3. User Space Allocation

Young W. Lim

2021-09-11 Sat

Outline

Based on

"Study of ELF loading and relocs", 1999
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

Compling 32-bit program on 64-bit gcc

- `gcc -v`
- `gcc -m32 t.c`
- `sudo apt-get install gcc-multilib`
- `sudo apt-get install g++-multilib`
- `gcc-multilib`
- `g++-multilib`
- `gcc -m32`
- `objdump -m i386`

TOC: `mmap` system call

(1) memory-mapped file I/O

- system call that maps *files* or *devices* into memory.
- a method of memory-mapped file I/O
- implements demand paging
  - file contents are not read from disk directly and initially do not use physical RAM at all.
  - the actual reads from disk are performed in a *lazy* manner, after a specific location is accessed.

`https://en.wikiversity.org/wiki/File:ARM.2ASM.Interrupt.20210707.pdf`

(2) allocating memory

- mmap() is the standard way to allocate large amounts of memory from user space
- while mmap() is often used for files, the MAP_ANONYMOUS flag causes mmap() to allocate normal memory for the process
- the MAP_SHARED flag can make the allocated pages sharable with other processes

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

(3) munmap, mprotect, madvise

- after the memory is no longer needed,
  it is important to munmap the pointers to it.
- protection information can be managed using mprotect
- special treatment can be enforced using madvise

https://en.wikiversity.org/wiki/File:ARM.2ASM.Interrupt.20210707.pdf

(4) virtual address mapping

mmap system call

```
void *mmap(void *addr, size_t length,
           int prot, int flags,int fd, off_t offset);
int munmap(void *addr, size_t length);
```

- mmap() creates a new mapping in the virtual address space of the calling process.
- the address of the new mapping is returned as the result of the call.
    - addr : the starting address for the new mapping
    - length : the length of the new mapping (which must be greater than 0).

https://man7.org/linux/man-pages/man2/mmap.2.html

(5) input parameter addr

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- If addr is NULL,
  then the kernel chooses the (page-aligned) address
  at which to create the mapping;
  - this is the most portable method of creating a new mapping.
- If addr is not NULL,
  then the kernel takes it as a hint
  about where to place the mapping;

https://man7.org/linux/man-pages/man2/mmap.2.html

(6) non-NULL input parameter addr

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- the kernel will pick a nearby page boundary and attempt to create the mapping there. (but always above or equal to the value specified by `/proc/sys/vm/mmap_min_addr`)
- if another mapping already exists there, the kernel picks a new address that may or may not depend on the hint.

https://man7.org/linux/man-pages/man2/mmap.2.html

(7) file-backed mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- file-backed mapping maps
  an area of the process's virtual memory to files;
  i.e. reading those areas of memory causes the file to be read.
- the default mapping type.
- without `MAP_ANONYMOUS` flag

https://en.wikipedia.org/wiki/Mmap

(8) file backed mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- The contents of a file mapping are initialized using `length` bytes starting at offset `offset` in the file referred to by the file descriptor `fd`.
- `offset` must be a multiple of the page size
- after the `mmap()` call has returned, the file descriptor, `fd`, can be closed immediately without invalidating the mapping.

https://man7.org/linux/man-pages/man2/mmap.2.html

(9) anonymous mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- anonymous mapping maps
  an area of the process's virtual memory not backed by any file
- the contents are initialized to zero
  - similar to `malloc`, and is used
    in some `malloc` implementations for certain allocations.
  - However, anonymous mappings are not part of the POSIX standard,
    though implemented by almost all operating systems
    by the `MAP_ANONYMOUS` and `MAP_ANON` flags.

https://en.wikipedia.org/wiki/Mmap

(10) anonymous mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- the mapping is not backed by any file
- its contents are initialized to zero.
- the `fd` argument is ignored; however, some implementations require `fd` to be `-1` if `MAP_ANONYMOUS` (or `MAP_ANON`) is specified, and portable applications should ensure this.
- the `offset` argument should be `0`

https://man7.org/linux/man-pages/man2/mmap.2.html

(11) memory protection

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- the `prot` argument describes
  the desired memory protection of the mapping
- must not conflict with the open mode of the file
- either `PROT_NONE` or the bitwise OR of
  one or more of the following flags:
    - `PROT_EXEC` Pages may be executed.
    - `PROT_READ` Pages may be read.
    - `PROT_WRITE` Pages may be written.
    - `PROT_NONE` Pages may not be accessed.

