system  write function (sys_write)
int 0x80    // call the kernel to execute the previous system call
            //Q1: Write what to what device?

mov eax, 1  //  Tel the OS to exit the program
            // call system exit function (sys_exit), int 0x80 has to come next

mov eax, 1  // store 1 in "eax" register
```

General Registers Tasks

- EBX Extended Base Register (32 bits)
  (EBX, BX, BH, BL)
  - Base pointer to the data section (.data // data section)
  - It is used as a base pointer for memory access
  - Gets some interrupt return values
  - Used as a general register

Examples:

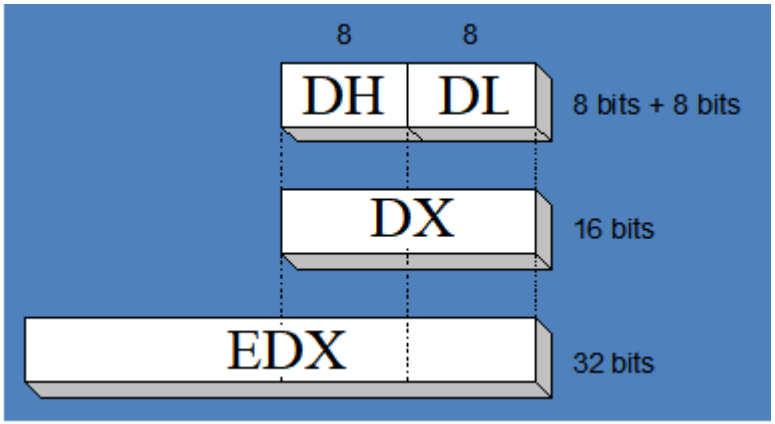
```
mov ebx, 1    // store 1 in ebx

mov ebx, 1    // Stdout (screen) file descriptor
              //The program will access the screen (output display device)
              //  int 0x80 has to become next

