Process Address Space: The way to create virtual address (page table) of userspace application

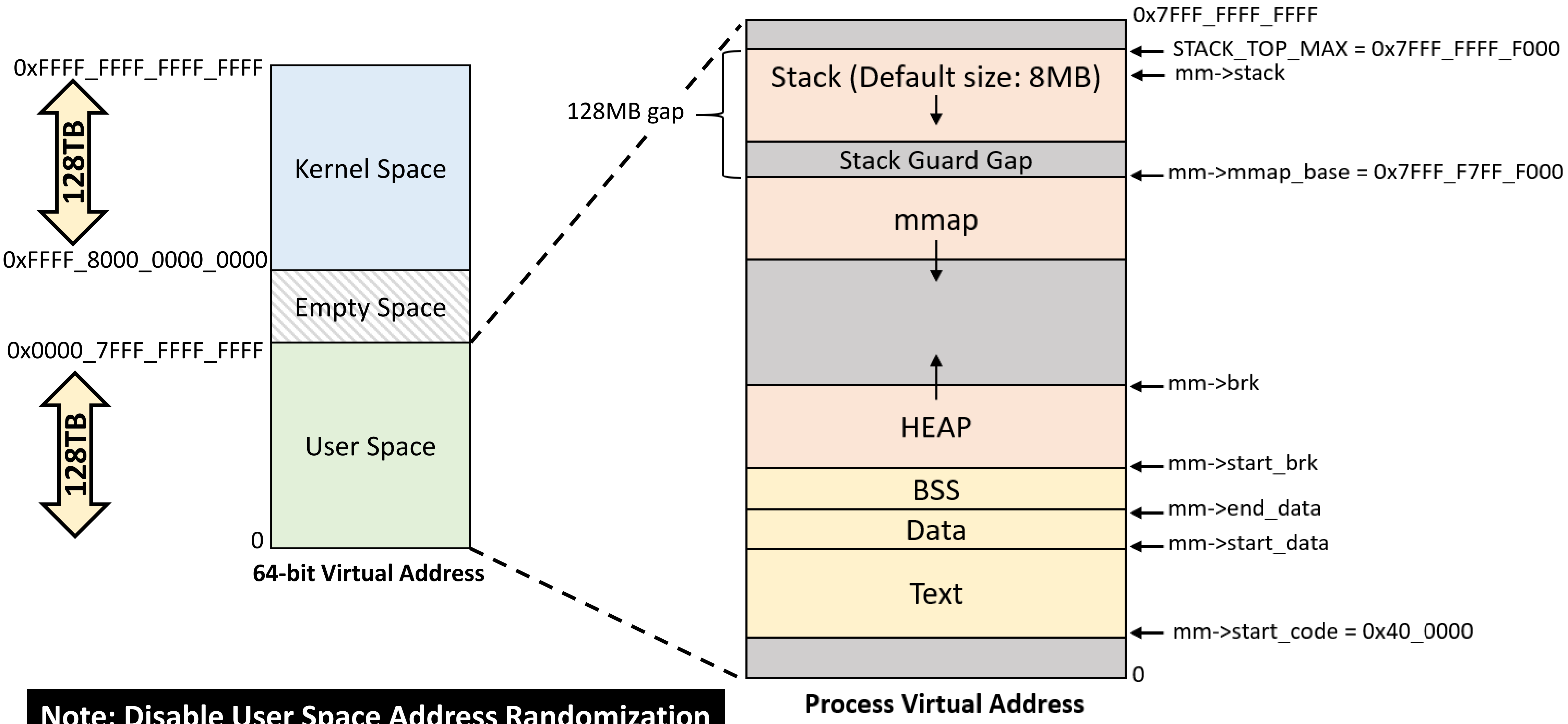
Adrian Huang | Oct, 2021

*** Based on kernel 5.11 (x86_64) – QEMU**
*** SMP (4 CPUs) and 8GB memory**
*** Kernel parameter: nokaslr norandmaps**
*** Userspace: ASLR is disabled**
*** Legacy BIOS**

Agenda

- 64-bit Virtual Address
- mm_struct & VMA
- Detail about stack
  - Stack configuration via bprm_execve()
  - Important function: load_elf_binary()
- Auxiliary Vector
- How does Linux call your main() function? The call path?
  - Statically-linked program
    - Base address: 0x400000
  - Dynamically-linked program
    - Base address: 0x555555554000
- Do you know why the base address of a dynamically-linked program is the base address ‘0x555555554000’?

64-bit Process Virtual Address



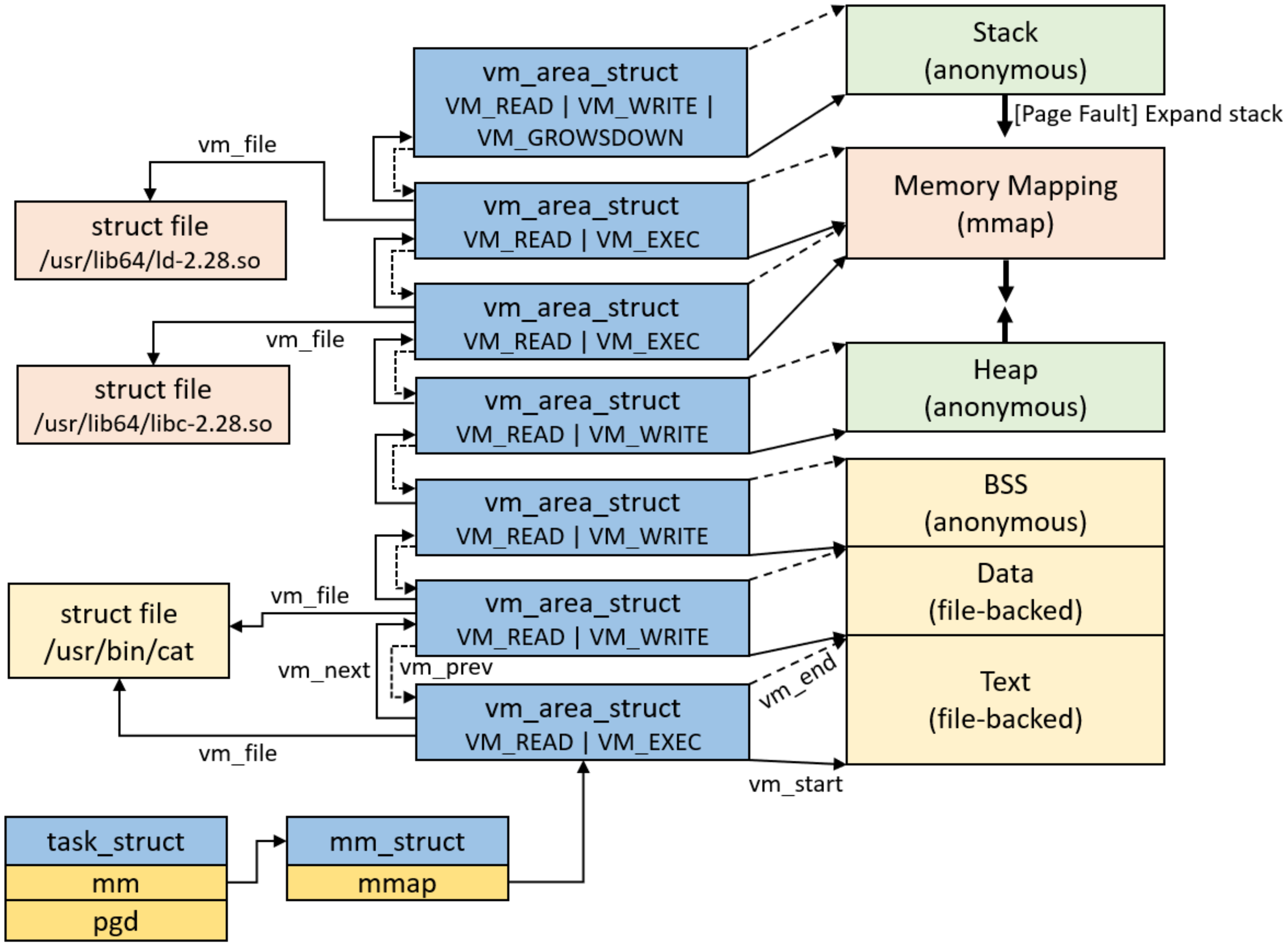
Note: Disable User Space Address Randomization

- [Option 1] Disable ASLR (Address Space Layout Randomization)

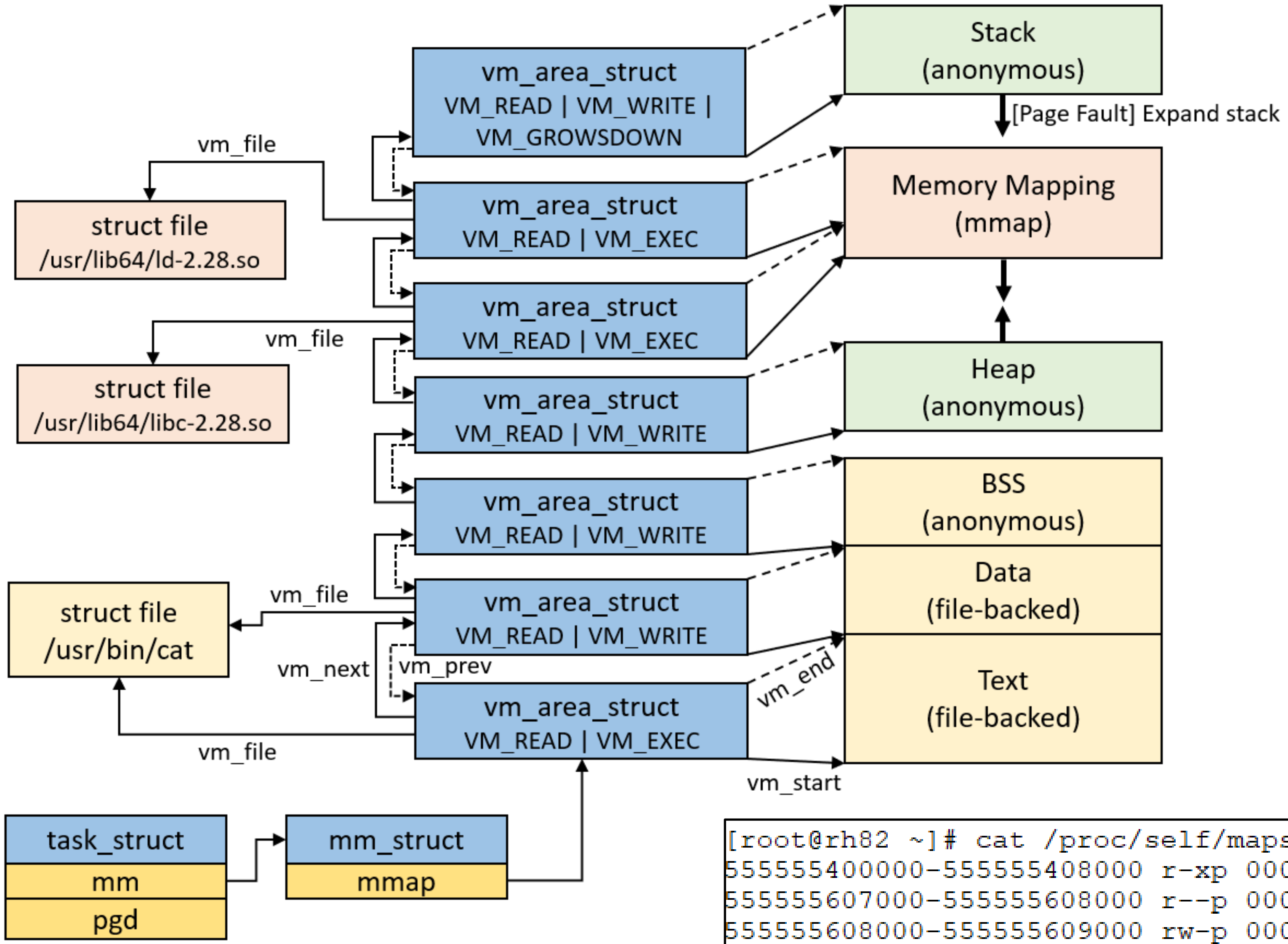
```
# echo 0 > /proc/sys/kernel/randomize_va_space
```

- [Option 2] kernel parameter: norandmaps

Process Address Space – mm_struct & VMA

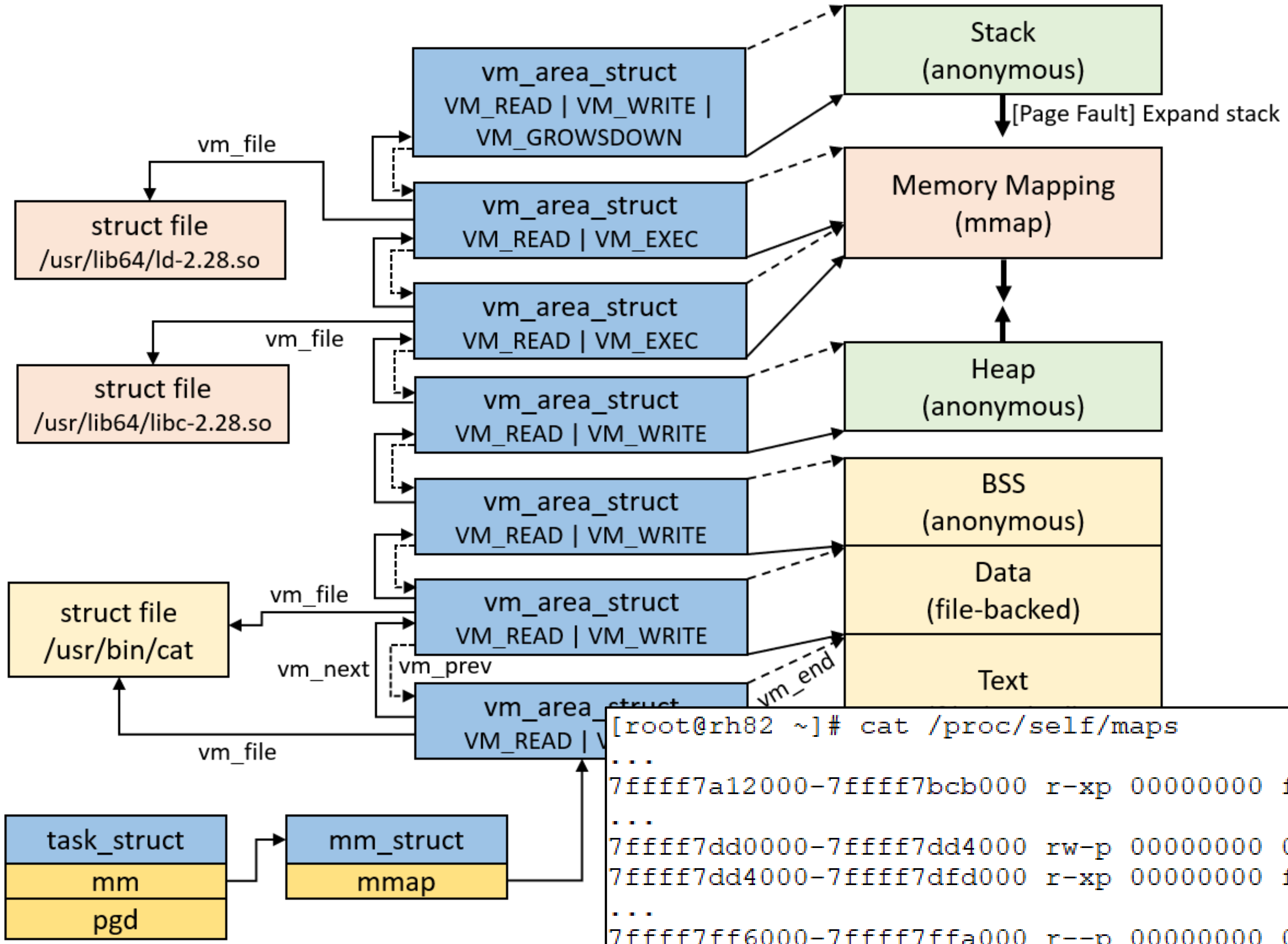


Process Address Space – mm_struct & VMA



```
[root@rh82 ~]# cat /proc/self/maps
555555400000-555555408000 r-xp 00000000 fd:00 262530          /usr/bin/cat
555555607000-555555608000 r--p 00007000 fd:00 262530          /usr/bin/cat
555555608000-555555609000 rw-p 00008000 fd:00 262530          /usr/bin/cat
555555609000-55555562a000 rw-p 00000000 00:00 0               [heap]
...
```

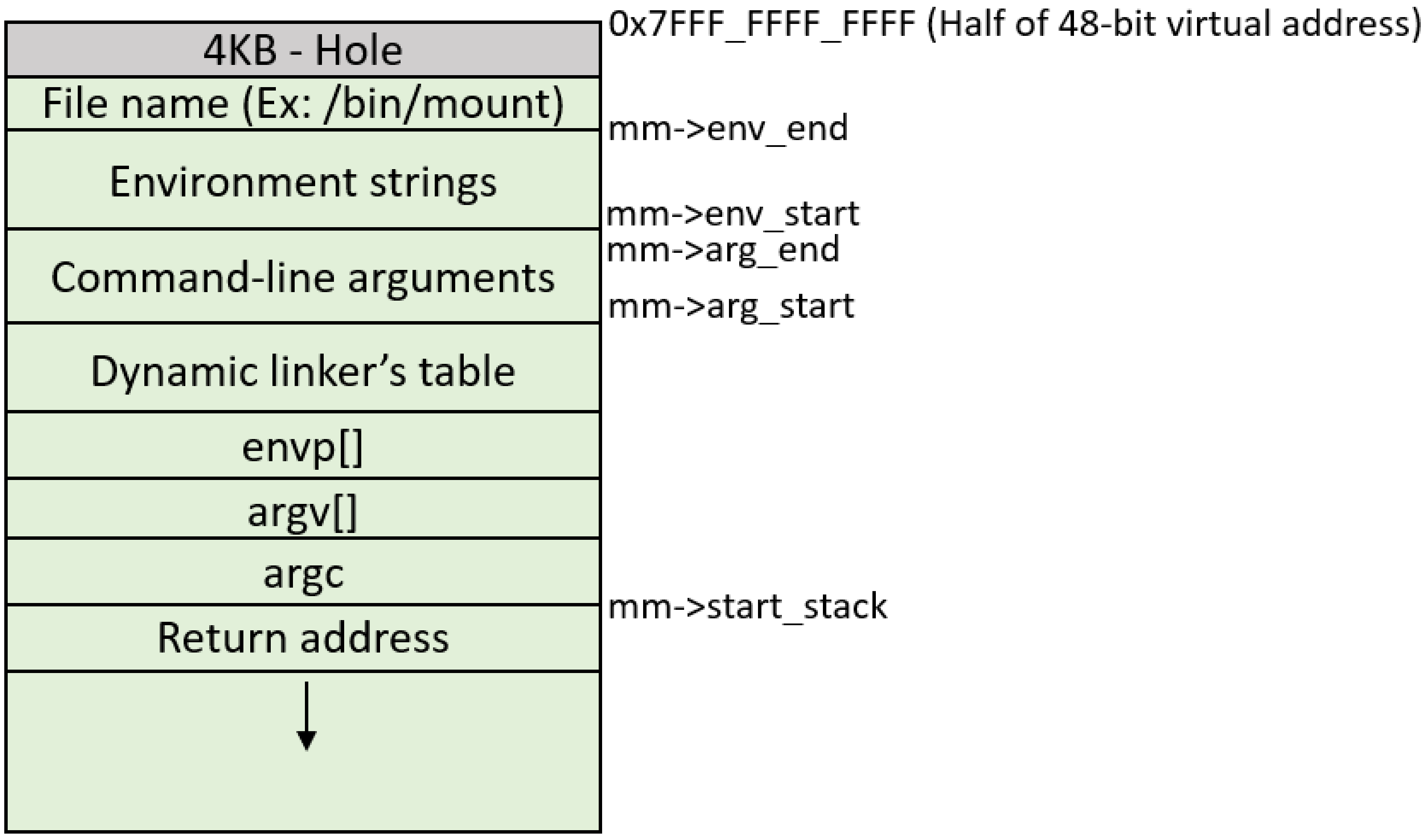
Process Address Space – mm_struct & VMA



```
handle_page_fault [arch/x86/mm/fault.c]
  |- do_user_addr_fault
    |- expand_stack
      |- expand_downwards
```

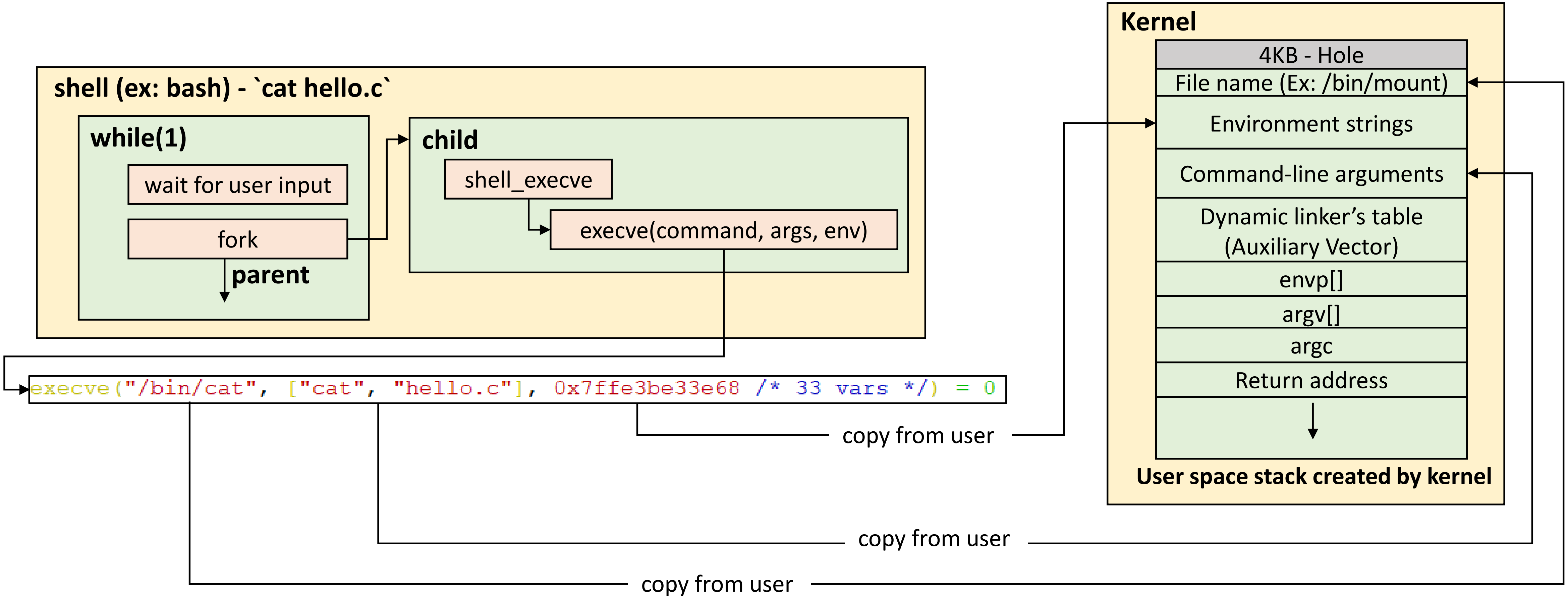
```
[root@rh82 ~]# cat /proc/self/maps
...
7ffff7a12000-7ffff7bcb000 r-xp 00000000 fd:00 33600280         /usr/lib64/libc-2.28.so
...
7ffff7dd0000-7ffff7dd4000 rw-p 00000000 00:00 0
7ffff7dd4000-7ffff7dfd000 r-xp 00000000 fd:00 33600273         /usr/lib64/ld-2.28.so
...
7ffff7ff6000-7ffff7ffa000 r--p 00000000 00:00 0                [vvar]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                [vdso]
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0        [vsyscall]
```

64-bit Process Virtual Address – Stack Layout



User Space Stack

Stack layout when executing a command



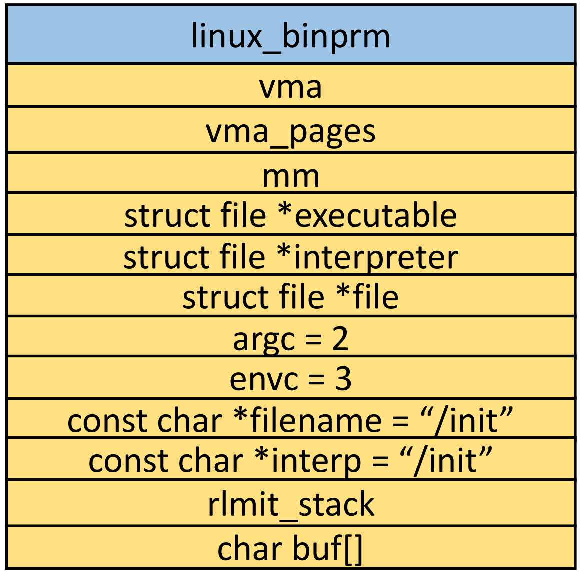
```
execve("/bin/cat", ["cat", "hello.c"], 0x7ffe3be33e68 /* 33 vars */) = 0
```

kernel_init -> run_init_process: init process (pid = 1) - Stack

```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- filename = getname_kernel(kernel_filename)
      /* Allocate and init a filename struct */
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
   bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```

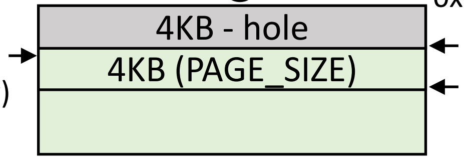
```
alloc_bprm [fs/exec.c]
  Allocate a linux_binprm struct from kzalloc
❶ [Executable file] bprm->filename = bprm->interp = filename->name
  |- bprm_mm_init
     bprm->mm = mm = mm_alloc()
     cfg bprm->rlim_stack (default: 8MB)
     |- __bprm_mm_init
       |- bprm->vma = vma = vm_area_alloc
       |- vma_set_anonymous
       vma->vm_end = STACK_TOP_MAX;
       vma->vm_start = vma->vm_end - PAGE_SIZE;
       |- insert_vm_struct(mm, vma);
       mm->stack_vm = mm->total_vm = 1;
❷      bprm->p = vma->vm_end - sizeof(void *);
```

❶



linux_binprm
vma
vma_pages
mm
struct file *executable
struct file *interpreter
struct file *file
argc = 2
envc = 3
const char *filename = “/init”
const char *interp = “/init”
rlmit_stack
char buf[]

❷



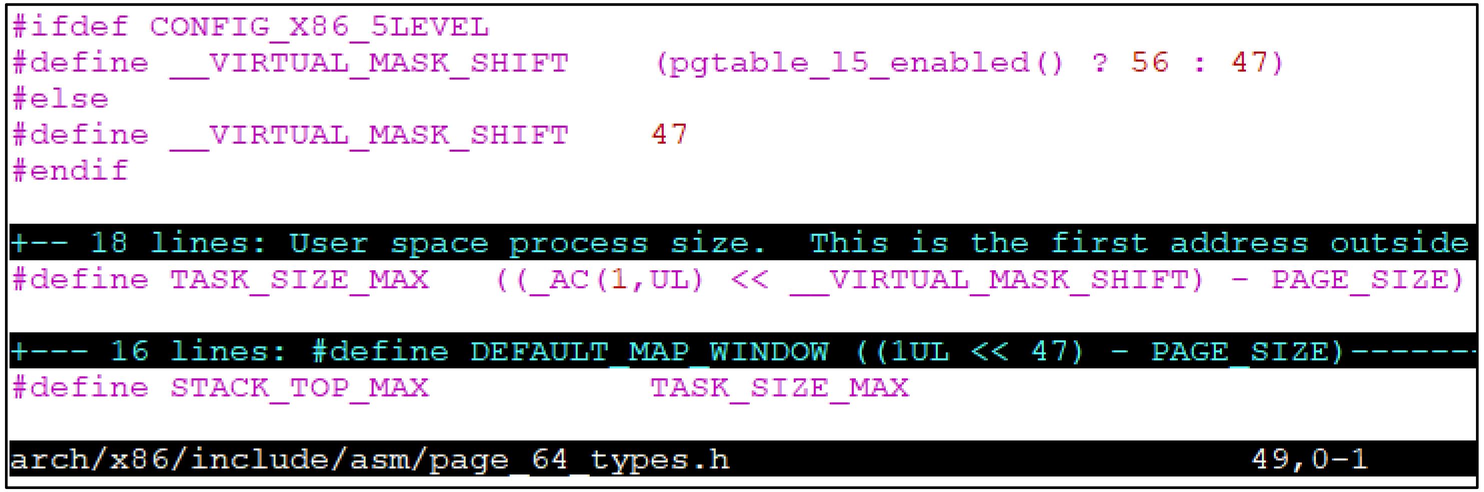
User Space Stack

bprm->p = vma->vm_end - sizeof(void *)
= 0x7FFF_FFFF_EFF8 (Current top of memory)

0x7FFF_FFFF_FFFF

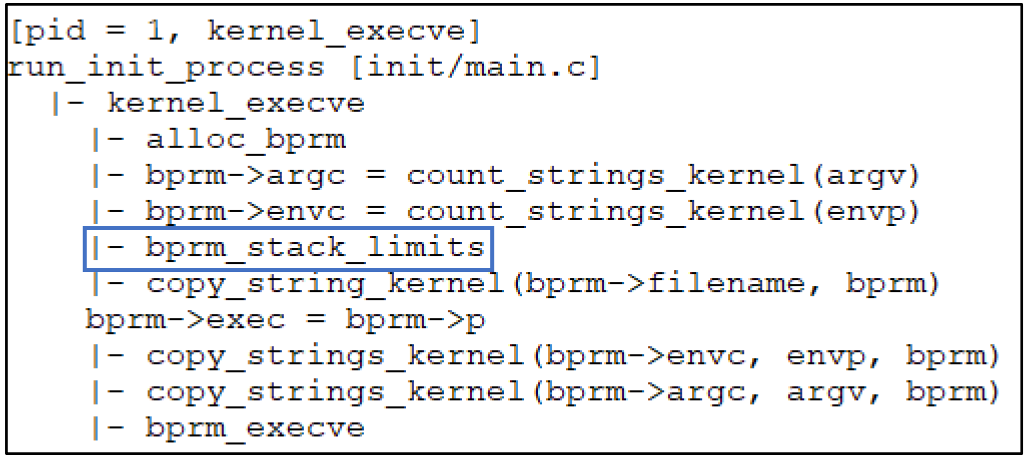
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000

vma->vm_start = 0x7FFF_FFFF_E000

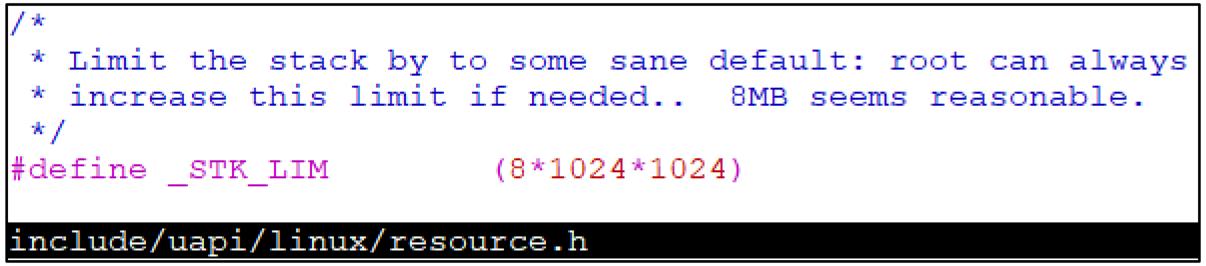


```
#ifdef CONFIG_X86_5LEVEL
#define __VIRTUAL_MASK_SHIFT    (pgtable_l5_enabled() ? 56 : 47)
#else
#define __VIRTUAL_MASK_SHIFT    47
#endif

+-- 18 lines: User space process size.  This is the first address outside
#define TASK_SIZE_MAX   ((_AC(1,UL) << __VIRTUAL_MASK_SHIFT) - PAGE_SIZE)