https://man7.org/linux/man-pages/man2/mmap.2.html

(12) shared mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- the `MAP_SHARED` flag is set :
  if the mapping is shared, then the mapping
  is preserved across a `fork` system call.
  - changes in a mapped area in one process are immediately visible in all related (parent, child or sibling) processes.
  - if the mapping is shared and backed by a file (`MAP_SHARED` and not `MAP_ANONYMOUS`) the underlying file medium is only guaranteed to be written after it is `msync`'ed.

https://en.wikipedia.org/wiki/Mmap

(13) private mapping

mmap system call

```
void *mmap
(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

- the MAP_PRIVATE flag is set :
  if the mapping is private, the changes will
  neither be seen by other processes
  nor written to the file.

https://en.wikipedia.org/wiki/Mmap

(14) munmap system call

munmap system call

```
int munmap(void *addr, size_t length);
```

- deletes the mappings for the specified address range,
- causes further references to addresses within the range to generate invalid memory references
- the region is also automatically unmapped when the process is terminated
- just closing the file descriptor does not unmap the region.

https://linux.die.net/man/2/munmap

(15) munmap system call

munmap system call

```
int munmap(void *addr, size_t length);
```

- the address addr must be a multiple of the page size
- all pages containing a part of the indicated range are unmapped
- subsequent references to these pages will generate SIGSEGV.
- It is not an error if the indicated range does not contain any mapped pages

https://linux.die.net/man/2/munmap

TOC: mmap example I

Example I (1) code skeleton

mmap code skeleton

```
const char str1[] = "string 1";
const char str2[] = "string 2";
char *anon, *zero;

anon = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANON|MAP_SHARED, -1, 0);
zero = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

if (anon == MAP_FAILED || zero == MAP_FAILED)
   errx(1, "either mmap");

strcpy(anon, str1);
strcpy(zero, str1);

printf("PID %d:\tanonymous %s, zero-backed %s\n", parpid, anon, zero);
```

https://en.wikipedia.org/wiki/Mmap

Example I (2) file-backed mapping

file-backed mapping call example

```
fd = open("/dev/zero", O_RDWR, 0);
zero = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
```

- NULL addr : the kernel chooses the mapping address
- PROT_READ|PROT_WRITE : RW protection
- no MAP_ANONYMOUS flag : file-backed mapping
- MAP_SHARED : shared mapping across related processes
- fd returned by open("/dev/zero", O_RDWR, 0)
  backed file : /dev/zero
- offset : 0

https://en.wikipedia.org/wiki/Mmap

Example I (3) anonymous mapping

anonymous mapping call example

```
anon = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANON|MAP_SHARED, -1, 0);
```

- NULL addr : the kernel chooses the mapping address
- PROT_READ|PROT_WRITE : RW protection
- with MAP_ANONYMOUS flag : anonymous mapping
- MAP_SHARED : shared mapping across related processes
- fd=-1 : no backed file
- offset : 0

https://en.wikipedia.org/wiki/Mmap

Example I (4) arguments example

mmap system call example

```
void *mmap
  (void *addr, size_t length, int prot, int flags,int fd, off_t offset);

anon = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANON|MAP_SHARED, -1, 0);
zero = (char*) mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED,          fd, 0);

//      addr, length,               prot,       flag,          fd, offset

fd = open("/dev/zero", O_RDWR, 0));
```

- anon : anonymously mapped, RW, 4096 bytes area
- zero : file-backed mapped, RW, 4096 bytes area

https://en.wikipedia.org/wiki/Mmap

Example code I part (1)

mmap system call example

```
const char str1[] = "string 1";
const char str2[] = "string 2";
pid_t parpid = getpid(), childpid;
int fd = -1;
char *anon, *zero;

if ((fd = open("/dev/zero", O_RDWR, 0)) == -1) err(1, "open");

anon = (char*)mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANON|MAP_SHARED, -1, 0);
zero = (char*)mmap
       (NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

if (anon == MAP_FAILED || zero == MAP_FAILED)errx(1, "either mmap");
```

https://en.wikipedia.org/wiki/Mmap

Example code I part (2)

mmap system call example

```
strcpy(anon, str1);
strcpy(zero, str1);

printf("PID %d:\tanonymous %s, 0-backed %s\n", parpid, anon, zero);
```

- write str1 ("string 1") to anon
- write str2 ("string 2") to zero
- this example shows how an mmap of /dev/zero is equivalent to using anonymous memory not connected to any file.

https://en.wikipedia.org/wiki/Mmap

Example code I part (3)

mmap system call example