mov eax, [ebx]    ; Move the 4 bytes in memory at the address contained in EBX into EAX
```

General Registers Tasks

- EDX: Extend Data Register (32 bits)

  (EDX, DX, DH, DL)

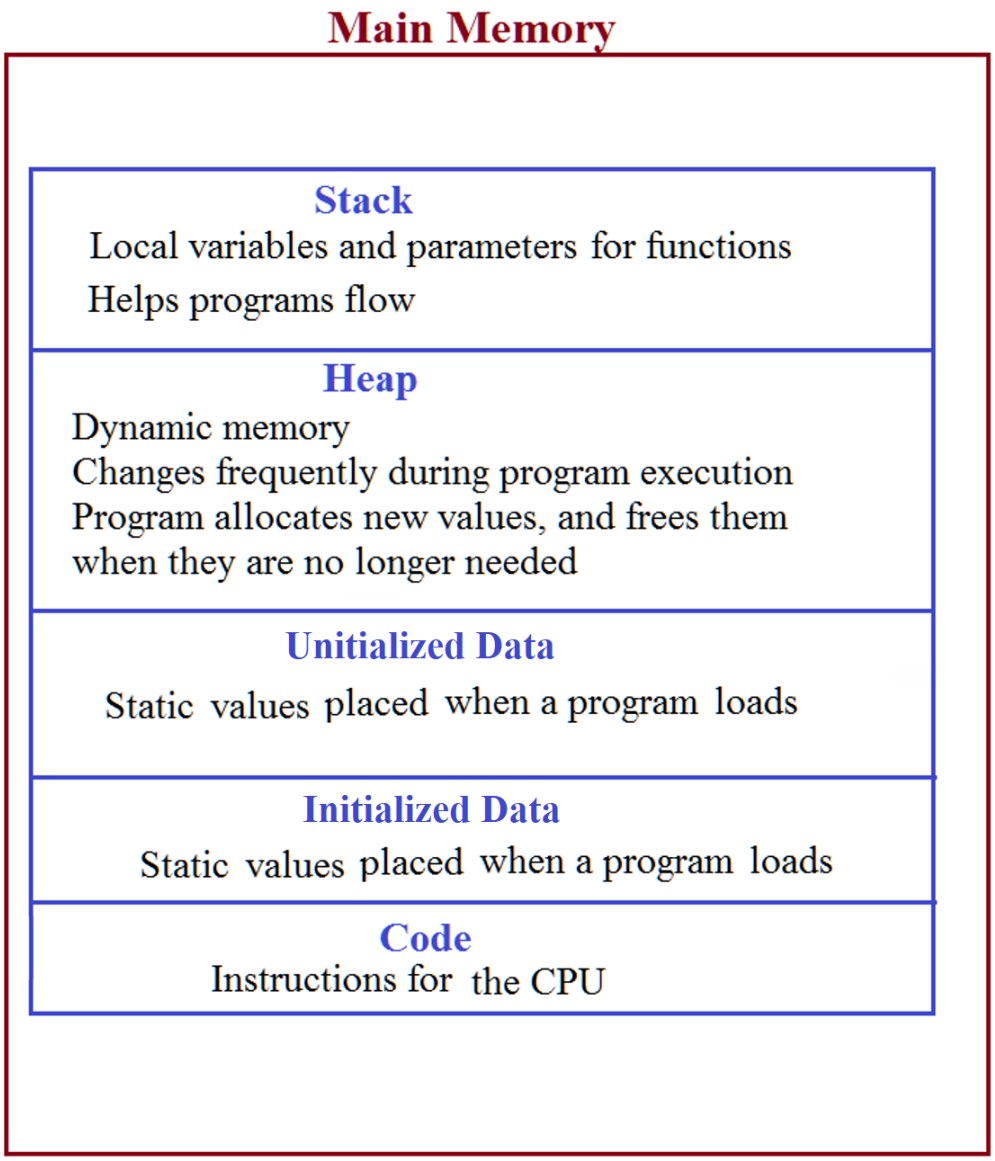
  - EDX is generally used for storing short-term variables within a function
  - I/O pointer



EIP (Extended Instruction Pointer)

- EIP is known as the *instruction pointer* or *program counter*
- Contains the memory address of the next instruction to be executed
- If EIP corrupted (contains wrong code), the CPU will fetch non-legitimate instructions and crash
- Attacks use Buffer overflow attack to gain control of EIP to point to a code to exploit a system

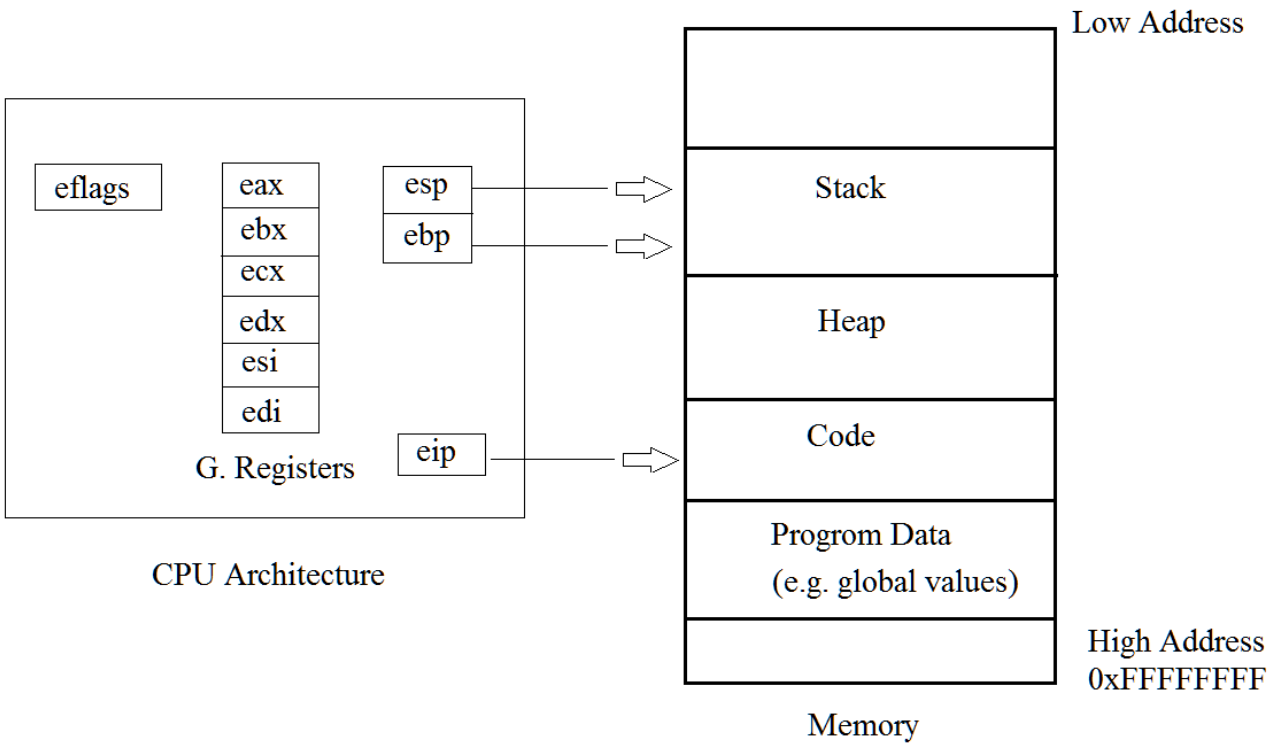
- ESP – Extend stack pointer (stack)
  Points to the top of the function's stack (memory structure)
  EBP – Extended Base (frame) pointer (stack)
  Used to reference to function's parameters and local variables on the stack.



Basic memory layout for a program

General Registers Tasks

- ESP – Extend stack pointer (stack)
- Points to the top of the function's stack (memory structure)
- EBP – Extended Base (frame) pointer (stack)
- Used to reference to function's parameters and local variables on the stack.
- EIP is known as the *instruction pointer* or *program counter*
- Contains the memory address of the next instruction to be executed



Status Flags

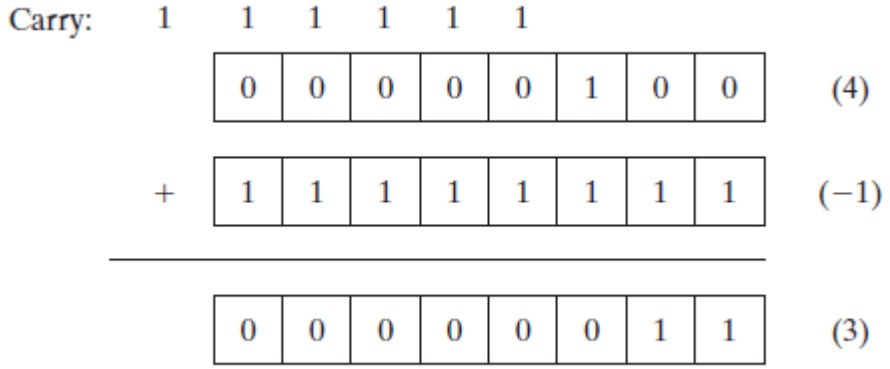
- Carry
  - unsigned arithmetic out of range
- Overflow
  - signed arithmetic out of range
- Sign
  - result is negative
- Zero
  - result is zero
- Auxiliary Carry
  - carry from bit 3 to bit 4
- Parity
  - sum of 1 bits is an even number

ZF and CF Flags Example

- In the sub instruction, the CPU can implement subtraction as a combination of negation and addition.
- Example: The expression 4 - 1 can be rewritten as

  4 + (-1)

  4 + NEG (1)



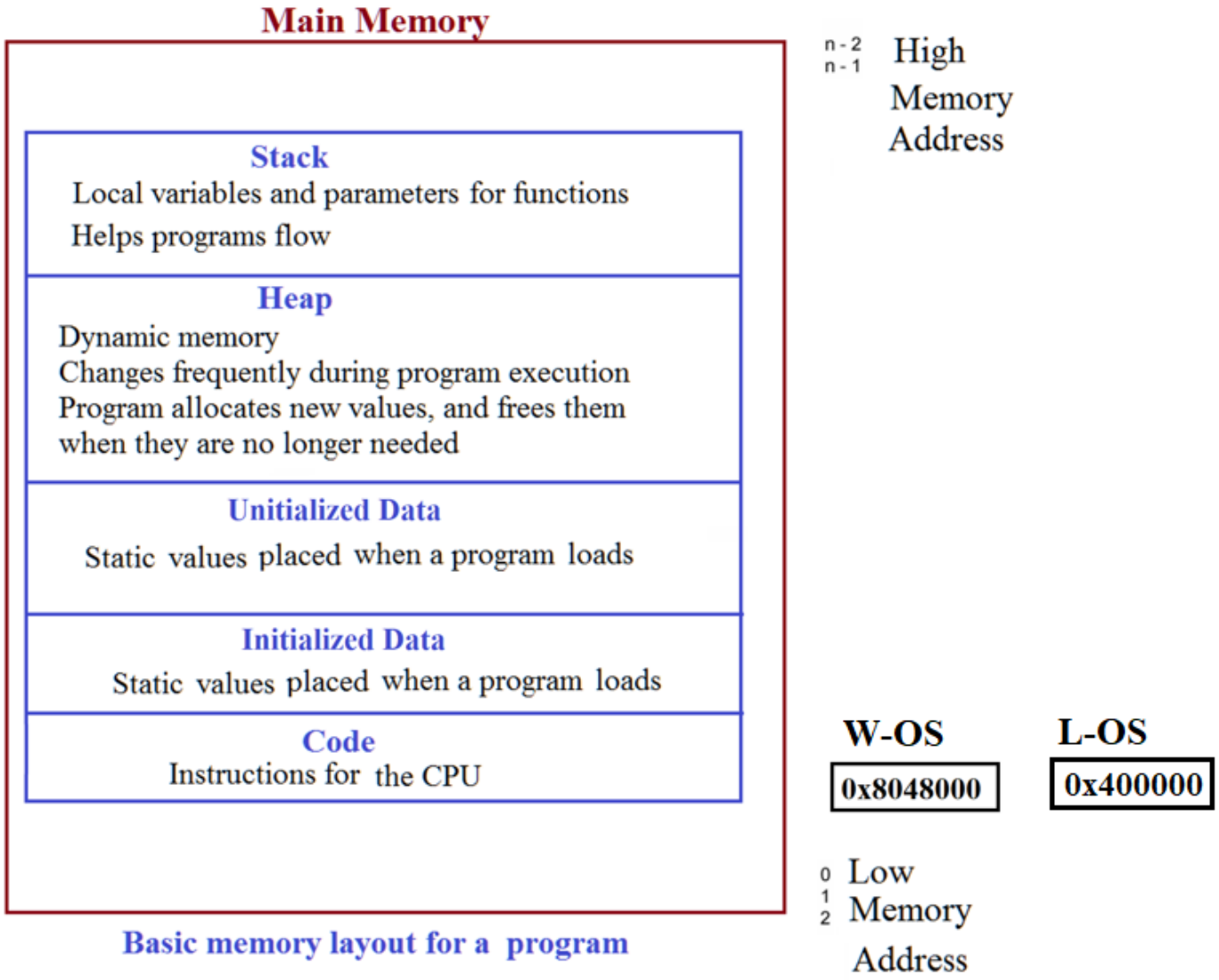
- The Carry, Zero, Sign, Overflow, Auxiliary Carry, and Parity flags are changed according to the value that is placed in the destination operand. (CF=0)
- CF: Set(=1) when a larger integer is subtracted from a smaller one

C-Programs, memory layout, and memory protection mechanism

Loading and Executing a Program

- As soon as the program begins running, it is called a *process*.
- The OS assigns the process an identification number (*process ID*), which is used to keep track of it while running.
- The *process* runs by itself. It is the OS's job to track the execution of the process and to respond to requests for system resources. Examples of resources are memory, disk files, and input-output devices.
- When the process ends, it is removed from memory.

Understand memory Layout



Basic memory layout for a program

Program Process Concept

- As soon as the program begins running, it is called a *process*.
- The OS assigns the process an identification number (*process ID*), which is used to keep track of it while running.
- The *process* runs by itself. It is the OS's job to track the execution of the process and to respond to requests for system resources. Examples of resources are memory, disk files, and input-output devices.
- When the process ends, it is removed from memory.
- How could you display the running processes ID?
- Where does Linux save information about a process?

Display running Processes

- You can use “ps -aux” command to display all running processes