+--- 16 lines: #define DEFAULT_MAP_WINDOW ((1UL << 47) - PAGE_SIZE)-------
#define STACK_TOP_MAX           TASK_SIZE_MAX

arch/x86/include/asm/page_64_types.h                          49,0-1
```

kernel_init -> run_init_process: init process (pid = 1) - Stack



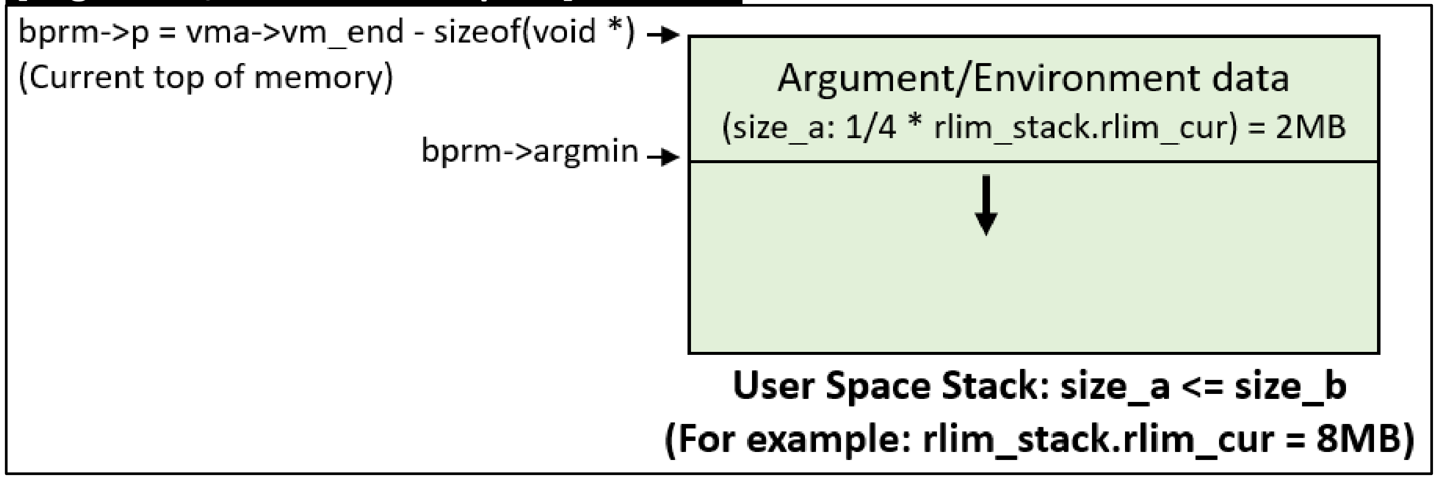
```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```



```
/*
 * Limit the stack by to some sane default: root can always
 * increase this limit if needed..  8MB seems reasonable.
 */
#define _STK_LIM        (8*1024*1024)
```

include/uapi/linux/resource.h

[Argument/environment space] Case #1



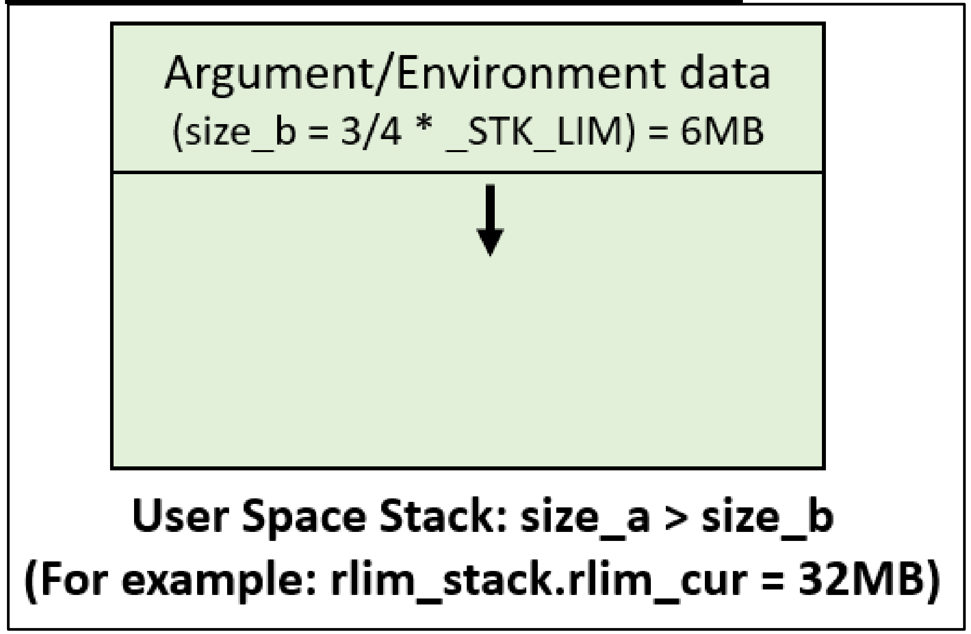
bprm->p = vma->vm_end - sizeof(void *)
(Current top of memory)

bprm->argmin

Argument/Environment data
(size_a: 1/4 * rlim_stack.rlim_cur) = 2MB

User Space Stack: size_a <= size_b
(For example: rlim_stack.rlim_cur = 8MB)

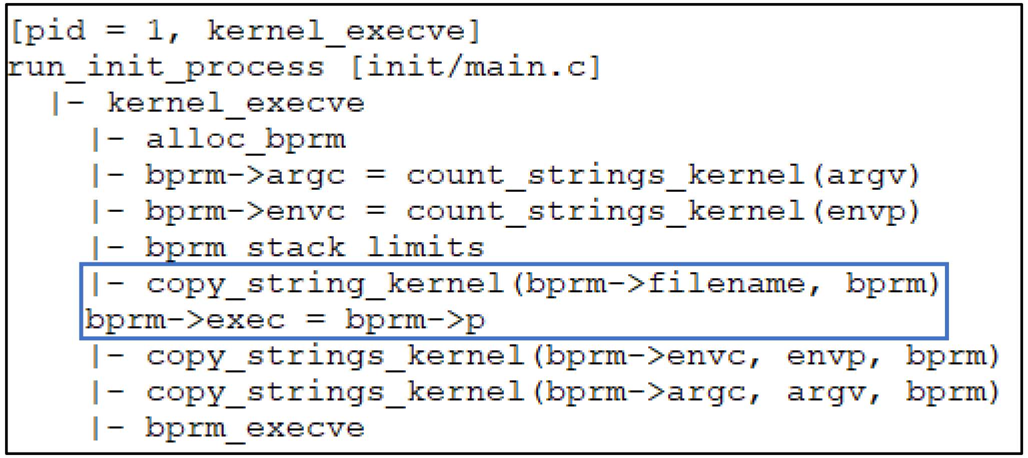
[Argument/environment space] Case #2



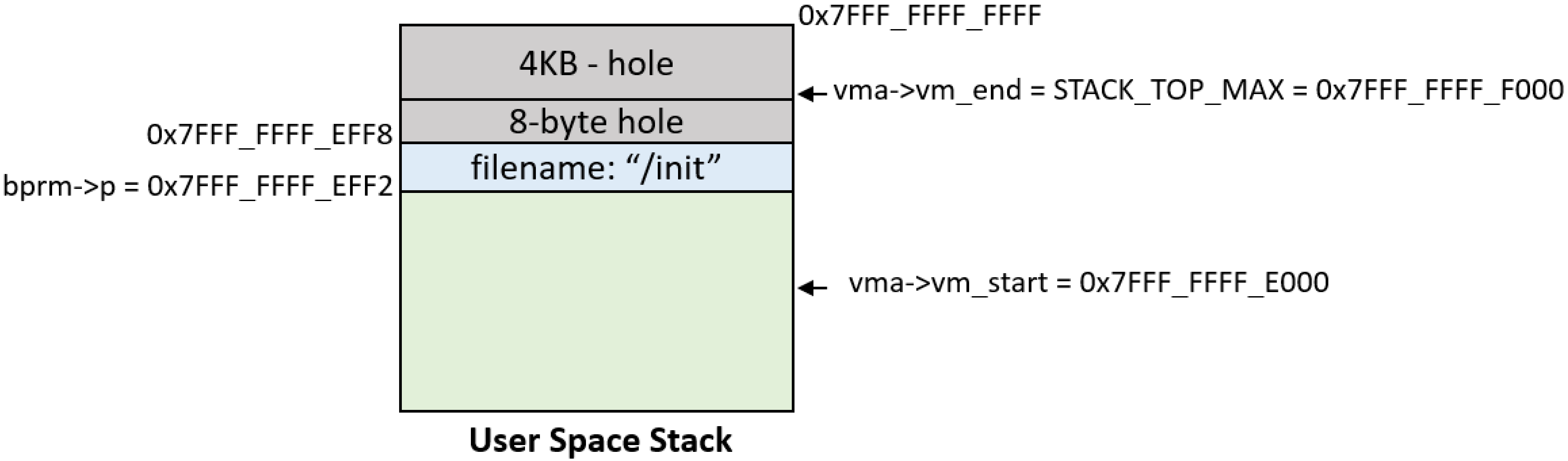
Argument/Environment data
(size_b = 3/4 * _STK_LIM) = 6MB

User Space Stack: size_a > size_b
(For example: rlim_stack.rlim_cur = 32MB)

kernel_init -> run_init_process: init process (pid = 1) - Stack

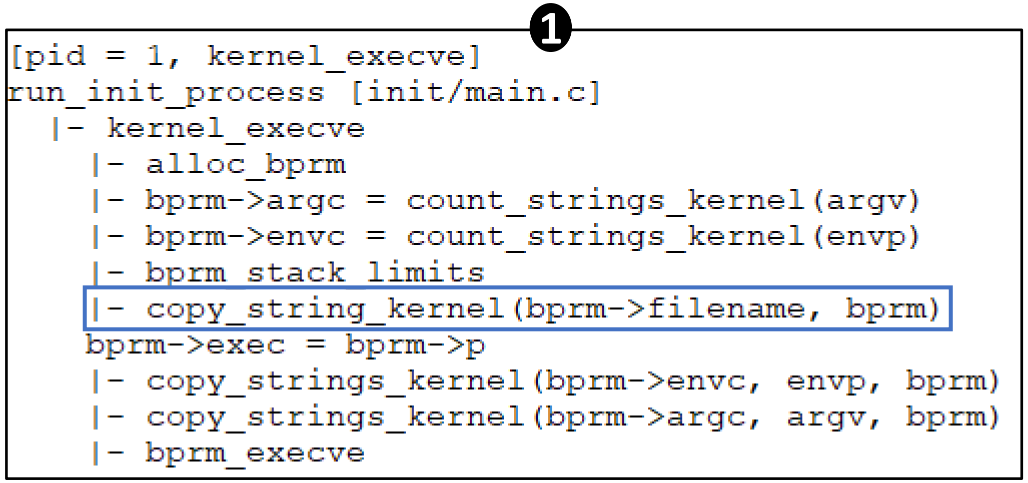


```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm stack limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```


0x7FFF_FFFF_FFFF
4KB - hole
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000
0x7FFF_FFFF_EFF8
8-byte hole
filename: “/init”
bprm->p = 0x7FFF_FFFF_EFF2
vma->vm_start = 0x7FFF_FFFF_E000
User Space Stack

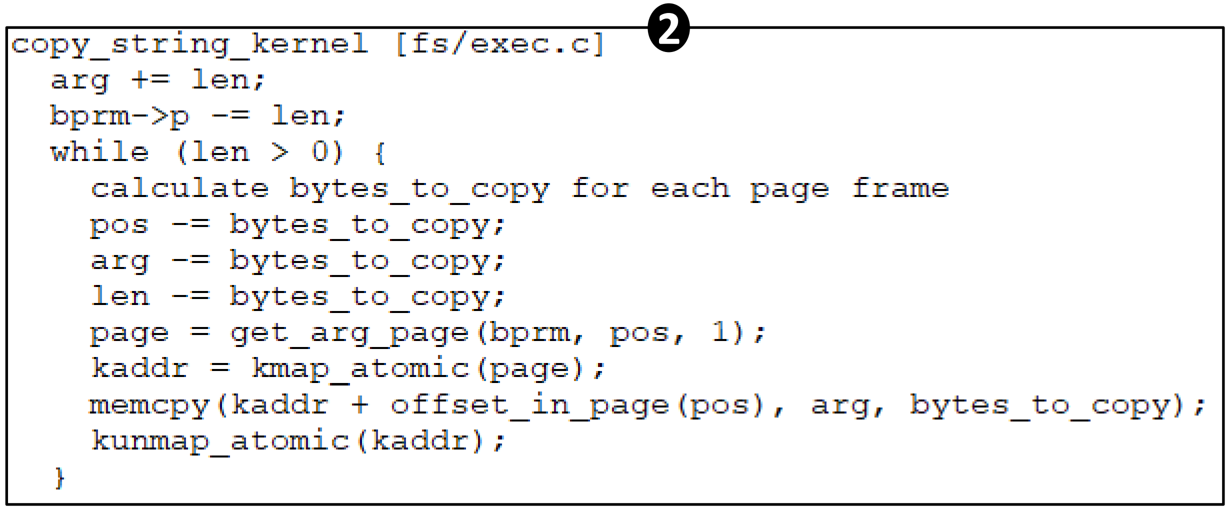
kernel_init -> run_init_process: init process (pid = 1) - Stack

❶



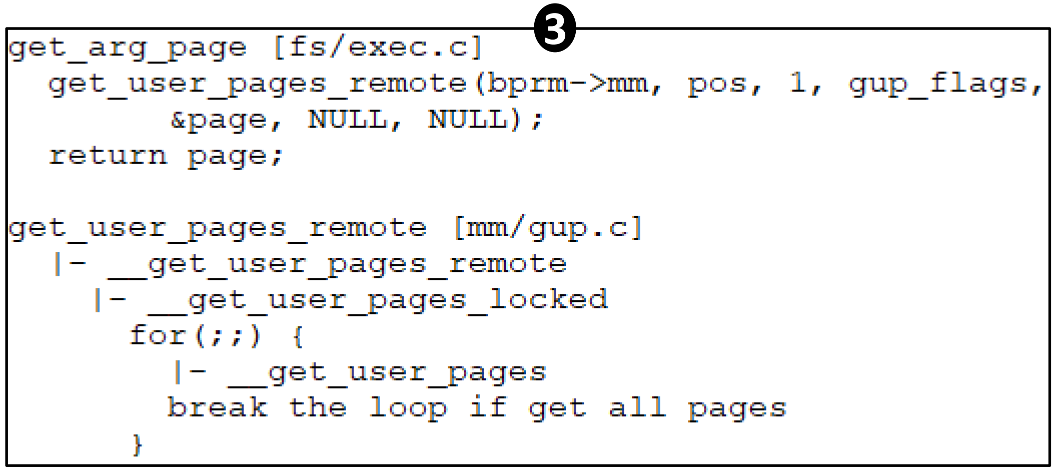
```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```

❷



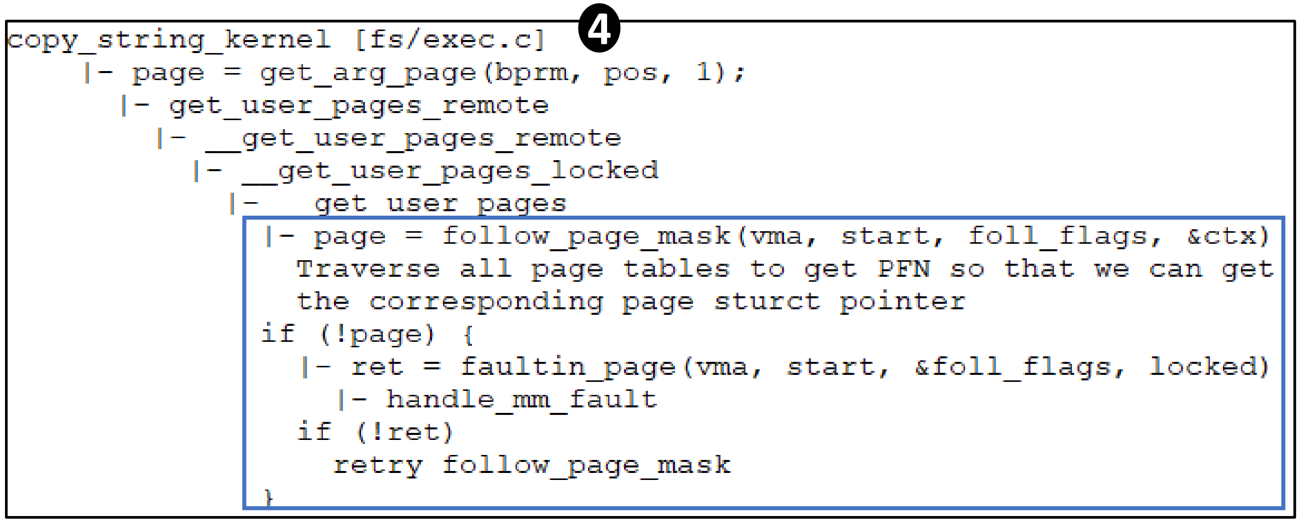
```
copy_string_kernel [fs/exec.c]
  arg += len;
  bprm->p -= len;
  while (len > 0) {
    calculate bytes_to_copy for each page frame
    pos -= bytes_to_copy;
    arg -= bytes_to_copy;
    len -= bytes_to_copy;
    page = get_arg_page(bprm, pos, 1);
    kaddr = kmap_atomic(page);
    memcpy(kaddr + offset_in_page(pos), arg, bytes_to_copy);
    kunmap_atomic(kaddr);
  }
```

❸



```
get_arg_page [fs/exec.c]
  get_user_pages_remote(bprm->mm, pos, 1, gup_flags,
        &page, NULL, NULL);
  return page;

get_user_pages_remote [mm/gup.c]
  |- __get_user_pages_remote
    |- __get_user_pages_locked
      for(;;) {
        |- __get_user_pages
        break the loop if get all pages
      }
```

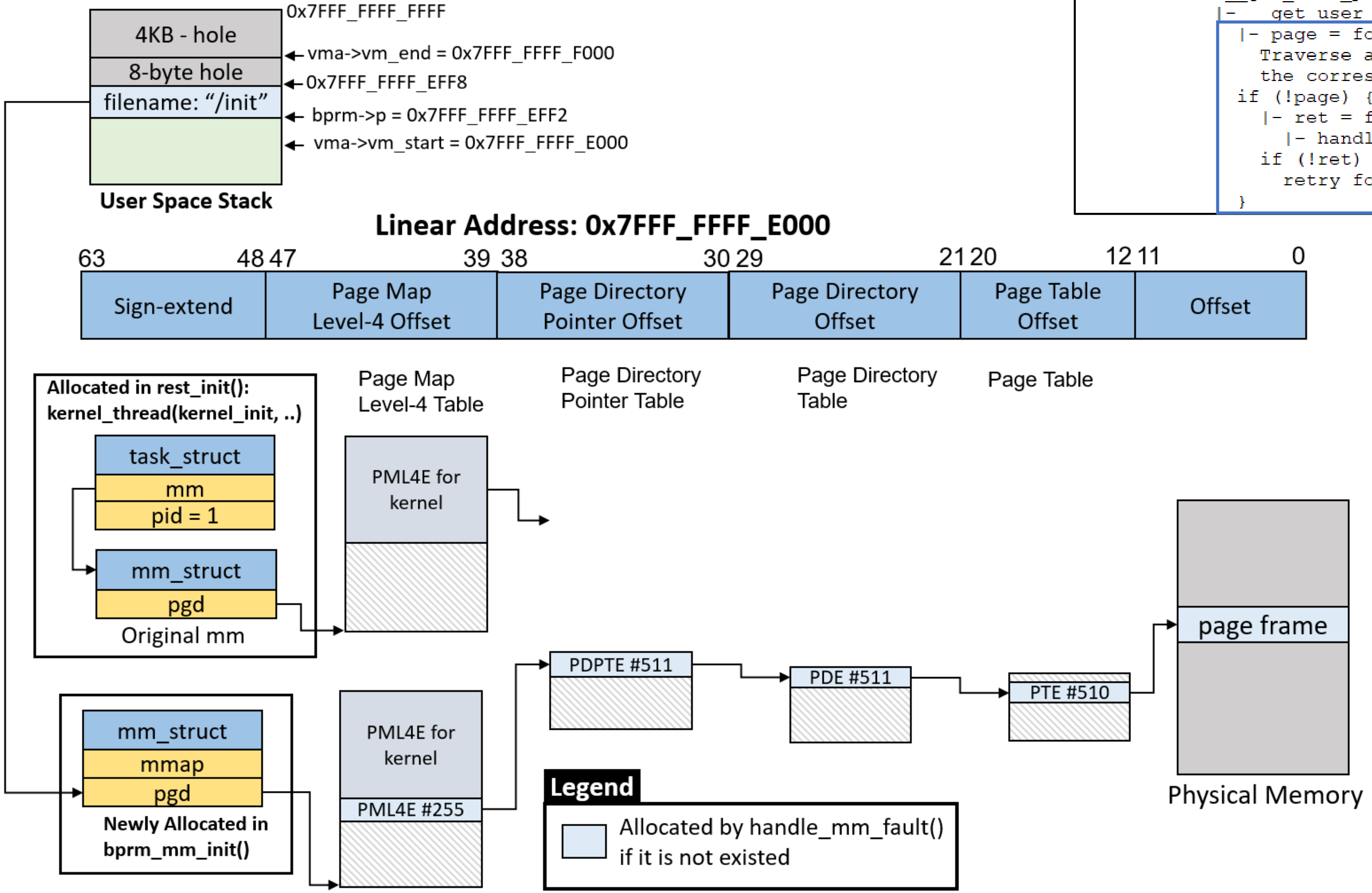
❹



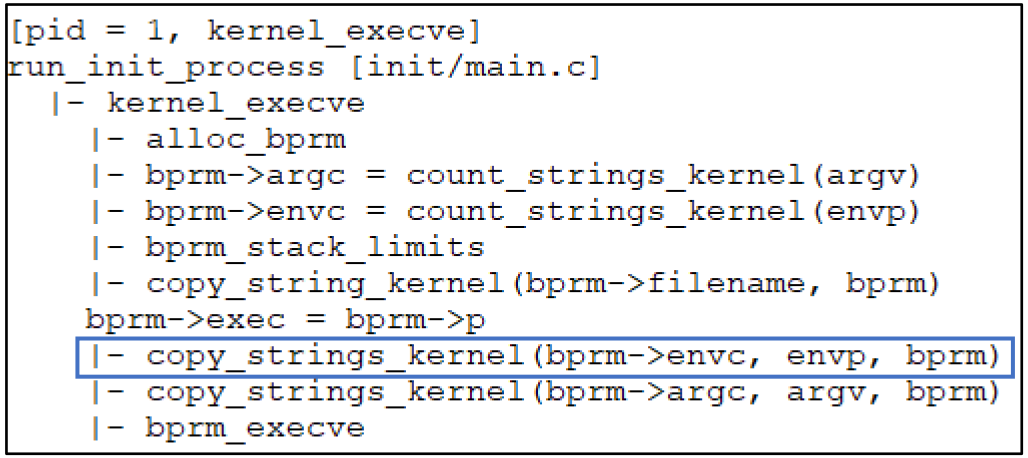
```
copy_string_kernel [fs/exec.c]
    |- page = get_arg_page(bprm, pos, 1);
      |- get_user_pages_remote
        |- __get_user_pages_remote
          |- __get_user_pages_locked
            |- __get_user_pages
              |- page = follow_page_mask(vma, start, foll_flags, &ctx)
                Traverse all page tables to get PFN so that we can get
                the corresponding page sturct pointer
              if (!page) {
                |- ret = faultin_page(vma, start, &foll_flags, locked)
                  |- handle_mm_fault
                if (!ret)
                  retry follow_page_mask
              }
```

kernel_init -> run_init_process: init process (pid = 1) - Stack

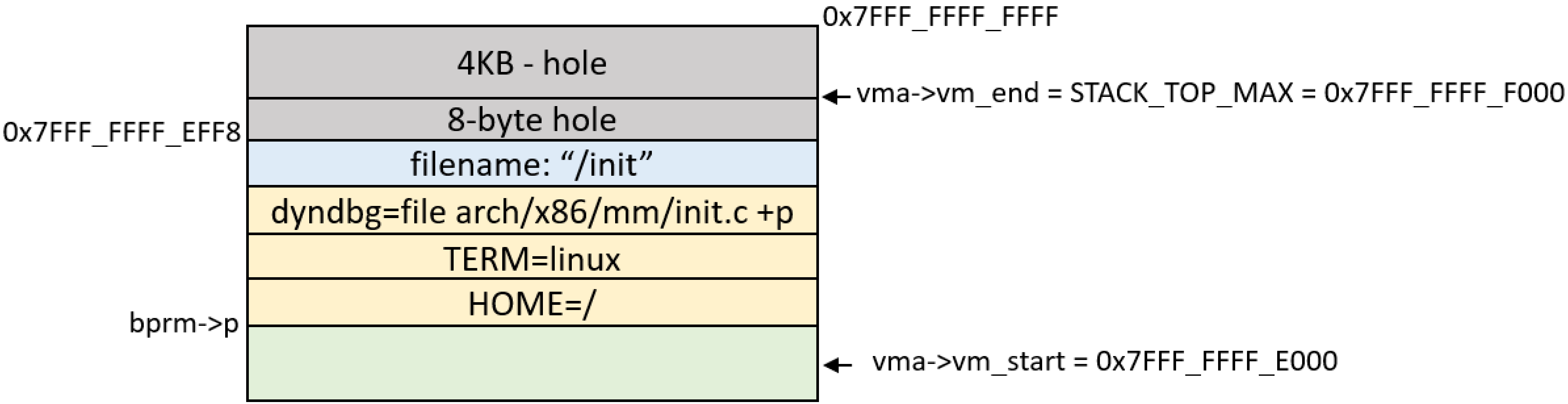
```
copy_string_kernel [fs/exec.c]
    |- page = get_arg_page(bprm, pos, 1);
      |- get_user_pages_remote
        |- __get_user_pages_remote
          |- __get_user_pages_locked
            |-   __get_user_pages
             |- page = follow_page_mask(vma, start, foll_flags, &ctx)
               Traverse all page tables to get PFN so that we can get
               the corresponding page sturct pointer
             if (!page) {
               |- ret = faultin_page(vma, start, &foll_flags, locked)
                 |- handle_mm_fault
               if (!ret)
                 retry follow_page_mask
             }
```



kernel_init -> run_init_process: init process (pid = 1) - Stack



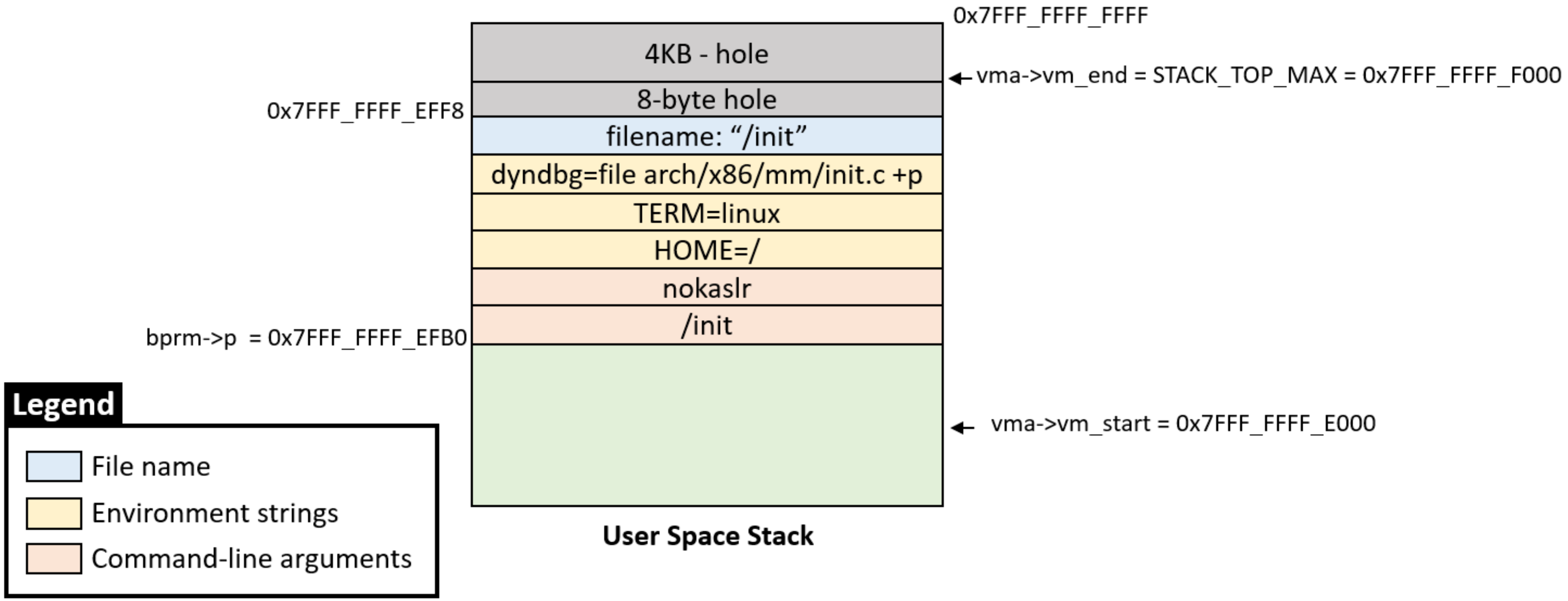
```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```


0x7FFF_FFFF_FFFF
4KB - hole
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000
8-byte hole
0x7FFF_FFFF_EFF8
filename: “/init”
dyndbg=file arch/x86/mm/init.c +p
TERM=linux
HOME=/
bprm->p
vma->vm_start = 0x7FFF_FFFF_E000

User Space Stack

kernel_init -> run_init_process: init process (pid = 1) - Stack

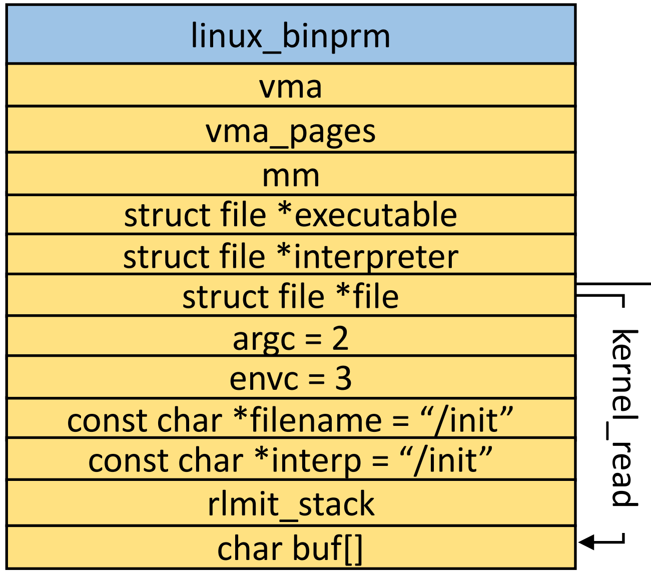
```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```



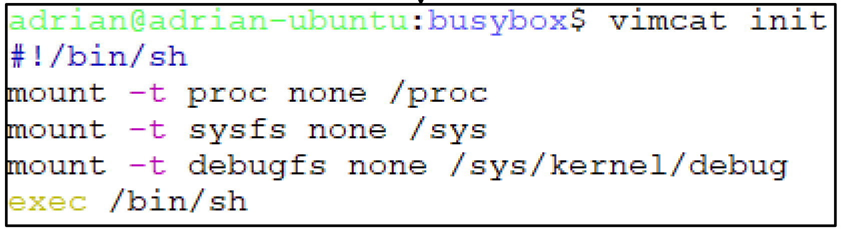
kernel_init -> run_init_process: init process (pid = 1) - Stack

```
[pid = 1, kernel_execve]
run_init_process [init/main.c]
  |- kernel_execve
    |- alloc_bprm
    |- bprm->argc = count_strings_kernel(argv)
    |- bprm->envc = count_strings_kernel(envp)
    |- bprm_stack_limits
    |- copy_string_kernel(bprm->filename, bprm)
    bprm->exec = bprm->p
    |- copy_strings_kernel(bprm->envc, envp, bprm)
    |- copy_strings_kernel(bprm->argc, argv, bprm)
    |- bprm_execve
```

```
bprm_execve [fs/exec.c]
  |- file = do_open_execat
    |- file = do_filp_open(fd, name, &open_exec_flags)
    return file
  bprm->file = file
  |- exec_binprm(bprm)
    for (depth = 0;; depth++) {
      if (depth > 5)
        return -ELOOP;

      |- search_binary_handler(bprm)
        |- prepare_binprm
          |- kernel_read
        |- load_elf_binary or load_script

      if (!bprm->interpreter)
        break;

      bprm->file = bprm->interpreter;
      bprm->interpreter = NULL;
    }
```



linux_binprm
vma
vma_pages
mm
struct file *executable
struct file *interpreter
struct file *file
argc = 2
envc = 3
const char *filename = “/init”
const char *interp = “/init”
rlmit_stack
char buf[]

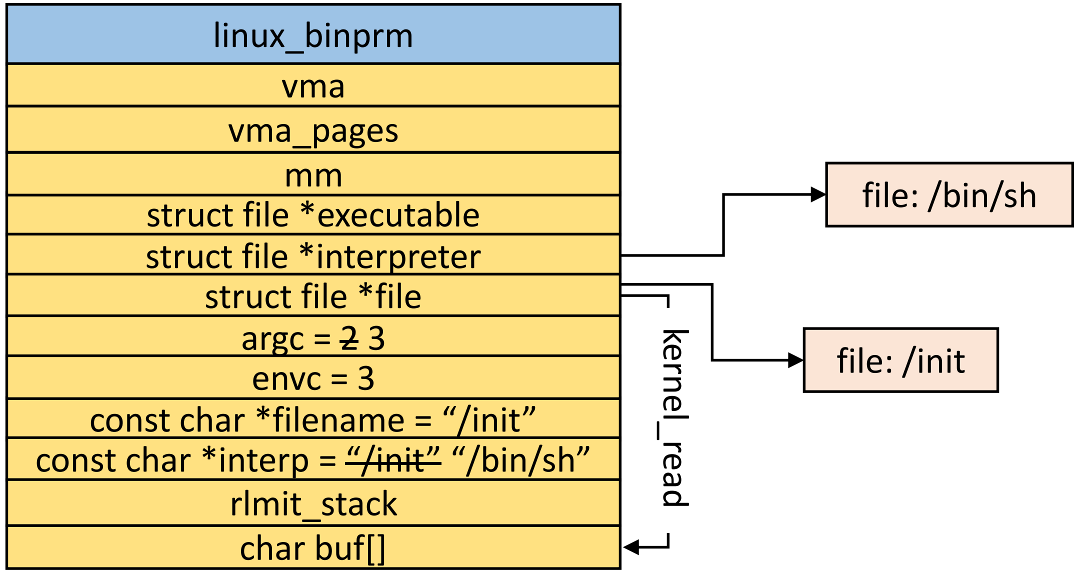


```
adrian@adrian-ubuntu:busybox$ vimcat init
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
mount -t debugfs none /sys/kernel/debug
exec /bin/sh
```

```
(gdb) p bprm->buf
$26 = "#!/bin/sh\000mount -t proc none /proc\nmount -t sysfs none /sys\nmount -t
 debugfs none /sys/kernel/debug\nexec /bin/sh\n", '\000' <repeats 142 times>
```

kernel_init -> run_init_process: init process (pid = 1) - Stack

```
load_script [fs/binfmt_script.c]
  Parse script interpreter /* Ex: #!/bin/sh */
  |- remove_arg_zero
  |- copy_string_kernel(bprm->interp, bprm)
  bprm->argc++
  Copy additional arguments if needed
  |- copy_string_kernel(i_name, bprm)
    /* Example: iname = "#!/bin/sh" */
  bprm->argc++
  |- bprm_change_interp(i_name, bprm)
    |- bprm->interp = kstrdup(interp, GFP_KERNEL)
  |- file = open_exec(i_name)
  bprm->interpreter = file
```



linux_binprm

vma

vma_pages

mm

struct file *executable

struct file *interpreter

struct file *file

argc = ~~2~~ 3

envc = 3

const char *filename = “/init”

const char *interp = ~~“/init”~~ “/bin/sh”

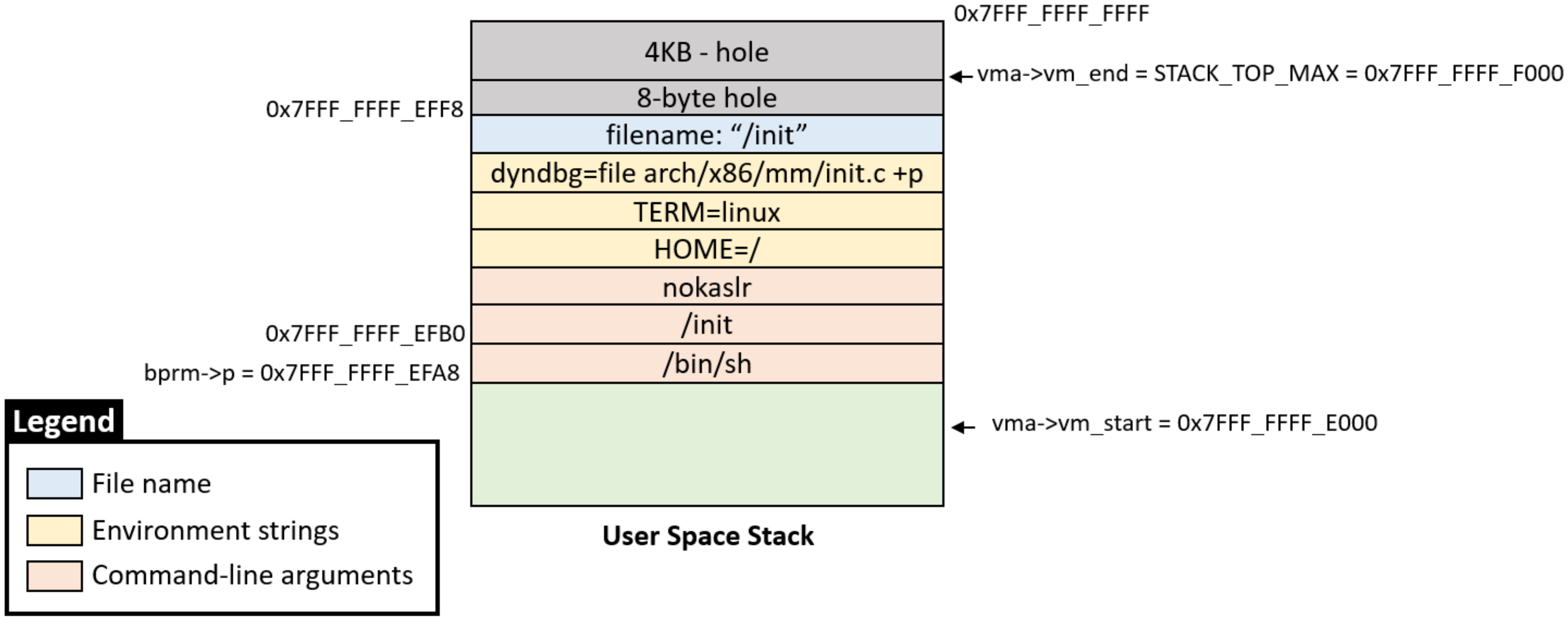
rlmit_stack

char buf[]

file: /bin/sh

file: /init

kernel_read



0x7FFF_FFFF_FFFF

4KB - hole

vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000

0x7FFF_FFFF_EFF8

8-byte hole

filename: “/init”

dyndbg=file arch/x86/mm/init.c +p

TERM=linux

HOME=/

nokaslr

0x7FFF_FFFF_EFB0

/init

bprm->p = 0x7FFF_FFFF_EFA8

/bin/sh

vma->vm_start = 0x7FFF_FFFF_E000

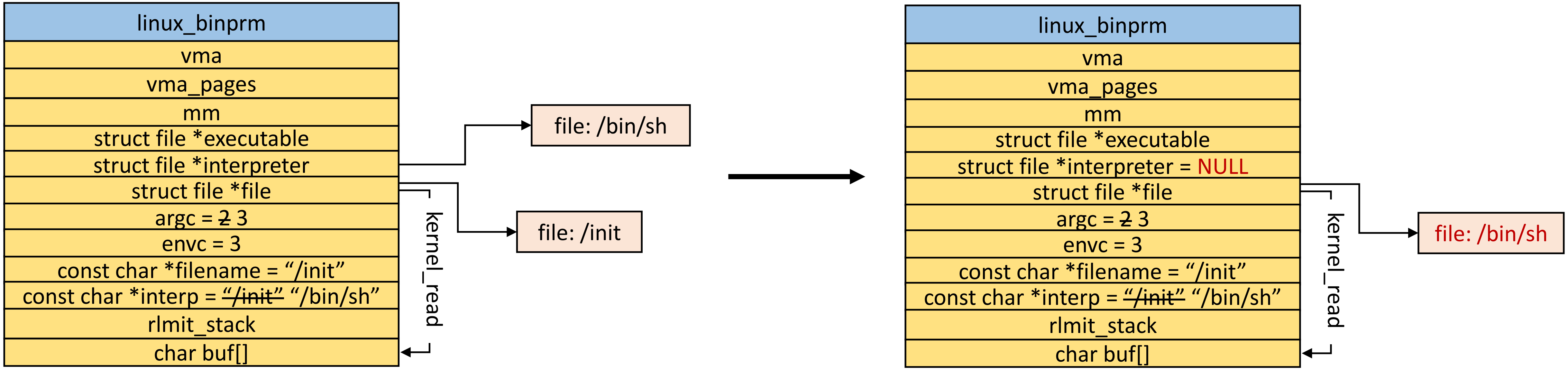
User Space Stack

Legend

- File name
- Environment strings
- Command-line arguments

kernel_init -> run_init_process: init process (pid = 1)

```
bprm_execve [fs/exec.c]
  |- file = do_open_execat
    |- file = do_filp_open(fd, name, &open_exec_flags)
    return file
  bprm->file = file
  |- exec_binprm(bprm)
    for (depth = 0;; depth++) {
      if (depth > 5)
        return -ELOOP;

      |- search_binary_handler(bprm)
        |- prepare_binprm
          |- kernel_read
        |- load_elf_binary or load_script

      if (!bprm->interpreter)
        break;

      bprm->file = bprm->interpreter;
      bprm->interpreter = NULL;
    }
```



linux_binprm

vma

vma_pages

mm

struct file *executable

struct file *interpreter

struct file *file

argc = ~~2~~ 3

envc = 3

const char *filename = “/init”

const char *interp = ~~“/init”~~ “/bin/sh”

rlmit_stack

char buf[]

linux_binprm

vma

vma_pages

mm

struct file *executable

struct file *interpreter = NULL

struct file *file

argc = ~~2~~ 3

envc = 3

const char *filename = “/init”

const char *interp = ~~“/init”~~ “/bin/sh”

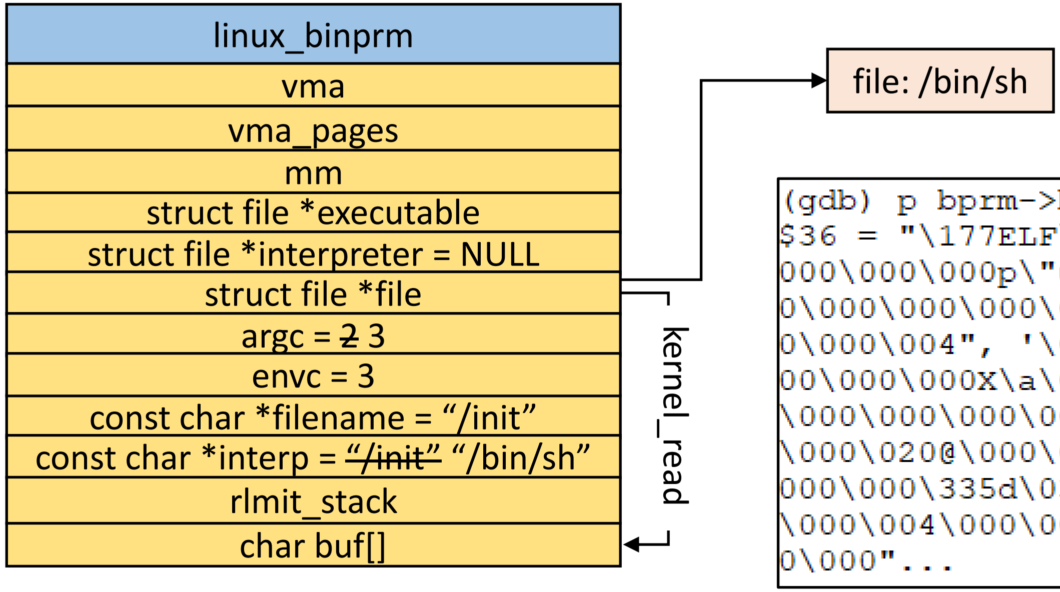
rlmit_stack

char buf[]

kernel_init -> run_init_process: init process (pid = 1)

```
bprm_execve [fs/exec.c]
  |- file = do_open_execat
    |- file = do_filp_open(fd, name, &open_exec_flags)
    return file
  bprm->file = file
  |- exec_binprm(bprm)
    for (depth = 0;; depth++) {
      if (depth > 5)
        return -ELOOP;

      |- search_binary_handler(bprm)
        |- prepare_binprm
          |- kernel_read
        |- load_elf_binary or load_script

      if (!bprm->interpreter)
        break;

      bprm->file = bprm->interpreter;
      bprm->interpreter = NULL;
    }
```

depth = 1



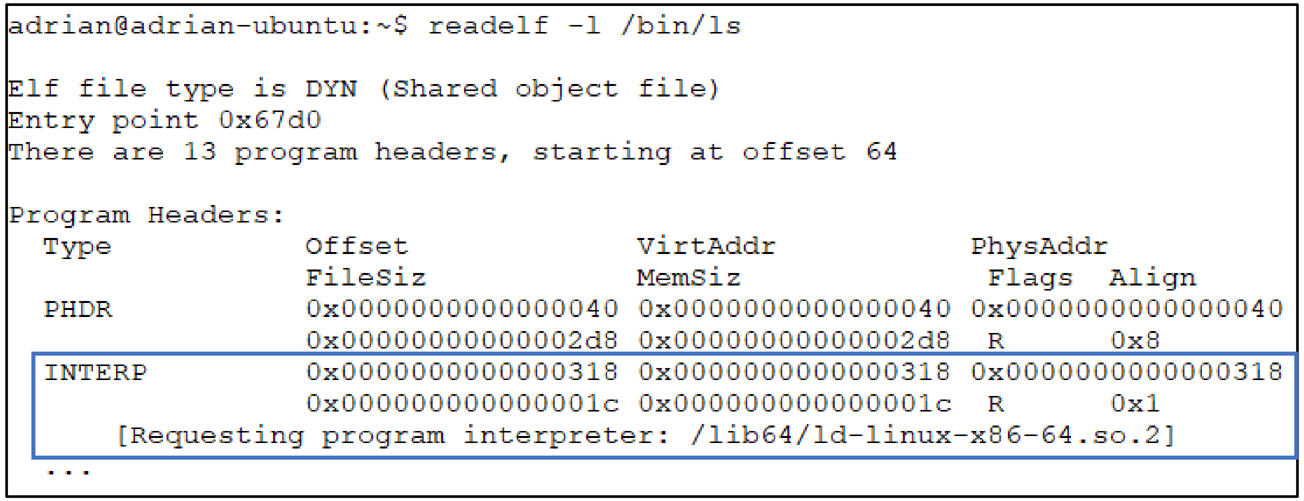
```
(gdb) p bprm->buf
$36 = "\177ELF\002\001\001\003\000\000\000\000\000\000\000\000\002\000>\000\001\
000\000\000p\"@\000\000\000\000\000@\000\000\000\000\000\000\000\300\002)\000\00
0\000\000\000\000\000\000\000@\000\070\000\n\000@\000\035\000\034\000\001\000\00
0\000\004", '\000' <repeats 13 times>, "@\000\000\000\000\000\000\000@\000\000\0
00\000\000X\a\000\000\000\000\000\000X\a\000\000\000\000\000\000\000\020\000\000
\000\000\000\000\001\000\000\000\005\000\000\000\000\020\000\000\000\000\000\000
\000\020@\000\000\000\000\000\000\020@\000\000\000\000\000\335d\037\000\000\000\
000\000\335d\037\000\000\000\000\000\000\020\000\000\000\000\000\000\001\000\000
\000\004\000\000\000\000\200\037\000\000\000\000\000\000\200_\000\000\000\000\00
0\000"...
```

kernel_init -> run_init_process: init process (pid = 1)

```
bprm_execve [fs/exec.c]
  |- file = do_open_execat
    |- file = do_filp_open(fd, name, &open_exec_flags)
    return file
  bprm->file = file
  |- exec_binprm(bprm)
    for (depth = 0;; depth++) {
      if (depth > 5)
        return -ELOOP;

      |- search_binary_handler(bprm)
        |- prepare_binprm
          |- kernel_read
        |- load_elf_binary or load_script

      if (!bprm->interpreter)
        break;

      bprm->file = bprm->interpreter;
      bprm->interpreter = NULL;
    }
```

depth = 1

```
load_elf_binary [fs/binfmt_elf.c]
  Parse ELF header and get program headers
  [Shared object file] Get/open interpreter from the program header
  Parse property program header - GNU_PROPERTY
  |- begin_new_exec
  |- setup_new_exec
  |- setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                               executable_stack)
    /* Finalize the stack vm_area_struct */
  Iterate each program header type 'PT_LOAD'
    |- elf_map
      |- vm_mmap
        |- vm_mmap_pgoff
  |- set_brk(elf_bss, elf_brk, bss_prot)
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- create_elf_tables
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```

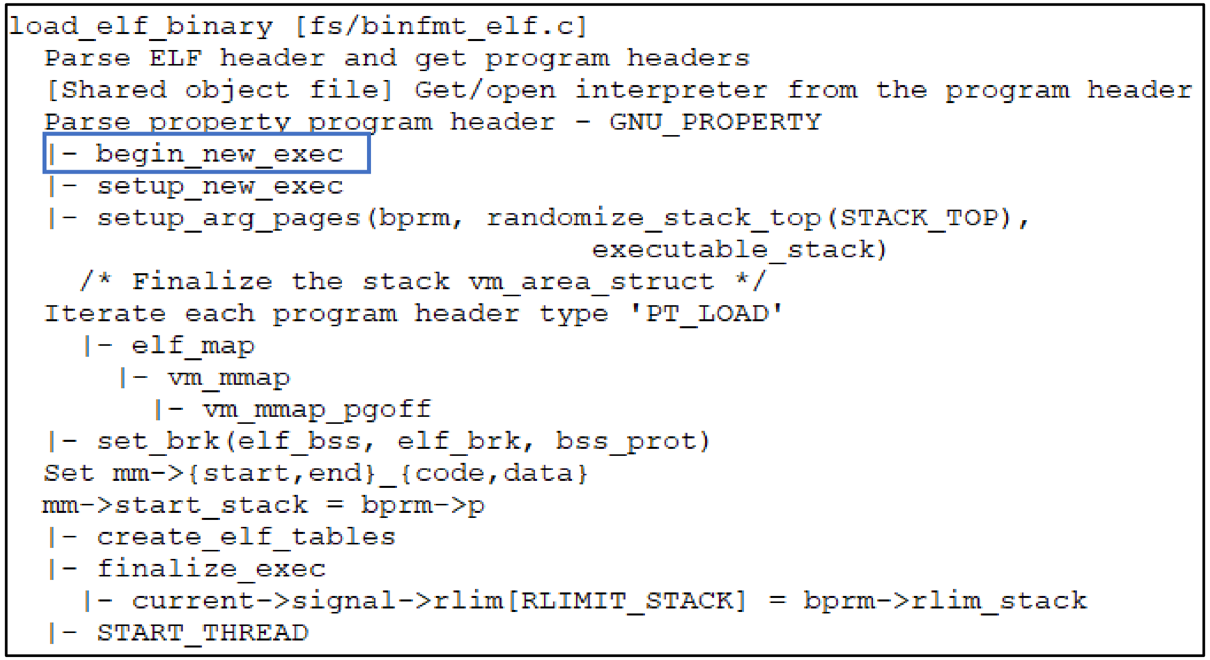


```
adrian@adrian-ubuntu:~$ readelf -l /bin/ls

Elf file type is DYN (Shared object file)
Entry point 0x67d0
There are 13 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  ...
```

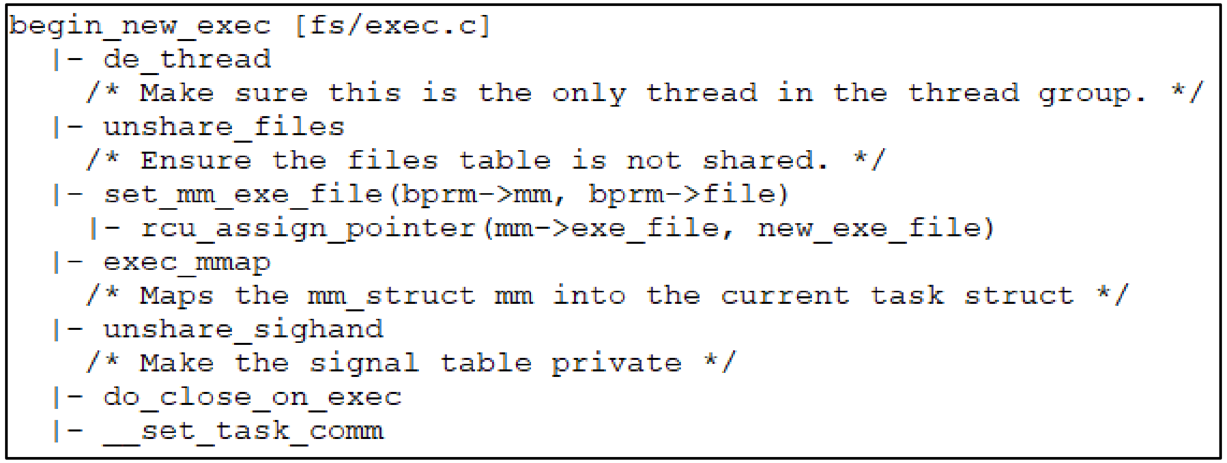
[Dynamic linking] ld-Linux.so: dynamic linker/loader

- Find and load the shared objects (shared libraries) needed by a program, prepare the program to run, and then run it
- `man ld-linux`

kernel_init -> run_init_process: init process (pid = 1)



```
load_elf_binary [fs/binfmt_elf.c]
  Parse ELF header and get program headers
  [Shared object file] Get/open interpreter from the program header
  Parse property program header - GNU_PROPERTY
  |- begin_new_exec
  |- setup_new_exec
  |- setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                              executable_stack)
    /* Finalize the stack vm_area_struct */
  Iterate each program header type 'PT_LOAD'
    |- elf_map
      |- vm_mmap
        |- vm_mmap_pgoff
  |- set_brk(elf_bss, elf_brk, bss_prot)
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- create_elf_tables
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```



```
begin_new_exec [fs/exec.c]
  |- de_thread
    /* Make sure this is the only thread in the thread group. */
  |- unshare_files
    /* Ensure the files table is not shared. */
  |- set_mm_exe_file(bprm->mm, bprm->file)
    |- rcu_assign_pointer(mm->exe_file, new_exe_file)
  |- exec_mmap
    /* Maps the mm_struct mm into the current task struct */
  |- unshare_sighand
    /* Make the signal table private */
  |- do_close_on_exec
  |- __set_task_comm
```

Description from `man execve`

de_thread(): All threads other than the calling thread are destroyed during an execve(). Mutexes, condition variables, and other pthreads objects are not preserved.

unshare_files(): The file descriptor table is unshared, undoing the effect of the CLONE_FILES flag of clone(2).

exec_mmap():

1. The program that is currently being run by the calling process to be replaced with a new program, with newly initialized stack, heap, and (initialized and uninitialized) data segments.
2. Memory mappings are not preserved.

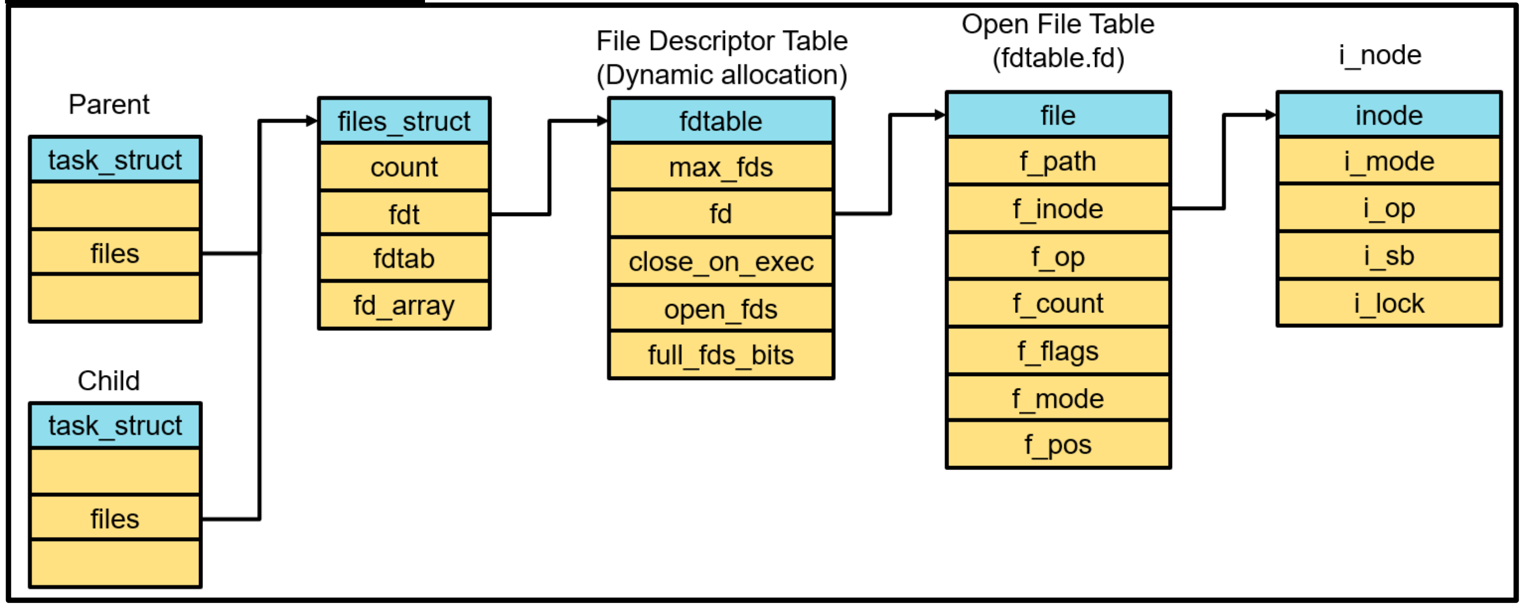
unshare_sighand(): The signal dispositions is unshared, undoing the effect of the CLONE_SIGHAND flag of clone(2) – **Not from `man execve`**

do_close_on_exec(): By default, file descriptors remain open across an execve(). File descriptors that are marked close-on-exec are closed.

kernel_init -> run_init_process: init process (pid = 1)

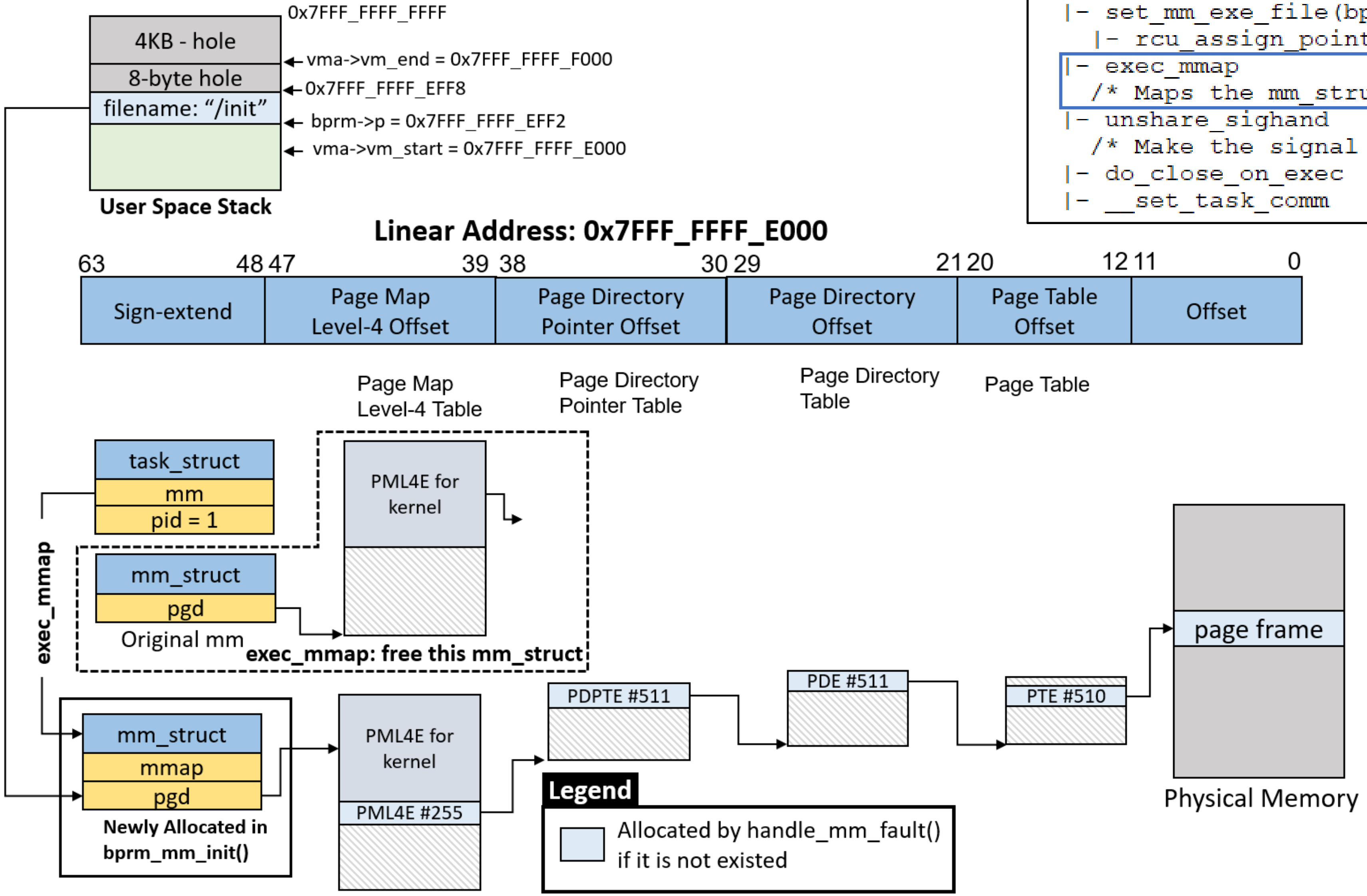
```
begin_new_exec [fs/exec.c]
  |- de_thread
    /* Make sure this is the only thread in the thread group. */
  |- unshare_files
    /* Ensure the files table is not shared. */
  |- set_mm_exe_file(bprm->mm, bprm->file)
    |- rcu_assign_pointer(mm->exe_file, new_exe_file)
  |- exec_mmap
    /* Maps the mm_struct mm into the current task struct */
  |- unshare_sighand
    /* Make the signal table private */
  |- do_close_on_exec
  |- __set_task_comm
```

CLONE_FILES flag is set



kernel_init -> run_init_process: init process (pid = 1)

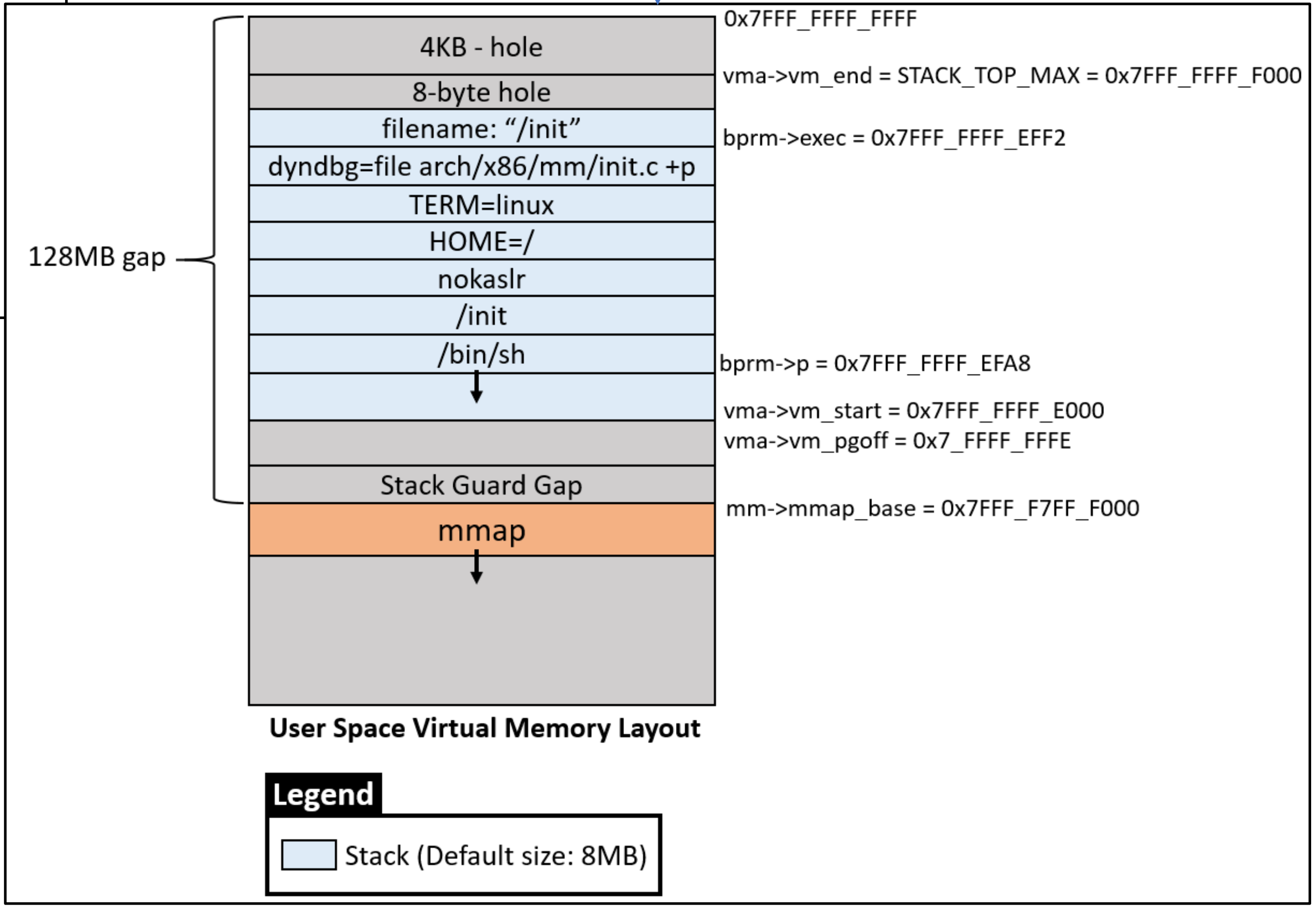
```
begin_new_exec [fs/exec.c]
  |- de_thread
    /* Make sure this is the only thread in the thread group. */
  |- unshare_files
    /* Ensure the files table is not shared. */
  |- set_mm_exe_file(bprm->mm, bprm->file)
    |- rcu_assign_pointer(mm->exe_file, new_exe_file)
  |- exec_mmap
    /* Maps the mm_struct mm into the current task struct */
  |- unshare_sighand
    /* Make the signal table private */
  |- do_close_on_exec
  |- __set_task_comm
```



load_elf_binary()->setup_new_exec()

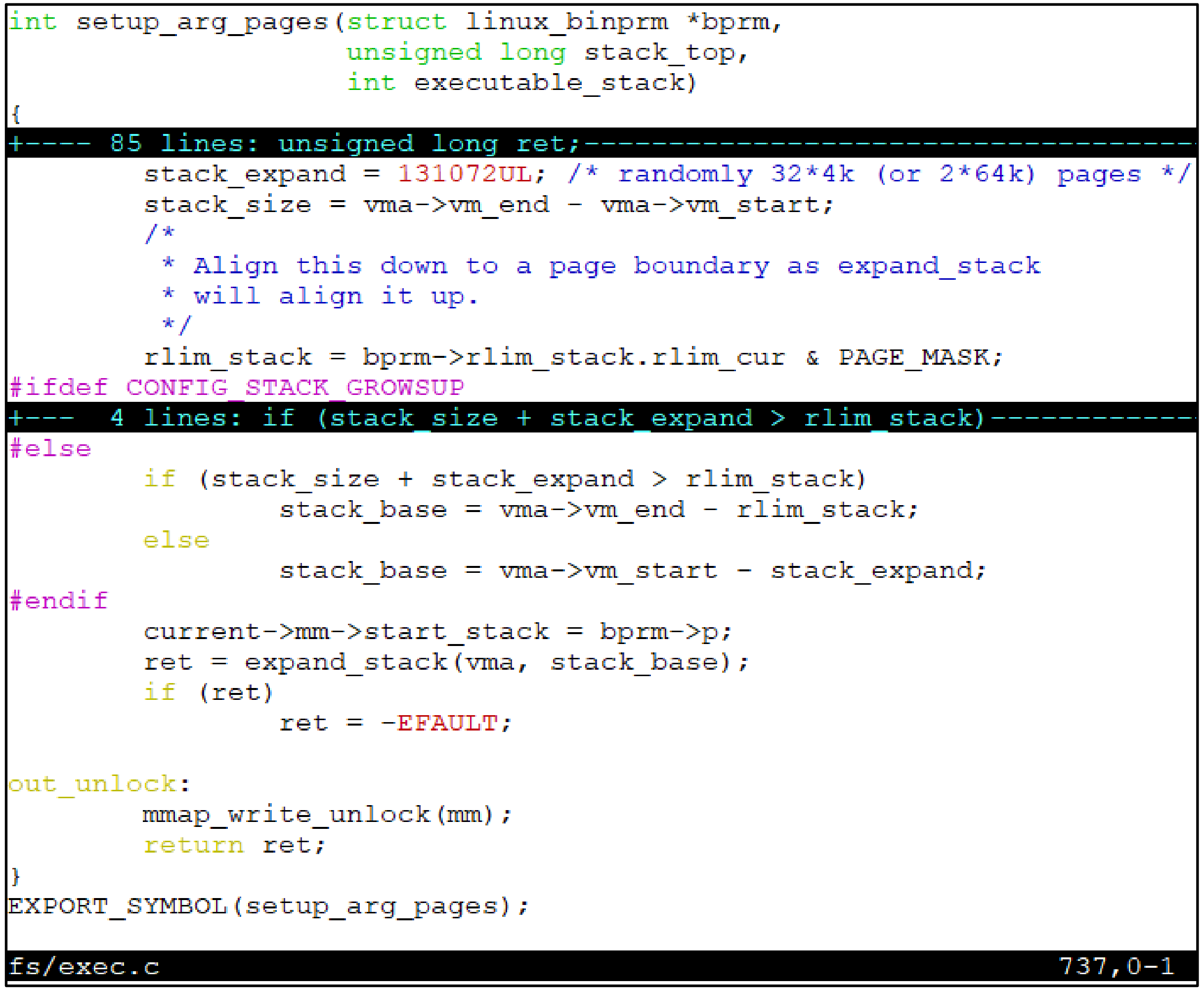
```
load_elf_binary [fs/binfmt_elf.c]
  Parse ELF header and get program headers
  [Shared object file] Get/open interpreter from the program header
  Parse property program header - GNU_PROPERTY
  |- begin_new_exec
  |- setup_new_exec
  |- setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                                 executable_stack)
    /* Finalize the stack vm_area_struct */
  Iterate each program header type 'PT_LOAD'
    |- elf_map
      |- vm_mmap
        |- vm_mmap_pgoff
  |- set_brk(elf_bss, elf_brk, bss_prot)
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- create_elf_tables
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```

```
setup_new_exec [fs/exec.c]
  |- arch_pick_mmap_layout
    /* mmap is NOT legacy */
    mm->get_unmapped_area = arch_get_unmapped_area_topdown
    |- arch_pick_mmap_base
      |- mmap_base
  me->mm->task_size = TASK_SIZE
```


0x7FFF_FFFF_FFFF
4KB - hole
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000
8-byte hole
filename: “/init”
bprm->exec = 0x7FFF_FFFF_EFF2
dyndbg=file arch/x86/mm/init.c +p
TERM=linux
HOME=/
128MB gap
nokaslr
/init
/bin/sh
bprm->p = 0x7FFF_FFFF_EFA8
vma->vm_start = 0x7FFF_FFFF_E000
vma->vm_pgoff = 0x7_FFFF_FFFE
Stack Guard Gap
mm->mmap_base = 0x7FFF_F7FF_F000
mmap
User Space Virtual Memory Layout
Legend
Stack (Default size: 8MB)

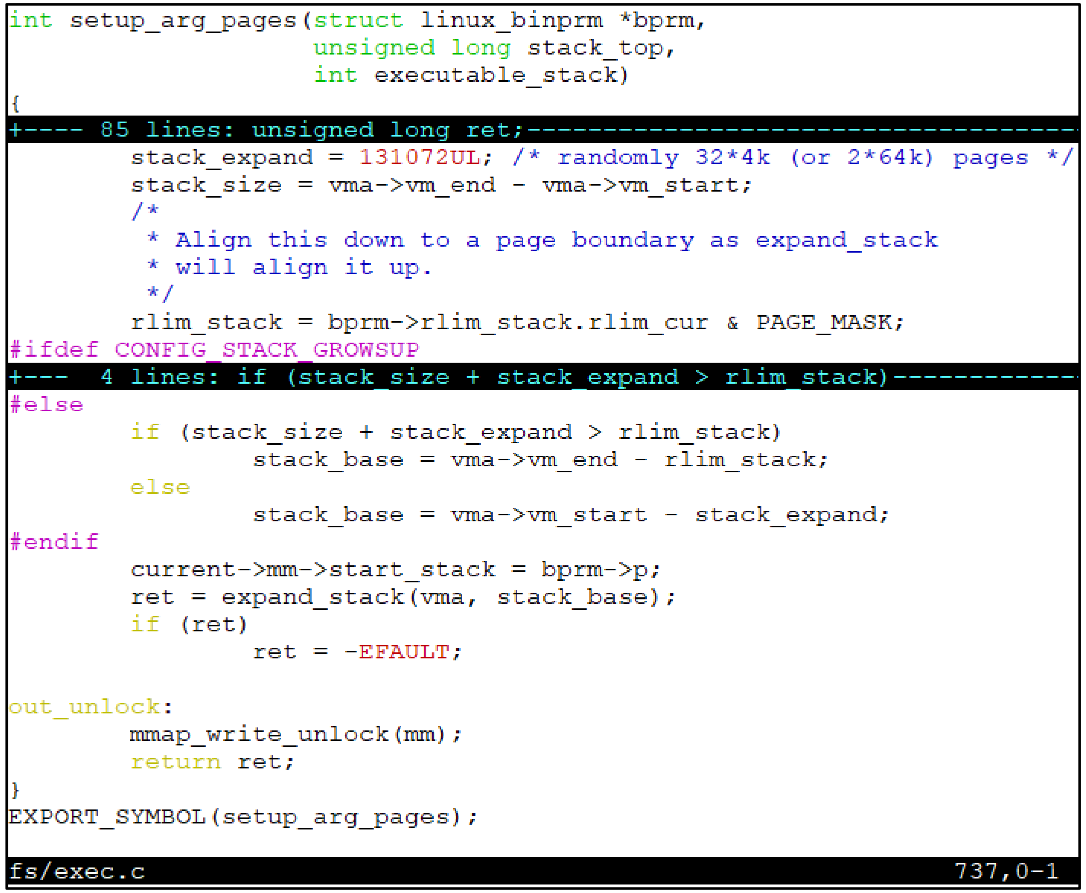
kernel_init -> run_init_process: init process (pid = 1)

```
load_elf_binary [fs/binfmt_elf.c]
  Parse ELF header and get program headers
  [Shared object file] Get/open interpreter from the program header
  Parse property program header - GNU_PROPERTY
  |- begin_new_exec
  |- setup_new_exec
  |- setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                           executable_stack)
    /* Finalize the stack vm_area_struct */
  Iterate each program header type 'PT_LOAD'
    |- elf_map
      |- vm_mmap
        |- vm_mmap_pgoff
  |- set_brk(elf_bss, elf_brk, bss_prot)
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- create_elf_tables
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```

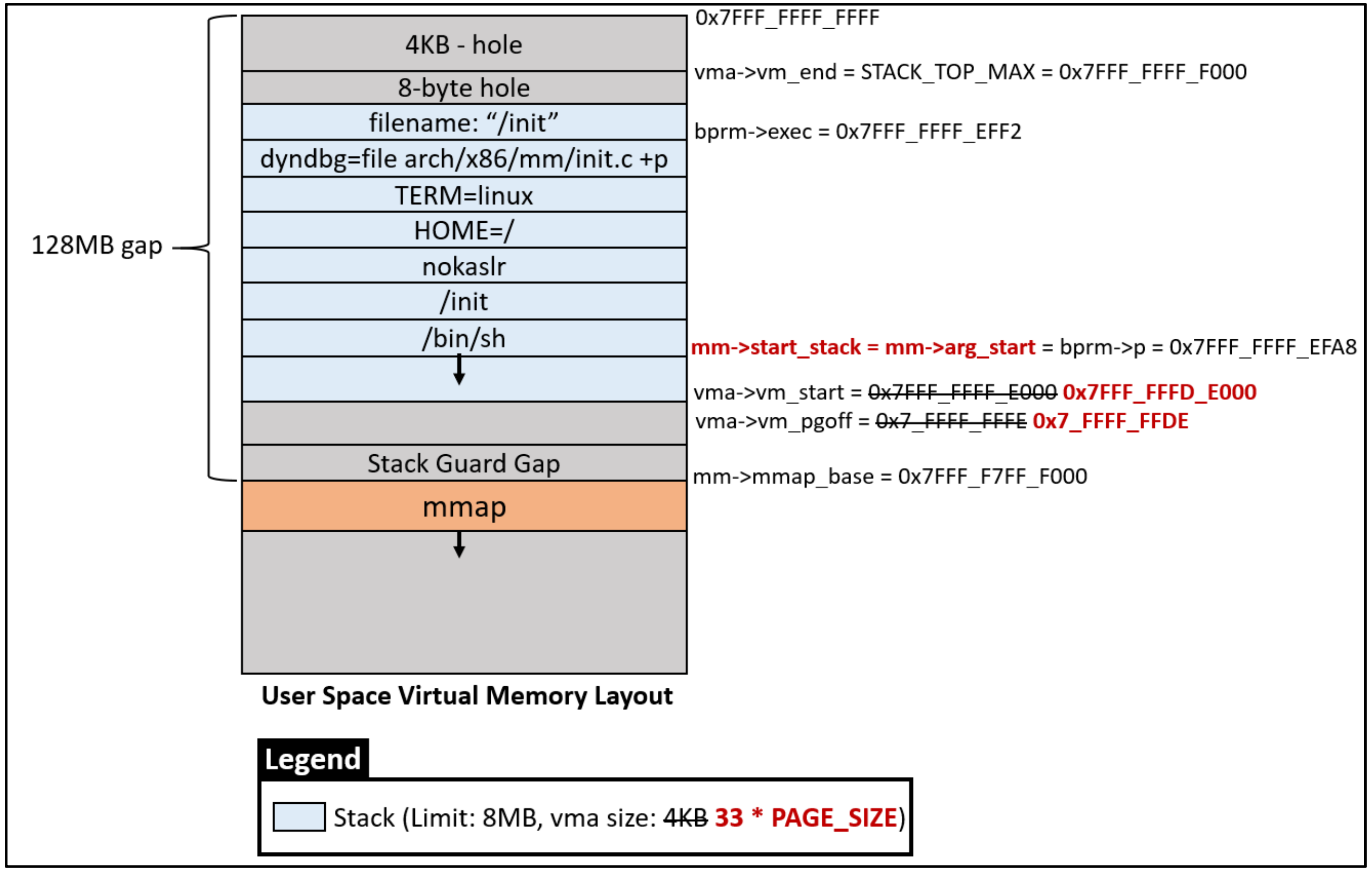


```
int setup_arg_pages(struct linux_binprm *bprm,
                    unsigned long stack_top,
                    int executable_stack)
{
+---- 85 lines: unsigned long ret;-----------------------------------------
        stack_expand = 131072UL; /* randomly 32*4k (or 2*64k) pages */
        stack_size = vma->vm_end - vma->vm_start;
        /*
         * Align this down to a page boundary as expand_stack
         * will align it up.
         */
        rlim_stack = bprm->rlim_stack.rlim_cur & PAGE_MASK;
#ifdef CONFIG_STACK_GROWSUP
+---  4 lines: if (stack_size + stack_expand > rlim_stack)-----------
#else
        if (stack_size + stack_expand > rlim_stack)
                stack_base = vma->vm_end - rlim_stack;
        else
                stack_base = vma->vm_start - stack_expand;
#endif
        current->mm->start_stack = bprm->p;
        ret = expand_stack(vma, stack_base);
        if (ret)
                ret = -EFAULT;

out_unlock:
        mmap_write_unlock(mm);
        return ret;
}
EXPORT_SYMBOL(setup_arg_pages);
fs/exec.c                                                     737,0-1
```

load_elf_binary()->setup_arg_pages()



```
int setup_arg_pages(struct linux_binprm *bprm,
                    unsigned long stack_top,
                    int executable_stack)
{
+---- 85 lines: unsigned long ret;---------------------------------------
        stack_expand = 131072UL; /* randomly 32*4k (or 2*64k) pages */
        stack_size = vma->vm_end - vma->vm_start;
        /*
         * Align this down to a page boundary as expand_stack
         * will align it up.
         */
        rlim_stack = bprm->rlim_stack.rlim_cur & PAGE_MASK;
#ifdef CONFIG_STACK_GROWSUP
+---  4 lines: if (stack_size + stack_expand > rlim_stack)------------
#else
        if (stack_size + stack_expand > rlim_stack)
                stack_base = vma->vm_end - rlim_stack;
        else
                stack_base = vma->vm_start - stack_expand;
#endif
        current->mm->start_stack = bprm->p;
        ret = expand_stack(vma, stack_base);
        if (ret)
                ret = -EFAULT;

out_unlock:
        mmap_write_unlock(mm);
        return ret;
}
EXPORT_SYMBOL(setup_arg_pages);
fs/exec.c                                                        737,0-1
```

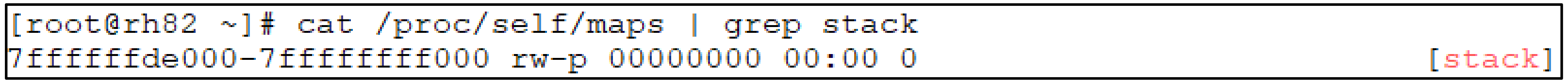


4KB - hole
8-byte hole
filename: “/init”
dyndbg=file arch/x86/mm/init.c +p
TERM=linux
HOME=/
nokaslr
/init
/bin/sh
Stack Guard Gap
mmap
128MB gap
0x7FFF_FFFF_FFFF
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000
bprm->exec = 0x7FFF_FFFF_EFF2
mm->start_stack = mm->arg_start = bprm->p = 0x7FFF_FFFF_EFA8
vma->vm_start = ~~0x7FFF_FFFF_E000~~ 0x7FFF_FFFD_E000
vma->vm_pgoff = ~~0x7_FFFF_FFFE~~ 0x7_FFFF_FFDE
mm->mmap_base = 0x7FFF_F7FF_F000

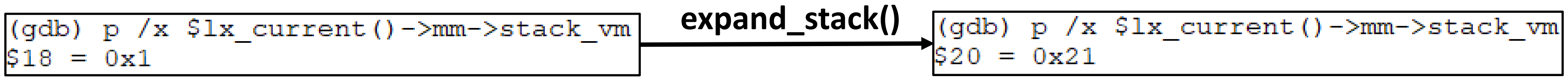
User Space Virtual Memory Layout

Legend

□ Stack (Limit: 8MB, vma size: ~~4KB~~ **33 * PAGE_SIZE**)



```
[root@rh82 ~]# cat /proc/self/maps | grep stack
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
```

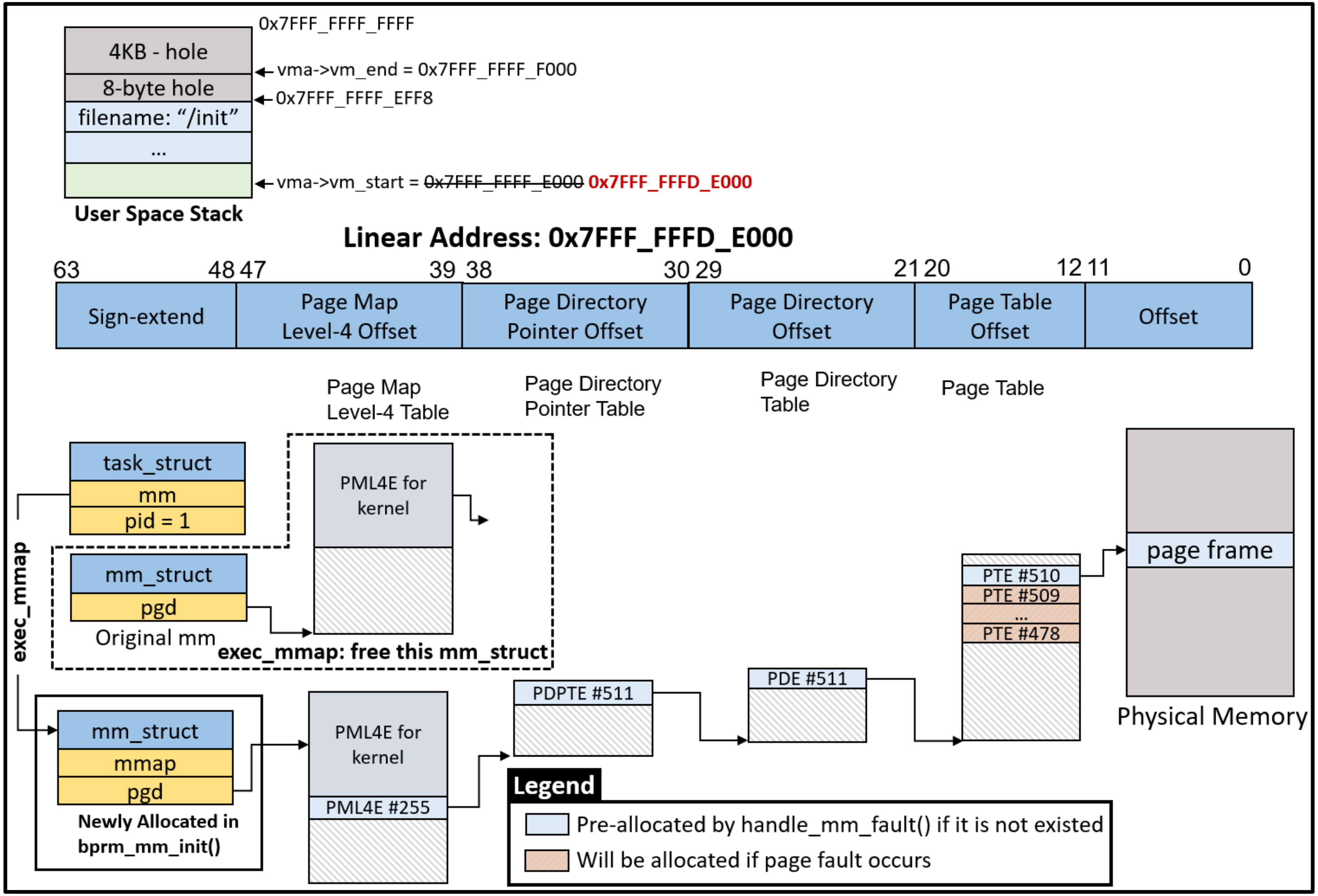


```
(gdb) p /x $lx_current()->mm->stack_vm
$18 = 0x1
```

expand_stack()

```
(gdb) p /x $lx_current()->mm->stack_vm
$20 = 0x21
```

load_elf_binary()->setup_arg_pages()



0x7FFF_FFFF_FFFF

4KB - hole

← vma->vm_end = 0x7FFF_FFFF_F000

8-byte hole

← 0x7FFF_FFFF_EFF8

filename: "/init"

...

← vma->vm_start = ~~0x7FFF_FFFF_E000~~ 0x7FFF_FFFD_E000

User Space Stack

Linear Address: 0x7FFF_FFFD_E000

63 – 48	47 – 39	38 – 30	29 – 21	20 – 12	11 – 0
Sign-extend	Page Map Level-4 Offset	Page Directory Pointer Offset	Page Directory Offset	Page Table Offset	Offset

Page Map Level-4 Table

Page Directory Pointer Table

Page Directory Table

Page Table

task_struct
mm
pid = 1

exec_mmap

mm_struct
pgd

Original mm

PML4E for kernel

exec_mmap: free this mm_struct

mm_struct
mmap
pgd

Newly Allocated in bprm_mm_init()

PML4E for kernel

PML4E #255

PDPTE #511

PDE #511

PTE #510
PTE #509
...
PTE #478

page frame

Physical Memory

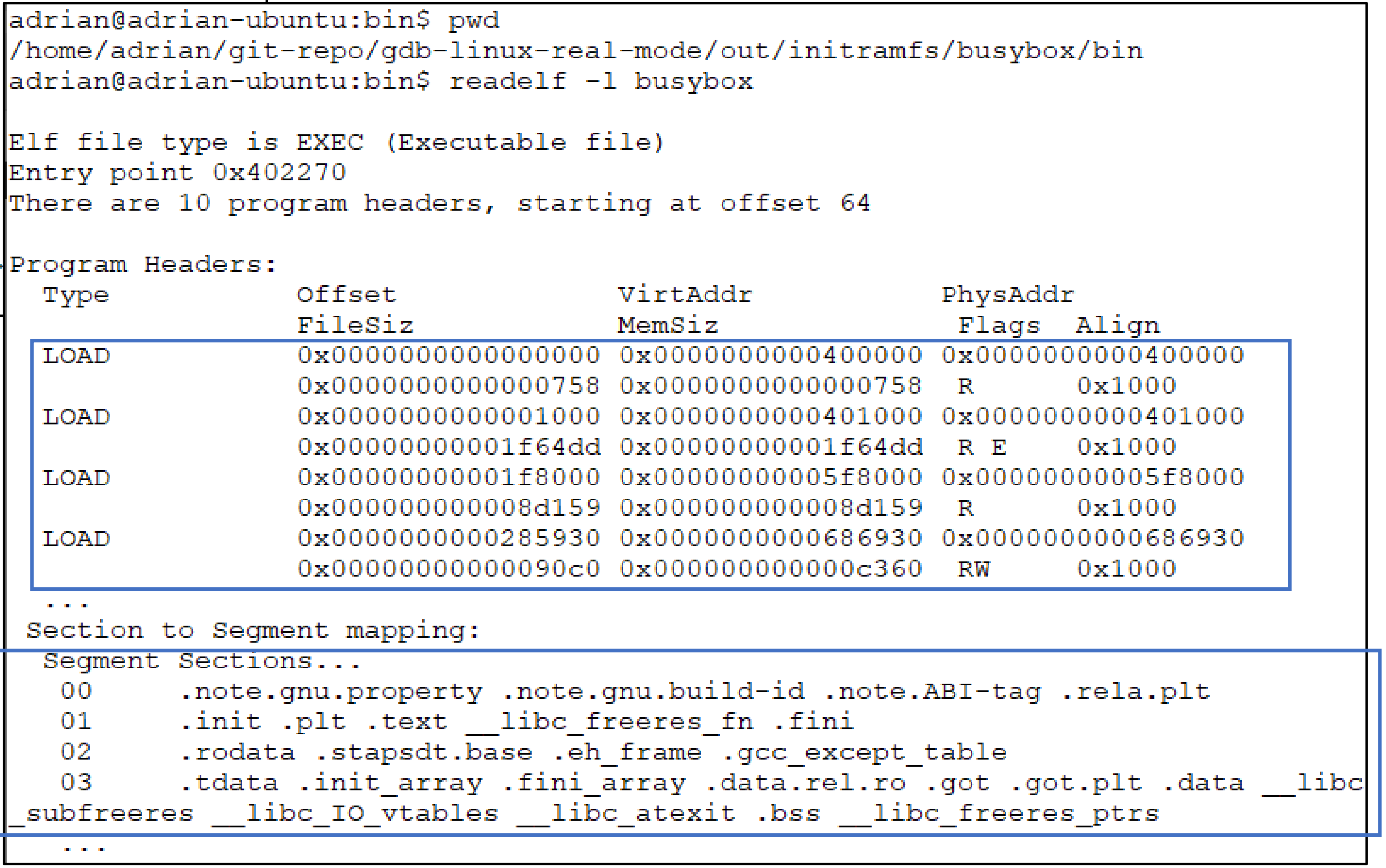
Legend

🟦 Pre-allocated by handle_mm_fault() if it is not existed

🟧 Will be allocated if page fault occurs

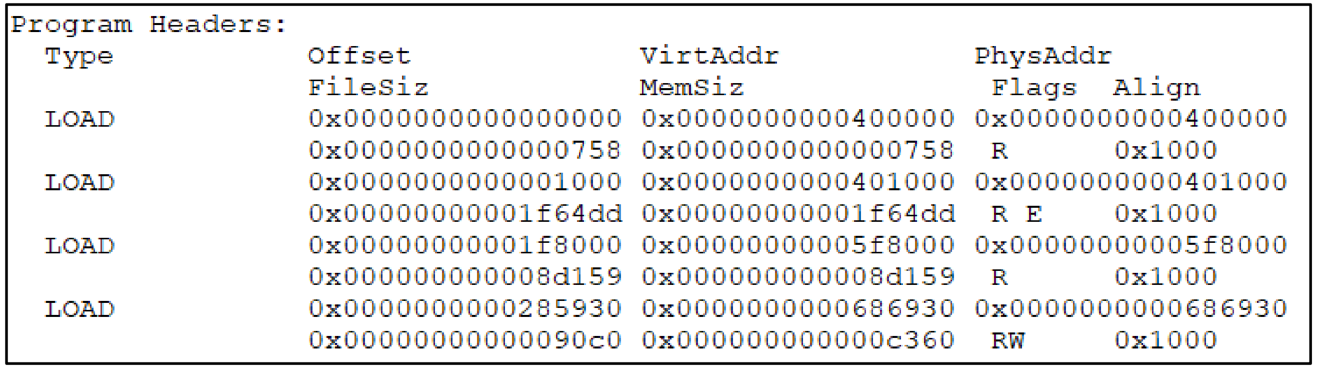
load_elf_binary(): load PT_LOAD program headers

```
load_elf_binary [fs/binfmt_elf.c]
  Parse ELF header and get program headers
  [Shared object file] Get/open interpreter from the program header
  Parse property program header - GNU_PROPERTY
  |- begin_new_exec
  |- setup_new_exec
  |- setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                              executable_stack)
    /* Finalize the stack vm_area_struct */
  Iterate each program header type 'PT_LOAD'
    |- elf_map
      |- vm_mmap
        |- vm_mmap_pgoff
  |- set_brk(elf_bss, elf_brk, bss_prot)
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- create_elf_tables
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->
  |- START_THREAD
```

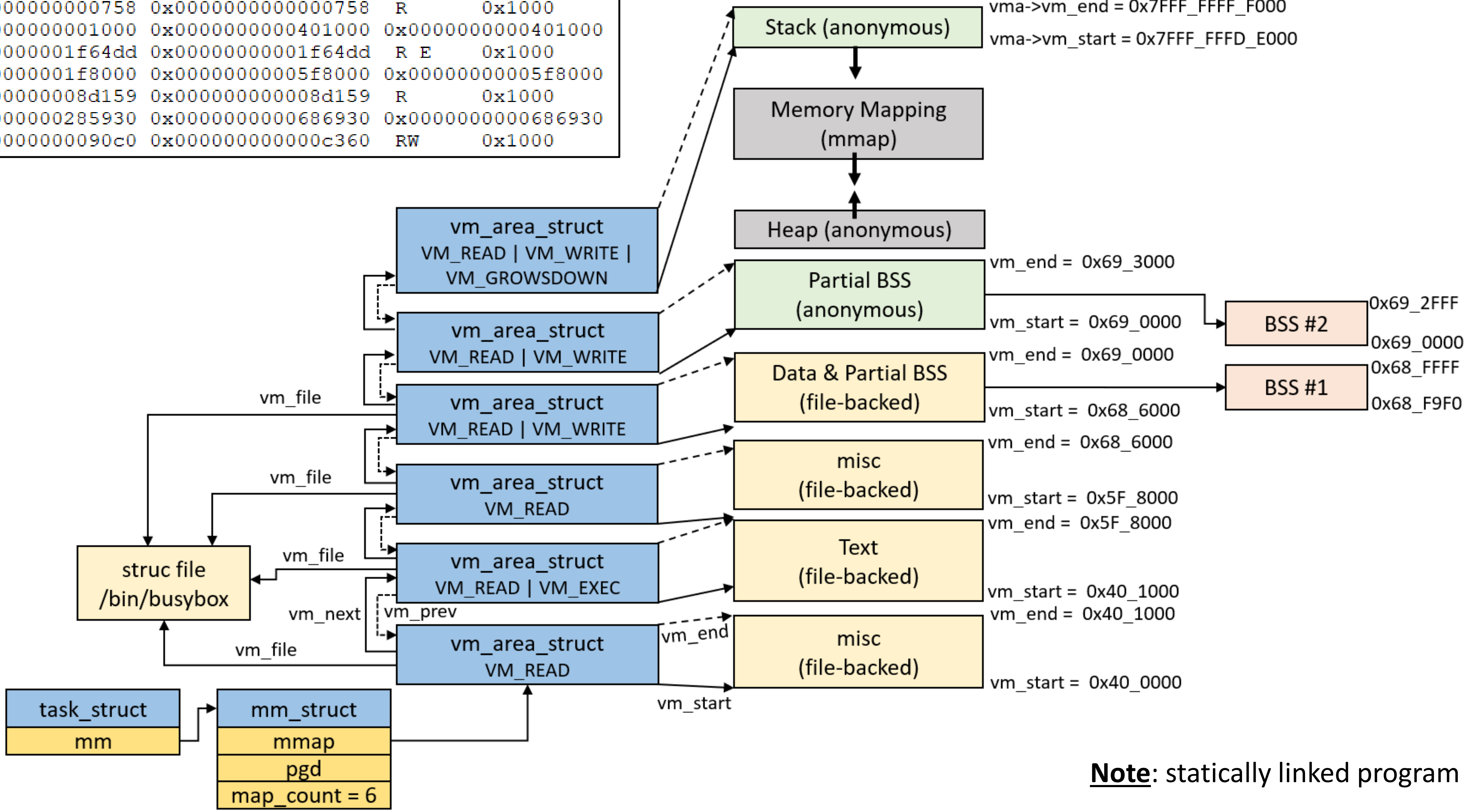


```
adrian@adrian-ubuntu:bin$ pwd
/home/adrian/git-repo/gdb-linux-real-mode/out/initramfs/busybox/bin
adrian@adrian-ubuntu:bin$ readelf -l busybox

Elf file type is EXEC (Executable file)
Entry point 0x402270
There are 10 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000000758 0x0000000000000758  R      0x1000
  LOAD           0x0000000000001000 0x0000000000401000 0x0000000000401000
                 0x00000000001f64dd 0x00000000001f64dd  R E    0x1000
  LOAD           0x00000000001f8000 0x00000000005f8000 0x00000000005f8000
                 0x000000000008d159 0x000000000008d159  R      0x1000
  LOAD           0x0000000000285930 0x0000000000686930 0x0000000000686930
                 0x00000000000090c0 0x000000000000c360  RW     0x1000
  ...
 Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.property .note.gnu.build-id .note.ABI-tag .rela.plt
   01     .init .plt .text __libc_freeres_fn .fini
   02     .rodata .stapsdt.base .eh_frame .gcc_except_table
   03     .tdata .init_array .fini_array .data.rel.ro .got .got.plt .data __libc
_subfreeres __libc_IO_vtables __libc_atexit .bss __libc_freeres_ptrs
   ...
```

load_elf_binary(): load PT_LOAD program headers



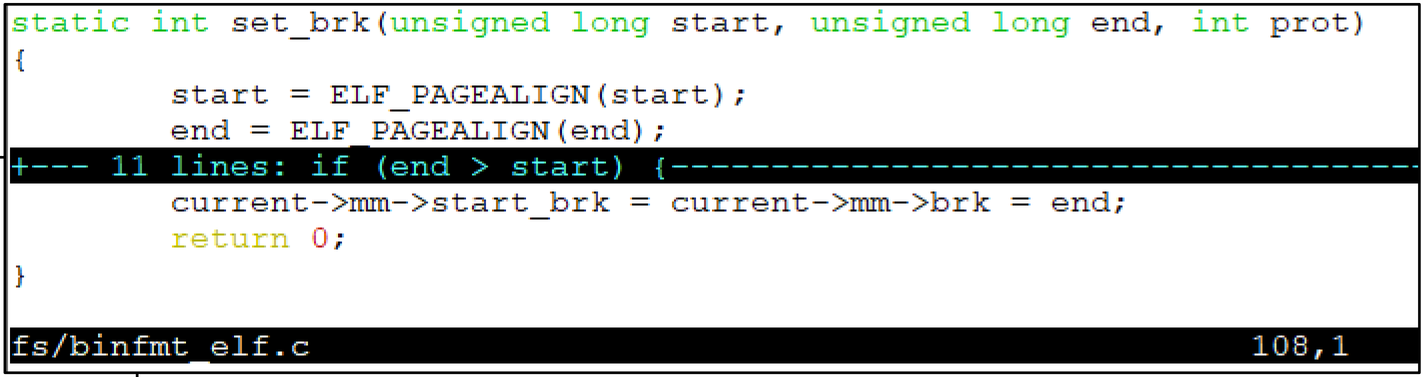
```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000000758 0x0000000000000758  R      0x1000
  LOAD           0x0000000000001000 0x0000000000401000 0x0000000000401000
                 0x00000000001f64dd 0x00000000001f64dd  R E    0x1000
  LOAD           0x00000000001f8000 0x00000000005f8000 0x00000000005f8000
                 0x000000000008d159 0x000000000008d159  R      0x1000
  LOAD           0x0000000000285930 0x0000000000686930 0x0000000000686930
                 0x00000000000090c0 0x000000000000c360  RW     0x1000
```



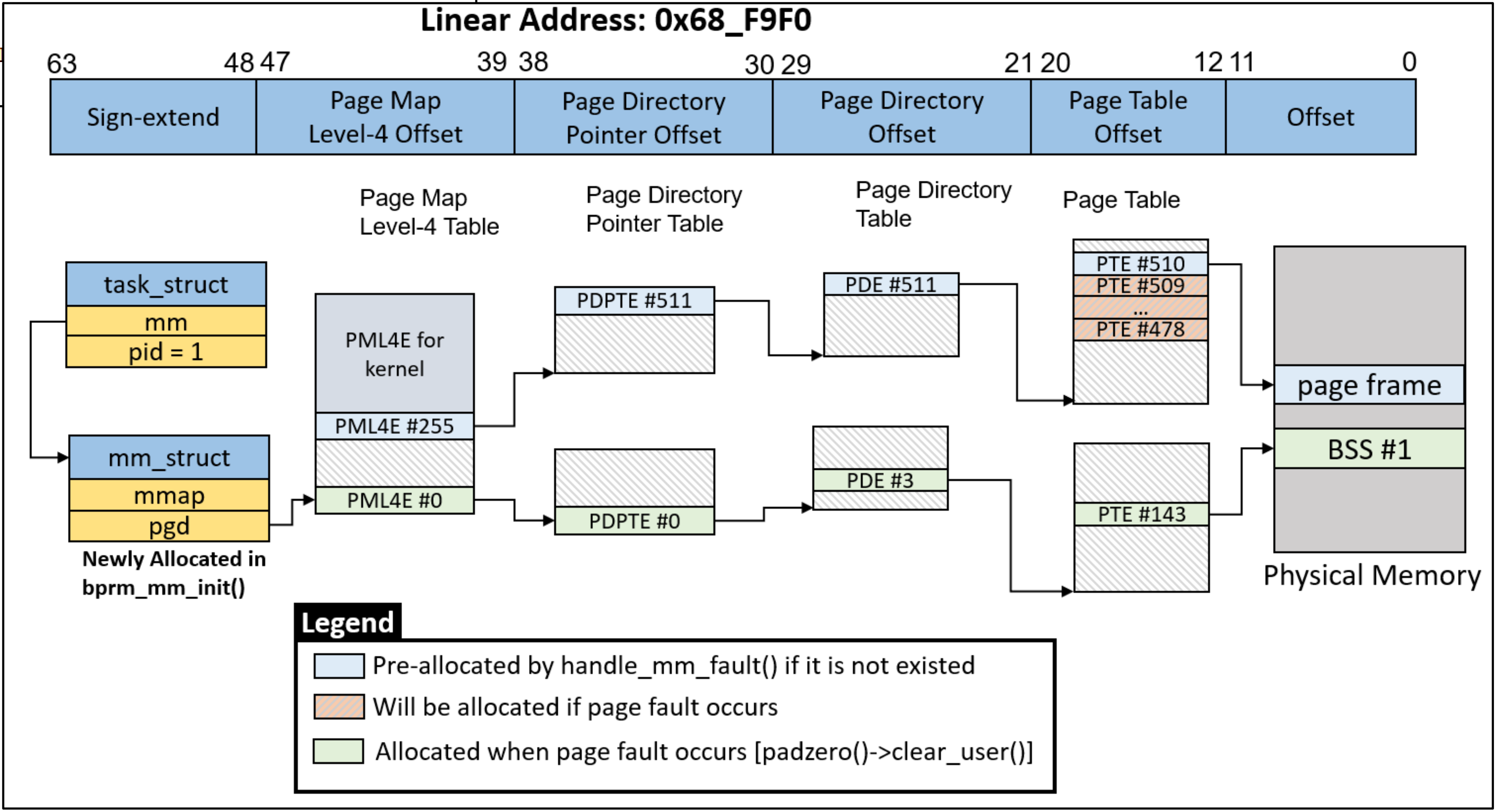
Note: statically linked program

set_brk() & padzero()

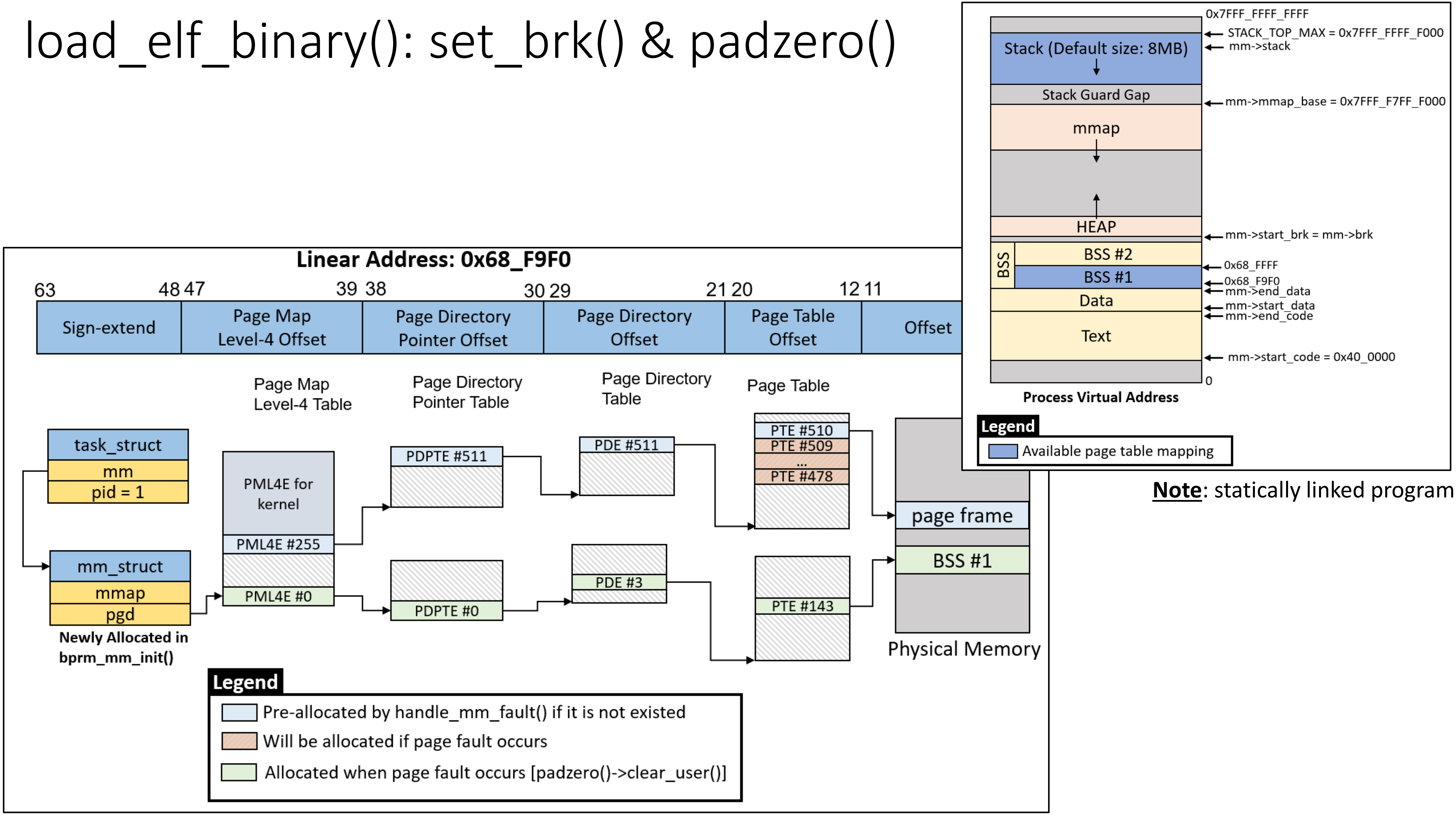
```
load_elf_binary [fs/binfmt_elf.c]
  ...
  |- set_brk(elf_bss, elf_brk, bss_prot)
  |- padzero(elf_bss)
  |- create_elf_tables
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- finalize_exec
    |- current->signal->rlim[RLI
  |- START_THREAD
```



```
static int set_brk(unsigned long start, unsigned long end, int prot)
{
        start = ELF_PAGEALIGN(start);
        end = ELF_PAGEALIGN(end);
+--- 11 lines: if (end > start) {------------------------------------
        current->mm->start_brk = current->mm->brk = end;
        return 0;
}

fs/binfmt_elf.c                                                108,1
```



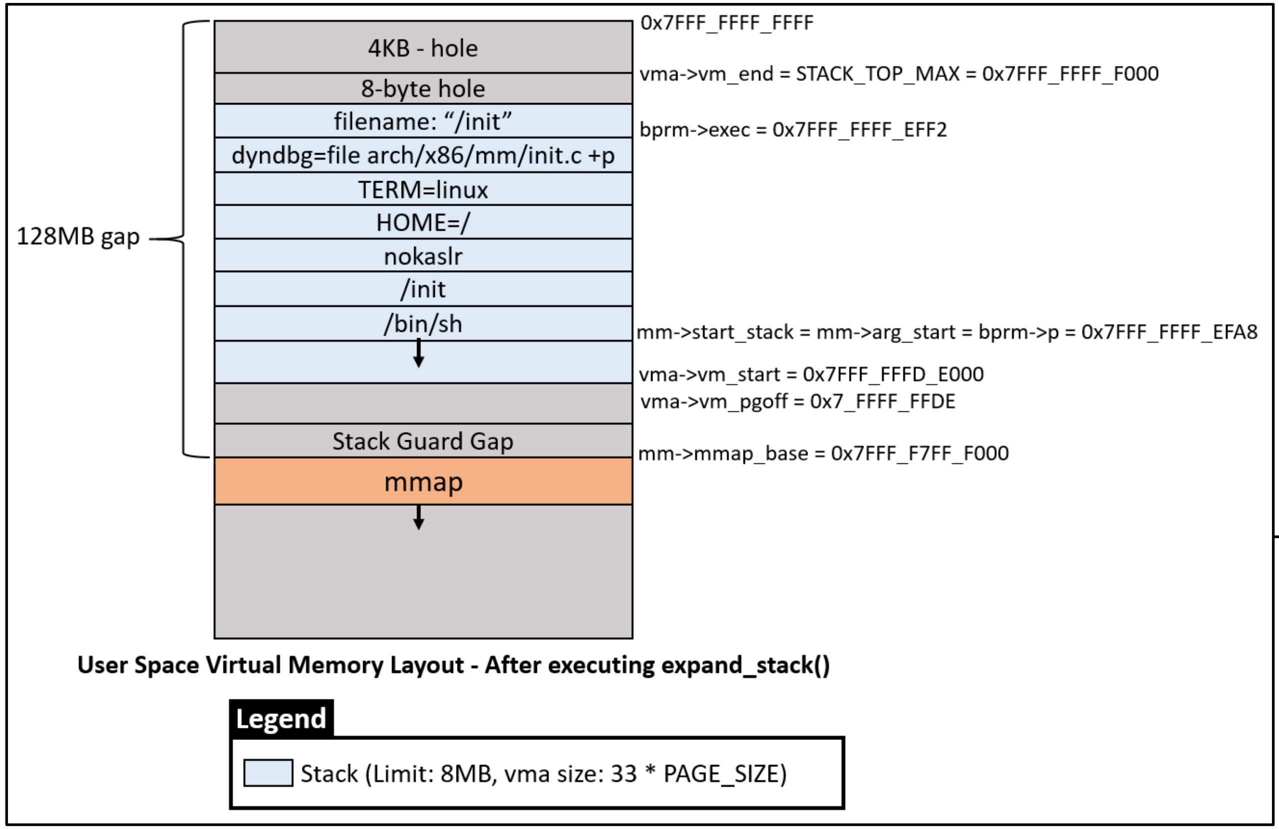
load_elf_binary(): set_brk() & padzero()



Note: statically linked program

create_elf_tables()

```
load_elf_binary [fs/binfmt_elf.c]
  ...
  |- set_brk(elf_bss, elf_brk, bss_prot)
  |- padzero(elf_bss)
  |- create_elf_tables
 Set mm->{start,end}_{code,data}
 mm->start_stack = bprm->p
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```



128MB gap

4KB - hole

8-byte hole

filename: “/init”

dyndbg=file arch/x86/mm/init.c +p

TERM=linux

HOME=/

nokaslr

/init

/bin/sh

Stack Guard Gap

mmap

0x7FFF_FFFF_FFFF

vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000

bprm->exec = 0x7FFF_FFFF_EFF2

mm->start_stack = mm->arg_start = bprm->p = 0x7FFF_FFFF_EFA8

vma->vm_start = 0x7FFF_FFFD_E000

vma->vm_pgoff = 0x7_FFFF_FFDE

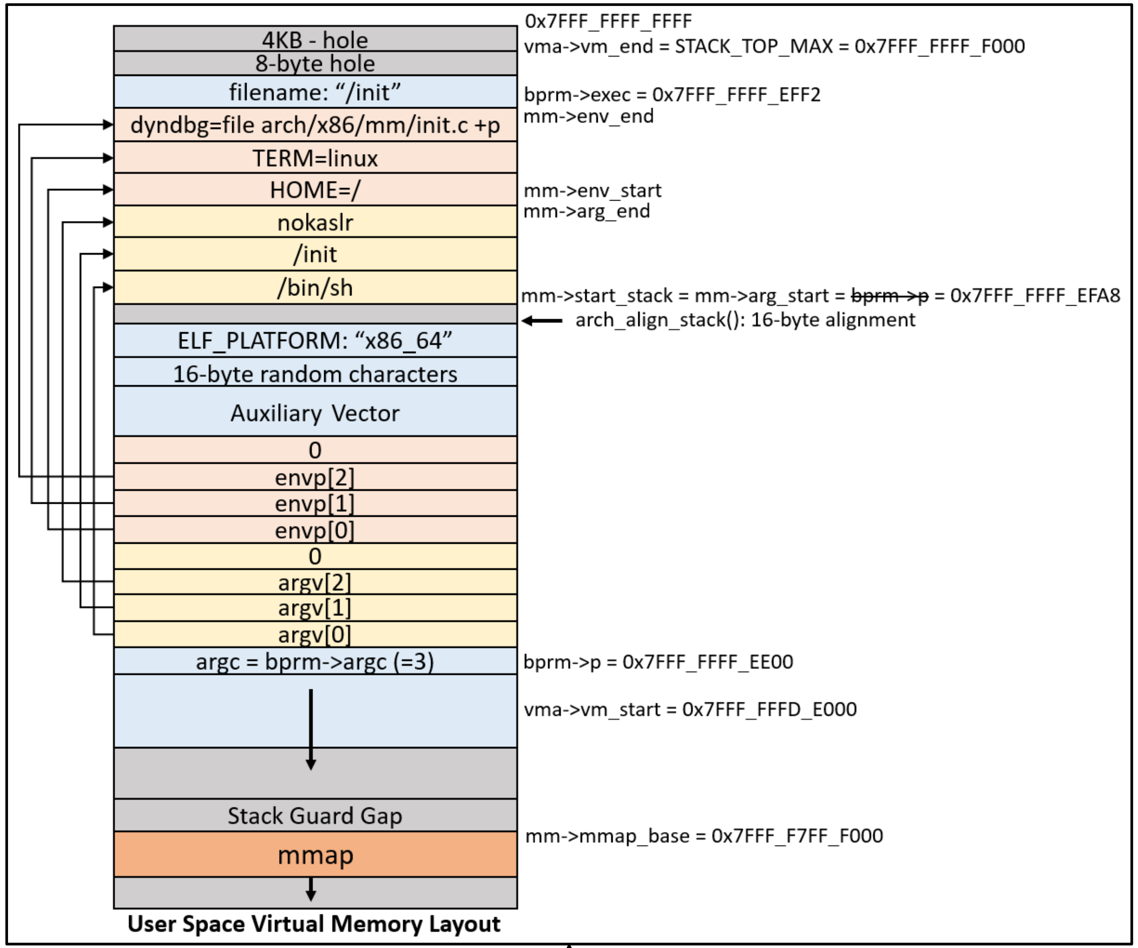
mm->mmap_base = 0x7FFF_F7FF_F000

User Space Virtual Memory Layout - After executing expand_stack()

Legend

▭ Stack (Limit: 8MB, vma size: 33 * PAGE_SIZE)

create_elf_tables()



4KB - hole

8-byte hole

filename: “/init”

dyndbg=file arch/x86/mm/init.c +p

TERM=linux

HOME=/

nokaslr

/init

/bin/sh

ELF_PLATFORM: “x86_64”

16-byte random characters

Auxiliary Vector

0

envp[2]

envp[1]

envp[0]

0

argv[2]

argv[1]

argv[0]

argc = bprm->argc (=3)

Stack Guard Gap

mmap

0x7FFF_FFFF_FFFF

vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000

bprm->exec = 0x7FFF_FFFF_EFF2

mm->env_end

mm->env_start

mm->arg_end

mm->start_stack = mm->arg_start = ~~bprm->p~~ = 0x7FFF_FFFF_EFA8

arch_align_stack(): 16-byte alignment

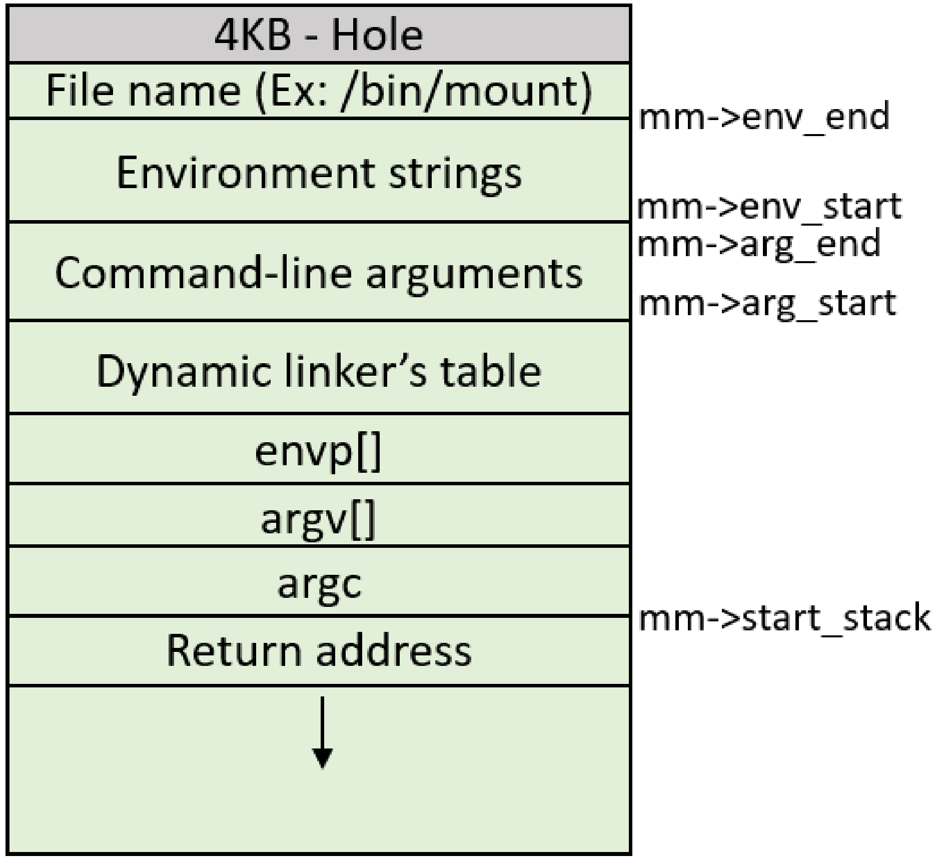
bprm->p = 0x7FFF_FFFF_EE00

vma->vm_start = 0x7FFF_FFFD_E000

mm->mmap_base = 0x7FFF_F7FF_F000

User Space Virtual Memory Layout

create_elf_tables() – User space stack



4KB - Hole

File name (Ex: /bin/mount)

mm->env_end

Environment strings

mm->env_start

mm->arg_end

Command-line arguments

mm->arg_start

Dynamic linker's table

envp[]

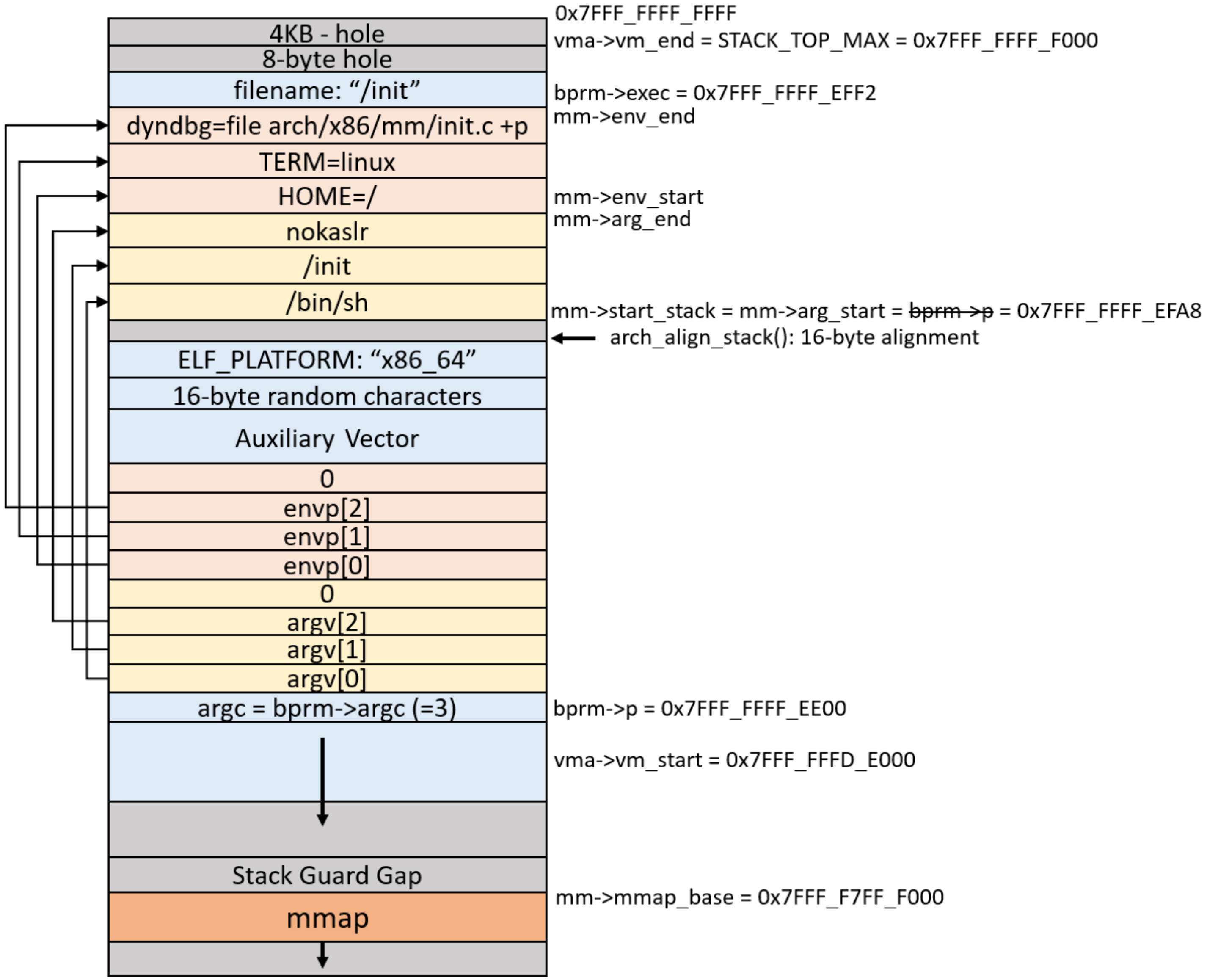
argv[]

argc

mm->start_stack

Return address

User Space Stack



0x7FFF_FFFF_FFFF

4KB - hole

vma->vm_end = STACK_TOP_MAX = 0x7FFF_FFFF_F000

8-byte hole

filename: "/init"

bprm->exec = 0x7FFF_FFFF_EFF2

mm->env_end

dyndbg=file arch/x86/mm/init.c +p

TERM=linux

HOME=/

mm->env_start

mm->arg_end

nokaslr

/init

/bin/sh

mm->start_stack = mm->arg_start = ~~bprm->p~~ = 0x7FFF_FFFF_EFA8

arch_align_stack(): 16-byte alignment

ELF_PLATFORM: "x86_64"

16-byte random characters

Auxiliary Vector

0

envp[2]

envp[1]

envp[0]

0

argv[2]

argv[1]

argv[0]

argc = bprm->argc (=3)

bprm->p = 0x7FFF_FFFF_EE00

vma->vm_start = 0x7FFF_FFFD_E000

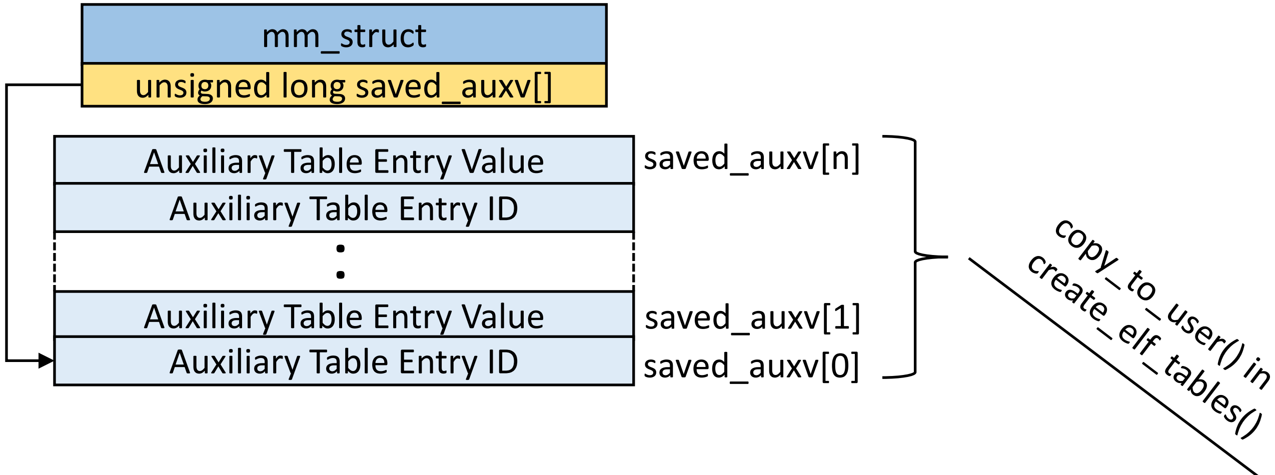
Stack Guard Gap

mm->mmap_base = 0x7FFF_F7FF_F000

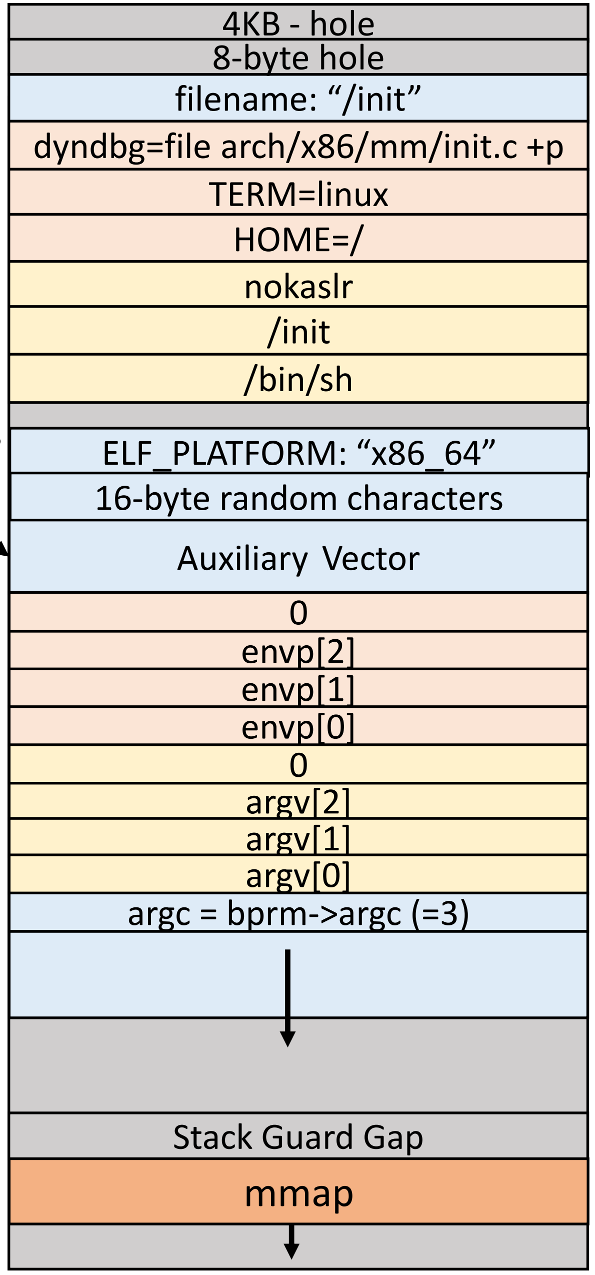
mmap

User Space Virtual Memory Layout

create_elf_tables() - Auxiliary Vector



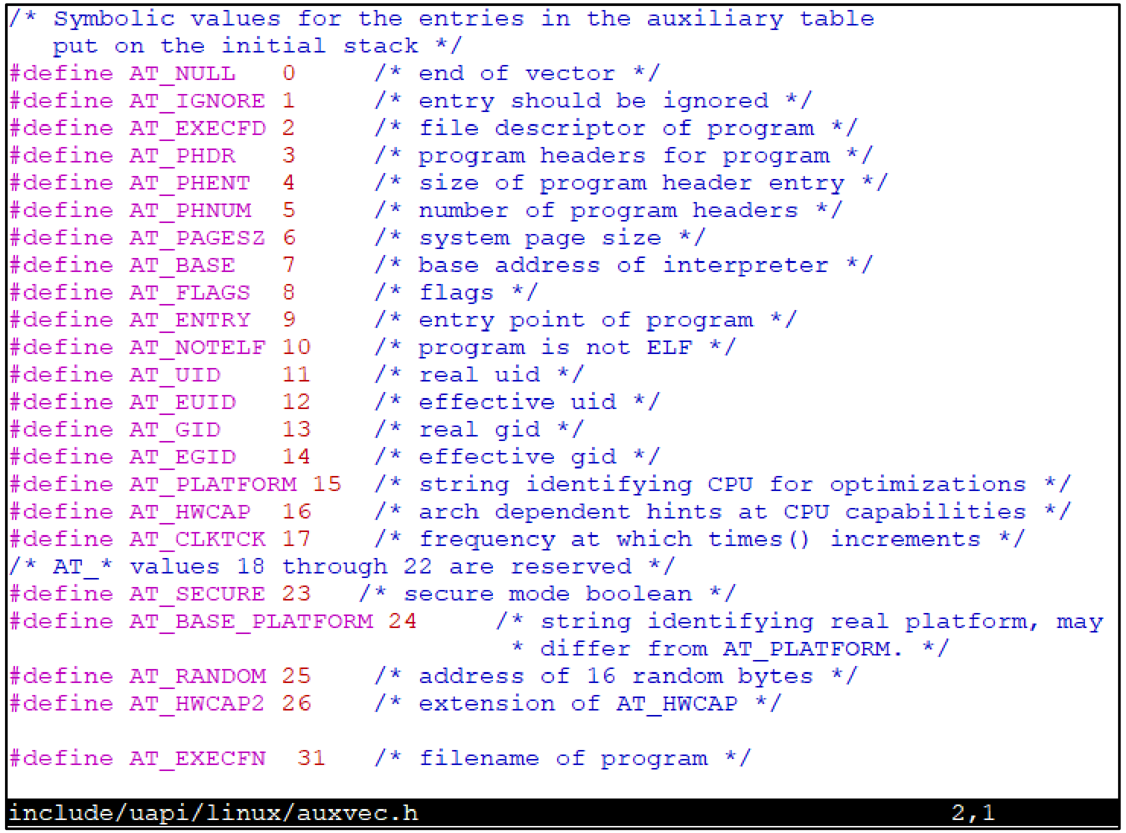
mm_struct
unsigned long saved_auxv[]
Auxiliary Table Entry Value
Auxiliary Table Entry ID
:
Auxiliary Table Entry Value
Auxiliary Table Entry ID
saved_auxv[n]
saved_auxv[1]
saved_auxv[0]
copy_to_user() in create_elf_tables()



User Space Virtual Memory Layout - Stack

More Info

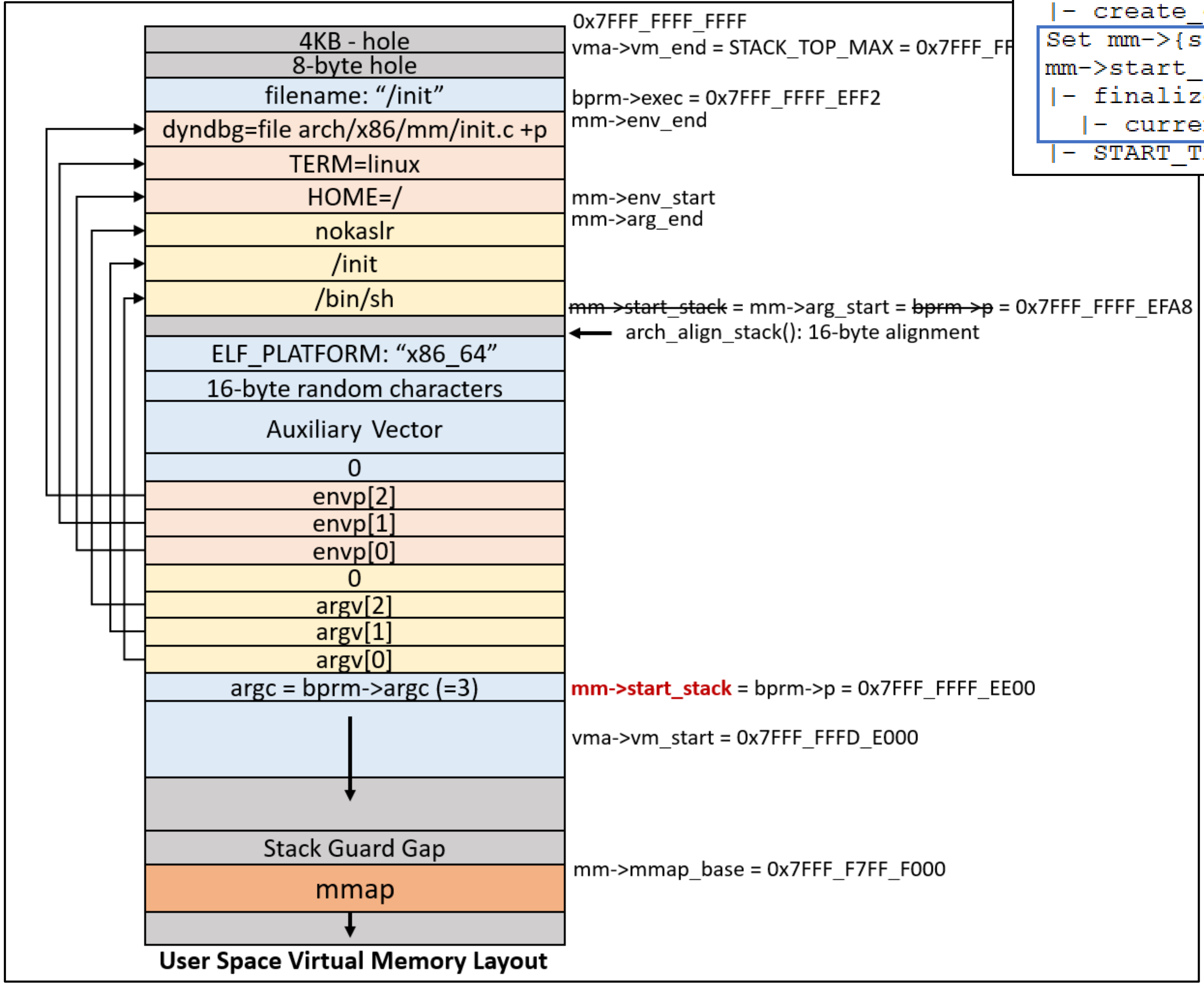
- `man getauxval`
- https://lwn.net/Articles/519085/



```
/* Symbolic values for the entries in the auxiliary table
   put on the initial stack */
#define AT_NULL   0     /* end of vector */
#define AT_IGNORE 1     /* entry should be ignored */
#define AT_EXECFD 2     /* file descriptor of program */
#define AT_PHDR   3     /* program headers for program */
#define AT_PHENT  4     /* size of program header entry */
#define AT_PHNUM  5     /* number of program headers */
#define AT_PAGESZ 6     /* system page size */
#define AT_BASE   7     /* base address of interpreter */
#define AT_FLAGS  8     /* flags */
#define AT_ENTRY  9     /* entry point of program */
#define AT_NOTELF 10    /* program is not ELF */
#define AT_UID    11    /* real uid */
#define AT_EUID   12    /* effective uid */
#define AT_GID    13    /* real gid */
#define AT_EGID   14    /* effective gid */
#define AT_PLATFORM 15  /* string identifying CPU for optimizations */
#define AT_HWCAP  16    /* arch dependent hints at CPU capabilities */
#define AT_CLKTCK 17    /* frequency at which times() increments */
/* AT_* values 18 through 22 are reserved */
#define AT_SECURE 23   /* secure mode boolean */
#define AT_BASE_PLATFORM 24     /* string identifying real platform, may
                                 * differ from AT_PLATFORM. */
#define AT_RANDOM 25    /* address of 16 random bytes */
#define AT_HWCAP2 26    /* extension of AT_HWCAP */

#define AT_EXECFN  31   /* filename of program */
include/uapi/linux/auxvec.h                                 2,1
```

load_elf_binary()

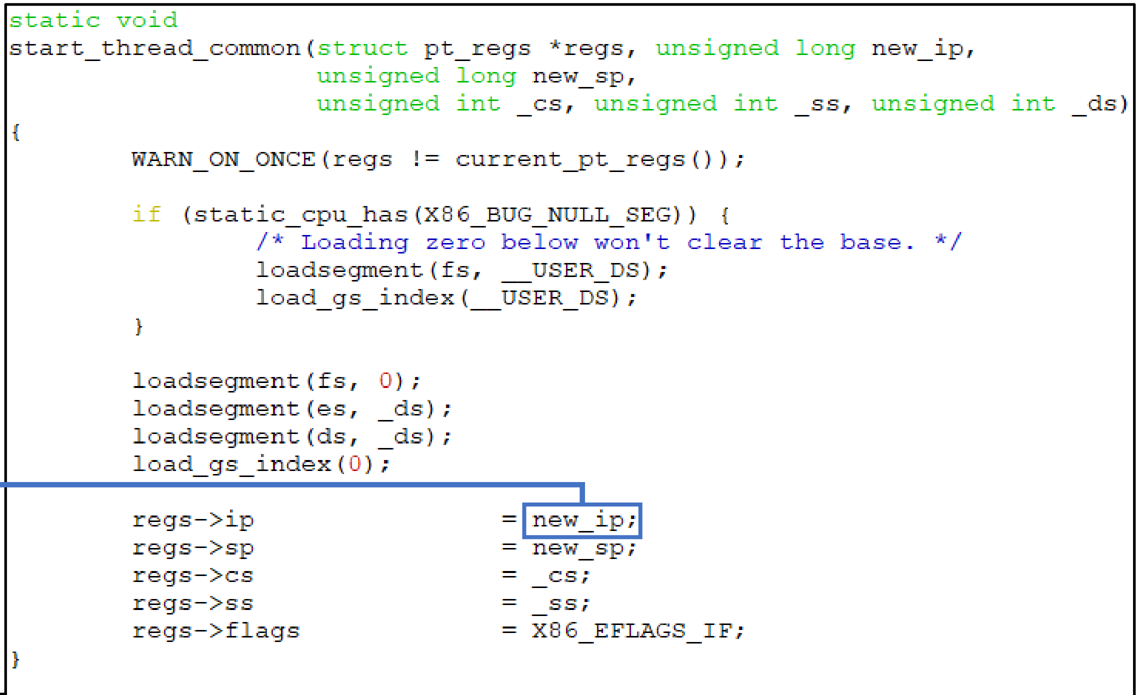
```
load_elf_binary [fs/binfmt_elf.c]
  ...
  |- set_brk(elf_bss, elf_brk, bss_prot)
  |- padzero(elf_bss)
  |- create_elf_tables
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD
```



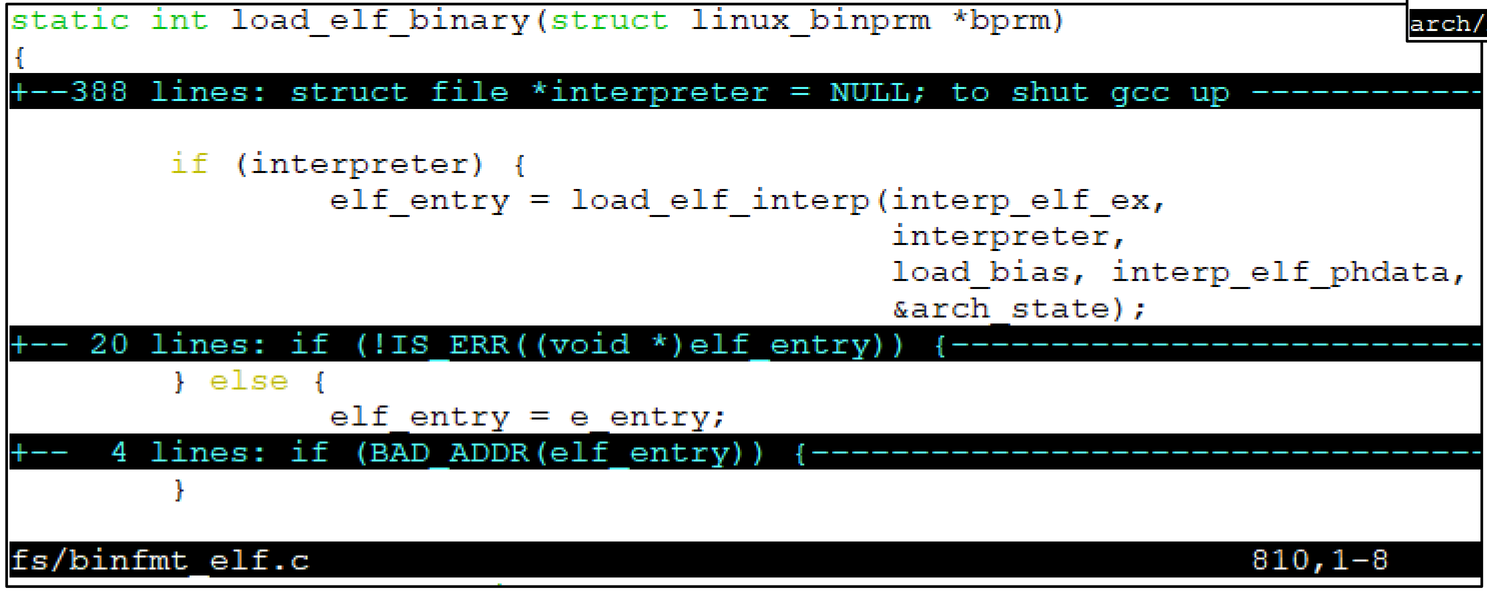
0x7FFF_FFFF_FFFF
4KB - hole
vma->vm_end = STACK_TOP_MAX = 0x7FFF_FF
8-byte hole
filename: "/init"
bprm->exec = 0x7FFF_FFFF_EFF2
dyndbg=file arch/x86/mm/init.c +p
mm->env_end
TERM=linux
HOME=/
mm->env_start
mm->arg_end
nokaslr
/init
/bin/sh
~~mm->start_stack~~ = mm->arg_start = ~~bprm->p~~ = 0x7FFF_FFFF_EFA8
arch_align_stack(): 16-byte alignment
ELF_PLATFORM: "x86_64"
16-byte random characters
Auxiliary Vector
0
envp[2]
envp[1]
envp[0]
0
argv[2]
argv[1]
argv[0]
argc = bprm->argc (=3)
mm->start_stack = bprm->p = 0x7FFF_FFFF_EE00
vma->vm_start = 0x7FFF_FFFD_E000
Stack Guard Gap
mmap
mm->mmap_base = 0x7FFF_F7FF_F000
User Space Virtual Memory Layout

load_elf_binary() -> START_THREAD()

```
load_elf_binary [fs/binfmt_elf.c]
  ...
  |- set_brk(elf_bss, elf_brk, bss_prot)
  |- padzero(elf_bss)
  |- create_elf_tables
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- finalize_exec
    |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD(elf_ex, regs, elf_entry, bprm->p);
```



```
static void
start_thread_common(struct pt_regs *regs, unsigned long new_ip,
                    unsigned long new_sp,
                    unsigned int _cs, unsigned int _ss, unsigned int _ds)
{
        WARN_ON_ONCE(regs != current_pt_regs());

        if (static_cpu_has(X86_BUG_NULL_SEG)) {
                /* Loading zero below won't clear the base. */
                loadsegment(fs, __USER_DS);
                load_gs_index(__USER_DS);
        }

        loadsegment(fs, 0);
        loadsegment(es, _ds);
        loadsegment(ds, _ds);
        load_gs_index(0);

        regs->ip                = new_ip;
        regs->sp                = new_sp;
        regs->cs                = _cs;
        regs->ss                = _ss;
        regs->flags             = X86_EFLAGS_IF;
}
arch/x86/kernel/process_64.c                                  478,1
```



```
static int load_elf_binary(struct linux_binprm *bprm)
{
+--388 lines: struct file *interpreter = NULL; to shut gcc up ------------

        if (interpreter) {
                elf_entry = load_elf_interp(interp_elf_ex,
                                            interpreter,
                                            load_bias, interp_elf_phdata,
                                            &arch_state);
+-- 20 lines: if (!IS_ERR((void *)elf_entry)) {----------------------------
        } else {
                elf_entry = e_entry;
+--  4 lines: if (BAD_ADDR(elf_entry)) {-----------------------------------
        }

fs/binfmt_elf.c                                             810,1-8
```

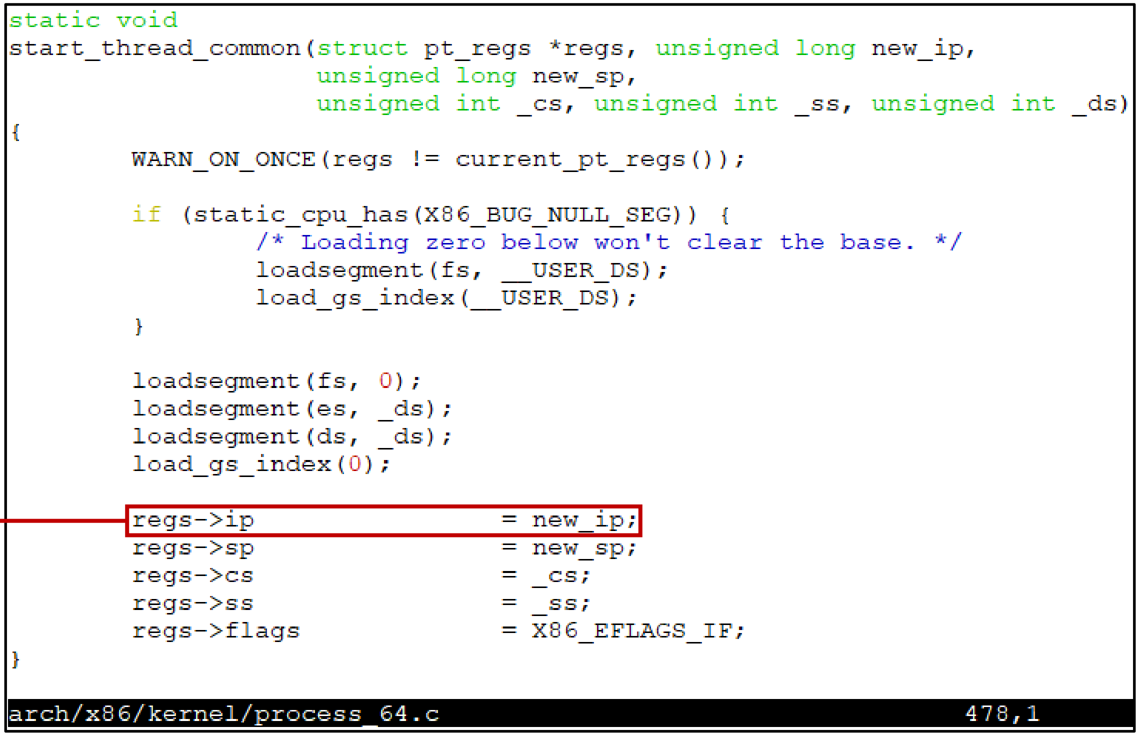
```
load_elf_interp [fs/binfmt_elf.c]
  |- map_addr = elf_map(...)
    |- vm_mmap
      |- vm_mmap_pgoff
```

load_elf_binary() -> START_THREAD() – Statically-linked program

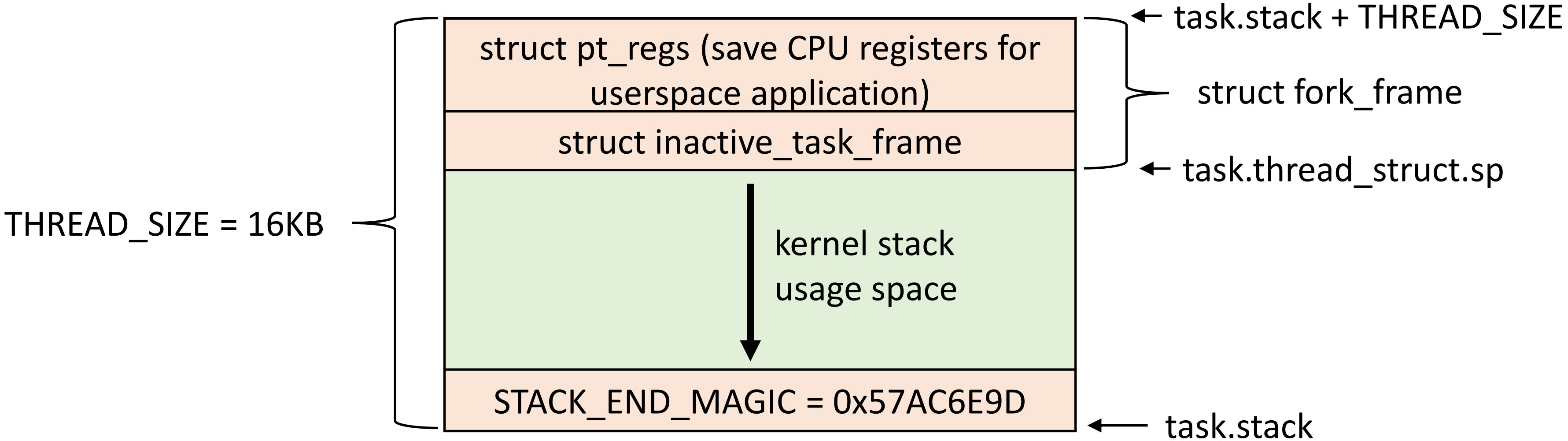
```
load_elf_binary [fs/binfmt_elf.c]
  ...
  |- set_brk(elf_bss, elf_brk, bss_prot)
  |- padzero(elf_bss)
  |- create_elf_tables
  Set mm->{start,end}_{code,data}
  mm->start_stack = bprm->p
  |- finalize_exec
     |- current->signal->rlim[RLIMIT_STACK] = bprm->rlim_stack
  |- START_THREAD(elf_ex, regs, elf_entry, bprm->p);
```

```
$ file bin/busybox
bin/busybox: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), staticall
y linked, BuildID[sha1]=0db801f555823db5126b6a7e8b0ed9294fc3c1a7, for GNU/Linux
3.2.0, stripped

$ readelf -h bin/busybox | grep Entry
  Entry point address:               0x402270
```

_start() in executable file 'busybox'



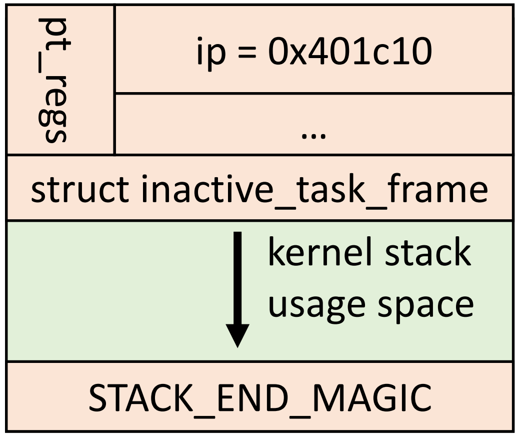
```
static void
start_thread_common(struct pt_regs *regs, unsigned long new_ip,
                    unsigned long new_sp,
                    unsigned int _cs, unsigned int _ss, unsigned int _ds)
{
        WARN_ON_ONCE(regs != current_pt_regs());

        if (static_cpu_has(X86_BUG_NULL_SEG)) {
                /* Loading zero below won't clear the base. */
                loadsegment(fs, __USER_DS);
                load_gs_index(__USER_DS);
        }

        loadsegment(fs, 0);
        loadsegment(es, _ds);
        loadsegment(ds, _ds);
        load_gs_index(0);

        regs->ip                = new_ip;
        regs->sp                = new_sp;
        regs->cs                = _cs;
        regs->ss                = _ss;
        regs->flags             = X86_EFLAGS_IF;
}
arch/x86/kernel/process_64.c                                              478,1
```



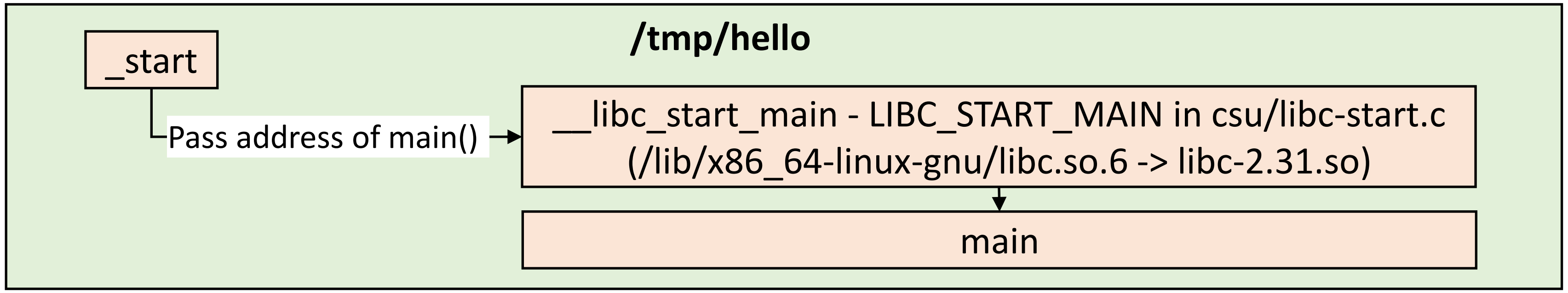
task.stack + THREAD_SIZE
struct pt_regs (save CPU registers for userspace application)
struct fork_frame
struct inactive_task_frame
task.thread_struct.sp
THREAD_SIZE = 16KB
kernel stack usage space
STACK_END_MAGIC = 0x57AC6E9D
task.stack

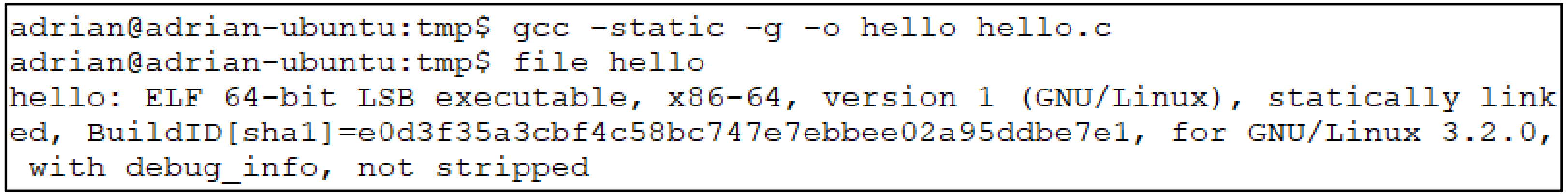
Kernel Stack

Statically-linked program (OS: Ubuntu 20.04.3)

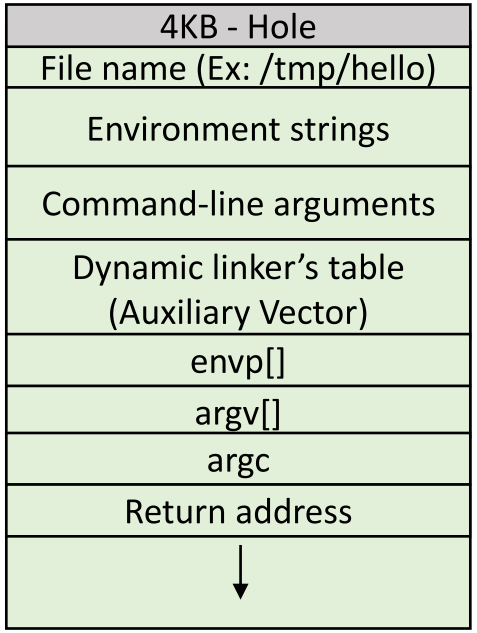


Kernel Stack

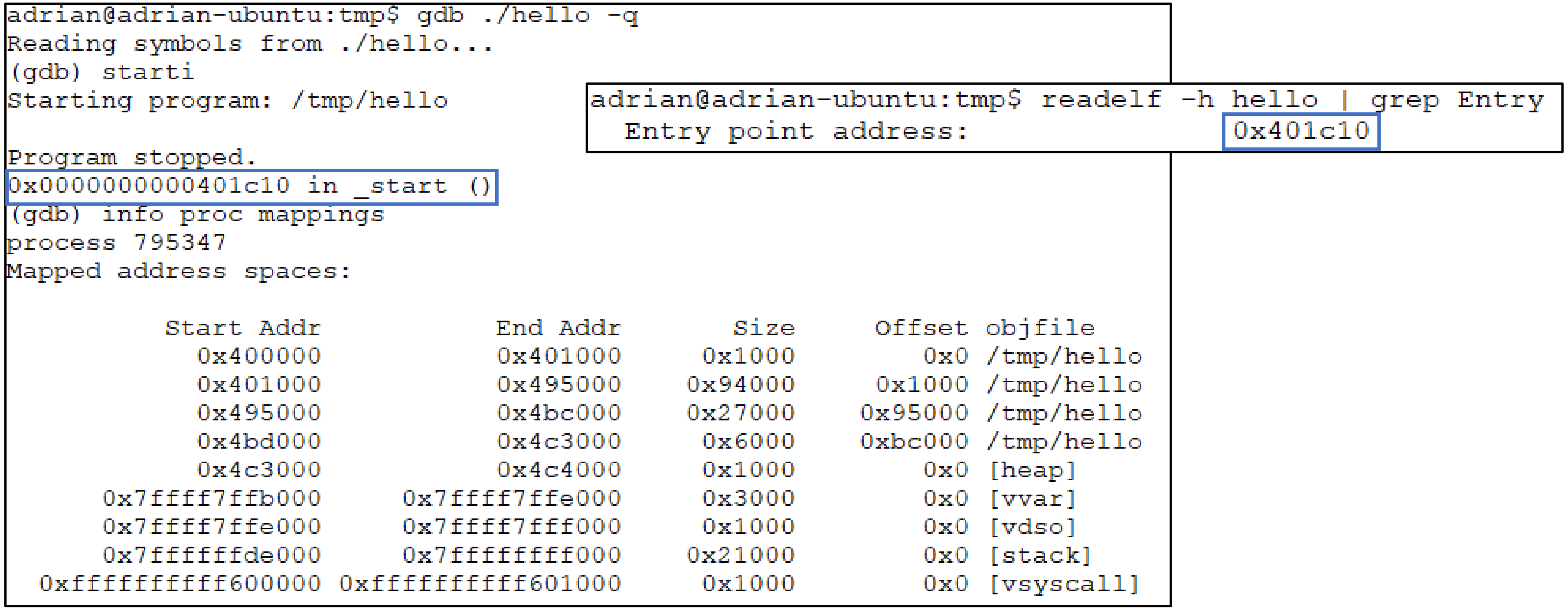

/tmp/hello
_start
Pass address of main()
__libc_start_main - LIBC_START_MAIN in csu/libc-start.c (/lib/x86_64-linux-gnu/libc.so.6 -> libc-2.31.so)
main



```
adrian@adrian-ubuntu:tmp$ gcc -static -g -o hello hello.c
adrian@adrian-ubuntu:tmp$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically link
ed, BuildID[sha1]=e0d3f35a3cbf4c58bc747e7ebbee02a95ddbe7e1, for GNU/Linux 3.2.0,
 with debug_info, not stripped
```



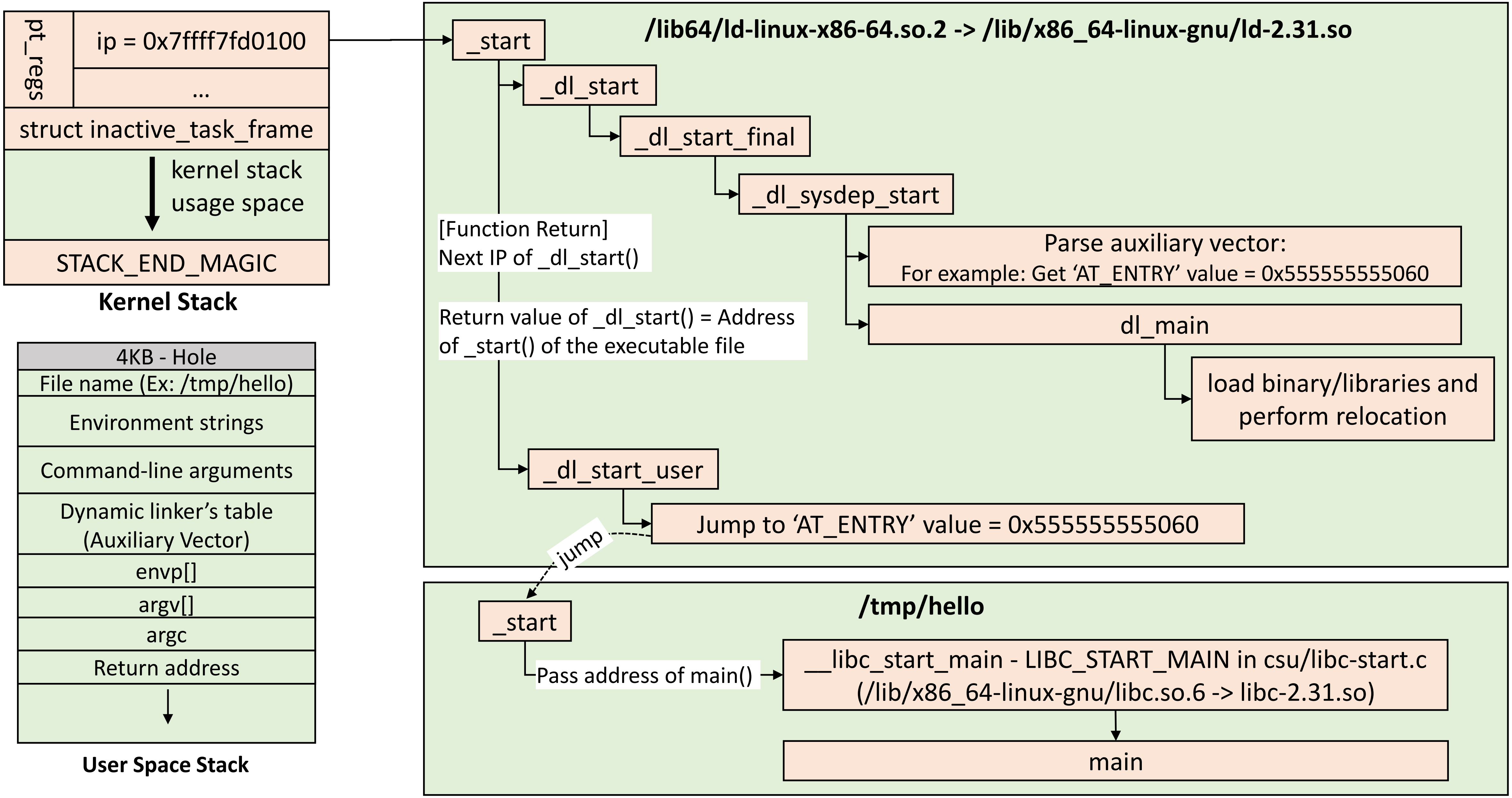
User Space Stack



```
adrian@adrian-ubuntu:tmp$ gdb ./hello -q
Reading symbols from ./hello...
(gdb) starti
Starting program: /tmp/hello

Program stopped.
0x0000000000401c10 in _start ()
(gdb) info proc mappings
process 795347
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /tmp/hello
            0x401000           0x495000    0x94000     0x1000 /tmp/hello
            0x495000           0x4bc000    0x27000    0x95000 /tmp/hello
            0x4bd000           0x4c3000     0x6000    0xbc000 /tmp/hello
            0x4c3000           0x4c4000     0x1000        0x0 [heap]
      0x7ffff7ffb000     0x7ffff7ffe000     0x3000        0x0 [vvar]
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0 [vdso]
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

```
adrian@adrian-ubuntu:tmp$ readelf -h hello | grep Entry
  Entry point address:               0x401c10
```

Dynamically-linked program (OS: Ubuntu 20.04.3)



pt_regs

ip = 0x7ffff7fd0100

…

struct inactive_task_frame

kernel stack usage space

STACK_END_MAGIC

Kernel Stack

4KB - Hole

File name (Ex: /tmp/hello)

Environment strings

Command-line arguments

Dynamic linker's table (Auxiliary Vector)

envp[]

argv[]

argc

Return address

User Space Stack

/lib64/ld-linux-x86-64.so.2 -> /lib/x86_64-linux-gnu/ld-2.31.so

_start

_dl_start

_dl_start_final

_dl_sysdep_start

Parse auxiliary vector:
For example: Get 'AT_ENTRY' value = 0x555555555060

dl_main

load binary/libraries and perform relocation

[Function Return]
Next IP of _dl_start()

Return value of _dl_start() = Address of _start() of the executable file

_dl_start_user

Jump to 'AT_ENTRY' value = 0x555555555060

jump

/tmp/hello

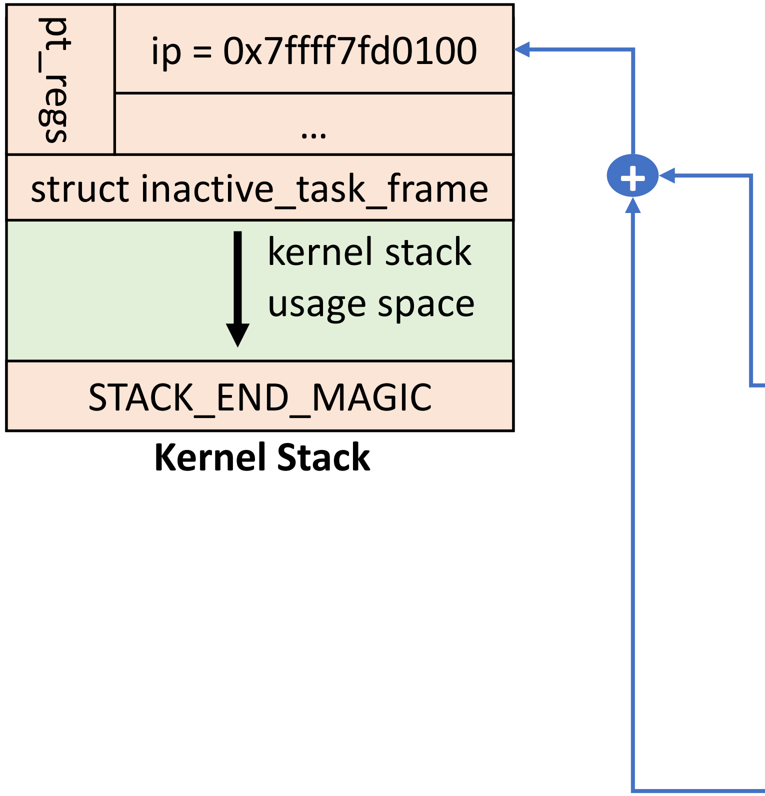
_start

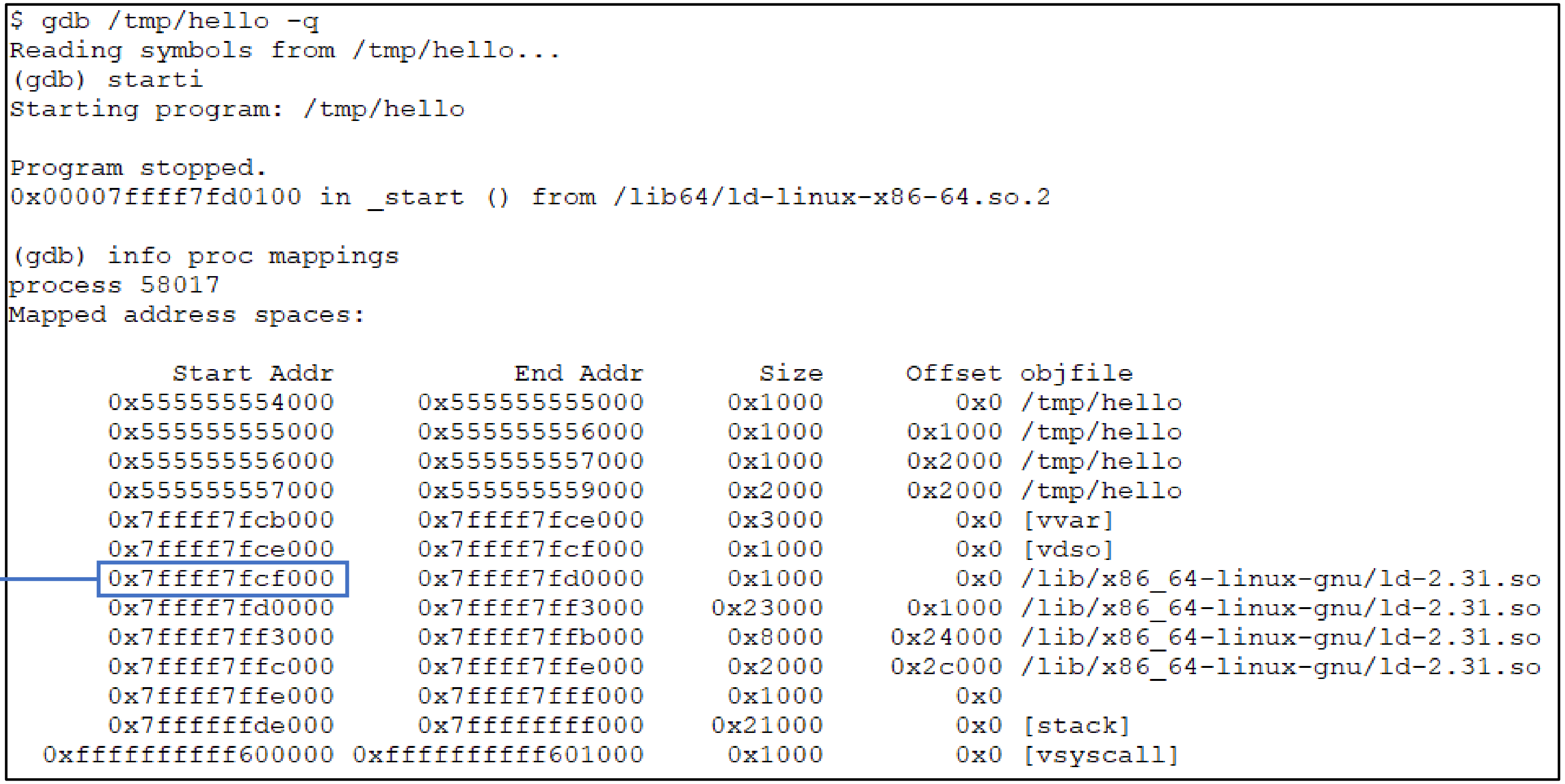
Pass address of main()

__libc_start_main - LIBC_START_MAIN in csu/libc-start.c
(/lib/x86_64-linux-gnu/libc.so.6 -> libc-2.31.so)

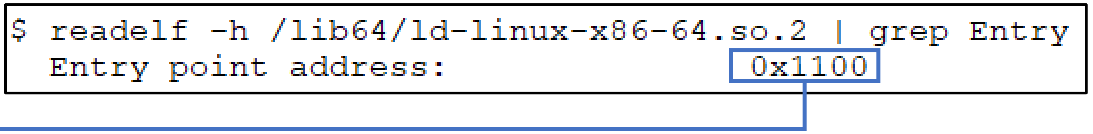
main

Dynamically-linked program (OS: Ubuntu 20.04.3)



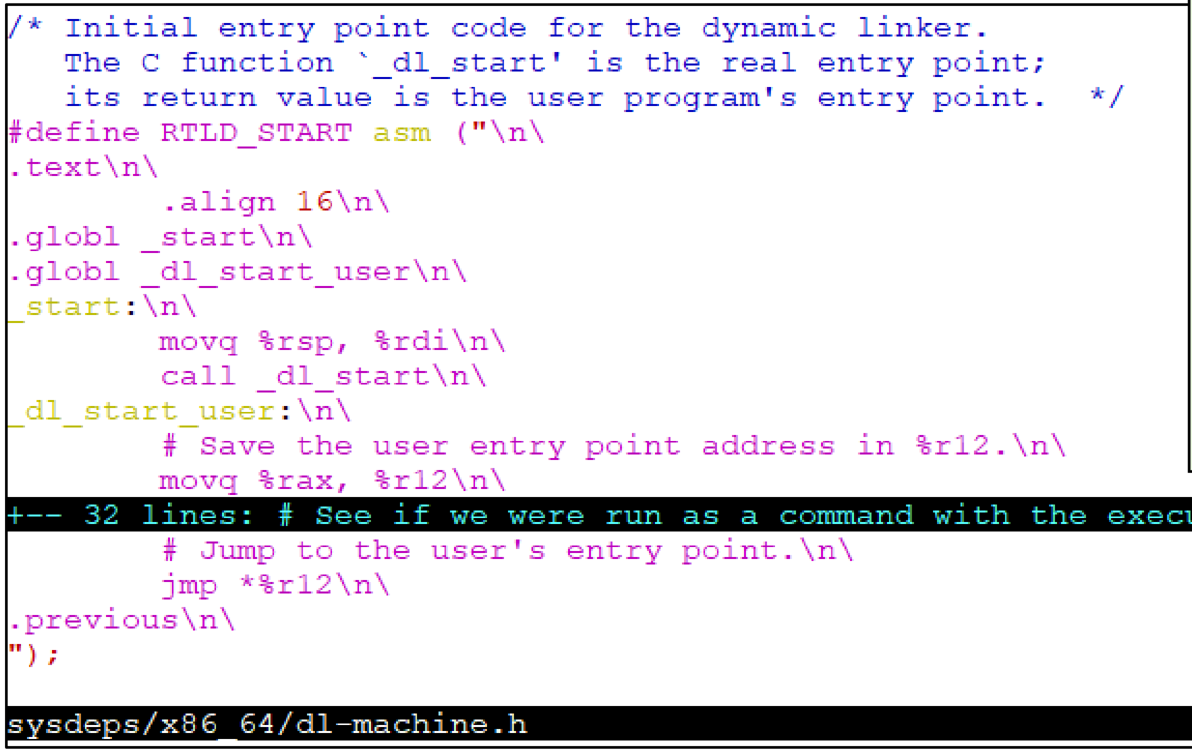


```
$ gdb /tmp/hello -q
Reading symbols from /tmp/hello...
(gdb) starti
Starting program: /tmp/hello

Program stopped.
0x00007ffff7fd0100 in _start () from /lib64/ld-linux-x86-64.so.2

(gdb) info proc mappings
process 58017
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x555555554000     0x555555555000     0x1000        0x0 /tmp/hello
      0x555555555000     0x555555556000     0x1000     0x1000 /tmp/hello
      0x555555556000     0x555555557000     0x1000     0x2000 /tmp/hello
      0x555555557000     0x555555559000     0x2000     0x2000 /tmp/hello
      0x7ffff7fcb000     0x7ffff7fce000     0x3000        0x0 [vvar]
      0x7ffff7fce000     0x7ffff7fcf000     0x1000        0x0 [vdso]
      0x7ffff7fcf000     0x7ffff7fd0000     0x1000        0x0 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7fd0000     0x7ffff7ff3000    0x23000     0x1000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ff3000     0x7ffff7ffb000     0x8000    0x24000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ffc000     0x7ffff7ffe000     0x2000    0x2c000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

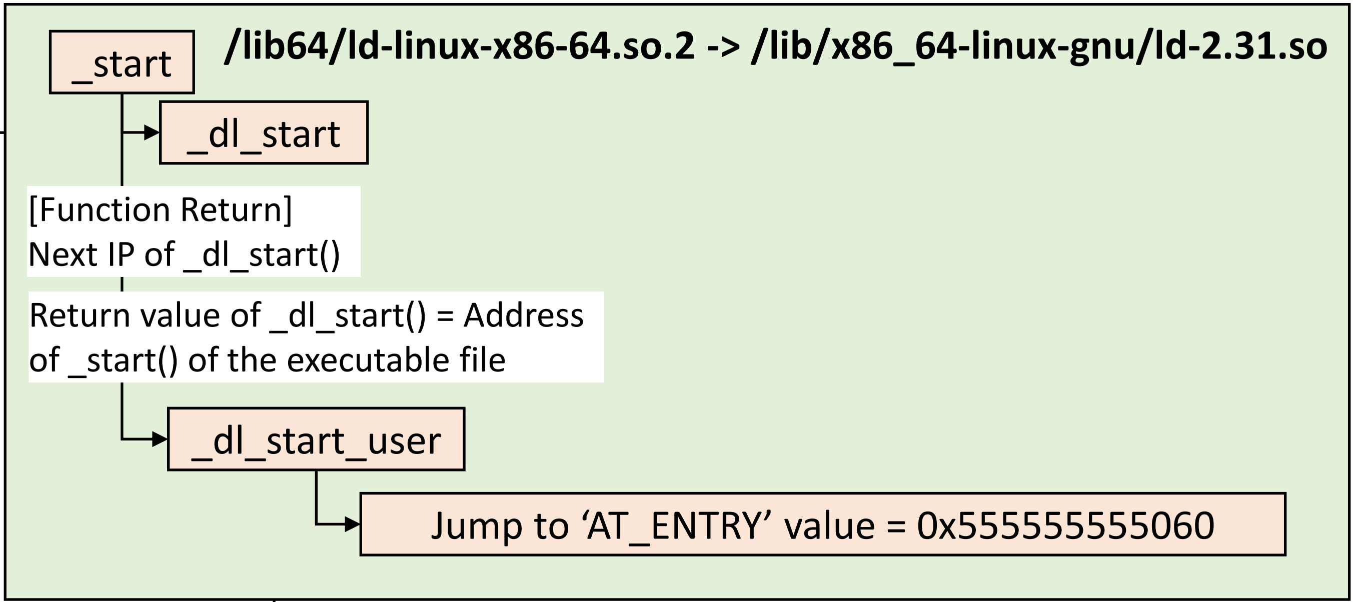


```
$ readelf -h /lib64/ld-linux-x86-64.so.2 | grep Entry
  Entry point address:               0x1100
```

/lib64/ld-linux-x86-64.so.2



```
/* Initial entry point code for the dynamic linker.
   The C function `_dl_start' is the real entry point;
   its return value is the user program's entry point.  */
#define RTLD_START asm ("\n\
.text\n\
        .align 16\n\
.globl _start\n\
.globl _dl_start_user\n\
_start:\n\
        movq %rsp, %rdi\n\
        call _dl_start\n\
_dl_start_user:\n\
        # Save the user entry point address in %r12.\n\
        movq %rax, %r12\n\
+-- 32 lines: # See if we were run as a command with the executable file\n\
        # Jump to the user's entry point.\n\
        jmp *%r12\n\
.previous\n\
");

sysdeps/x86_64/dl-machine.h                              133,1
```



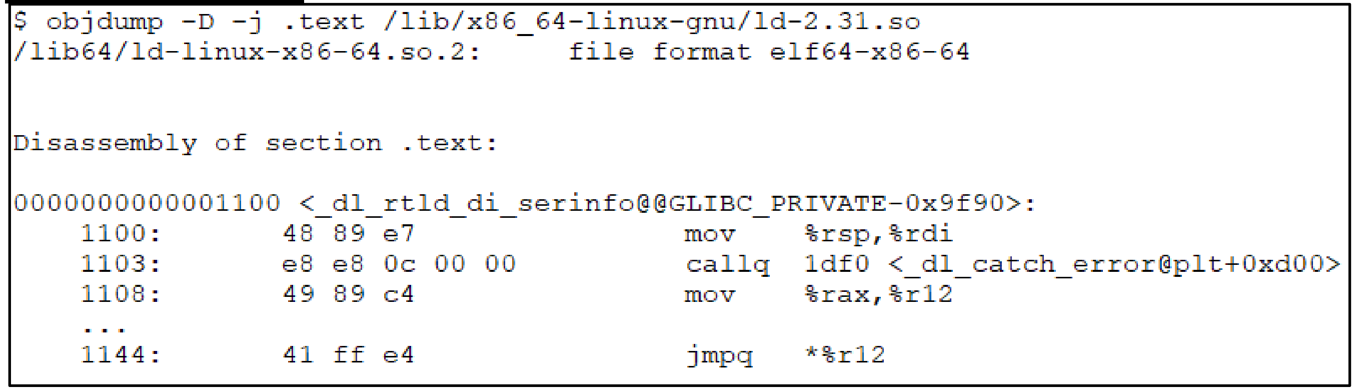
/lib64/ld-linux-x86-64.so.2 -> /lib/x86_64-linux-gnu/ld-2.31.so

[Function Return]
Next IP of _dl_start()

Return value of _dl_start() = Address of _start() of the executable file

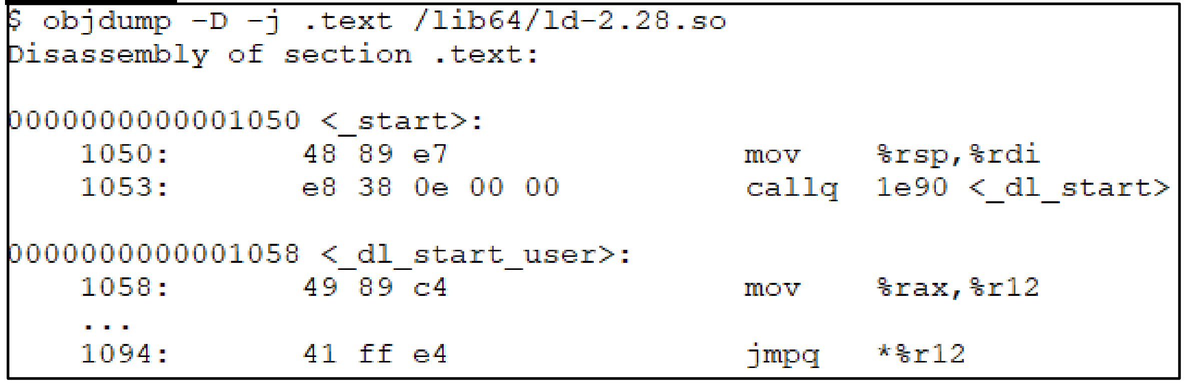
Jump to 'AT_ENTRY' value = 0x555555555060

Ubuntu 20.04.3



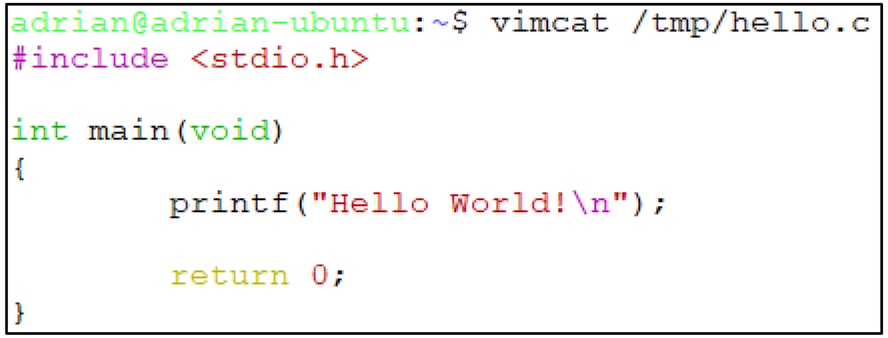
```
$ objdump -D -j .text /lib/x86_64-linux-gnu/ld-2.31.so
/lib64/ld-linux-x86-64.so.2:     file format elf64-x86-64


Disassembly of section .text:

0000000000001100 <_dl_rtld_di_serinfo@@GLIBC_PRIVATE-0x9f90>:
    1100:       48 89 e7                mov    %rsp,%rdi
    1103:       e8 e8 0c 00 00          callq  1df0 <_dl_catch_error@plt+0xd00>
    1108:       49 89 c4                mov    %rax,%r12
    ...
    1144:       41 ff e4                jmpq   *%r12
```

RHEL8.2

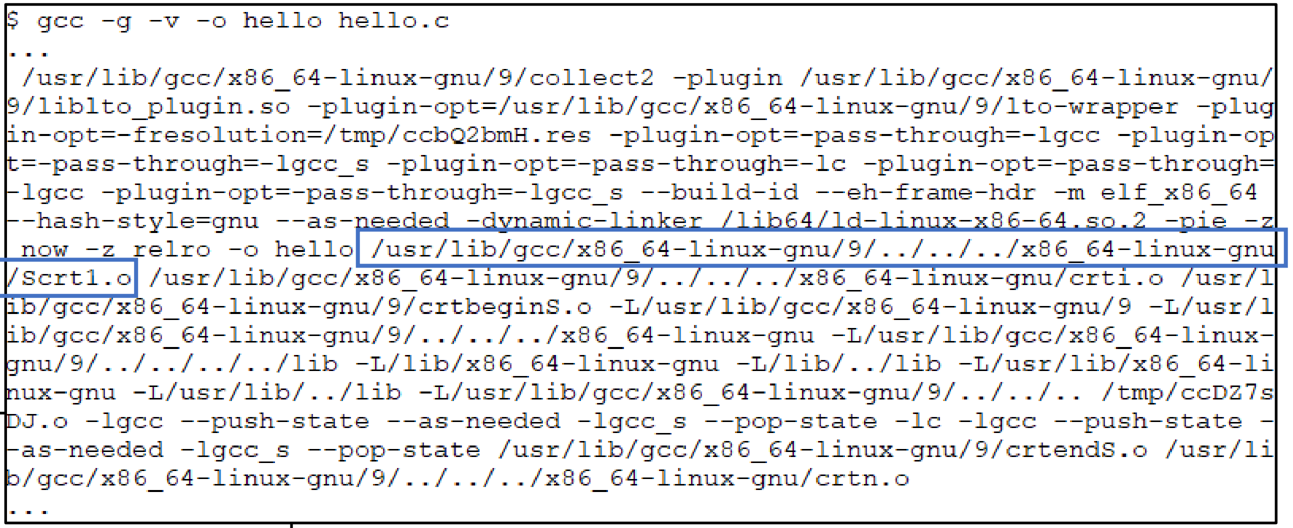


```
$ objdump -D -j .text /lib64/ld-2.28.so
Disassembly of section .text:

0000000000001050 <_start>:
    1050:       48 89 e7                mov    %rsp,%rdi
    1053:       e8 38 0e 00 00          callq  1e90 <_dl_start>

0000000000001058 <_dl_start_user>:
    1058:       49 89 c4                mov    %rax,%r12
    ...
    1094:       41 ff e4                jmpq   *%r12
```

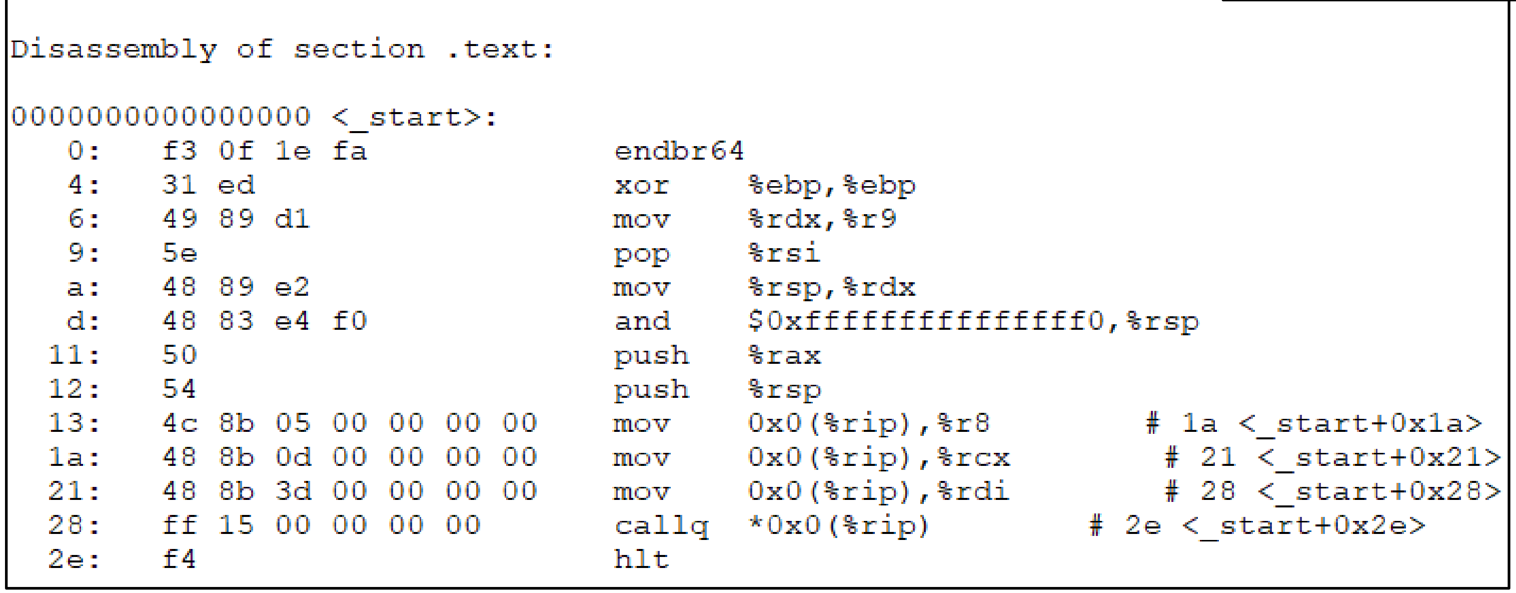
[Dynamically-linked program] Executable file: where is “_start()” from?



```
adrian@adrian-ubuntu:~$ vimcat /tmp/hello.c
#include <stdio.h>

int main(void)
{
        printf("Hello World!\n");

        return 0;
}
```



```
$ gcc -g -v -o hello hello.c
...
 /usr/lib/gcc/x86_64-linux-gnu/9/collect2 -plugin /usr/lib/gcc/x86_64-linux-gnu/
9/liblto_plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper -plug
in-opt=-fresolution=/tmp/ccbQ2bmH.res -plugin-opt=-pass-through=-lgcc -plugin-op
t=-pass-through=-lgcc_s -plugin-opt=-pass-through=-lc -plugin-opt=-pass-through=
-lgcc -plugin-opt=-pass-through=-lgcc_s --build-id --eh-frame-hdr -m elf_x86_64
--hash-style=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie -z
 now -z relro -o hello /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu
/Scrt1.o /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/crti.o /usr/l
ib/gcc/x86_64-linux-gnu/9/crtbeginS.o -L/usr/lib/gcc/x86_64-linux-gnu/9 -L/usr/l
ib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-
gnu/9/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-li
nux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/9/../../.. /tmp/ccDZ7s
DJ.o -lgcc --push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state -
-as-needed -lgcc_s --pop-state /usr/lib/gcc/x86_64-linux-gnu/9/crtendS.o /usr/li
b/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/crtn.o
...
```

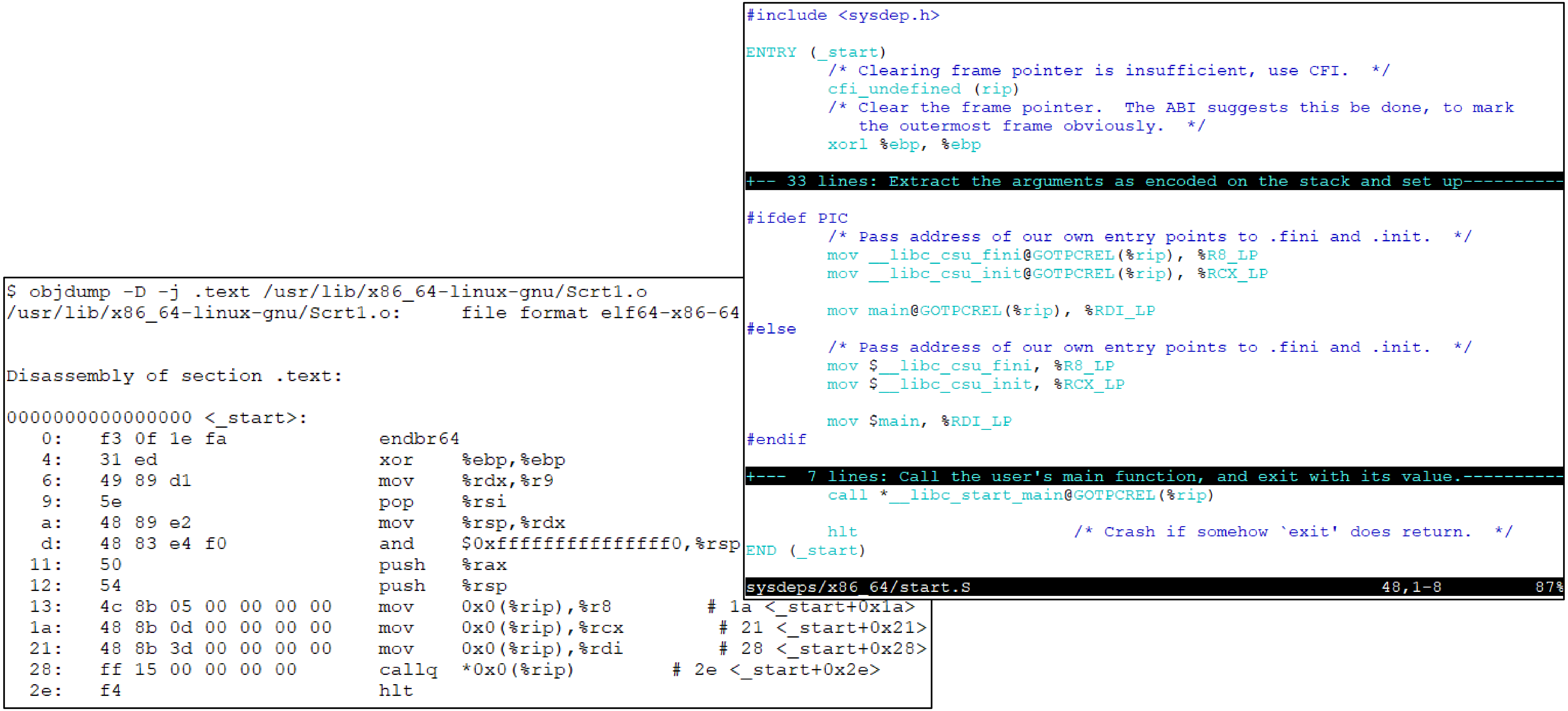


```
$ objdump -D -j .text /usr/lib/x86_64-linux-gnu/Scrt1.o
/usr/lib/x86_64-linux-gnu/Scrt1.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_start>:
   0:   f3 0f 1e fa             endbr64
   4:   31 ed                   xor    %ebp,%ebp
   6:   49 89 d1                mov    %rdx,%r9
   9:   5e                      pop    %rsi
   a:   48 89 e2                mov    %rsp,%rdx
   d:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
  11:   50                      push   %rax
  12:   54                      push   %rsp
  13:   4c 8b 05 00 00 00 00    mov    0x0(%rip),%r8        # 1a <_start+0x1a>
  1a:   48 8b 0d 00 00 00 00    mov    0x0(%rip),%rcx        # 21 <_start+0x21>
  21:   48 8b 3d 00 00 00 00    mov    0x0(%rip),%rdi        # 28 <_start+0x28>
  28:   ff 15 00 00 00 00       callq  *0x0(%rip)        # 2e <_start+0x2e>
  2e:   f4                      hlt
```

Reference
https://dev.gentoo.org/~vapier/crt.txt
https://en.wikipedia.org/wiki/Crt0

***crt*.o (C Runtime): A set of execution startup routines linked into a C program that performs initialization work before calling the program’s main function.**

[Dynamically-linked program] Executable file: where is “_start()” from?

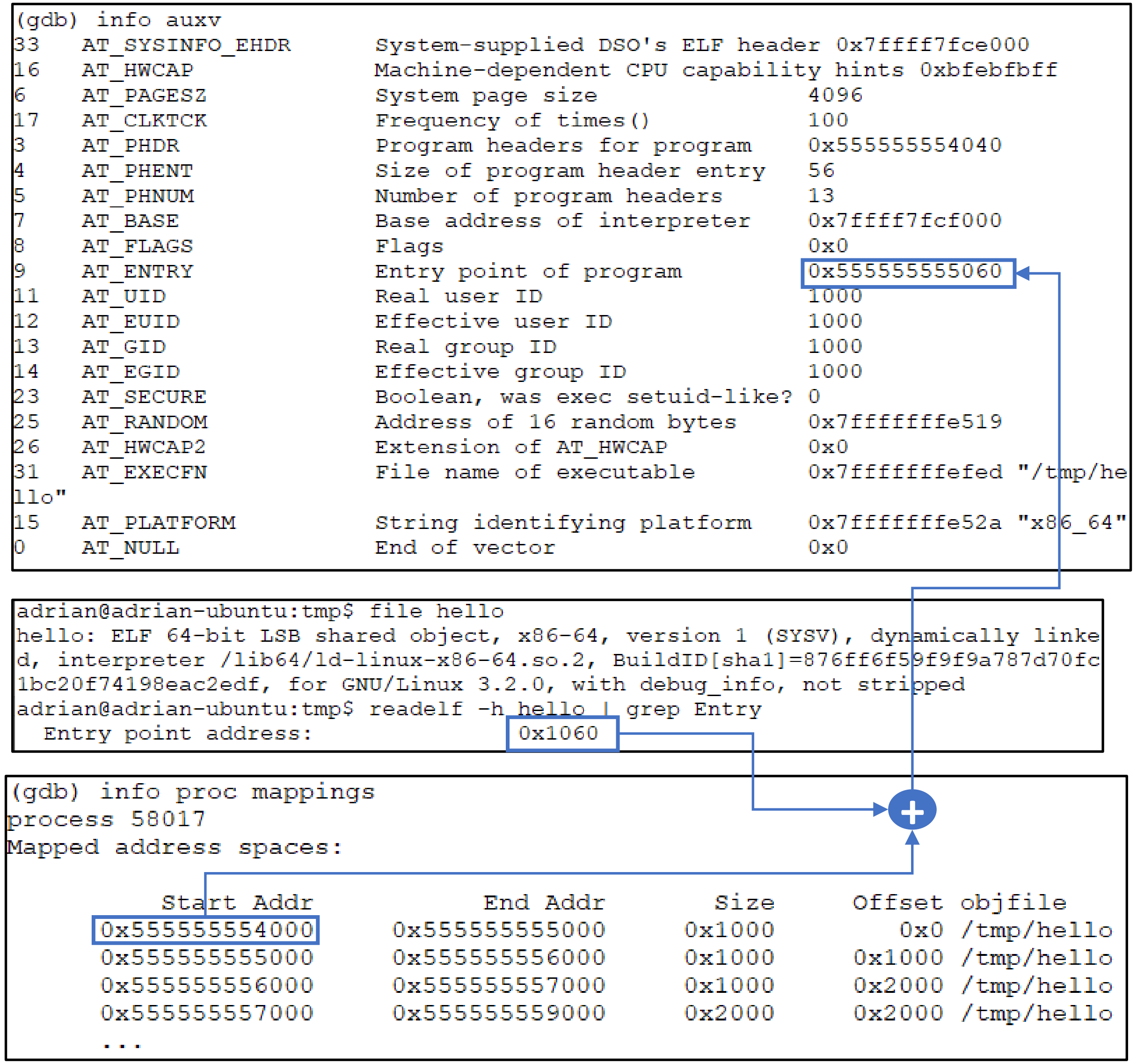


```
$ objdump -D -j .text /usr/lib/x86_64-linux-gnu/Scrt1.o
/usr/lib/x86_64-linux-gnu/Scrt1.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_start>:
   0:   f3 0f 1e fa             endbr64
   4:   31 ed                   xor    %ebp,%ebp
   6:   49 89 d1                mov    %rdx,%r9
   9:   5e                      pop    %rsi
   a:   48 89 e2                mov    %rsp,%rdx
   d:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
  11:   50                      push   %rax
  12:   54                      push   %rsp
  13:   4c 8b 05 00 00 00 00    mov    0x0(%rip),%r8        # 1a <_start+0x1a>
  1a:   48 8b 0d 00 00 00 00    mov    0x0(%rip),%rcx        # 21 <_start+0x21>
  21:   48 8b 3d 00 00 00 00    mov    0x0(%rip),%rdi        # 28 <_start+0x28>
  28:   ff 15 00 00 00 00       callq  *0x0(%rip)        # 2e <_start+0x2e>
  2e:   f4                      hlt
```

```
#include <sysdep.h>

ENTRY (_start)
        /* Clearing frame pointer is insufficient, use CFI.  */
        cfi_undefined (rip)
        /* Clear the frame pointer.  The ABI suggests this be done, to mark
           the outermost frame obviously.  */
        xorl %ebp, %ebp

+-- 33 lines: Extract the arguments as encoded on the stack and set up----------

#ifdef PIC
        /* Pass address of our own entry points to .fini and .init.  */
        mov __libc_csu_fini@GOTPCREL(%rip), %R8_LP
        mov __libc_csu_init@GOTPCREL(%rip), %RCX_LP

        mov main@GOTPCREL(%rip), %RDI_LP
#else
        /* Pass address of our own entry points to .fini and .init.  */
        mov $__libc_csu_fini, %R8_LP
        mov $__libc_csu_init, %RCX_LP

        mov $main, %RDI_LP
#endif

+---  7 lines: Call the user's main function, and exit with its value.----------
        call *__libc_start_main@GOTPCREL(%rip)

        hlt                     /* Crash if somehow `exit' does return.  */
END (_start)

sysdeps/x86_64/start.S                                        48,1-8         87%
```

Auxiliary vector & base address of a program

Dynamically-linked program

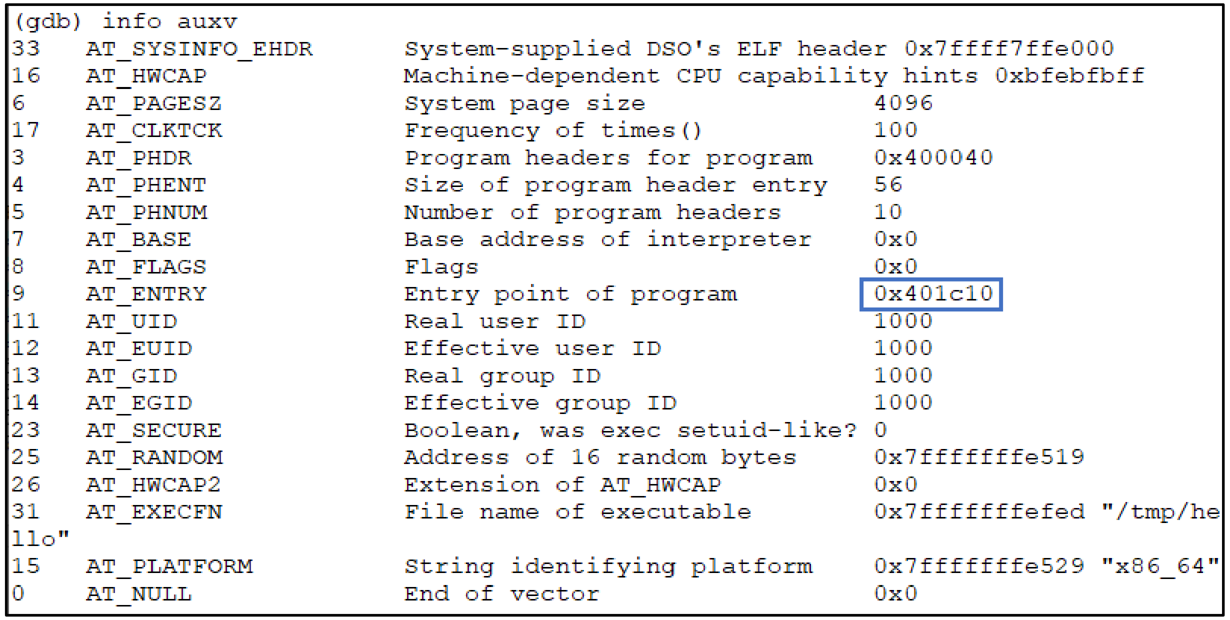


```
(gdb) info auxv
33   AT_SYSINFO_EHDR      System-supplied DSO's ELF header 0x7ffff7fce000
16   AT_HWCAP             Machine-dependent CPU capability hints 0xbfebfbff
6    AT_PAGESZ            System page size               4096
17   AT_CLKTCK            Frequency of times()           100
3    AT_PHDR              Program headers for program    0x555555554040
4    AT_PHENT             Size of program header entry   56
5    AT_PHNUM             Number of program headers      13
7    AT_BASE              Base address of interpreter    0x7ffff7fcf000
8    AT_FLAGS             Flags                          0x0
9    AT_ENTRY             Entry point of program         0x555555555060
11   AT_UID               Real user ID                   1000
12   AT_EUID              Effective user ID              1000
13   AT_GID               Real group ID                  1000
14   AT_EGID              Effective group ID             1000
23   AT_SECURE            Boolean, was exec setuid-like? 0
25   AT_RANDOM            Address of 16 random bytes     0x7fffffffe519
26   AT_HWCAP2            Extension of AT_HWCAP          0x0
31   AT_EXECFN            File name of executable        0x7fffffffefed "/tmp/he
llo"
15   AT_PLATFORM          String identifying platform    0x7fffffffe52a "x86_64"
0    AT_NULL              End of vector                  0x0
```

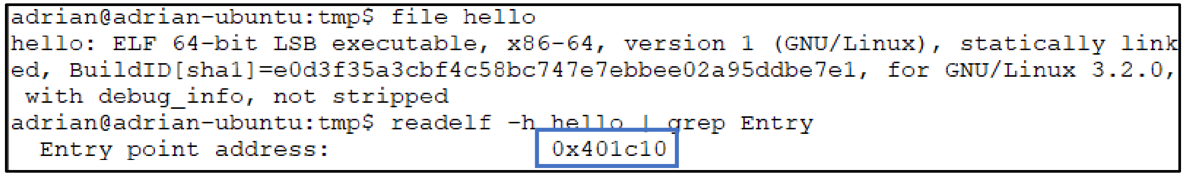
```
adrian@adrian-ubuntu:tmp$ file hello
hello: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linke
d, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=876ff6f59f9f9a787d70fc
1bc20f74198eac2edf, for GNU/Linux 3.2.0, with debug_info, not stripped
adrian@adrian-ubuntu:tmp$ readelf -h hello | grep Entry
  Entry point address:               0x1060
```

```
(gdb) info proc mappings
process 58017
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x555555554000     0x555555555000     0x1000        0x0 /tmp/hello
      0x555555555000     0x555555556000     0x1000     0x1000 /tmp/hello
      0x555555556000     0x555555557000     0x1000     0x2000 /tmp/hello
      0x555555557000     0x555555559000     0x2000     0x2000 /tmp/hello
      ...
```

Statically-linked program



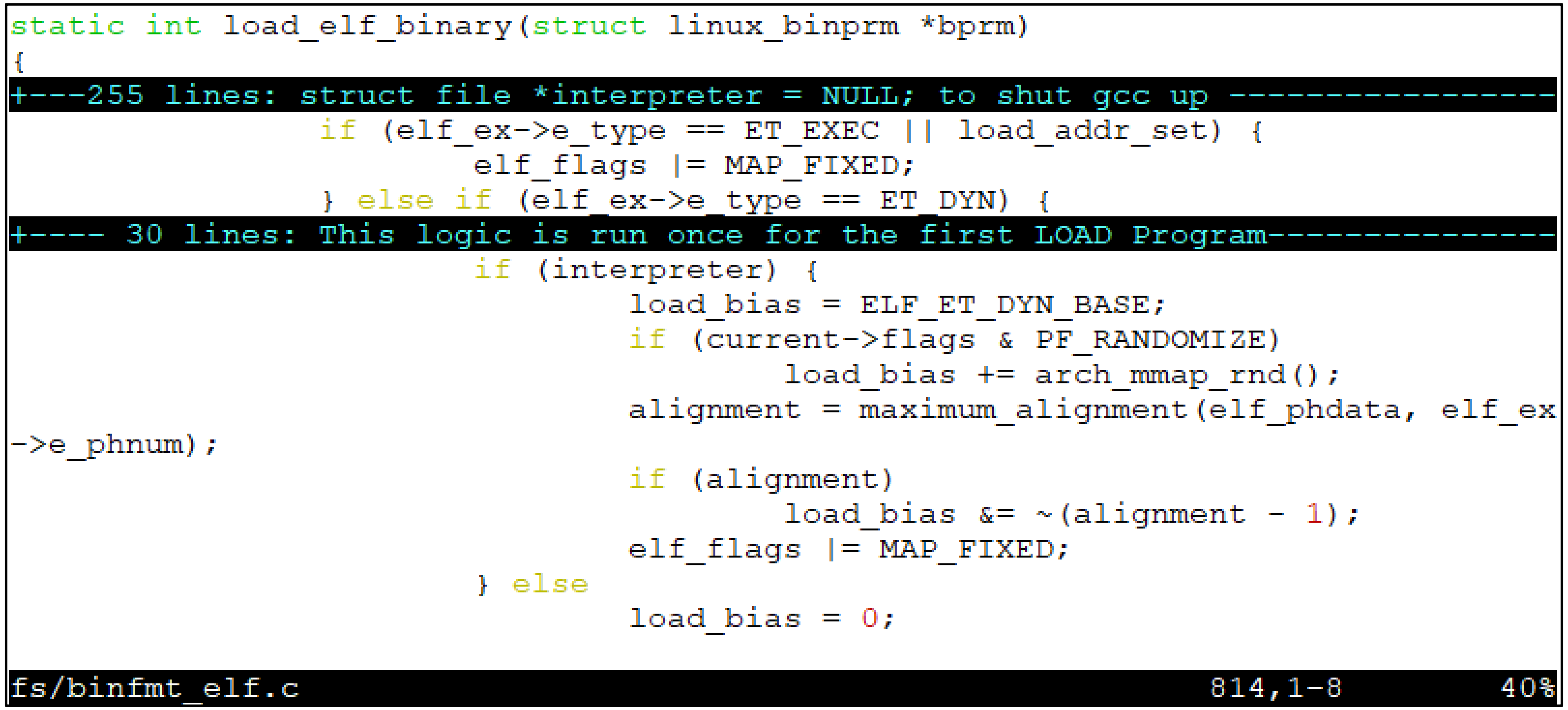
```
(gdb) info auxv
33   AT_SYSINFO_EHDR      System-supplied DSO's ELF header 0x7ffff7ffe000
16   AT_HWCAP             Machine-dependent CPU capability hints 0xbfebfbff
6    AT_PAGESZ            System page size               4096
17   AT_CLKTCK            Frequency of times()           100
3    AT_PHDR              Program headers for program    0x400040
4    AT_PHENT             Size of program header entry   56
5    AT_PHNUM             Number of program headers      10
7    AT_BASE              Base address of interpreter    0x0
8    AT_FLAGS             Flags                          0x0
9    AT_ENTRY             Entry point of program         0x401c10
11   AT_UID               Real user ID                   1000
12   AT_EUID              Effective user ID              1000
13   AT_GID               Real group ID                  1000
14   AT_EGID              Effective group ID             1000
23   AT_SECURE            Boolean, was exec setuid-like? 0
25   AT_RANDOM            Address of 16 random bytes     0x7fffffffe519
26   AT_HWCAP2            Extension of AT_HWCAP          0x0
31   AT_EXECFN            File name of executable        0x7fffffffefed "/tmp/he
llo"
15   AT_PLATFORM          String identifying platform    0x7fffffffe529 "x86_64"
0    AT_NULL              End of vector                  0x0
```



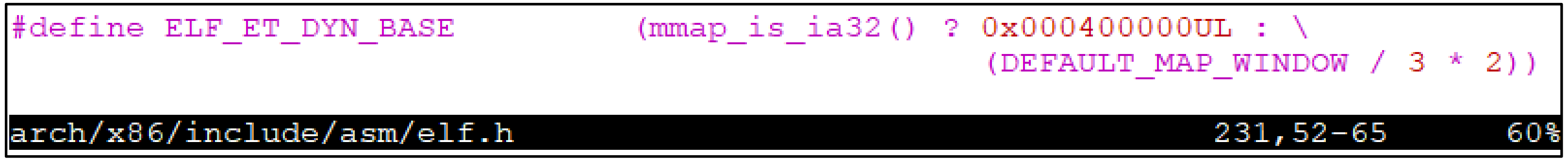
```
adrian@adrian-ubuntu:tmp$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically link
ed, BuildID[sha1]=e0d3f35a3cbf4c58bc747e7ebbee02a95ddbe7e1, for GNU/Linux 3.2.0,
 with debug_info, not stripped
adrian@adrian-ubuntu:tmp$ readelf -h hello | grep Entry
  Entry point address:               0x401c10
```

Why is the base address '0x555555554000' for a dynamically-linked program?

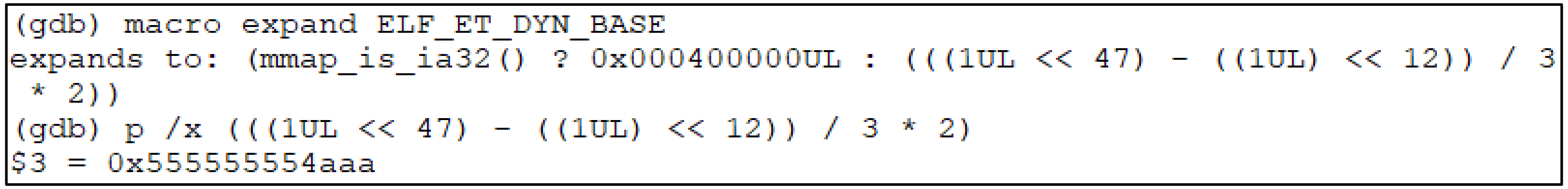
Why is the base address ‘0x555555554000’ for a dynamically-linked program?



```
static int load_elf_binary(struct linux_binprm *bprm)
{
+---255 lines: struct file *interpreter = NULL; to shut gcc up ------------------
                if (elf_ex->e_type == ET_EXEC || load_addr_set) {
                        elf_flags |= MAP_FIXED;
                } else if (elf_ex->e_type == ET_DYN) {
+---- 30 lines: This logic is run once for the first LOAD Program---------------
                        if (interpreter) {
                                load_bias = ELF_ET_DYN_BASE;
                                if (current->flags & PF_RANDOMIZE)
                                        load_bias += arch_mmap_rnd();
                                alignment = maximum_alignment(elf_phdata, elf_ex
->e_phnum);
                                if (alignment)
                                        load_bias &= ~(alignment - 1);
                                elf_flags |= MAP_FIXED;
                        } else
                                load_bias = 0;

fs/binfmt_elf.c                                                  814,1-8     40%
```

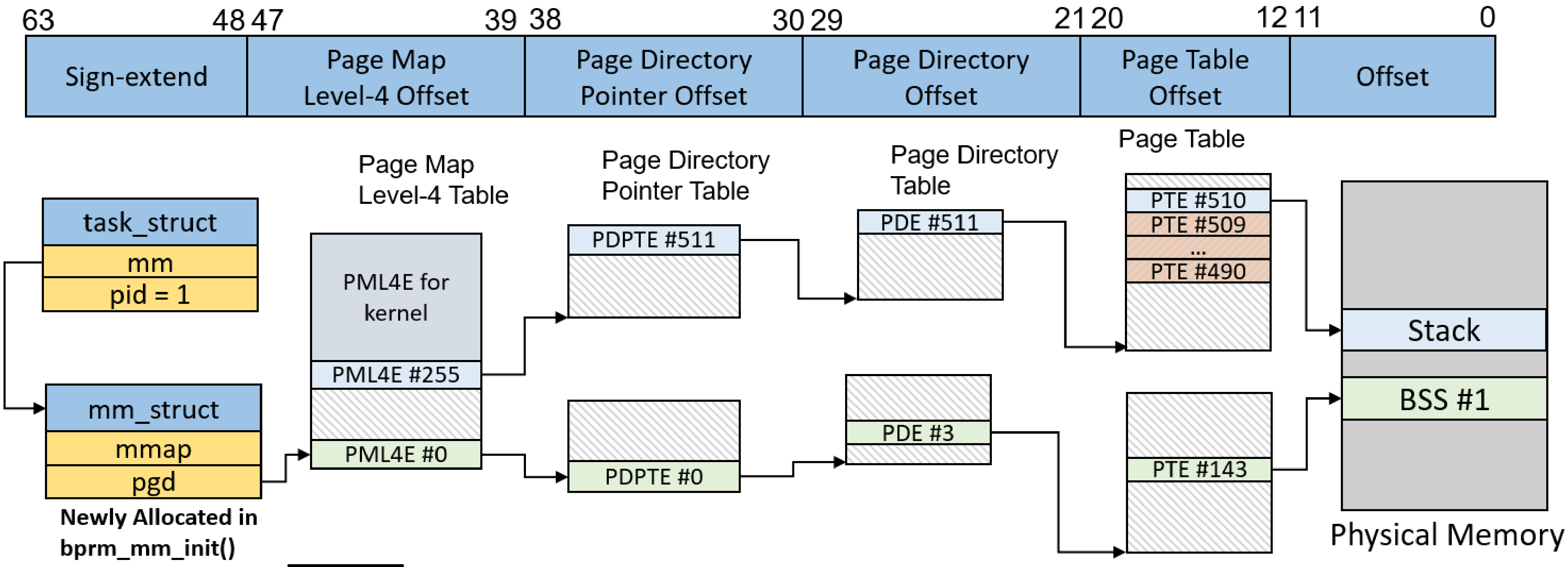


```
#define ELF_ET_DYN_BASE         (mmap_is_ia32() ? 0x000400000UL : \
                                                  (DEFAULT_MAP_WINDOW / 3 * 2))

arch/x86/include/asm/elf.h                                       231,52-65     60%
```



```
(gdb) macro expand ELF_ET_DYN_BASE
expands to: (mmap_is_ia32() ? 0x000400000UL : (((1UL << 47) - ((1UL) << 12)) / 3
 * 2))
(gdb) p /x (((1UL << 47) - ((1UL) << 12)) / 3 * 2)
$3 = 0x555555554aaa
```

Demand paging

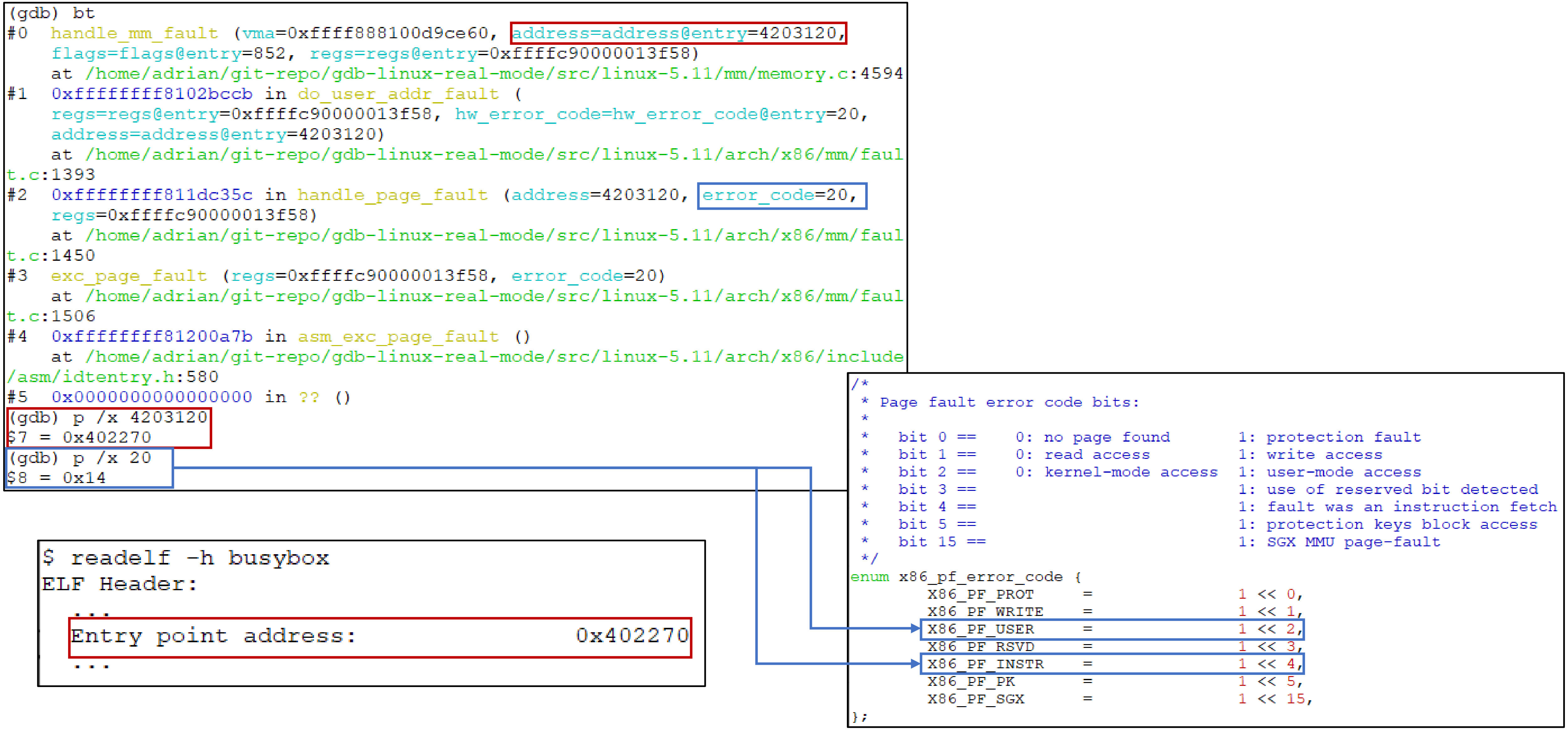


Legend

- ▭ Pre-allocated by handle_mm_fault() if it is not existed
- ▨ Will be allocated if page fault occurs
- 🟩 Allocated when page fault occurs [padzero()->clear_user()]

Demand paging: copy a disk page into physical memory if a page fault occurs

Demand paging



```
(gdb) bt
#0  handle_mm_fault (vma=0xffff888100d9ce60, address=address@entry=4203120,
    flags=flags@entry=852, regs=regs@entry=0xffffc90000013f58)
    at /home/adrian/git-repo/gdb-linux-real-mode/src/linux-5.11/mm/memory.c:4594
#1  0xffffffff8102bccb in do_user_addr_fault (
    regs=regs@entry=0xffffc90000013f58, hw_error_code=hw_error_code@entry=20,
    address=address@entry=4203120)
    at /home/adrian/git-repo/gdb-linux-real-mode/src/linux-5.11/arch/x86/mm/faul
t.c:1393
#2  0xffffffff811dc35c in handle_page_fault (address=4203120, error_code=20,
    regs=0xffffc90000013f58)
    at /home/adrian/git-repo/gdb-linux-real-mode/src/linux-5.11/arch/x86/mm/faul
t.c:1450
#3  exc_page_fault (regs=0xffffc90000013f58, error_code=20)
    at /home/adrian/git-repo/gdb-linux-real-mode/src/linux-5.11/arch/x86/mm/faul
t.c:1506
#4  0xffffffff81200a7b in asm_exc_page_fault ()
    at /home/adrian/git-repo/gdb-linux-real-mode/src/linux-5.11/arch/x86/include
/asm/idtentry.h:580
#5  0x0000000000000000 in ?? ()
(gdb) p /x 4203120
$7 = 0x402270
(gdb) p /x 20
$8 = 0x14
```

```
$ readelf -h busybox
ELF Header:
  ...
  Entry point address:               0x402270
  ...
```

```
/*
 * Page fault error code bits:
 *
 *   bit 0 ==    0: no page found       1: protection fault
 *   bit 1 ==    0: read access         1: write access
 *   bit 2 ==    0: kernel-mode access  1: user-mode access
 *   bit 3 ==                           1: use of reserved bit detected
 *   bit 4 ==                           1: fault was an instruction fetch
 *   bit 5 ==                           1: protection keys block access
 *   bit 15 ==                          1: SGX MMU page-fault
 */
enum x86_pf_error_code {
        X86_PF_PROT     =               1 << 0,
        X86_PF_WRITE    =               1 << 1,
        X86_PF_USER     =               1 << 2,
        X86_PF_RSVD     =               1 << 3,
        X86_PF_INSTR    =               1 << 4,
        X86_PF_PK       =               1 << 5,
        X86_PF_SGX      =               1 << 15,
};
```

Demand paging: copy a disk page into physical memory if a page fault occurs