rongguang ou

csc 139

9/24/19

Programming Assignment 1 – Linux Kernel Modules

Define (in your own words) the function of each unique system call in the trace. There should be about 10 unique system calls in the trace.

execve()

int execve(const char* path , char* const argv[], char* const envp[])

Description : replace current process with new process at location pointed by <path>. The argv is a pointer to where all the args are stored if any was passed into the function. parameter <envp> is a pointer to environment variables. The return value will be -1 if there are error, 0 otherwise.

brk()

void * brk(const void* addr)

Description : brk set lowest address of a process's data(uninitializaed data) to address immmediately above bss. This essentially sets the program break to allow more/less space available for allocation dependong on the input. The return value will be a pointer to the new of memory if successful; otherwise returns -1 with <errno>.

access()

int access(const char* pathname, int mode)

Description : checks whether the calling process can access the file specified in the parameter <pathname> with certain mode such as R_OK,W_OK,X_OK if file exists and grants read,write and execute respectively. Function returns 0 if success to access at a given <mode> else return -1 and errno is also returned for what the cause of the error might be.

open()

int open(const char* pathname, int flags, mode_t mode)

Description: opens the file at located at “pathname” with particular mode. If the file does not exist or encounters error during open then the return value will be negative with error hint indicating why it was

not able to open the file. If the file was able to open successfully then the return value will be a file descriptor(a non negative value). Second parameter indicates if the file should be open as readonly,writeonly or both.

close()

int close(int fd)

close() closes a file descriptor so OS can reuse them for future open(). close() will return 0 on success else -1.

fstat64

int stat(const char* path, struct stat* buf)

Description: Returns file status that is information about the status of the file specified by path. First parameter is also the file descriptor. Second parameter is a struct pointer that is the location in which the information of the path file will be written to. This version of the function is used for file size larger than 2gb. If the file information was able to obtain and write to buf then return value will be 0.

read()

ssize_t read(int fd, void* buf, size_t count)

Description: read() takes a file description <fd> and attempts to read up to <count> bytes into <buf>. read() returns number of bytes read else return -1 if error is found and errno is set appropriately.

set_thread_area()

int set_thread_area(struct user_desc *u_info)

Sets thread local storage entry to value pointed by info->entry_numberd. When entry_number is -1 then set_thread_area() will free TLS entry. If set_thread_area() finds a free TLS entry, the value of u_info->entry_number is set upon return to show which entry was changed.

write()

ssize_t write(int fd, const void* buf, size_t count)

write count bytes from source pointed by buf to destination file pointed by fd. On success, returns the number of bytes written successfully else -1 if any error.

exit_group()

exit_group(int status)

kills all threads in a process.

Write a description of the process loading protocol in terms of the sequence of system calls in strace.linux. With the exception of mmap, mprotect, and munmap you should be able to provide a detailed description of what the system call is doing and what object it is operating on. This must answer the question: What is the *purpose* of the operation and the object (file) it is operating on?

//replace current process code with the code in "pr1_c89_32" passing in also argv for 2nd parameter and list of environment variables. the return value is 0 meaning no error.

execve("./pr1_c89_32", ["./pr1_c89_32"], [/* 49 vars */]) = 0

// sets memory address of the lowest address of data above bss. return value is the starting address of the data block.

brk(0) = 0x81ef000

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77f7000

// check if the file located at "/etc/ld.so.preload" has read access , -1 means error due to either no such file or false directory

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

// open the file located at "/etc/ld.so.cache" with read only access , file descriptor of 3 is returned

open("/etc/ld.so.cache", O_RDONLY) = 3

// get file 3's information that was just opened previously , return value of 0 means no error.

fstat64(3, {st_mode=S_IFREG|0644, st_size=92826, ...}) = 0

//

mmap2(NULL, 92826, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb77e0000

// close file descriptor 3 , return value 0 means success.

close(3) = 0

// open file "/lib/libm.so.6" for reach only access , returned file descriptor 3

open("/lib/libm.so.6", O_RDONLY) = 3

// Read 512 bytes from file 3 to buffer that's the second arg passed in.

read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\244q\0004\0\0\0"..., 512) = 512

// Obtain file 3's status and write status to struct pointed by 2nd arg , return value is 0 means success.

fstat64(3, {st_mode=S_IFREG|0755, st_size=202680, ...}) = 0

mmap2(0x717000, 168064, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x717000

mmap2(0x73f000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x27000) = 0x73f000

//close file 3 , return value is 0 means close without error

close(3) = 0

//open file "/lib/libc.so.6" for read only , returned 3 as the file descriptor number

open("/lib/libc.so.6", O_RDONLY) = 3

//Read 512 bytes from file 3 to second arg the buffer.

read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\200^V\0004\0\0\0"..., 512) = 512

//Get file 3's status to second arg, return value of 0 means operation success.

fstat64(3, {st_mode=S_IFREG|0755, st_size=1912920, ...}) = 0

mmap2(0x54f000, 1665484, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x1f2000

mmap2(0x383000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x191000) = 0x383000

mmap2(0x386000, 10700, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x386000

//Close file 3 , return value of 0 means operation success.

close(3) = 0

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77df000

//Sets a TLS entry passing in ->6 and use base address of 0xb77df6c0. Return value 0 means operation success.

set_thread_area({entry_number:-1 -> 6, base_addr:0xb77df6c0, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0

mprotect(0x383000, 8192, PROT_READ) = 0

mprotect(0x73f000, 4096, PROT_READ) = 0

mprotect(0x54b000, 4096, PROT_READ) = 0

munmap(0xb77e0000, 92826) = 0

//Get file 1 status and write to 2nd arg, return value is 0 means operation success.

fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 11), ...}) = 0

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77f6000

// Write 50 bytes from src "0x0000000804a954 (4)   "... to file 1.

write(1, "0x0000000804a954 (4)   "..., 50) = 50

write(1, "0x0000000804a958 (4)   "..., 50) = 50

write(1, "0x0000000804a92c (4)   stati"..., 50) = 50

write(1, "0x0000000804a930 (4)   stati"..., 50) = 50

write(1, "0x0000000804a960 (4)   "..., 50) = 50

write(1, "0x0000000804a95c (4)   "..., 50) = 50

write(1, "0x000000bfc324e4 (4)   lo"..., 58) = 58

write(1, "0x000000bfc324e0 (4)   lo"..., 56) = 56

write(1, "0x0000000804a944 (4) static_lo"..., 50) = 50

write(1, "0x0000000804a948 (4) static_lo"..., 50) = 50

write(1, "0x000000bfc3248c (4)   "..., 58) = 58

write(1, "0x000000bfc32488 (4)   "..., 56) = 56

write(1, "0x0000000804a93c (4)   static_"..., 50) = 50

write(1, "0x0000000804a940 (4)   static_"..., 50) = 50

write(1, "0x000000bfc3248c (4)   l"..., 58) = 58

write(1, "0x000000bfc32488 (4)   l"..., 56) = 56

write(1, "0x0000000804a934 (4)   static_l"..., 50) = 50

write(1, "0x0000000804a938 (4)   static_l"..., 50) = 50

write(1, "0x000000bfc3248c (4)   "..., 58) = 58

write(1, "0x000000bfc32488 (4)   "..., 56) = 56

write(1, "0x0000000804a94c (4)   static_"..., 50) = 50

write(1, "0x0000000804a950 (4)   static_"..., 50) = 50

write(1, "0x000000bfc324a0 (4)   test_"..., 50) = 50

write(1, "0x000000bfc324a4 (4)   test_"..., 50) = 50

write(1, "0x000000bfc324a8 (4)   test_"..., 50) = 50

write(1, "0x000000bfc3238c (4)   test_r"..., 50) = 50

write(1, "0x000000bfc323a0 (4)   test"..., 50) = 50

write(1, "0x000000bfc323cc (4)   test_r"..., 50) = 50

write(1, "0x000000bfc323e0 (4)   test"..., 50) = 50

write(1, "0x000000bfc3240c (4)   test_r"..., 50) = 50

write(1, "0x000000bfc32420 (4)   test"..., 50) = 50

write(1, "0x000000bfc3244c (4)   test_r"..., 50) = 50

write(1, "0x000000bfc32460 (4)   test"..., 50) = 50

write(1, "0x000000bfc3248c (4)   test_r"..., 50) = 50

write(1, "0x000000bfc324a0 (4)   test"..., 50) = 50

write(1, "0x00000008048450   "..., 47) = 47

write(1, "0x00000008048460   "..., 47) = 47

write(1, "0x00000008048440   "..., 47) = 47

write(1, "0x00000008048430   "..., 47) = 47

write(1, "0x00000008048570   "..., 47) = 47

write(1, "0x00000008048f9a   "..., 47) = 47

write(1, "0x00000008049057   sta"..., 47) = 47

write(1, "0x0000000804921c   "..., 47) = 47

write(1, "0x00000008049114   test"..., 47) = 47

write(1, "0x0000000804919d   "..., 47) = 47

write(1, "0x0000000804950a (4)   "..., 54) = 54

write(1, "0x000000080494fd (7)   "..., 57) = 57

write(1, "0x000000080494fd (7)   local_m"..., 57) = 57

write(1, "0x000000bfc324dc (4)   local_m"..., 65) = 65

write(1, "0x000000bfc32510 (4)   "..., 50) = 50

write(1, "0x000000bfc32514 (4)   "..., 65) = 65

write(1, "0x000000bfc325b4 (4)   "..., 67) = 67

write(1, "0x000000bfc333b6 (13)   "..., 67) = 67

write(1, "0x000000bfc325b8 (4)   "..., 67) = 67

//Set a new base memory address for data section. return address is the starting address of the lowest address of data(uninitialized data)