```
root@kali-Test:~# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.2  0.3  29728  6960 ?        Ss   16:41   0:01 /sbin/init
root           2  0.0  0.0      0     0 ?        S    16:41   0:00 [kthreadd]
root           4  0.0  0.0      0     0 ?        S<   16:41   0:00 [kworker/0:0H]
root           5  0.0  0.0      0     0 ?        S    16:41   0:00 [kworker/u8:0]
root           6  0.0  0.0      0     0 ?        S<   16:41   0:00 [mm_percpu_wq]
root           7  0.0  0.0      0     0 ?        S    16:41   0:00 [ksoftirqd/0]
root           8  0.0  0.0      0     0 ?        S    16:41   0:00 [rcu_sched]
root           9  0.0  0.0      0     0 ?        S    16:41   0:00 [rcu_bh]
root          10  0.0  0.0      0     0 ?        S    16:41   0:00 [migration/0]
root          11  0.0  0.0      0     0 ?        S    16:41   0:00 [watchdog/0]
root          12  0.0  0.0      0     0 ?        S    16:41   0:00 [cpuhp/0]
root          13  0.0  0.0      0     0 ?        S    16:41   0:00 [cpuhp/1]
root          14  0.0  0.0      0     0 ?        S    16:41   0:00 [watchdog/1]
root          15  0.0  0.0      0     0 ?        S    16:41   0:00 [migration/1]
root          16  0.0  0.0      0     0 ?        S    16:41   0:00 [ksoftirqd/1]
root          18  0.0  0.0      0     0 ?        S<   16:41   0:00 [kworker/1:0H]
root          19  0.0  0.0      0     0 ?        S    16:41   0:00 [cpuhp/2]
root          20  0.0  0.0      0     0 ?        S    16:41   0:00 [watchdog/2]
root          21  0.0  0.0      0     0 ?        S    16:41   0:00 [migration/2]
root          22  0.0  0.0      0     0 ?        S    16:41   0:00 [ksoftirqd/2]
root          24  0.0  0.0      0     0 ?        S<   16:41   0:00 [kworker/2:0H]
root          25  0.0  0.0      0     0 ?        S    16:41   0:00 [cpuhp/3]
root          26  0.0  0.0      0     0 ?        S    16:41   0:00 [watchdog/3]
```

Display running Processes

- You can use “ps -aux” command to display all running processes

```
Debian-+   642  0.0  0.2  30436  5252 ?        Sl   16:41   0:00 /usr/lib/at-spi2-core/at-spi
Debian-+   656  1.0  6.3 868476 131128 tty1    Sl+  16:41   0:06 /usr/bin/gnome-shell
root       664  0.0  0.3  53432  7368 ?        Ssl  16:41   0:00 /usr/lib/upower/upowerd
Debian-+   682  0.0  0.4 888664  9816 ?        Ssl  16:41   0:00 /usr/bin/pulseaudio --daemon
rtkit      683  0.0  0.1  23988  3044 ?        SNsl 16:41   0:00 /usr/lib/rtkit/rtkit-daemon
root       694  0.9  0.7  69572 16536 ?        Ssl  16:41   0:05 /usr/lib/packagekit/packagek
Debian-+   695  0.0  0.8  54224 17812 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   696  0.1  0.9  73340 19096 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   698  0.1  0.8  63112 17668 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   699  0.0  0.3  38432  6604 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   700  0.0  0.8  53556 17592 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   703  0.1  0.9  92216 19748 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   704  0.0  0.5  84332 11048 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   707  0.0  0.8  62812 17488 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   714  0.0  0.8  72304 17860 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   723  0.0  1.0 339224 21172 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   724  0.0  0.2  26484  4780 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   725  0.0  0.9  73504 19892 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   726  0.0  0.9  61672 19880 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   741  0.0  0.2  26500  4784 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   742  0.0  0.2  35708  4592 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   743  0.0  0.4  45332  8936 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   749  0.0  0.4  51920  8488 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
Debian-+   750  0.0  0.4  45488  8456 tty1     Sl+  16:41   0:00 /usr/lib/gnome-settings-daem
colord     771  0.0  0.6  47920 13936 ?        Ssl  16:41   0:00 /usr/lib/colord/colord
```

Display running Processes

- You can use “ps -aux” with “grep” to display all running processes for certain program

```
root@kali-Test:~# ps -aux | grep rtkit
rtkit      683  0.0  0.1  23988  3044 ?        SNsl 16:41   0:00 /usr/lib/rtkit/rtkit-daemon
root      1659  0.0  0.0   4808   792 pts/0    S+   16:58   0:00 grep rtkit
root@kali-Test:~# ps -aux | grep colord
colord     771  0.0  0.6  47920 13936 ?        Ssl  16:41   0:00 /usr/lib/colord/colord
root      1661  0.0  0.0   4808   840 pts/0    S+   16:58   0:00 grep colord
root@kali-Test:~#
```

Where does Linux save information about a process?

- The "/proc" directory holds all the runtime information about all running processes and other information about the status of the kernel
- Each process holds a directory that contains information about that process

```
root@kali-Test:~# ls /proc
1     1066  1190  166   20   33   463  577  694  750  964          fb
10    1072  12    167   201  333  464  583  695  771  977          filesystems
1001  1074  1217  1678  204  35   47   584  696  8    981          fs
1004  1076  1237  168   21   356  471  595  698  811  990          interrupts
1014  1080  1264  169   22   36   472  597  699  827  993          iomem
1018  1083  1280  170   24   368  473  6    7    828  acpi         ioports
1023  1087  1282  1701  25   37   474  605  700  838  asound       irq
1027  1092  13    1702  254  38   48   608  703  839  buddyinfo    kallsyms
1031  1096  1301  1709  255  39   49   619  704  844  bus          kcore
1032  1098  1335  171   26   4    492  623  707  848  cgroups      key-users
1034  11    135   172   27   40   493  628  714  850  cmdline      keys
1039  1135  136   173   28   41   5    634  723  861  consoles     kmsg
1040  1150  137   18    293  42   50   642  724  9    cpuinfo      kpagecgroup
1041  1158  14    19    30   43   504  656  725  914  crypto       kpagecount
1042  1161  15    190   307  44   505  66   726  918  devices      kpageflags
1047  1162  1557  196   308  45   51   664  741  923  diskstats    loadavg
1050  1163  1563  197   31   457  52   67   742  938  dma          locks
1057  1164  16    199   313  458  540  682  743  943  driver       meminfo
1062  1165  165   2     32   46   572  683  749  946  execdomains  misc
root@kali-Test:~#
```

Where does Linux save information about a process?

- Each process holds a directory that contains information about that process.
- For example, “maps” file contains the memory layout of process ID = 683

```
root@kali-Test:~# cd /proc/683
root@kali-Test:/proc/683# ls
attr          comm             fd         map_files   net             personality  setgroups  syscall
autogroup     coredump_filter  fdinfo     maps        ns              projid_map   smaps      task
auxv          cpuset           gid_map    mem         oom_adj         root         stack      timers
cgroup        cwd              io         mountinfo   oom_score       sched        stat       timerslack_ns
clear_refs    environ          limits     mounts      oom_score_adj   schedstat    statm      uid_map
cmdline       exe              loginuid   mountstats  pagemap         sessionid    status     wchan
root@kali-Test:/proc/683#
```

Where does Linux save information about a process?

```
root@kali-Test:/proc/683# cat maps
004d2000-004e1000 r-xp 00000000 08:01 140957     /usr/lib/rtkit/rtkit-daemon
004e1000-004e2000 r--p 0000e000 08:01 140957     /usr/lib/rtkit/rtkit-daemon
004e2000-004e3000 rw-p 0000f000 08:01 140957     /usr/lib/rtkit/rtkit-daemon
01e8b000-01eac000 rw-p 00000000 00:00 0          [heap]
b5f00000-b5f21000 rw-p 00000000 00:00 0
b5f21000-b6000000 ---p 00000000 00:00 0
b6100000-b6121000 rw-p 00000000 00:00 0
b6121000-b6200000 ---p 00000000 00:00 0
b6243000-b6244000 ---p 00000000 00:00 0
b6244000-b6a44000 rw-p 00000000 00:00 0
b6a44000-b6a45000 ---p 00000000 00:00 0
b6a45000-b7245000 rw-p 00000000 00:00 0
b7245000-b7250000 r-xp 00000000 08:01 1443493    /lib/i386-linux-gnu/libnss_files-2.24.so
b7250000-b7251000 r--p 0000a000 08:01 1443493    /lib/i386-linux-gnu/libnss_files-2.24.so
b7251000-b7252000 rw-p 0000b000 08:01 1443493    /lib/i386-linux-gnu/libnss_files-2.24.so
b7252000-b7258000 rw-p 00000000 00:00 0
b7258000-b7263000 r-xp 00000000 08:01 1443504    /lib/i386-linux-gnu/libnss_nis-2.24.so
b7263000-b7264000 r--p 0000a000 08:01 1443504    /lib/i386-linux-gnu/libnss_nis-2.24.so
b7264000-b7265000 rw-p 0000b000 08:01 1443504    /lib/i386-linux-gnu/libnss_nis-2.24.so
b7265000-b727b000 r-xp 00000000 08:01 1443487    /lib/i386-linux-gnu/libnsl-2.24.so
b727b000-b727c000 r--p 00016000 08:01 1443487    /lib/i386-linux-gnu/libnsl-2.24.so
b727c000-b727d000 rw-p 00017000 08:01 1443487    /lib/i386-linux-gnu/libnsl-2.24.so
b727d000-b727f000 rw-p 00000000 00:00 0
b727f000-b7287000 r-xp 00000000 08:01 1443489    /lib/i386-linux-gnu/libnss_compat-2.24.so
b7287000-b7288000 r--p 00007000 08:01 1443489    /lib/i386-linux-gnu/libnss_compat-2.24.so
b7288000-b7289000 rw-p 00008000 08:01 1443489    /lib/i386-linux-gnu/libnss_compat-2.24.so
```

Where does Linux save information about a process?

```
b74ef000-b74f0000 rw-p 00090000 08:01 1443550    /lib/i386-linux-gnu/libsystemd.so.0.19.0
b74f0000-b76a1000 r-xp 00000000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b76a1000-b76a3000 r--p 001b0000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b76a3000-b76a4000 rw-p 001b2000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b76a4000-b76a7000 rw-p 00000000 00:00 0
b76a7000-b76c0000 r-xp 00000000 08:01 1443529    /lib/i386-linux-gnu/libpthread-2.24.so
b76c0000-b76c1000 r--p 00018000 08:01 1443529    /lib/i386-linux-gnu/libpthread-2.24.so
b76c1000-b76c2000 rw-p 00019000 08:01 1443529    /lib/i386-linux-gnu/libpthread-2.24.so
b76c2000-b76c4000 rw-p 00000000 00:00 0
b76c4000-b76cb000 r-xp 00000000 08:01 1443537    /lib/i386-linux-gnu/librt-2.24.so
b76cb000-b76cc000 r--p 00006000 08:01 1443537    /lib/i386-linux-gnu/librt-2.24.so
b76cc000-b76cd000 rw-p 00007000 08:01 1443537    /lib/i386-linux-gnu/librt-2.24.so
b76cd000-b76d1000 r-xp 00000000 08:01 1443410    /lib/i386-linux-gnu/libcap.so.2.25
b76d1000-b76d2000 r--p 00003000 08:01 1443410    /lib/i386-linux-gnu/libcap.so.2.25
b76d2000-b76d3000 rw-p 00004000 08:01 1443410    /lib/i386-linux-gnu/libcap.so.2.25
b76d3000-b772d000 r-xp 00000000 08:01 1443420    /lib/i386-linux-gnu/libdbus-1.so.3.14.13
b772d000-b772e000 ---p 0005a000 08:01 1443420    /lib/i386-linux-gnu/libdbus-1.so.3.14.13
b772e000-b772f000 r--p 0005a000 08:01 1443420    /lib/i386-linux-gnu/libdbus-1.so.3.14.13
b772f000-b7730000 rw-p 0005b000 08:01 1443420    /lib/i386-linux-gnu/libdbus-1.so.3.14.13
b7750000-b7753000 rw-p 00000000 00:00 0
b7753000-b7756000 r--p 00000000 00:00 0          [vvar]
b7756000-b7758000 r-xp 00000000 00:00 0          [vdso]
b7758000-b777b000 r-xp 00000000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b777b000-b777c000 r--p 00022000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b777c000-b777d000 rw-p 00023000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
bfbea000-bfc0b000 rw-p 00000000 00:00 0          [stack]
root@kali-Test:/proc/683#
```

User Mode C-Program

- Create cprogram1.c
- Compile the program
- Run the program
- Display the program's memory information

Create cprogram1.c

```
// cprogram1.c //

#include <stdio.h>
#include <stdlib.h>

int add (int x, int y)
{
     int result = 0;
     result = x + y;
     return result;
}

main (int argc, char **argv)
{
    int a = atoi(argv[1]);
    int b = atoi(argv[2]);
    int c;
    char buffer [100];

    gets(buffer);
    puts(buffer);

    c = add (a, b);

    printf("Sum of %d and %d = %d\n", a, b, c);
    exit (0);
}
```

The **atoi() function** converts a character string to an integer value

Compile the C-Program

- Compile the C-program
- Run the program
- Display the program's memory information

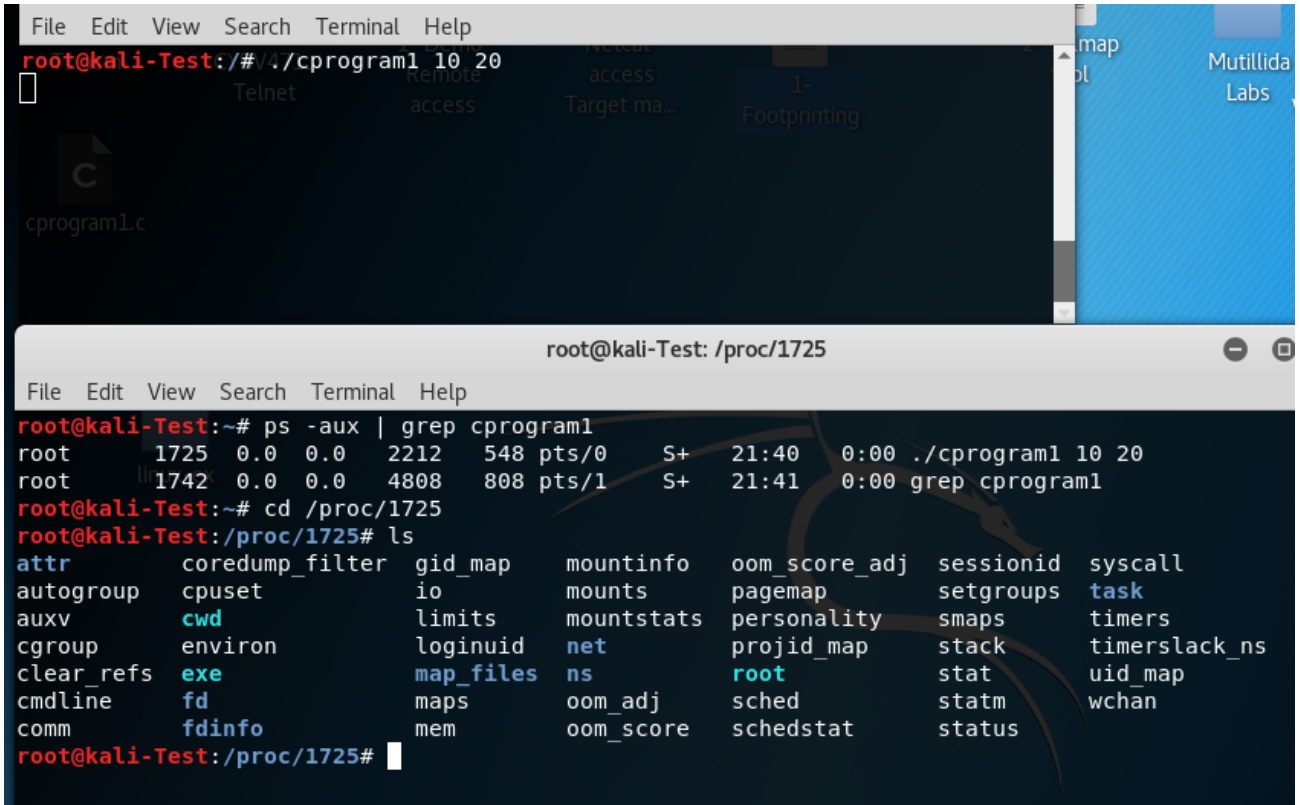
```
root@kali-Test:/# gcc -g cprogram1.c -o cprogram1
cprogram1.c:13:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main (int argc, char **argv)
 ^~~~
cprogram1.c: In function 'main':
cprogram1.c:20:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'
? [-Wimplicit-function-declaration]
     gets(buffer);
     ^~~~
     fgets
/tmp/cc2HY9jG.o: In function `main':
//cprogram1.c:20: warning: the `gets' function is dangerous and should not be used.
root@kali-Test:/#
```

```
root@kali-Test:/# ./cprogram1 10 20
Hello
Hello
Sum of 10 and 20 = 30
root@kali-Test:/#
```

Compile the C-Program

- Display the program's memory information

```
root@kali-Test:/# ./cprogram1 10 20
Hello
Hello
Sum of 10 and 20 = 30
root@kali-Test:/# ps -aux | grep cprogram1
root      1717  0.0  0.0   4808   800 pts/0    S+   21:39   0:00 grep cprogram1
root@kali-Test:/# cd /proc/1717
bash: cd: /proc/1717: No such file or directory
root@kali-Test:/#
```



Compile the C-Program

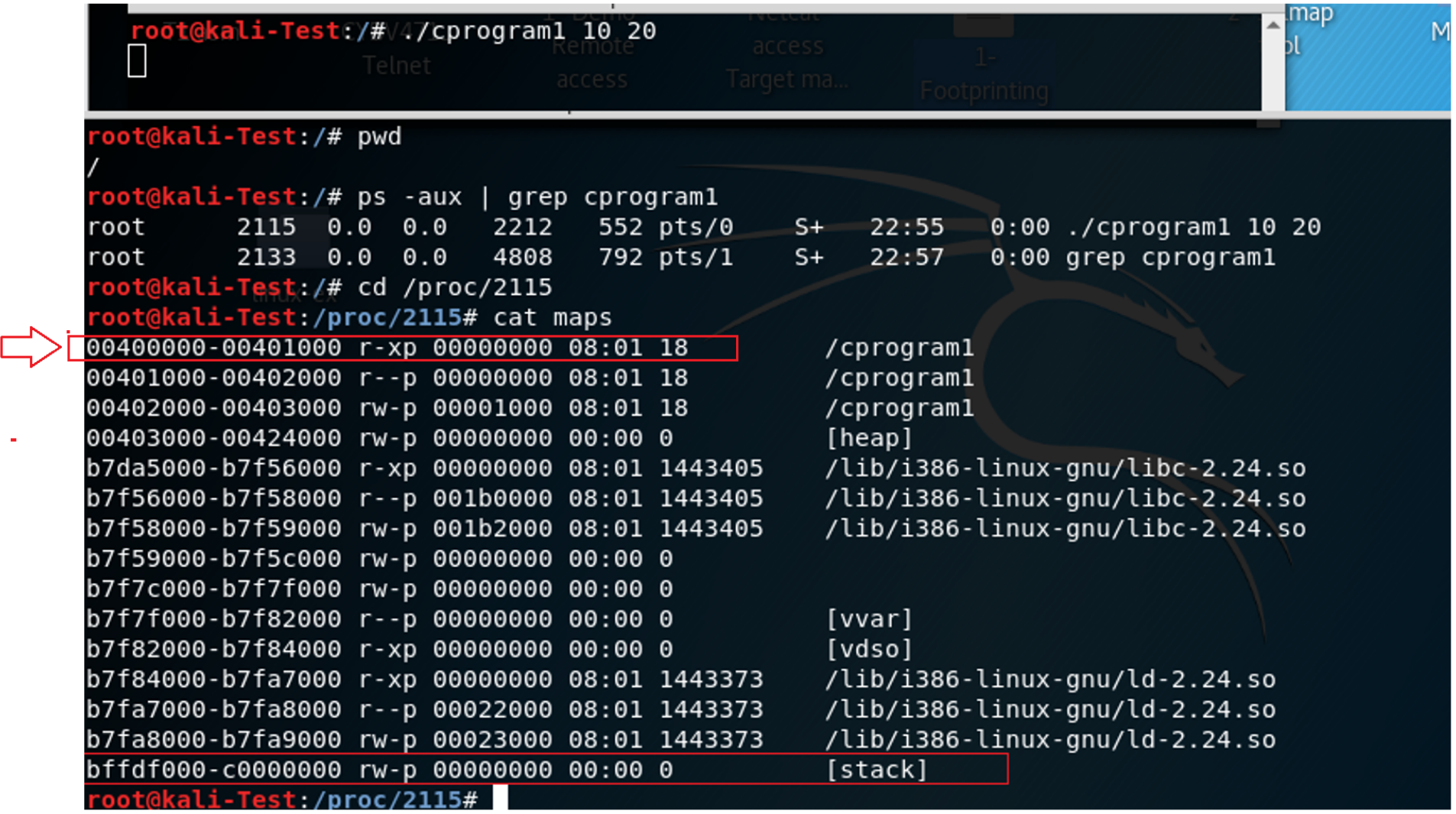
- Display the program's memory information
- Will the stack start in same location?

```
root@kali-Test:/proc/1725# cat maps
004c5000-004c6000 r-xp 00000000 08:01 18         /cprogram1
004c6000-004c7000 r--p 00000000 08:01 18         /cprogram1
004c7000-004c8000 rw-p 00001000 08:01 18         /cprogram1
0121a000-0123b000 rw-p 00000000 00:00 0          [heap]
b7552000-b7703000 r-xp 00000000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7703000-b7705000 r--p 001b0000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7705000-b7706000 rw-p 001b2000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7706000-b7709000 rw-p 00000000 00:00 0
b7729000-b772c000 rw-p 00000000 00:00 0
b772c000-b772f000 r--p 00000000 00:00 0          [vvar]
b772f000-b7731000 r-xp 00000000 00:00 0          [vdso]
b7731000-b7754000 r-xp 00000000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b7754000-b7755000 r--p 00022000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b7755000-b7756000 rw-p 00023000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
bffa0000-bffc1000 rw-p 00000000 00:00 0          [stack]
root@kali-Test:/proc/1725#
```

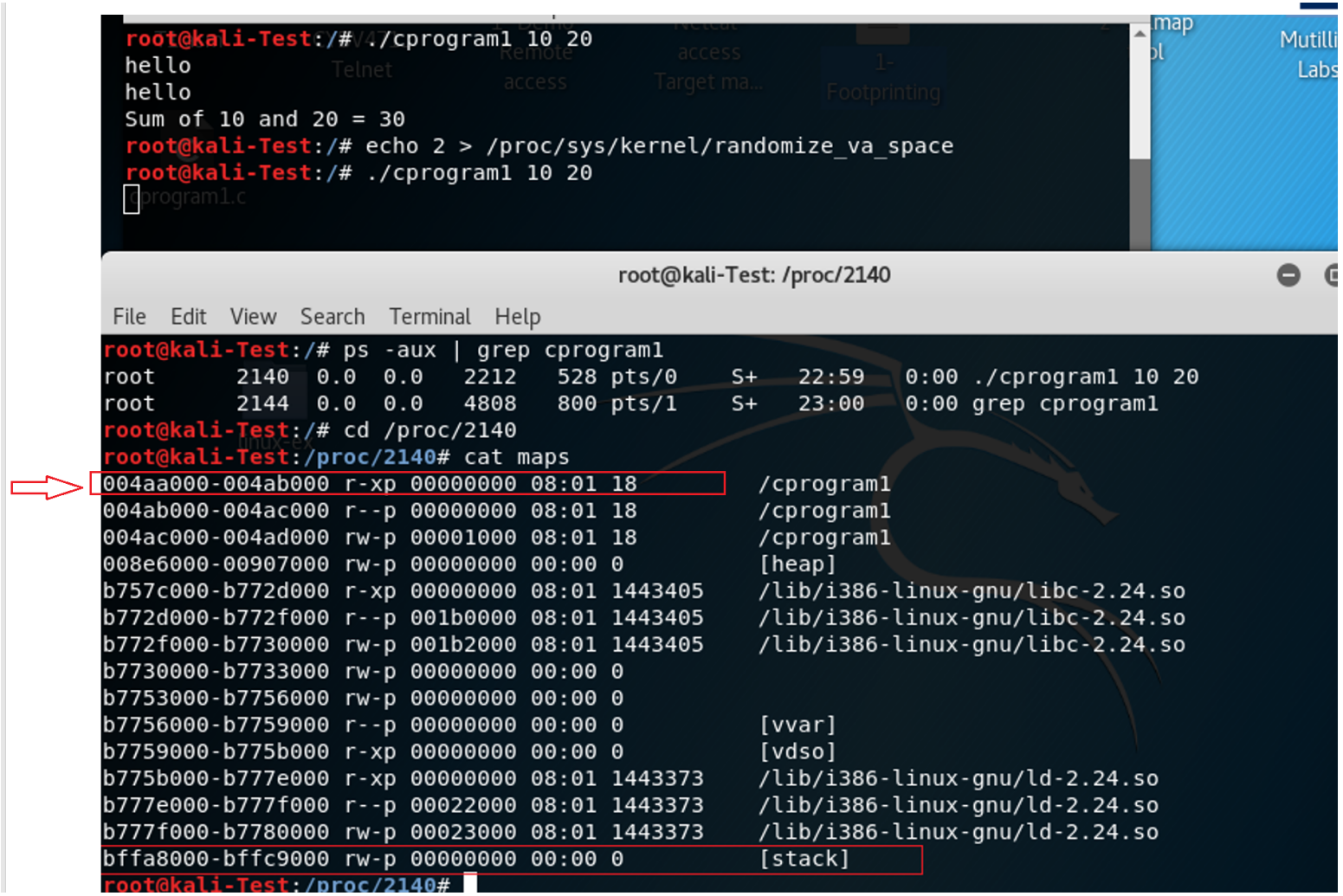
Protecting memory from buffer overflow attacks

```
root@kali-Test:/# sysctl -a --pattern "randomize"
kernel.randomize_va_space = 2
root@kali-Test:/# echo 0 > /proc/sys/kernel/randomize_va_space
root@kali-Test:/# sysctl -a --pattern "randomize"
kernel.randomize_va_space = 0
root@kali-Test:/# ps -aux | grep cprogram1
root      2063  0.0  0.0   2212   552 pts/0    S+   22:46   0:00 ./cprogram1 10 20
root      2071  0.0  0.0   4808   820 pts/1    S+   22:46   0:00 grep cprogram1
root@kali-Test:/# cd /proc/2063
root@kali-Test:/proc/2063# cat maps
00400000-00401000 r-xp 00000000 08:01 18         /cprogram1
00401000-00402000 r--p 00000000 08:01 18         /cprogram1
00402000-00403000 rw-p 00001000 08:01 18         /cprogram1
00403000-00424000 rw-p 00000000 00:00 0          [heap]
b7dc5000-b7f76000 r-xp 00000000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7f76000-b7f78000 r--p 001b0000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7f78000-b7f79000 rw-p 001b2000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7f79000-b7f7c000 rw-p 00000000 00:00 0
b7f9c000-b7f9f000 rw-p 00000000 00:00 0
b7f9f000-b7fa2000 r--p 00000000 00:00 0          [vvar]
b7fa2000-b7fa4000 r-xp 00000000 00:00 0          [vdso]
b7fa4000-b7fc7000 r-xp 00000000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b7fc7000-b7fc8000 r--p 00022000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b7fc8000-b7fc9000 rw-p 00023000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
root@kali-Test:/proc/2063#
```

Protecting memory from buffer overflow attacks



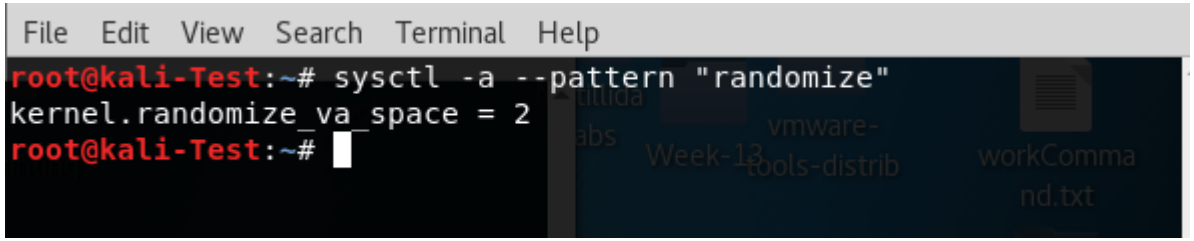
Protecting memory from buffer overflow attacks

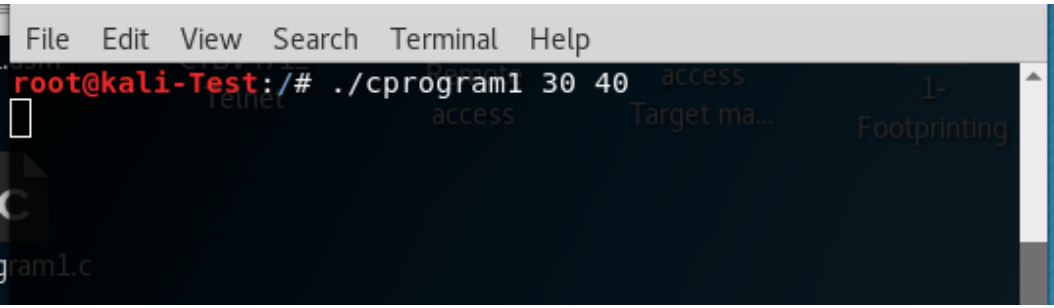


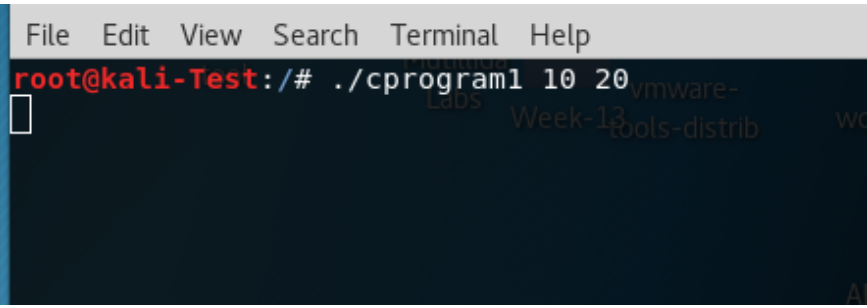
Running same C-Program Twice

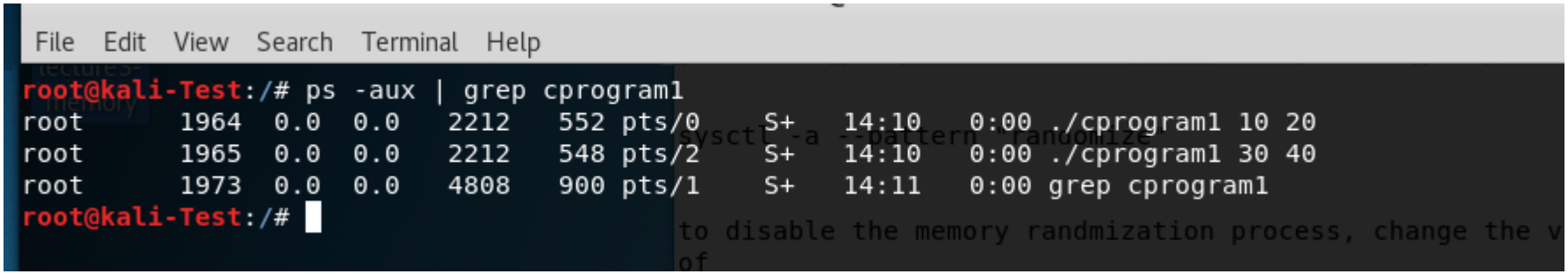
- Run one instance of cprogram1.c
- Run second instance of cprogram1.c
- Compare the memory layout of each program

Running same C-Program Twice

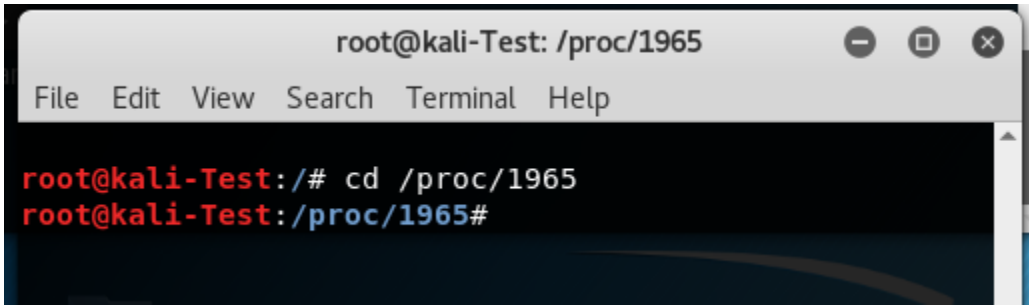


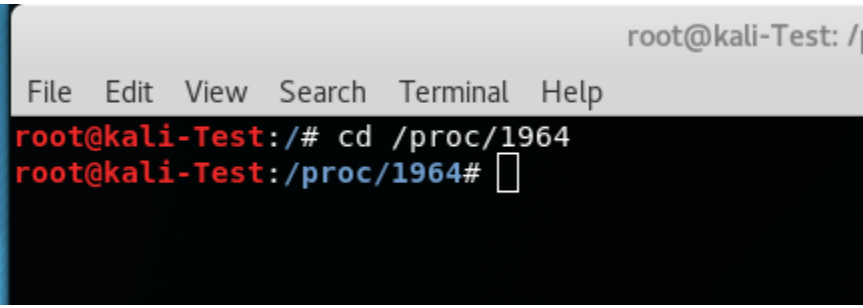






Running same C-Program Twice With Memory Protection






```
root@kali-Test:/# cd /proc/1965
root@kali-Test:/proc/1965# cat maps
00409000-0040a000 r-xp 00000000 08:01 18       /cprogram1
0040a000-0040b000 r--p 00000000 08:01 18       /cprogram1
0040b000-0040c000 rw-p 00001000 08:01 18       /cprogram1
00a7c000-00a9d000 rw-p 00000000 00:00 0        [heap]
b7567000-b7718000 r-xp 00000000 08:01 1443405  /lib/i386-linux-gnu/libc-2.24.so
b7718000-b771a000 r--p 001b0000 08:01 1443405  /lib/i386-linux-gnu/libc-2.24.so
b771a000-b771b000 rw-p 001b2000 08:01 1443405  /lib/i386-linux-gnu/libc-2.24.so
b771b000-b771e000 rw-p 00000000 00:00 0
b773e000-b7741000 rw-p 00000000 00:00 0
b7741000-b7744000 r--p 00000000 00:00 0        [vvar]
b7744000-b7746000 r-xp 00000000 00:00 0        [vdso]
b7746000-b7769000 r-xp 00000000 08:01 1443373  /lib/i386-linux-gnu/ld-2.24.so
b7769000-b776a000 r--p 00022000 08:01 1443373  /lib/i386-linux-gnu/ld-2.24.so
b776a000-b776b000 rw-p 00023000 08:01 1443373  /lib/i386-linux-gnu/ld-2.24.so
bfabc000-bfadd000 rw-p 00000000 00:00 0        [stack]
root@kali-Test:/proc/1965#
```

```
root@kali-Test:/# cd /proc/1965
root@kali-Test:/proc/1965# cat maps
00409000-0040a000 r-xp 00000000 08:01 18         /cprogram1
0040a000-0040b000 r--p 00000000 08:01 18         /cprogram1
0040b000-0040c000 rw-p 00001000 08:01 18         /cprogram1
00a7c000-00a9d000 rw-p 00000000 00:00 0          [heap]
b7567000-b7718000 r-xp 00000000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7718000-b771a000 r--p 001b0000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b771a000-b771b000 rw-p 001b2000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b771b000-b771e000 rw-p 00000000 00:00 0
b773e000-b7741000 rw-p 00000000 00:00 0
b7741000-b7744000 r--p 00000000 00:00 0          [vvar]
b7744000-b7746000 r-xp 00000000 00:00 0          [vdso]
b7746000-b7769000 r-xp 00000000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b7769000-b776a000 r--p 00022000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b776a000-b776b000 rw-p 00023000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
bfabc000-bfadd000 rw-p 00000000 00:00 0          [stack]
root@kali-Test:/proc/1965#
```

```
File  Edit  View  Search  Terminal  Help
root@kali-Test:/# cd /proc/1964
root@kali-Test:/proc/1964# cat maps
004bc000-004bd000 r-xp 00000000 08:01 18         /cprogram1
004bd000-004be000 r--p 00000000 08:01 18         /cprogram1
004be000-004bf000 rw-p 00001000 08:01 18         /cprogram1
0227e000-0229f000 rw-p 00000000 00:00 0          [heap]
b75c5000-b7776000 r-xp 00000000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7776000-b7778000 r--p 001b0000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7778000-b7779000 rw-p 001b2000 08:01 1443405    /lib/i386-linux-gnu/libc-2.24.so
b7779000-b777c000 rw-p 00000000 00:00 0
b779c000-b779f000 rw-p 00000000 00:00 0
b779f000-b77a2000 r--p 00000000 00:00 0          [vvar]
b77a2000-b77a4000 r-xp 00000000 00:00 0          [vdso]
b77a4000-b77c7000 r-xp 00000000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b77c7000-b77c8000 r--p 00022000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
b77c8000-b77c9000 rw-p 00023000 08:01 1443373    /lib/i386-linux-gnu/ld-2.24.so
bfcbb000-bfcdc000 rw-p 00000000 00:00 0          [stack]
root@kali-Test:/proc/1964#
```

Putting It All Together

You should know:

- **General and special registers**
- **Loading and Executing a Program**
- **Understand memory Layout**
- **Understand C-program layout**
- **Program Process Concept**
- **Display memory layout in Linux OS**
- **Protecting memory from buffer overflow attacks**

Questions?

Coming Next Week

GDB debugger, debug C and assembly language programs, and disassemble C-programs