```
switch ((childpid = fork())) {
  case -1:
    err(1, "fork");
    /* NOTREACHED */
  case 0:
    childpid = getpid();
    printf("PID %d:\tanonymous %s, 0-backed %s\n", childpid, anon, zero);
    sleep(3);

    printf("PID %d:\tanonymous %s, 0-backed %s\n", childpid, anon, zero);
    munmap(anon, 4096);
    munmap(zero, 4096);
    close(fd);
    return EXIT_SUCCESS;
}
```

https://en.wikipedia.org/wiki/Mmap

Example code I part (4)

mmap system call example

```
sleep(2);
strcpy(anon, str2);
strcpy(zero, str2);

printf("PID %d:\tanonymous %s, 0-backed %s\n", parpid, anon, zero);
munmap(anon, 4096);
munmap(zero, 4096);
close(fd);
return EXIT_SUCCESS;
```

https://en.wikipedia.org/wiki/Mmap

Example code I output

mmap system call example

```
PID 22475:      anonymous string 1, zero-backed string 1
PID 22476:      anonymous string 1, zero-backed string 1
PID 22475:      anonymous string 2, zero-backed string 2
PID 22476:      anonymous string 2, zero-backed string 2
```

parent:	printf-1	sleep(2)	printf-2	
child:		printf-1	sleep(3)	printf-2

https://en.wikipedia.org/wiki/Mmap

TOC: mmap example II

Example II (1) code skeleton

mmap code skeleton

```
fd = open(argv[1], O_RDONLY);     // argv[1] : file name
offset = atoi(argv[2]);           // argv[2] : offset
length = atoi(argv[3]);           // argv[3] : length
length = sb.st_size - offset;

addr = mmap(NULL, length + offset - pa_offset, PROT_READ,
                MAP_PRIVATE, fd, pa_offset);

s = write(STDOUT_FILENO, addr + offset - pa_offset, length);
```

https://en.wikipedia.org/wiki/Mmap

Example II (2) stat.h

structure stat in sys/stat.h

dev_t	st_dev	ID of device containing file
ino_t	st_ino	file serial number
mode_t	st_mode	mode of file (see below)
nlink_t	st_nlink	number of links to the file
uid_t	st_uid	user ID of file
gid_t	st_gid	group ID of file
dev_t	st_rdev	device ID (if file is character or block special)
off_t	st_size	file size in bytes (if file is a regular file)
time_t	st_atime	time of last access
time_t	st_mtime	time of last data modification
time_t	st_ctime	time of last status change
blksize_t	st_blksize	a filesystem-specific preferred I/O block size for this object. In some filesystem types, this may vary from file to file
blkcnt_t	st_blocks	number of blocks allocated for this object

https://pubs.opengroup.org/onlinepubs/007908799/xsh/sysstat.h.html

Example II (3) sb

sb variable

```
#include <sys/stat.h>

struct stat sb;

    if (fstat(fd, &sb) == -1)           /* To obtain file size */

    if (offset >= sb.st_size) {

        if (offset + length > sb.st_size)
            length = sb.st_size - offset;

        length = sb.st_size - offset;
```

https://linux.die.net/man/2/munmap

Example II (4) fstat()

fstat()

- `int fstat(int fildes, struct stat *buf);`
- The `fstat()` function obtains information about an open file associated with the file descriptor `fildes`,
  and writes it to the area pointed to by `buf`.

https://pubs.opengroup.org/onlinepubs/007908799/xsh/fstat.html

Example code II part 1

mmap code part 1

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define handle_error(msg) \
    do { perror(msg); exit(EXIT_FAILURE); } while (0)
```

https://linux.die.net/man/2/munmap

Example code II part 2

mmap code part 2

```
int
main(int argc, char *argv[])
{
    char *addr;
    int fd;
    struct stat sb;
    off_t offset, pa_offset;
    size_t length;
    ssize_t s;
    if (argc < 3 || argc > 4) {
        fprintf(stderr, "%s file offset [length]\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    fd = open(argv[1], O_RDONLY);
    if (fd == -1)
        handle_error("open");
```

https://linux.die.net/man/2/munmap

Example code II part 3

mmap code part 3

```
if (fstat(fd, &sb) == -1)           /* To obtain file size */
    handle_error("fstat");
offset = atoi(argv[2]);
pa_offset = offset & ~(sysconf(_SC_PAGE_SIZE) - 1);
    /* offset for mmap() must be page aligned */
if (offset >= sb.st_size) {
    fprintf(stderr, "offset is past end of file\n");
    exit(EXIT_FAILURE);
}
if (argc == 4) {
    length = atoi(argv[3]);
    if (offset + length > sb.st_size)
        length = sb.st_size - offset;
            /* Canaqt display bytes past end of file */
} else {    /* No length arg ==> display to end of file */
    length = sb.st_size - offset;
}
```