brk(0)   = 0x81ef000

//Same as before, except this time the new address will be 0x8210000

brk(0x8210000) = 0x8210000

write(1, "0x0000000804a90c (8) "..., 58) = 58

write(1, "0x000000bfc32518 (4) "..., 65) = 65

write(1, "0x000000bfc325bc (4) "..., 67) = 67

write(1, "0x000000bfc333c3 (17) "..., 71) = 71

write(1, "0x000000bfc325c0 (4) "..., 67) = 67

write(1, "0x000000bfc333d4 (9) "..., 63) = 63

write(1, "0x000000bfc325c4 (4) "..., 67) = 67

write(1, "0x000000bfc333dd (12) "..., 66) = 66

write(1, "0x000000bfc325c8 (4) "..., 67) = 67

write(1, "0x000000bfc333e9 (29) "..., 83) = 83

write(1, "0x000000bfc325cc (4) "..., 67) = 67

write(1, "0x000000bfc33406 (228) "..., 283) = 283

write(1, "0x000000bfc325d0 (4) "..., 67) = 67

write(1, "0x000000bfc334ea (19) "..., 73) = 73

write(1, "0x000000bfc325d4 (4) "..., 67) = 67

write(1, "0x000000bfc334fd (15) "..., 69) = 69

write(1, "0x000000bfc325d8 (4) "..., 67) = 67

write(1, "0x000000bfc3350c (35) "..., 89) = 89

write(1, "0x000000bfc325dc (4) "..., 67) = 67

write(1, "0x000000bfc3352f (53) "..., 107) = 107

write(1, "0x000000bfc325e0 (4) "..., 67) = 67

write(1, "0x000000bfc33564 (20) "..., 74) = 74

write(1, "0x000000bfc325e4 (4) "..., 68) = 68

write(1, "0x000000bfc33578 (20) "..., 75) = 75

write(1, "0x000000bfc325e8 (4) "..., 68) = 68

write(1, "0x000000bfc3358c (20) "..., 75) = 75

write(1, "0x000000bfc325ec (4) "..., 68) = 68

write(1, "0x000000bfc335a0 (13) "..., 68) = 68

write(1, "0x000000bfc325f0 (4)   "..., 68) = 68

write(1, "0x000000bfc335ad (13)   "..., 68) = 68

write(1, "0x000000bfc325f4 (4)   "..., 68) = 68

write(1, "0x000000bfc335ba (14)   "..., 69) = 69

write(1, "0x000000bfc325f8 (4)   "..., 68) = 68

write(1, "0x000000bfc335c8 (8)   "..., 63) = 63

write(1, "0x000000bfc325fc (4)   "..., 68) = 68

write(1, "0x000000bfc335d0 (41)   "..., 96) = 96

write(1, "0x000000bfc32600 (4)   "..., 68) = 68

write(1, "0x000000bfc335f9 (13)   "..., 68) = 68

write(1, "0x000000bfc32604 (4)   "..., 68) = 68

write(1, "0x000000bfc33606 (25)   "..., 80) = 80

write(1, "0x000000bfc32608 (4)   "..., 68) = 68

write(1, "0x000000bfc3361f (26)   "..., 81) = 81

write(1, "0x000000bfc3260c (4)   "..., 68) = 68

write(1, "0x000000bfc33639 (29)   "..., 84) = 84

write(1, "0x000000bfc32610 (4)   "..., 68) = 68

write(1, "0x000000bfc33656 (1549)   "..., 1024) = 1024

write(1, "8;5;13:*.ogm=38;5;13:*.mp4=38;5;"..., 582) = 582

write(1, "0x000000bfc32614 (4)   "..., 68) = 68

write(1, "0x000000bfc33c63 (12)   "..., 67) = 67

write(1, "0x000000bfc32618 (4)   "..., 68) = 68

write(1, "0x000000bfc33c6f (21)   "..., 76) = 76

write(1, "0x000000bfc3261c (4)   "..., 68) = 68

write(1, "0x000000bfc33c84 (51)   "..., 106) = 106

write(1, "0x000000bfc32620 (4)   "..., 68) = 68

write(1, "0x000000bfc33cb7 (35)   "..., 90) = 90

write(1, "0x000000bfc32624 (4)   "..., 68) = 68

write(1, "0x000000bfc33cda (22)   "..., 77) = 77

write(1, "0x000000bfc32628 (4)   "..., 68) = 68

write(1, "0x000000bfc33cf0 (30) "..., 85) = 85

write(1, "0x000000bfc3262c (4) "..., 68) = 68

write(1, "0x000000bfc33d0e (26) "..., 81) = 81

write(1, "0x000000bfc32630 (4) "..., 68) = 68

write(1, "0x000000bfc33d28 (68) "..., 123) = 123

write(1, "0x000000bfc32634 (4) "..., 68) = 68

write(1, "0x000000bfc33d6c (15) "..., 70) = 70

write(1, "0x000000bfc32638 (4) "..., 68) = 68

write(1, "0x000000bfc33d7b (29) "..., 84) = 84

write(1, "0x000000bfc3263c (4) "..., 68) = 68

write(1, "0x000000bfc33d98 (90) "..., 145) = 145

write(1, "0x000000bfc32640 (4) "..., 68) = 68

write(1, "0x000000bfc33df2 (46) "..., 101) = 101

write(1, "0x000000bfc32644 (4) "..., 68) = 68

write(1, "0x000000bfc33e20 (42) "..., 97) = 97

write(1, "0x000000bfc32648 (4) "..., 68) = 68

write(1, "0x000000bfc33e4a (28) "..., 83) = 83

write(1, "0x000000bfc3264c (4) "..., 68) = 68

write(1, "0x000000bfc33e66 (40) "..., 95) = 95

write(1, "0x000000bfc32650 (4) "..., 68) = 68

write(1, "0x000000bfc33e8e (24) "..., 79) = 79

write(1, "0x000000bfc32654 (4) "..., 68) = 68

write(1, "0x000000bfc33ea6 (29) "..., 84) = 84

write(1, "0x000000bfc32658 (4) "..., 68) = 68

write(1, "0x000000bfc33ec3 (46) "..., 101) = 101

write(1, "0x000000bfc3265c (4) "..., 68) = 68

write(1, "0x000000bfc33ef1 (44) "..., 99) = 99

write(1, "0x000000bfc32660 (4) "..., 68) = 68

write(1, "0x000000bfc33f1d (54) "..., 109) = 109

write(1, "0x000000bfc32664 (4) "..., 68) = 68

write(1, "0x000000bfc33f53 (20) "..., 75) = 75

write(1, "0x000000bfc32668 (4) "..., 68) = 68

write(1, "0x000000bfc33f67 (38) "..., 93) = 93

write(1, "0x000000bfc3266c (4) "..., 68) = 68

write(1, "0x000000bfc33f8d (20) "..., 75) = 75

write(1, "0x000000bfc32670 (4) "..., 68) = 68

write(1, "0x000000bfc33fa1 (21) "..., 76) = 76

write(1, "0x000000bfc32674 (4) "..., 68) = 68

write(1, "0x000000bfc33fb6 (18) "..., 73) = 73

write(1, "0x000000bfc32678 (4) "..., 68) = 68

write(1, "0x000000bfc33fc8 (24) "..., 79) = 79

write(1, "0x000000bfc3267c (4) "..., 68) = 68

write(1, "0x000000bfc33fe0 (15) "..., 70) = 70

write(1, "0x000000bfc32680 (4) "..., 68) = 68

write(1, "0x0000000804a920 (4) "..., 65) = 65

write(1, "0x000000081ef008 (4) "..., 67) = 67

write(1, "0x000000bfc333c3 (17) "..., 71) = 71

write(1, "0x000000081ef00c (4) "..., 67) = 67

write(1, "0x000000bfc333d4 (9) "..., 63) = 63

write(1, "0x000000081ef010 (4) "..., 67) = 67

write(1, "0x000000bfc333dd (12) "..., 66) = 66

write(1, "0x000000081ef014 (4) "..., 67) = 67

write(1, "0x000000bfc333e9 (29) "..., 83) = 83

write(1, "0x000000081ef018 (4) "..., 67) = 67

write(1, "0x000000bfc33406 (228) "..., 283) = 283

write(1, "0x000000081ef01c (4) "..., 67) = 67

write(1, "0x000000bfc334ea (19) "..., 73) = 73

write(1, "0x000000081ef020 (4) "..., 67) = 67

write(1, "0x000000bfc334fd (15) "..., 69) = 69

write(1, "0x000000081ef024 (4) "..., 67) = 67

write(1, "0x000000bfc3350c (35) "..., 89) = 89

write(1, "0x000000081ef028 (4) "..., 67) = 67

write(1, "0x000000bfc3352f (53) "..., 107) = 107

write(1, "0x000000081ef02c (4) "..., 67) = 67

write(1, "0x000000bfc33564 (20) "..., 74) = 74

write(1, "0x000000081ef030 (4) "..., 68) = 68

write(1, "0x000000bfc33578 (20) "..., 75) = 75

write(1, "0x000000081ef034 (4) "..., 68) = 68

write(1, "0x000000bfc3358c (20) "..., 75) = 75

write(1, "0x000000081ef038 (4) "..., 68) = 68

write(1, "0x000000bfc335a0 (13) "..., 68) = 68

write(1, "0x000000081ef03c (4) "..., 68) = 68

write(1, "0x000000bfc335ad (13) "..., 68) = 68

write(1, "0x000000081ef040 (4) "..., 68) = 68

write(1, "0x000000bfc335ba (14) "..., 69) = 69

write(1, "0x000000081ef044 (4) "..., 68) = 68

write(1, "0x000000bfc335c8 (8) "..., 63) = 63

write(1, "0x000000081ef048 (4) "..., 68) = 68

write(1, "0x000000bfc335d0 (41) "..., 96) = 96

write(1, "0x000000081ef04c (4) "..., 68) = 68

write(1, "0x000000bfc335f9 (13) "..., 68) = 68

write(1, "0x000000081ef050 (4) "..., 68) = 68

write(1, "0x000000bfc33606 (25) "..., 80) = 80

write(1, "0x000000081ef054 (4) "..., 68) = 68

write(1, "0x000000bfc3361f (26) "..., 81) = 81

write(1, "0x000000081ef058 (4) "..., 68) = 68

write(1, "0x000000bfc33639 (29) "..., 84) = 84

write(1, "0x000000081ef05c (4) "..., 68) = 68

write(1, "0x000000bfc33656 (1549) "..., 1024) = 1024

write(1, "8;5;13:*.ogm=38;5;13:*.mp4=38;5;"..., 582) = 582

write(1, "0x000000081ef060 (4)   "..., 68) = 68

write(1, "0x000000bfc33c63 (12)   "..., 67) = 67

write(1, "0x000000081ef064 (4)   "..., 68) = 68

write(1, "0x000000bfc33c6f (21)   "..., 76) = 76

write(1, "0x000000081ef068 (4)   "..., 68) = 68

write(1, "0x000000bfc33c84 (51)   "..., 106) = 106

write(1, "0x000000081ef06c (4)   "..., 68) = 68

write(1, "0x000000bfc33cb7 (35)   "..., 90) = 90

write(1, "0x000000081ef070 (4)   "..., 68) = 68

write(1, "0x000000bfc33cda (22)   "..., 77) = 77

write(1, "0x000000081ef074 (4)   "..., 68) = 68

write(1, "0x000000bfc33cf0 (30)   "..., 85) = 85

write(1, "0x000000081ef078 (4)   "..., 68) = 68

write(1, "0x000000bfc33d0e (26)   "..., 81) = 81

write(1, "0x000000081ef07c (4)   "..., 68) = 68

write(1, "0x000000bfc33d28 (68)   "..., 123) = 123

write(1, "0x000000081ef080 (4)   "..., 68) = 68

write(1, "0x000000bfc33d6c (15)   "..., 70) = 70

write(1, "0x000000081ef084 (4)   "..., 68) = 68

write(1, "0x000000bfc33d7b (29)   "..., 84) = 84

write(1, "0x000000081ef088 (4)   "..., 68) = 68

write(1, "0x000000bfc33d98 (90)   "..., 145) = 145

write(1, "0x000000081ef08c (4)   "..., 68) = 68

write(1, "0x000000bfc33df2 (46)   "..., 101) = 101

write(1, "0x000000081ef090 (4)   "..., 68) = 68

write(1, "0x000000bfc33e20 (42)   "..., 97) = 97

write(1, "0x000000081ef094 (4)   "..., 68) = 68

write(1, "0x000000bfc33e4a (28)   "..., 83) = 83

write(1, "0x000000081ef098 (4)   "..., 68) = 68

write(1, "0x000000bfc33e66 (40)   "..., 95) = 95

write(1, "0x000000081ef09c (4)   "..., 68) = 68

write(1, "0x000000bfc33e8e (24)   "..., 79) = 79

write(1, "0x000000081ef0a0 (4)   "..., 68) = 68

write(1, "0x000000bfc33ea6 (29)   "..., 84) = 84

write(1, "0x000000081ef0a4 (4)   "..., 68) = 68

write(1, "0x000000bfc33ec3 (46)   "..., 101) = 101

write(1, "0x000000081ef0a8 (4)   "..., 68) = 68

write(1, "0x000000bfc33ef1 (44)   "..., 99) = 99

write(1, "0x000000081ef0ac (4)   "..., 68) = 68

write(1, "0x000000bfc33f1d (54)   "..., 109) = 109

write(1, "0x000000081ef0b0 (4)   "..., 68) = 68

write(1, "0x000000bfc33f53 (20)   "..., 75) = 75

write(1, "0x000000081ef0b4 (4)   "..., 68) = 68

write(1, "0x000000bfc33f67 (38)   "..., 93) = 93

write(1, "0x000000081ef0b8 (4)   "..., 68) = 68

write(1, "0x000000bfc33f8d (20)   "..., 75) = 75

write(1, "0x000000081ef0bc (4)   "..., 68) = 68

write(1, "0x000000bfc33fa1 (21)   "..., 76) = 76

write(1, "0x000000081ef0c0 (4)   "..., 68) = 68

write(1, "0x000000bfc33fb6 (18)   "..., 73) = 73

write(1, "0x000000081ef0c4 (4)   "..., 68) = 68

write(1, "0x000000bfc33fc8 (24)   "..., 79) = 79

write(1, "0x000000081ef0c8 (4)   "..., 68) = 68

write(1, "0x000000bfc33fe0 (15)   "..., 70) = 70

write(1, "0x000000081ef0cc (4)   "..., 68) = 68

write(1, "0x000000081ef0d8 (8)   "..., 63) = 63

write(1, "0x000000081ef0d0 (4)   "..., 68) = 68

mmap2(NULL, 1052672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb76de000

mmap2(NULL, 1073745920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x776dd000

mmap2(NULL, 2147487744, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

mmap2(NULL, 2147618816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

mmap2(NULL, 2097152, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x774dd000

munmap(0x774dd000, 143360) = 0

munmap(0x77600000, 905216) = 0

mprotect(0x77500000, 135168, PROT_READ|PROT_WRITE) = 0

mmap2(NULL, 2147487744, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

write(1, "0x000000bfc324d8 (4) "..., 65) = 65

write(1, "0x000000081ef100 "..., 50) = 50

write(1, "0x000000bfc324d4 (4) "..., 65) = 65

write(1, "0x000000081ef118 "..., 50) = 50

write(1, "0x000000bfc324d0 (4) "..., 65) = 65

write(1, "0x000000b76de008 "..., 50) = 50

write(1, "0x000000bfc324cc (4) "..., 65) = 65

write(1, "0x000000776dd008 "..., 50) = 50

write(1, "0x000000bfc324c8 (4) "..., 65) = 65

write(1, " (nil) "..., 50) = 50

write(1, "0x000000bfc324c4 (4) "..., 65) = 65

write(1, "0x00000077500470 "..., 50) = 50

//kill all thread in this process

exit_group(0) = ?

+++ exited with 0 +++

For mmap determine whether the system call is mapping data from a file (specify the file), providing a block of zero'd memory, or causing memory to be unmapped.

Mmap2 is providing a block of zero'd memory. The address is NULL so the kernel chooses the address at which to create the mapping. Because of MAP_ANONYMOUS the mapping is not backed by any file; its contents are initialized to zero.

munmap and mprotect affect specific memory segments in the process's address space. Identify the memory segment by specifying which mmap command mapped the memory initially. You can simply identify the mmap command by number (don't have to write it out). One mprotect appears not to correspond to the mmap'd memory

Mmap2() = 0x77452000 mapped the memory initially.

The libraries libc.so.* and libm.so.* are loaded (via mmap) after the executable by a mechanism known as *dynamic linking*. Define dynamic linking (in your own words), including a contrast with static linking.

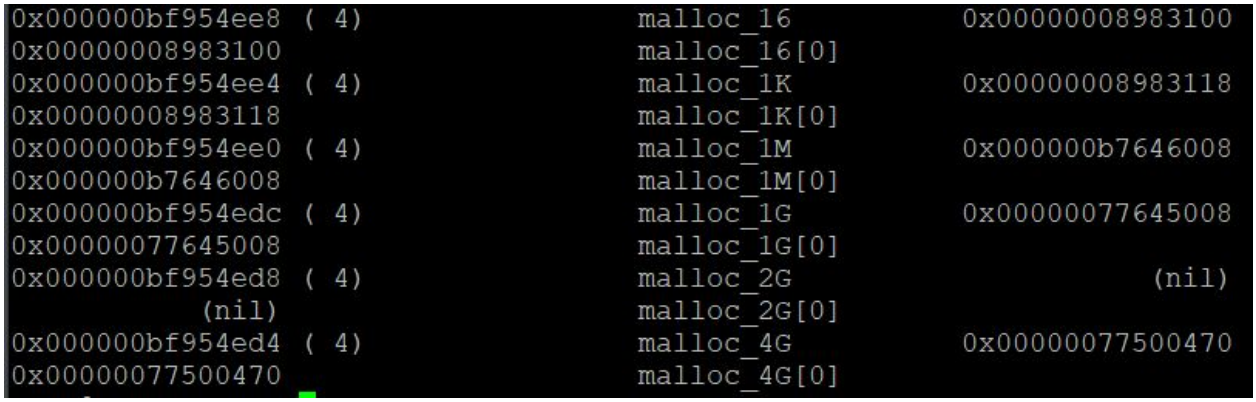
Dynamic Linking - Dynamic linking means only the names of the external library are placed in the final executable but the code is not. The code or actual linking happens during runtime whenever a section of code requires external library then will link at that time hence dynamic linking. There is only one copy of the shared library and so the size of the executable is reduced. Dynamically linking library can be updated and recompiled. Contrast to static linking, all external library need to be copied during compile time. the size of the final executable is larger and if any of the library requires update, everything needs to be recompiled by the compiler including program code. Lastly, static linking is usually faster than dll.

Which variables are global? Which variables are local? What is difference in their memory locations?

```
0x0000000804a954 ( 4)            global_var_1          0
0x0000000804a958 ( 4)            global_var_2          0
0x0000000804a92c ( 4)     static_global_var_1          0
0x0000000804a930 ( 4)     static_global_var_2          0
0x0000000804a960 ( 4)            extern_var_1          0
0x0000000804a95c ( 4)            extern_var_2          0
0x000000bf954ef4 ( 4)        local_main_var_1          134513409
0x000000bf954ef0 ( 4)        local_main_var_2          7209432
0x0000000804a944 ( 4)  static_local_main_var_1          0
0x0000000804a948 ( 4)  static_local_main_var_2          0
0x000000bf954e9c ( 4)          local_lf_var_1          134517693
0x000000bf954e98 ( 4)          local_lf_var_2          7219552
0x0000000804a93c ( 4)    static_local_lf_var_1          0
0x0000000804a940 ( 4)    static_local_lf_var_2          0
0x000000bf954e9c ( 4)         local_slf_var_1          134517693
0x000000bf954e98 ( 4)         local_slf_var_2          7219552
0x0000000804a934 ( 4)   static_local_slf_var_1          0
0x0000000804a938 ( 4)   static_local_slf_var_2          0
0x000000bf954e9c ( 4)          local_ef_var_1          134517693
0x000000bf954e98 ( 4)          local_ef_var_2          7219552
0x0000000804a94c ( 4)    static_local_ef_var_1          0
0x0000000804a950 ( 4)    static_local_ef_var_2          0
```

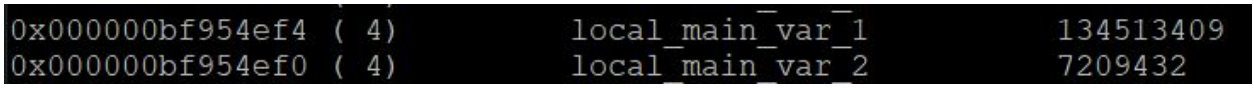
global and static variables are assigned at lower address than local variables.

Allocate a variable using malloc(). Where is this variable located in the address space relative to the others?



malloc variables have higher range address compare to local var indicate they are located in the stack

Where are local variables allocated?



local vars that have value have higher address range compare to ones that is not initialized.

How about function arguments (argv, argc)?

function args are assigned in the stack area with local vars. in the 0x000000bfxxxxxx range

What about the environment variables? You should familiarize yourself with environment variables.

```
0x000000bf954f28 ( 4)                    envp          0x000000bf954fcc
0x000000bf954fcc ( 4)                    envp[0]       0x000000bf9564d2
0x000000bf9564d2 ( 9)                    envp[0]->     "USER=our"
0x000000bf954fd0 ( 4)                    envp[1]       0x000000bf9564db
0x000000bf9564db (12)                    envp[1]->     "LOGNAME=our"
0x000000bf954fd4 ( 4)                    envp[2]       0x000000bf9564e7
0x000000bf9564e7 (29)                    envp[2]->     "HOME=/gaia/class/student/our"
0x000000bf954fd8 ( 4)                    envp[3]       0x000000bf956504
0x000000bf956504 (228)                    envp[3]->    "PATH=/bin:/usr/bin:/sbin:/usr/sbin:/u
X11R6/bin:/usr/pkg/fm/bin:/usr/pkg/mv/bin:/usr/pkg/pts/bin:/usr/pkg/sml/bin:/usr/pkg/syn/bin:/
0x000000bf954fdc ( 4)                    envp[4]       0x000000bf9565e8
0x000000bf9565e8 (19)                    envp[4]->     "MAIL=/var/mail/our"
0x000000bf954fe0 ( 4)                    envp[5]       0x000000bf9565fb
0x000000bf9565fb (15)                    envp[5]->     "SHELL=/bin/csh"
0x000000bf954fe4 ( 4)                    envp[6]       0x000000bf95660a
0x000000bf95660a (34)                    envp[6]->     "SSH_CLIENT=67.161.185.31 57347 22"
```

envp also is allocated to stack area with in the range 0x000000bf95xxxx