https://linux.die.net/man/2/munmap

Example code II part 4

mmap code part 4

```
    addr = mmap(NULL, length + offset - pa_offset, PROT_READ,
                MAP_PRIVATE, fd, pa_offset);
    if (addr == MAP_FAILED)
        handle_error("mmap");
    s = write(STDOUT_FILENO, addr + offset - pa_offset, length);
    if (s != length) {
        if (s == -1)
            handle_error("write");
        fprintf(stderr, "partial write");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

https://linux.die.net/man/2/munmap

Example code II part 5

mmap code part 1

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define handle_error(msg) \
    do { perror(msg); exit(EXIT_FAILURE); } while (0)
```

https://linux.die.net/man/2/munmap

Memory layout of C programs

memory layout of c programs

```
|-------------------|.....................
|                   | command-line arg's
|                   | and environ var's
|-------------------|.....................
| stack             |
|vvvvvvvvvvvvvvvvvvv|
|                   |
|^^^^^^^^^^^^^^^^^^^|
| heap              |
|-------------------|......................
| uninitialized data | initialized to zero
| bss               | by exec
|-------------------|...................... <- program break
| initialized data  |  read from
| data              |  program file
|-------------------|  by exec
|                   |
| text              |
|-------------------|.......................
```

https://www.geeksforgeeks.org/memory-layout-of-c-program/

brk() / sbrk() (1) program break

- brk() and sbrk() change the location of the program break, which defines the end of the process's data segment
- the program break is the first location after the end of the uninitialized data segment
- increasing the program break has the effect of allocating memory to the process;
- decreasing the program break deallocates memory.

https://man7.org/linux/man-pages/man2/brk.2.html

brk() / sbrk() (2) setting / increasing the program break

- brk() sets the top of the program break
  - this is the top of the data segment
    but inspecton of kernel/sys.c shows
    it separates from the data segment
  - this in effect increases the size of the heap
- sbrk() increases the program break
  rather than setting it directly

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

brk() / sbrk() (3) lazy allocation

- lazy allocation
- see mm/mmap.c for do_brk()
- do_brk() is implemented similar to mmap()
- modify the page tables for the new area
- wait for the page fault
- optionally, do_brk() can pre-fault the new area
  and allocate it
  see mlock(2) to control this behavior

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

brk() system call (4) brk() synopsys

brk system call

```
#include <unistd.h>

int brk(void *addr);
```

- brk() sets the end of the data segment to the value specified by addr,
  - when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size

https://man7.org/linux/man-pages/man2/brk.2.html

sbrk() system call (5) sbrk() synopsys

sbrk system call

```
#include <unistd.h>

void *sbrk(intptr_t increment);
```

- sbrk() increments the program's data space by increment bytes.
  - calling sbrk() with an increment of 0 can be used to find the current location of the program break

https://man7.org/linux/man-pages/man2/brk.2.html

brk() / sbrk() examples (1)

- You can use brk and sbrk yourself
  to avoid the malloc overhead
- cannot be easily used with malloc
- cannot free anything
- should avoid any library calls
  which may use malloc internally.
    - strlen is probably safe
    - fopen probably isn't

https://stackoverflow.com/questions/6988487/what-does-the-brk-system-call-do

brk() / sbrk() examples (2)

my allocate

```
void *myallocate(int n){
    return sbrk(n);
}
```

- Call `sbrk` just like you would call `malloc`.
  it returns a pointer to the current break
  and increments the break by that amount.

- `malloc()` and `calloc()` will use
  either `brk()` or `mmap()`
  depending on the requested allocation size

https://stackoverflow.com/questions/6988487/what-does-the-brk-system-call-do

brk() / sbrk() examples (3)

init and reset memory pool

```
void *memorypool;

void initmemorypool(void){
    memorypool = sbrk(0);
}

void resetmemorypool(void){
    brk(memorypool);
}
```

- frist, save the current break : `memorypool = sbrk(0)`
- then, use allocate space : `addr = myallocate(n)`
- finally, rewinding call `brk(memorypool)`

https://stackoverflow.com/questions/6988487/what-does-the-brk-system-call-do

High level implementation

- small allocations use `brk()`
- large allocaion use `mmap()`
- see `mallopt(3)` and the `M_MMAP_THRESHOD` parameter
  to control this behavio

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

Stack

- Stack expansion
- if a process accesses memory beyond its stack, the CPU will trigger a page fault
- the page fault handler detects the address is just beyond the stack, and allocates a new page to extend the stack
- the new page will not be physically contiguous with the rest of the stack
- see `__do_page_fault()` in `/arch/arm/mm/fault.c`

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf