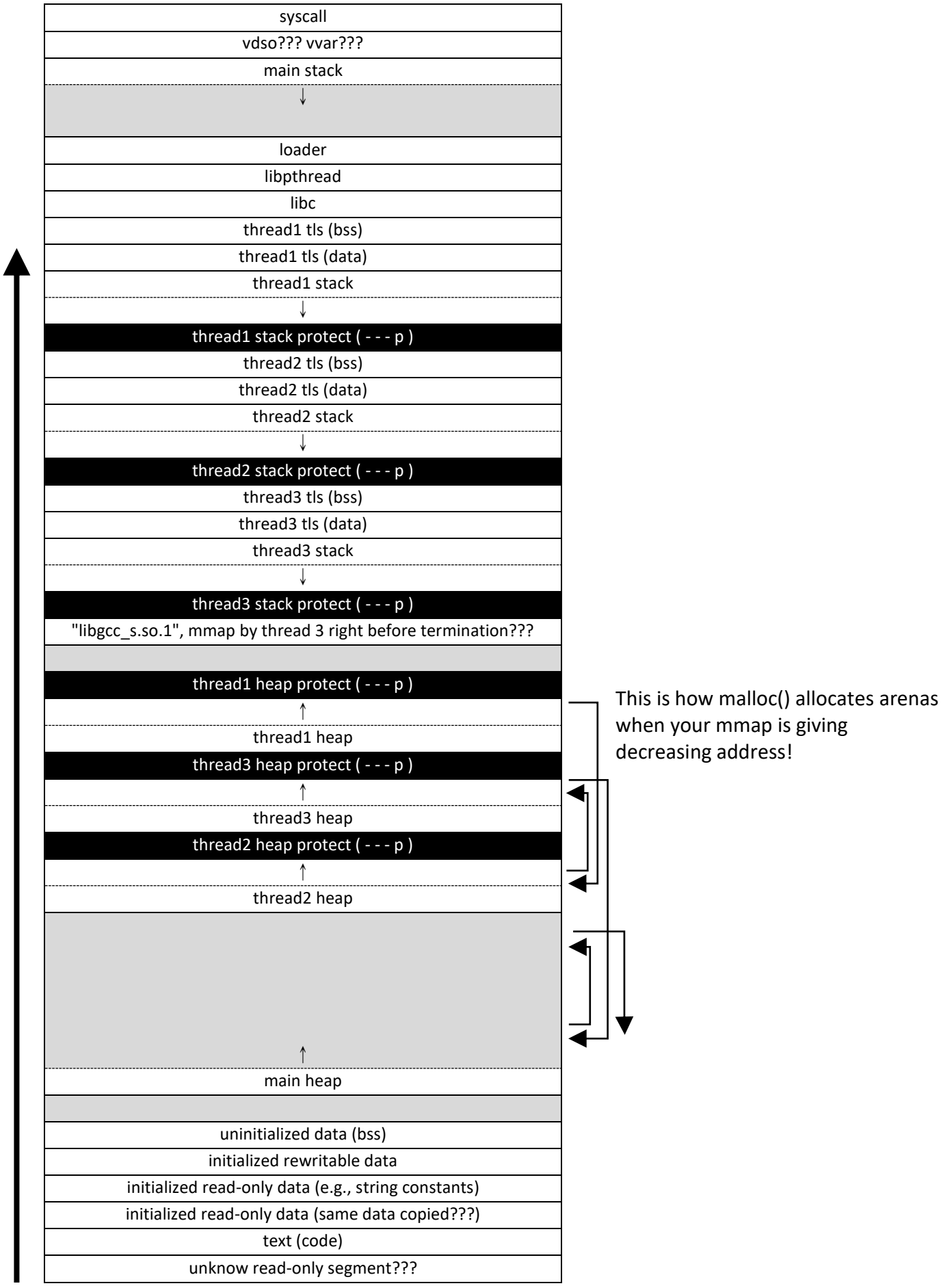
第 9 組

黃允誠 107502520
王伯綸 110521095
黃柏儒 110522136

the 圖

※大小未完全按照比例。



基本

哪一段 kernel code 造成同 process 的 threads 共享 memory？

fork.c → sys_clone → kernel_clone → copy_process → copy_mm

額外補充：

copy_process → … p = dup_task_struct …

copy_process → thread stack 跟 tls 到底在哪配置？→ copy_thread (machine dependent)

在 copy_mm 當中，若 clone_flags 裡的 CLONE_VM 為 true，則會將 parent process (current)的"mm"直接 assign 給 child thread 的"mm"。但我們知道 task_struct 中的"mm"是指標形式，因此指標 assign 過去等於完全拿同一份 mm_struct (memory mapping)。

clone 和 fork 的關係？thread 跟 process 的差別？

首先 clone 和 fork 明顯可以看出第一層下去之後都是一樣的呼叫路徑，其中最主要肉眼可見的差別就是 clone 會傳入五個參數，而 fork 不能傳入任何參數。但這還不夠，我們 trace 下去並密切留意這個差別究竟在哪裡造成影響，最終發現在這次 project 重點之一 copy_mm 當中，先前只注意到它會複製 mm，並沒有注意到它也可能複製 mm。耶？這裡就要說到中文的弱點了：demo 當天助教提醒我們要講這點，我回助教說 thread 會複製 process 的各種資源，結果助教說「fork 產生新 process 的時候也會複製啊！都是 kernel_clone 啊！」我們 strace 觀察 call clone 時傳入的參數可以發現傳入了許多的"flags"，從名字就可以看出大部分 flag 都是在指示特定一種資源要不要「複製」。例如 copy_mm 就受到「CLONE_VM」這個 flag 影響，當其為 true 就如前一題直接"copy"，但若為 false，則會 call dup_mm，也就是"duplicate"。clone、copy、duplicate 在中文裡全部都是「複製」，但在英文裡有微妙的差異，中文是相對模糊的一種語言。仔細觀察，"copy"做的事情如前一題，兩個人拿著的是同一個東西；但"duplicate"是複製出「長得一模一樣」的另一個東西，兩個人拿的看起來一樣其實不是同一個，之後也可能變得不一樣。

為甚麼說「人」呢？進到 thread 跟 process 的部分，我們可以使用一個很完美的類比：Linux 中 thread 跟 process 都是 task，我們把 task 都當作人，那麼 thread 當乘客，而 process 當司機，剩下當然就是車了。司機一定要「有車」才能開車，但乘客不用，乘客只要「上車」就行了。車就是 process 的各種資源，我們 clone 一個 thread 就是一個乘客上車，直接讓它上車；但如果要 fork 出一個 process，那麼就必須從無到有打造出一整台車讓它開。

那是甚麼決定你是哪種人？是在創造你的當下有沒有給你「車票」，也就是 call clone 時傳入的那些 flags。fork 既然不傳入任何參數，在進入 kernel_clone 前也只加一個 exit signal，這些 flags 很多都是 false 怎麼上車？不能上車那就只能蓋一台車當司機(process)了。我們可以另外查看在 copy_process 尾端、copy_mm 前後類似的函式，許多裡面都是一樣的道理，都有對應的 flag。

trace mmap: segments 是如何建立的?

strace 就可以看到許多東西，最一開始 main heap 其實是以 brk 來配置的：

sys_brk → /* Ok, looks good - let it rip. */ → do_brk_flags

剩下的就是各種 mmap，一直 mmap：

mmap.c → mmap_pgoff → ksys_mmap_pgoff → vm_mmap_pgoff → do_mmap → … addr = get_unmapped_area …

→ … addr = mmap_region …

mmap 各種參數在文件找得到，strace 裡也可以看出端睨：第一個是 addr，文件是這麼說的：null 的話就是隨便 kernel 配，但如果給 address 也不保證直接配給你，只是「盡量」配到那裡(as a hint)，這就導致建立在這之上的應用要想辦法處理這種不確定因子。第二個參數是長度，長度如果配就一定要配完整的，要是做不到就不能配置。第三個是權限(rwx)，第四個是各種 flag，第五個是 file descriptor，最後一個是 offset，如果開檔的話，可能不一定要從檔案最前面 mapping，此時可以用 offset 往後挪 mapping 的起始位置。第四個 flag 當中最值得講的是 MAP_ANONYMOUS：其實只有最一開始幾條需要開檔，剩下大部分一般的 segment 都不是檔案，設定這個 flag。文件提到此時 file descriptor 有些實作會無視，但有些要求設定-1，所以為了 portability 大家應該都放-1，另外 offset 要放 0。

get_unmapped_area 就是尋找可用的一段記憶體，mmap_region 是實際配置。

助教提示

getpid 真的是 getpid 嗎？

助教其實是直接跟我們說 getpid 並不是真的拿 pid。我原本以為 threads 的 pid 都是跟著 process 沒有錯，結果助教說這不對，threads 也有自己的 pid。Project1 就有印象 pid_t 有兩個東西。一樣 trace code：

sys_getpid → task_tgid_vnr(current) → __task_pid_nr_ns(current, PIDTYPE_TGID, NULL)
→ ns = task_active_pid_ns(current)
→ nr = pid_nr_ns(rcu_dereference(*task_pid_ptr(task, PIDTYPE_TGID)), ns)
sys_gettid → task_pid_vnr(current) → __task_pid_nr_ns(current, PIDTYPE_PID, NULL)
→ ns = task_active_pid_ns(current)
→ nr = pid_nr_ns(rcu_dereference(*task_pid_ptr(task, PIDTYPE_PID)), ns)

要 getpid，結果拿到的是 tgid；要 gettid，拿到的卻是 pid。看起來非常詭異，究竟是怎麼回事？根據助教的說法和我們查到的資料總結，本來 Linux 是只有 pid 的，也沒有平行處理的打算，所以 Linux 只有 task_struct 一種架構。因為這樣，Linux 所謂的 pid 其實是人家的 tid。本來架構就沒有打算要平行處理，但忽然發現平行處理很猛，怎麼辦呢？POSIX 標準要求同樣 process 的 thread 必須共用一個 pid。Linux 就沒有這種架構啊，於是直接加上去，也就是所謂的 tgid (thread group id)。

這樣一來邏輯就很清楚了，Linux 的 pid 是大家的 tid，每個 thread 唯一；而大家的 pid 是 Linux 的 tgid，同 process 的所有 thread 共用。

但是這邊多出現一個問題：trace code 到後面就看不太懂它在做甚麼，最後是直接上網查資料，總算是把 Linux namespace 相關的觀念搞懂了。task_active_pid_ns 從 current(也就是 call syscall 的 task)取得最底層 namespace。然後 pid_nr_ns……那一串負責取得在最底層 namespace(也就是 userspace 的 task 看自己)的 id。如果在 task_struct 直接取那兩個 pid_t 的話，取得的會是 global 的 id(最頂層、init 那一層 namespace 看到的 id)，但當 userspace call 這兩個 syscall 的時候，希望 kernel 給它們的是它們最底層視角看到的 id。

之後我試著想把搞懂的觀念跟 code 的細節做對應，結果還是無法。大部分都沒有問題，但是中間那個 rcu_dereference 的 macro 實在是太多層，碰不到底，也看不出它在做甚麼。一層兩層三層四層……中間還出現一個直接 return 1 的怪東西，喪心病狂，只好作罷……

最後以助教的話作結：「Linux 一開始沒打算平行處理，全部都是 task。所以後來就變成 thread 跟 process 都有自己的 task_struct，只是同個 process 的 threads 會共用一些 task_struct 的內容。」

為何不同 thread 執行 malloc 配置的位置差距那麼大？

這個問題其實很早就發現了，但較晚找到答案。我們的程式會把 virtual address 印出來，搭配 /proc/<pid>/maps 對照，就算不知道答案也大概會猜到發生了甚麼事。

不過其實我本來真正想找的答案是：為甚麼 threads 的 heap 跟 main heap 中間跳那麼一大段？結果誤打誤撞看到關鍵字 per-thread arenas。多個 thread 在執行，如果它們很不巧同時對某個位置做 malloc 相關的操作就會打架出錯(類似 racing condition)。當然可以用上鎖來處理，但這樣不斷搶鎖又會影響效能。所以最後就出現這種分蛋糕的方法，每個 thread 都切一塊，彼此不要互搶，也就不用一直搶鎖了。從 maps 可以看出我這台一塊蛋糕是 64MiB，聽說這也是普遍的默認值。

所以這樣一來這個問題解決了，但我本來想找的問題反而沒解決，因為回答的人其實沒有針對他的問題回答，發文的那個人之後追問結果都沒人理他。不過後來我想通了：main heap 是一開始由 brk 配置的，而所有其他 thread heap 都是由 mmap 配置的。brk 是由低位址往高位址配置，而**我的** mmap 是**由高位址往低位址**配置，換句話說，thread heaps 跟 main heap 雖然都是 heap，但配置的方式不一樣，這兩邊本來就是從兩頭夾過來的。所以中間的那一大段空隙並不是「跳」過去的，而是本來就在中間的。

額外研究

mprotect 究竟是負責保護還是放行？

一個小問題，答案我認為是都可以。聽助教說 protect 在做甚麼之後，直覺會認為它是把要保護用的 protect 段，所有權限禁止，從而達到「保護」的作用，我記得 demo 那天助教也是說「把權限壓掉」。但是我後來看 strace 發現，竟然有反過來的作法，它是先 mmap 一大塊「完全沒有權限」的記憶體，然後才把需要的部分「mprotect」把相應的權限「打開」。

仔細想兩種結果是一樣的，而且其實我這邊看起來大部分都是後面這種變形較不直觀的作法。我推測這可能跟效能有關係：在需要 protect 段的情況下，要關權限的部分跟要開權限的部分一定都小於你整體 mmap 的大小，所以這時候我們想像不論是把沒有的權限打開還是把權限關掉都要花效能去改動某些數值，那麼比起 mmap 同時把需要的權限都打開，之後再 mprotect 把一部份記憶體權限關掉，當然是 mmap 不開權限，然後 mprotect 把需要權限的部分打開比較省力。

當然如果記憶體權限開開關關那也就無所謂開權限比關權限費力的問題，不過觀察到這個情況，一般也都是程式剛開始執行一堆 mmap 的時候，而每次程式關閉前理論上它也要負責把記憶體都恢復原狀，下次執行新的程式就算碰巧動到同樣的虛擬記憶體時應該也是初始狀態，所以這可能性蠻大的。

當然也是不能排除助教回答其它問題時說的一種可能性：「因為它爽」(非原話)。

threads 的 tls 裡真的沒有分 data 段跟 bss 段嗎？

我們先不考慮 threads，先說一般 single process 全域的 data 段跟 bss 段。data 段是有初始值的全域變數，bss 段是沒初始值的全域變數，其實還有一點，data 段有分「唯獨段」跟「可讀寫段」。仔細想應該就有兩點很奇怪：第一，我們從 maps 裡看到主程式的可讀寫段只有一段，這段到底是 data 段還是 bss 段？第二，唯獨的 data 段呢？

第二個問題很簡單先答：使用 gdb 秀出 virtual address 在可讀寫段「下方」(較低一邊，maps 來看通常是上面)的那一段唯獨段，就會發現其中存了很多 string constant，例如 printf("hello");的"hello"就會被存在這裡。這些資料很明顯是有初始值的，應該放在 data 段，但它們必須唯讀，所以放在這裡。這邊插播個小問題，再往旁邊會看到另一段一樣大小的唯讀段，裡面存著一模一樣的東西，這個問題我還無解，但也不是這邊的重點所以只稍微提一下有發現這件事。

來到第一個問題，看似很好回答又不是那麼好回答。簡單的答案或許有一種：data 跟 bss 都在這裡，Linux 作業系統不區分 data 或 bss，只區分權限。但我目前還沒看過有人這樣說，一般而言大家還是認為全域變數有分 data 或 bss 的對吧？

這不只是說有沒有這麼簡單而已。一般而言當我們宣告很多全域變數時，compiler 會由低位址往高位址，按照我們在 source code 中宣告的順序配置變數。那如果我們把有初始值的變數(data)跟沒初始值的變數(bss)交錯宣告會發生甚麼事？很明顯地，compiler 會由低至高先把有初始值的配置完，再配置沒初始值的變數，而且重點是：除了上面這條規則，剩下的部分都依舊按照 source code 中宣告的順序。

所以總結一下這部分我認為的結論：「由低位往高位，唯讀 data，銜接可讀寫 data，銜接 bss，後面兩個合併在 maps 上顯示為同一區段」。

我們都知道變數配置的位置主要是由 compiler 決定的，從 Linux 的角度我們用 strace 觀察 mmap 可以得知它基本上主要只關注記憶體區段的權限。當然現在說的這部分是一個 execve 就解決了，裡面我還沒有時間 trace，然而不論 Linux 有沒有在插手或觀察這件事，不論重點是 OS 還是 compiler：如果我們都能接受即使 maps 中沒有顯示，甚至 Linux kernel 中可能也沒有記錄相關信息，我們還是認為在 Linux 中執行 c 語言程式時，全域變數在概念上是有區分 data 或 bss 段的……那麼，是甚麼原因讓你這麼認為？不論是甚麼原因，把它記著。

現在如果我告訴你，tls 中跟全域一樣會發生「一模一樣」的事呢？

我們對加了__thread 的 tls 變數做一樣的測試，也會有同樣的特性，而且是在各 thread 的 tls 中：「由低往高，先配置有初始值的變數，再配置沒初始值的，剩下所有順序依照宣告次序」。那麼又是甚麼原因讓大家認為全域中有分為 data 段跟 bss 段，但是一樣的情形發生在 tls 中就不算數呢？

深度研究——到底是誰在跳？

前言

說深度其實是我私心，畢竟助教都說了他覺得這個沒有特別重要了……但是解這題真的好累 Orz。其實我前面有劇透兩次，不知道有沒有引起興趣……先簡單說一下問題：

首先是除了助教提到的「thread call malloc 時彼此間位址差距很大」的問題之外，我發現了 thread heaps 配置的順序很奇怪。一開始依照題目要求 3 個 threads，tls 的部分由低位址往高位址按照 thread321 配置因為有 stack 所以我覺得還算正常，但 heap 的部分，明明我有用 sleep 控制 123 的順序，由低往高卻是 231。於是我一時興起，竄改程式變成一次開 7 個 threads，結果順序更怪，6745231。由於當時我使用 strace 可以看到大部分的東西，卻看不到這部分的 mmap，導致我想不通所以然，便跑到 demo 上問助教。

助教給我的答案是，由於都是 mmap 在配置，一般 call mmap 的時候通常不會指定 addr，所以都是隨 mmap 高興配哪就配哪。我問助教，但是每次執行順序是固定的耶？助教就說那可能真的有些原因，但不是這門課的重點。此時我鍥而不捨，追問助教為甚麼我下 strace 無法看到這部分的 mmap，最後助教提到可能因為我的 strace 只有 trace main thread，回去下-f 試試看。

於是我回家就迫不及待(?)的打開我特製版的 7 threads 來-f，那個文字量真是大海撈針，但最後我還是理出頭緒了……其實 mmap 是無辜的。它從頭到尾都非常乖，並不像助教說的那樣想放哪就放哪，相反地，是人家叫它放哪就放哪，左右跳來跳去的，看起來十分可憐。那麼究竟是誰這麼壞呢？沒錯，就是 malloc！！！

好了前情提要結束……

找尋問題點

剛才說我使用-f 看到了一切，由於助教說是 mmap 的問題所以我特別的關注它，而又由於助教提醒的基本問答中有一題要 trace mmap，所以對於其中傳入的參數我也很清楚。我很快就發現了 mmap 被某人叫來叫去：首先 thread1 malloc 時，mmap 先被要求隨便配置 128MiB 的記憶體，接著把前面的一部份丟掉(munmap)，再把後面的一部份丟掉。我第一個想法是「這人有病吧？」，由於實在太奇怪，我決定先跳過 thread1，先看後面。而之後的 thread 就產生很強的規律：在偶數 thread，這人又會叫 mmap 配置 128MiB 的記憶體，此時我注意到 mmap 一直都是往低位址配置過去，而由於前面已經有配置過 thread heap 了，這時直接要求 128MiB mmap 就往下跳了。但配置完之後，這人做甚麼事？要求把比較高的一半(64MiB)丟掉，然後留下比較低的一半給這個 thread 用。看到這我都快對電腦發火了(其實主要是因為想不通為甚麼)，但還沒完！偶數的 thread 配置完之後，輪到奇數的 thread，這人直接叫 mmap 配在中間！就是上一個 thread 刪掉的一半，這個 thread 剛好塞進去，用指定位址的方式，要求 64MiB。之後就如此重複，我就看可憐的 mmap 這樣跳過來跳過去，配了又刪，刪了又配……這邊文字敘述太抽象的話可以搭配第一頁那張圖旁邊的「劇透」看。不過這時我又發現疑點：這些 heap 連在一起，連在一起不奇怪，重點是 address 都非常「平整」。此時我才想起：我這不是在配置 heap 嗎？我 call 的不是 malloc 嗎？為甚麼我在 trace mmap？malloc 人呢？？？

trace code

解了上面 per-thread arenas 那題之後這邊就很清楚，malloc 在維持每個 thread 的 arena 大小是默認的 64MiB。而且同時也解開我對 thread heap 配置順序的疑惑，因為很明顯就是有人故意這樣搞的，strace -f 裡面傳入參數都寫得很清楚。但是主要的兩個問題就是：第一，為甚麼要這樣跳來跳去刪掉又配又刪掉？第二，後面都是很規律的「配兩倍後拿掉高的一半→塞進高的一半→循環」，為甚麼 thread1 這麼複雜？

此時我忽然想到 demo 時助教也提醒過，其實 bootlin 只要下拉選單拉一下也可以 trace libc，於是進入了 trace code 環節。剛才說很累主要就是這裡，因為一開始沒進入狀況真的是看不太懂，東西一堆無從下手：

malloc → __libc_malloc → arena_get → arena_get2 → _int_new_arena → new_heap

其他部分坦白說我也沒能完全看懂，大概知道意思找對路而已，重點都在 new_heap 了：

```
/* If consecutive mmap (0, HEAP_MAX_SIZE << 1, ...) calls return decreasing
   addresses as opposed to increasing, new_heap would badly fragment the
   address space.  In that case remember the second HEAP_MAX_SIZE part
   aligned to HEAP_MAX_SIZE from last mmap (0, HEAP_MAX_SIZE << 1, ...)
   call (if it is already aligned) and try to reuse it next time.  We need
   no locking for it, as kernel ensures the atomicity for us - worst case
   we'll call mmap (addr, HEAP_MAX_SIZE, ...) for some value of addr in
   multiple threads, but only one will succeed.  */
static char *aligned_heap_area;
/* Create a new heap.  size is automatically rounded up to a multiple
   of the page size. */
static heap_info *
new_heap (size_t size, size_t top_pad){
    size_t pagesize = GLRO (dl_pagesize);
    char *p1, *p2;
    unsigned long ul;
    heap_info *h;
    if (size + top_pad < HEAP_MIN_SIZE)
        size = HEAP_MIN_SIZE;
    else if (size + top_pad <= HEAP_MAX_SIZE)
        size += top_pad;
    else if (size > HEAP_MAX_SIZE)
        return 0;
    else
        size = HEAP_MAX_SIZE;
    size = ALIGN_UP (size, pagesize);
    /* A memory region aligned to a multiple of HEAP_MAX_SIZE is needed.
       No swap space needs to be reserved for the following large
       mapping (on Linux, this is the case for all non-writable mappings
       anyway). */
    p2 = MAP_FAILED;
    if (aligned_heap_area){
        p2 = (char *) MMAP (aligned_heap_area, HEAP_MAX_SIZE, PROT_NONE, MAP_NORESERVE);
        aligned_heap_area = NULL;
        if (p2 != MAP_FAILED && ((unsigned long) p2 & (HEAP_MAX_SIZE - 1))){
            __munmap (p2, HEAP_MAX_SIZE);
            p2 = MAP_FAILED;
        }
    }
    if (p2 == MAP_FAILED){
        p1 = (char *) MMAP (0, HEAP_MAX_SIZE << 1, PROT_NONE, MAP_NORESERVE);
        if (p1 != MAP_FAILED){
            p2 = (char *) (((unsigned long) p1 + (HEAP_MAX_SIZE - 1)) & ~(HEAP_MAX_SIZE - 1));
            ul = p2 - p1;
            if (ul)
                __munmap (p1, ul);
            else
                aligned_heap_area = p2 + HEAP_MAX_SIZE;
            __munmap (p2 + HEAP_MAX_SIZE, HEAP_MAX_SIZE - ul);
        }else{
            /* Try to take the chance that an allocation of only HEAP_MAX_SIZE
               is already aligned. */
            p2 = (char *) MMAP (0, HEAP_MAX_SIZE, PROT_NONE, MAP_NORESERVE);
            if (p2 == MAP_FAILED)
                return 0;
            if ((unsigned long) p2 & (HEAP_MAX_SIZE - 1)){
                __munmap (p2, HEAP_MAX_SIZE);
                return 0;
            }
        }
    }
    if (__mprotect (p2, size, PROT_READ | PROT_WRITE) != 0){
        __munmap (p2, HEAP_MAX_SIZE);
        return 0;
    }
    h = (heap_info *) p2;
    h->size = size;
    h->mprotect_size = size;
    LIBC_PROBE (memory_heap_new, 2, h, h->size);
    return h;
}
```

thread 2n+1

thread 2n

x2

thread 1

上面的部分只有解釋了為甚麼會這樣 mmap，只是結果跟程式碼成功對應而已，主要是搭配作者的註解：我們可以看到作者不確定 mmap 到底是往上配置還是往下配置，應該是有不同版本的實作。接著作者提到他希望 arenas 彼此間的分界線是在 arena 大小的整數倍(aligned)，這部分沒有說為甚麼，我推測跟教授最近上課常講到 aligned 原因都差不多，應該是方便之後 malloc 管理這些 arenas，反正不是省空間就是省時間。

那麼來回答問題了：為甚麼 thread1 這麼複雜？從上面就可以看出來，由於第一下要打開第一個 arena 的時候，malloc 完全沒有任何資訊，所以它只能讓 mmap 自己配置，然後把傳回來的位址當成第一個資訊。但是它又希望要對齊，如果只要一個 arena 大小的記憶體，第一次要回來的多半都是沒對齊的位置，而這時候 kernel mmap 給你的位置以外，都有可能是不能使用的記憶體，就很尷尬了。所以它就耍無賴，直接要兩倍的，然後自己切一切，就對齊了……所以就是 strace -f 一開始看到的一次 mmap 兩次 munmap。

那為甚麼後面要跳來跳去地配置？作者有說，因為不確定 mmap 是往哪邊配置，所以他這種要兩倍空間的作法，如果他單純每次要兩倍空間，mmap 每次配置又是往下走，但配置是從 addr 往上配置，這時候就會切出一大堆的空氣(fragment)，非常糟糕。所以他的做法就是像這樣：每次要兩倍之後，把其中一倍記下來，這一邊一定也會是對齊的，下一次還需要的時候就直接用，不要繼續往前要。這個寫法如果 mmap 是往上走也是一樣沒有問題的，接得好好的。

最後多一個問題：這時候我就在想，那為甚麼不第一下要兩倍，之後都乖乖一塊一塊要呢？不是完美了嗎？其實不是，我們現在的情況是因為我只不斷地開 thread，然後每個 thread 一開 malloc 一小塊，就這樣很簡單的用法。但實際應用上是會有很多很複雜的情況，這邊 malloc 那邊 free，那邊又 free 再 malloc……所以他其實每一次都沒法保證 syscall 拿回來的 addr 有對齊(除非前面就有記著另外一半)，並不是像我們現在看起來只有程式執行第一下會歪掉，後面都剛好。所以他用這種方式，不斷確保不論怎樣拿到的 arenas 都是 aligned 的。

然後稍微再說一個小點：__libc_malloc 裡面有寫道，single process 的部分，前面就會算完 return 了。這邊這條路是專門針對 multi threads 的情況下，才會走 arenas 這邊。

最後最後簡單結論就是：剛好我的 mmap 是 new_heap 比較討厭的那種，所以讓我有機會發現問題體會到這個煉獄……

好了，剩下的部分就是 code 跟執行結果，這部分呢……哀，我把它們放上來的作用，就……只是放上來了而已……東西非常多、非常雜，特別是 strace 下了-f 之後真的不是人看的……真的很抱歉我沒有時間心力整理最後這些了，已經太久沒睡覺……這時候還要把比較不重要的東西刪掉的話，可能一不小心把重要的東西刪掉了……所以就全部放著以作參考，如果助教們有想要確認甚麼東西，用 ctrl + f 應該多少可以瀏覽一下……可能這篇報告也難以組織得很好，可能有點冗長，助教們辛苦了。

Code

kernel space syscall

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/string.h>
#include <linux/uaccess.h>
#include <linux/init_task.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>
#include <linux/pgtable.h>
#include <linux/mm_types.h>
#include <linux/mm.h>
SYSCALL_DEFINE3(project2, int, len, void* __user, vaddr, void* __user, paddr) {
	if(len < 0){
		return len;
	}
	long err = 0;
	unsigned long vaddr_k[len];
	unsigned long paddr_k[len];
	copy_from_user(&vaddr_k, vaddr, sizeof(unsigned long)*len);
	int i = 0;
	for(; i < len; i++){
		pmd_t *pmd = pmd_off(current -> mm, vaddr_k[i]);
		pte_t *pte = pte_offset_map(pmd, vaddr_k[i]);
		if (pte_none(*pte)){
			err++;
		}
		paddr_k[i] = page_to_phys(pte_page(*pte)) + (vaddr_k[i] & ~PAGE_MASK);
	}
	copy_to_user(paddr, &paddr_k, sizeof(unsigned long)*len);
	return err;
}
```

user space test

```
#include <syscall.h>
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#define main_tests 12
#define thread_count 5
#define thread_tests 13
int bss;
int data = 1;
int bss_2;
int data_2 = 1;
__thread int bss_thread;
__thread int data_thread = 1;
__thread int bss_thread_2;
__thread int data_thread_2 = 1;
__thread void *code_p;
__thread void *data_p;
__thread void *data_2_p;
__thread void *bss_p;
__thread void *bss_2_p;
int *heap_main_p;
__thread int *heap_thread_p;
__thread void *libc_p;
```

```
void *stack_main_p;
__thread void *stack_thread_p;
__thread void *stack_thread_2_p;
__thread void *data_thread_p;
__thread void *data_thread_2_p;
__thread void *bss_thread_p;
__thread void *bss_thread_2_p;
__thread unsigned long vaddr[thread_tests];
__thread unsigned long paddr[thread_tests];

long thread_test_with_stack_direction() {
	int stack_thread_2 = 1;
	stack_thread_2_p = &stack_thread_2;
	vaddr[0] = code_p;
	vaddr[1] = data_p;
	vaddr[2] = bss_p;
	vaddr[3] = heap_main_p;
	vaddr[4] = heap_thread_p;
	vaddr[5] = libc_p;
	vaddr[6] = stack_main_p;
	vaddr[7] = stack_thread_p;
	vaddr[8] = stack_thread_2_p;
	vaddr[9] = data_thread_p;
	vaddr[10] = data_thread_2_p;
	vaddr[11] = bss_thread_p;
	vaddr[12] = bss_thread_2_p;
	return syscall(443, thread_tests, (void*)&vaddr, (void*)&paddr);
}

void *thread_f(void *thread_ID) {
	int ID = *((int *) thread_ID);
	sleep(ID);

	printf("thread %d:\npid: %d, tid: %d, &printf: %lx\n", ID, getpid(), gettid(), &printf);
	int stack_thread = 1;
	data_thread = data_thread + 1;
	data_thread_2 = data_thread_2 + 1;
	bss_thread = bss_thread + 1;
	bss_thread_2 = bss_thread_2 + 1;

	code_p = &thread_f;
	data_p = &data;
	bss_p = &bss;
	heap_thread_p = malloc(sizeof(int) * 1);
	*heap_thread_p = *heap_thread_p + 1;
	libc_p = &printf;
	stack_thread_p = &stack_thread;
	data_thread_p = &data_thread;
	data_thread_2_p = &data_thread_2;
	bss_thread_p = &bss_thread;
	bss_thread_2_p = &bss_thread_2;

	thread_test_with_stack_direction();
	char segements[thread_tests][15] = {"code", "data", "bss", "heap_main", "heap_thread", "libc", "stack_main",
"stack_thread", "stack_thread_2", "data_thread", "data_thread_2", "bss_thread", "bss_thread_2"};
	for(int i=0; i < thread_tests; i++){
		printf("%-15s\tvaddr: %lx, paddr: %lx\n", segements[i], vaddr[i], paddr[i]);
	}

	free(heap_thread_p);
	sleep(thread_count+1-ID);
	pthread_exit(NULL);
}
```

```
int main() {
	printf("main:\npid: %d, tid: %d, &printf: %lx\n", getpid(), gettid(), &printf);
	int stack_main = 1;
	data = data + 1;
	data_2 = data_2 + 1;
	bss = bss + 1;
	bss_2 = bss_2 + 1;
	data_thread = data_thread + 1;
	data_thread_2 = data_thread_2 + 1;
	bss_thread = bss_thread + 1;
	bss_thread_2 = bss_thread_2 + 1;

	code_p = &thread_f;
	data_p = &data;
	data_2_p = &data_2;
	bss_p = &bss;
	bss_2_p = &bss_2;
	heap_main_p = malloc(sizeof(int) * 1);
	*heap_main_p = *heap_main_p + 1;
	libc_p = &printf;
	stack_main_p = &stack_main;
	data_thread_p = &data_thread;
	data_thread_2_p = &data_thread_2;
	bss_thread_p = &bss_thread;
	bss_thread_2_p = &bss_thread_2;

	vaddr[0] = code_p;
	vaddr[1] = data_p;
	vaddr[2] = data_2_p;
	vaddr[3] = bss_p;
	vaddr[4] = bss_2_p;
	vaddr[5] = heap_main_p;
	vaddr[6] = libc_p;
	vaddr[7] = stack_main_p;
	vaddr[8] = data_thread_p;
	vaddr[9] = data_thread_2_p;
	vaddr[10] = bss_thread_p;
	vaddr[11] = bss_thread_2_p;
	syscall(443, main_tests, (void*)&vaddr, (void*)&paddr);
	char segements[main_tests][15] = {"code", "data", "data_2", "bss", "bss_2", "heap_main", "libc", "stack_main",
"data_thread", "data_thread_2", "bss_thread", "bss_thread_2"};
	for(int i=0; i < main_tests; i++){
		printf("%-15s\tvaddr: %lx, paddr: %lx\n", segements[i], vaddr[i], paddr[i]);
	}

	int thread_IDs[thread_count+1];
	for(int i = 1; i <= thread_count; i++){
		thread_IDs[i] = i;
	}
	pthread_t threads[thread_count+1];
	for(int i = 1; i <= thread_count; i++){
		pthread_create(&threads[i], NULL, &thread_f, &thread_IDs[i]);
	}
	for(int i = 1; i <= thread_count; i++){
		pthread_join(threads[i], NULL);
	}
	printf("Press Enter to end.\n");
	char c;
	scanf("%c", &c);
	printf("End.\n");
	free(heap_main_p);
	return 0;
}
```

Result

一般輸出，兩個 process 同時 running。

```
main:
pid: 9458, tid: 9458, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
data_2          vaddr: 5566b4ab4014, paddr: 17c03f014
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
bss_2           vaddr: 5566b4ab4028, paddr: 17c03f028
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
data_thread     vaddr: 7f9c7ad72720, paddr: 1ba3fb720
data_thread_2   vaddr: 7f9c7ad72724, paddr: 1ba3fb724
bss_thread      vaddr: 7f9c7ad72730, paddr: 1ba3fb730
bss_thread_2    vaddr: 7f9c7ad72734, paddr: 1ba3fb734
thread 1:
pid: 9458, tid: 9459, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
heap_thread     vaddr: 7f9c74000b60, paddr: 165df1b60
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
stack_thread    vaddr: 7f9c7ad70cc4, paddr: 17997acc4
stack_thread_2  vaddr: 7f9c7ad70c94, paddr: 17997ac94
data_thread     vaddr: 7f9c7ad715a0, paddr: 1763765a0
data_thread_2   vaddr: 7f9c7ad715a4, paddr: 1763765a4
bss_thread      vaddr: 7f9c7ad715b0, paddr: 1763765b0
bss_thread_2    vaddr: 7f9c7ad715b4, paddr: 1763765b4
thread 2:
pid: 9458, tid: 9460, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
heap_thread     vaddr: 7f9c6c000b60, paddr: 1a0feab60
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
stack_thread    vaddr: 7f9c7a56fcc4, paddr: 1b9d39cc4
stack_thread_2  vaddr: 7f9c7a56fc94, paddr: 1b9d39c94
data_thread     vaddr: 7f9c7a5705a0, paddr: 1727b25a0
data_thread_2   vaddr: 7f9c7a5705a4, paddr: 1727b25a4
bss_thread      vaddr: 7f9c7a5705b0, paddr: 1727b25b0
bss_thread_2    vaddr: 7f9c7a5705b4, paddr: 1727b25b4
thread 3:
pid: 9458, tid: 9461, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
heap_thread     vaddr: 7f9c70000b60, paddr: 1a032bb60
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
stack_thread    vaddr: 7f9c79d6ecc4, paddr: 179807cc4
stack_thread_2  vaddr: 7f9c79d6ec94, paddr: 179807c94
data_thread     vaddr: 7f9c79d6f5a0, paddr: 17166f5a0
data_thread_2   vaddr: 7f9c79d6f5a4, paddr: 17166f5a4
bss_thread      vaddr: 7f9c79d6f5b0, paddr: 17166f5b0
bss_thread_2    vaddr: 7f9c79d6f5b4, paddr: 17166f5b4
thread 4:
pid: 9458, tid: 9462, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
heap_thread     vaddr: 7f9c64000b60, paddr: 1ba3d8b60
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
stack_thread    vaddr: 7f9c7956dcc4, paddr: 1b7127cc4
stack_thread_2  vaddr: 7f9c7956dc94, paddr: 1b7127c94
data_thread     vaddr: 7f9c7956e5a0, paddr: 16f8685a0
data_thread_2   vaddr: 7f9c7956e5a4, paddr: 16f8685a4
bss_thread      vaddr: 7f9c7956e5b0, paddr: 16f8685b0
bss_thread_2    vaddr: 7f9c7956e5b4, paddr: 16f8685b4
thread 5:
pid: 9458, tid: 9463, &printf: 7f9c7add9e10
code            vaddr: 5566b4ab141d, paddr: 18dfe741d
data            vaddr: 5566b4ab4010, paddr: 17c03f010
bss             vaddr: 5566b4ab402c, paddr: 17c03f02c
heap_main       vaddr: 5566b67546b0, paddr: 1b7f0e6b0
heap_thread     vaddr: 7f9c68000b60, paddr: 1b9e1ab60
libc            vaddr: 7f9c7add9e10, paddr: 11f749e10
stack_main      vaddr: 7ffd45e0c17c, paddr: 1ae54d17c
stack_thread    vaddr: 7f9c78d6ccc4, paddr: 197852cc4
stack_thread_2  vaddr: 7f9c78d6cc94, paddr: 197852c94
data_thread     vaddr: 7f9c78d6d5a0, paddr: 17fbf15a0
data_thread_2   vaddr: 7f9c78d6d5a4, paddr: 17fbf15a4
bss_thread      vaddr: 7f9c78d6d5b0, paddr: 17fbf15b0
bss_thread_2    vaddr: 7f9c78d6d5b4, paddr: 17fbf15b4
Press Enter to end.
```

```
main:
pid: 9480, tid: 9480, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
data_2          vaddr: 560a57c35014, paddr: 1b7402014
bss             vaddr: 560a57c3502c, paddr: 1b740202c
bss_2           vaddr: 560a57c35028, paddr: 1b7402028
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
data_thread     vaddr: 7f9337e6e720, paddr: 1ba352720
data_thread_2   vaddr: 7f9337e6e724, paddr: 1ba352724
bss_thread      vaddr: 7f9337e6e730, paddr: 1ba352730
bss_thread_2    vaddr: 7f9337e6e734, paddr: 1ba352734
thread 1:
pid: 9480, tid: 9481, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
bss             vaddr: 560a57c3502c, paddr: 1b740202c
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
heap_thread     vaddr: 7f9330000b60, paddr: 1650e4b60
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
stack_thread    vaddr: 7f9337e6ccc4, paddr: 18df2ccc4
stack_thread_2  vaddr: 7f9337e6cc94, paddr: 18df2cc94
data_thread     vaddr: 7f9337e6d5a0, paddr: 18df2e5a0
data_thread_2   vaddr: 7f9337e6d5a4, paddr: 18df2e5a4
bss_thread      vaddr: 7f9337e6d5b0, paddr: 18df2e5b0
bss_thread_2    vaddr: 7f9337e6d5b4, paddr: 18df2e5b4
thread 2:
pid: 9480, tid: 9482, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
bss             vaddr: 560a57c3502c, paddr: 1b740202c
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
heap_thread     vaddr: 7f9328000b60, paddr: 1a7a01b60
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
stack_thread    vaddr: 7f933766bcc4, paddr: 1a0055cc4
stack_thread_2  vaddr: 7f933766bc94, paddr: 1a0055c94
data_thread     vaddr: 7f933766c5a0, paddr: 18df225a0
data_thread_2   vaddr: 7f933766c5a4, paddr: 18df225a4
bss_thread      vaddr: 7f933766c5b0, paddr: 18df225b0
bss_thread_2    vaddr: 7f933766c5b4, paddr: 18df225b4
thread 3:
pid: 9480, tid: 9483, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
bss             vaddr: 560a57c3502c, paddr: 1b740202c
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
heap_thread     vaddr: 7f932c000b60, paddr: 17a2f8b60
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
stack_thread    vaddr: 7f9336e6acc4, paddr: 1a0d19cc4
stack_thread_2  vaddr: 7f9336e6ac94, paddr: 1a0d19c94
data_thread     vaddr: 7f9336e6b5a0, paddr: 1a17d45a0
data_thread_2   vaddr: 7f9336e6b5a4, paddr: 1a17d45a4
bss_thread      vaddr: 7f9336e6b5b0, paddr: 1a17d45b0
bss_thread_2    vaddr: 7f9336e6b5b4, paddr: 1a17d45b4
thread 4:
pid: 9480, tid: 9484, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
bss             vaddr: 560a57c3502c, paddr: 1b740202c
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
heap_thread     vaddr: 7f9320000b60, paddr: 18bfa6b60
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
stack_thread    vaddr: 7f9336669cc4, paddr: 1aea0dcc4
stack_thread_2  vaddr: 7f9336669c94, paddr: 1aea0dc94
data_thread     vaddr: 7f933666a5a0, paddr: 1b7cc55a0
data_thread_2   vaddr: 7f933666a5a4, paddr: 1b7cc55a4
bss_thread      vaddr: 7f933666a5b0, paddr: 1b7cc55b0
bss_thread_2    vaddr: 7f933666a5b4, paddr: 1b7cc55b4
thread 5:
pid: 9480, tid: 9485, &printf: 7f9337ed5e10
code            vaddr: 560a57c3241d, paddr: 18dfe741d
data            vaddr: 560a57c35010, paddr: 1b7402010
bss             vaddr: 560a57c3502c, paddr: 1b740202c
heap_main       vaddr: 560a590476b0, paddr: 1b7bcd6b0
heap_thread     vaddr: 7f9324000b60, paddr: 17ff2fb60
libc            vaddr: 7f9337ed5e10, paddr: 11f749e10
stack_main      vaddr: 7ffd8662f54c, paddr: 1a93d654c
stack_thread    vaddr: 7f9335e68cc4, paddr: 17a19ecc4
stack_thread_2  vaddr: 7f9335e68c94, paddr: 17a19ec94
data_thread     vaddr: 7f9335e695a0, paddr: 1a00455a0
data_thread_2   vaddr: 7f9335e695a4, paddr: 1a00455a4
bss_thread      vaddr: 7f9335e695b0, paddr: 1a00455b0
bss_thread_2    vaddr: 7f9335e695b4, paddr: 1a00455b4
Press Enter to end.
```

/proc/9538/maps

```
55a03e83e000-55a03e83f000 r--p 00000000 08:05 1048624                    /home/fabin/project2testQ1
55a03e83f000-55a03e840000 r-xp 00001000 08:05 1048624                    /home/fabin/project2testQ1
55a03e840000-55a03e841000 r--p 00002000 08:05 1048624                    /home/fabin/project2testQ1
55a03e841000-55a03e842000 r--p 00002000 08:05 1048624                    /home/fabin/project2testQ1
55a03e842000-55a03e843000 rw-p 00003000 08:05 1048624                    /home/fabin/project2testQ1
55a03f61a000-55a03f63b000 rw-p 00000000 00:00 0                          [heap]
7fd26c000000-7fd26c021000 rw-p 00000000 00:00 0
7fd26c021000-7fd270000000 ---p 00000000 00:00 0
7fd270000000-7fd270021000 rw-p 00000000 00:00 0
7fd270021000-7fd274000000 ---p 00000000 00:00 0
7fd274000000-7fd274021000 rw-p 00000000 00:00 0
7fd274021000-7fd278000000 ---p 00000000 00:00 0
7fd278000000-7fd278021000 rw-p 00000000 00:00 0
7fd278021000-7fd27c000000 ---p 00000000 00:00 0
7fd27c000000-7fd27c021000 rw-p 00000000 00:00 0
7fd27c021000-7fd280000000 ---p 00000000 00:00 0
7fd28390c000-7fd28390f000 r--p 00000000 08:05 2367418                    /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
7fd28390f000-7fd283921000 r-xp 00003000 08:05 2367418                    /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
7fd283921000-7fd283925000 r--p 00015000 08:05 2367418                    /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
7fd283925000-7fd283926000 r--p 00018000 08:05 2367418                    /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
7fd283926000-7fd283927000 rw-p 00019000 08:05 2367418                    /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
7fd283927000-7fd283928000 ---p 00000000 00:00 0
7fd283928000-7fd284128000 rw-p 00000000 00:00 0
7fd284128000-7fd284129000 ---p 00000000 00:00 0
7fd284129000-7fd284929000 rw-p 00000000 00:00 0
7fd284929000-7fd28492a000 ---p 00000000 00:00 0
7fd28492a000-7fd28512a000 rw-p 00000000 00:00 0
7fd28512a000-7fd28512b000 ---p 00000000 00:00 0
7fd28512b000-7fd28592b000 rw-p 00000000 00:00 0
7fd28612c000-7fd28612f000 rw-p 00000000 00:00 0
7fd28612f000-7fd286154000 r--p 00000000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd286154000-7fd2862cc000 r-xp 00025000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd2862cc000-7fd286316000 r--p 0019d000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd286316000-7fd286317000 ---p 001e7000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd286317000-7fd28631a000 r--p 001e7000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd28631a000-7fd28631d000 rw-p 001ea000 08:05 2367125                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd28631d000-7fd286321000 rw-p 00000000 00:00 0
7fd286321000-7fd286328000 r--p 00000000 08:05 2368044                    /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fd286328000-7fd286339000 r-xp 00007000 08:05 2368044                    /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fd286339000-7fd28633e000 r--p 00018000 08:05 2368044                    /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fd28633e000-7fd28633f000 r--p 0001c000 08:05 2368044                    /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fd28633f000-7fd286340000 rw-p 0001d000 08:05 2368044                    /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fd286340000-7fd286346000 rw-p 00000000 00:00 0
7fd286355000-7fd286356000 r--p 00000000 08:05 2366909                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd286356000-7fd286379000 r-xp 00001000 08:05 2366909                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd286379000-7fd286381000 r--p 00024000 08:05 2366909                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd286382000-7fd286383000 r--p 0002c000 08:05 2366909                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd286383000-7fd286384000 rw-p 0002d000 08:05 2366909                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd286384000-7fd286385000 rw-p 00000000 00:00 0
7fffa48af000-7fffa48d0000 rw-p 00000000 00:00 0                          [stack]
7fffa49d0000-7fffa49d4000 r--p 00000000 00:00 0                          [vvar]
7fffa49d4000-7fffa49d6000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

strace -f

```
execve("./project2testQ1", ["./project2testQ1"], 0x7ffff7bd7298 /* 57 vars */) = 0
brk(NULL)                               = 0x55a03f61a000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffa48ccfc0) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=61202, ...}) = 0
mmap(NULL, 61202, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd286346000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd286344000
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68
mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd286321000
mmap(0x7fd286328000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7fd286328000
mmap(0x7fd286339000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7fd286339000
mmap(0x7fd28633e000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c000) = 0x7fd28633e000
mmap(0x7fd286340000, 13432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd286340000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd28612f000
mprotect(0x7fd286154000, 1847296, PROT_NONE) = 0
mmap(0x7fd286154000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7fd286154000
mmap(0x7fd2862cc000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7fd2862cc000
mmap(0x7fd286317000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fd286317000
mmap(0x7fd28631d000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd28631d000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd28612c000
arch_prctl(ARCH_SET_FS, 0x7fd28612c880) = 0
mprotect(0x7fd286317000, 12288, PROT_READ) = 0
mprotect(0x7fd28633e000, 4096, PROT_READ) = 0
mprotect(0x55a03e841000, 4096, PROT_READ) = 0
```

```
mprotect(0x7fd286382000, 4096, PROT_READ) = 0
munmap(0x7fd286346000, 61202)          = 0
set_tid_address(0x7fd28612cb50)         = 9538
set_robust_list(0x7fd28612cb60, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7fd286328bf0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7fd2863363c0},
NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7fd286328c90, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7fd2863363c0}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
gettid()                               = 9538
getpid()                               = 9538
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                              = 0x55a03f61a000
brk(0x55a03f63b000)                      = 0x55a03f63b000
write(1, "main:\n", 6main:
)                 = 6
write(1, "pid: 9538, tid: 9538, &printf: 7"..., 44pid: 9538, tid: 9538, &printf: 7fd286193e10
) = 44
syscall_0x1bb(0xc, 0x7fd28612c7a0, 0x7fd28612c810, 0x55a03f61a6b0, 0x32, 0) = 0
write(1, "code         \tvaddr: 55a03e83f"..., 54code          vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
write(1, "data         \tvaddr: 55a03e842"..., 54data          vaddr: 55a03e842010, paddr: 1ba532010
) = 54
write(1, "data_2       \tvaddr: 55a03e842"..., 54data_2        vaddr: 55a03e842014, paddr: 1ba532014
) = 54
write(1, "bss          \tvaddr: 55a03e842"..., 54bss           vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
write(1, "bss_2        \tvaddr: 55a03e842"..., 54bss_2         vaddr: 55a03e842028, paddr: 1ba532028
) = 54
write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main         vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
write(1, "libc         \tvaddr: 7fd286193"..., 54libc          vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
write(1, "data_thread   \tvaddr: 7fd28612c"..., 54data_thread      vaddr: 7fd28612c720, paddr: 1baaf8720
) = 54
write(1, "data_thread_2  \tvaddr: 7fd28612c"..., 54data_thread_2    vaddr: 7fd28612c724, paddr: 1baaf8724
) = 54
write(1, "bss_thread    \tvaddr: 7fd28612c"..., 54bss_thread        vaddr: 7fd28612c730, paddr: 1baaf8730
) = 54
write(1, "bss_thread_2   \tvaddr: 7fd28612c"..., 54bss_thread_2     vaddr: 7fd28612c734, paddr: 1baaf8734
) = 54
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fd28592b000
mprotect(0x7fd28592c000, 8388608, PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7fd28612ae70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEART
IDstrace: Process 9539 attached
, parent_tid=[9539], tls=0x7fd28612b700, child_tidptr=0x7fd28612b9d0) = 9539
[pid  9538] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid  9539] set_robust_list(0x7fd28612b9e0, 24 <unfinished ...>
[pid  9538] <... mmap resumed>)        = 0x7fd28512a000
[pid  9539] <... set_robust_list resumed>) = 0
[pid  9538] mprotect(0x7fd28512b000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid  9539] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  9538] <... mprotect resumed>)    = 0
[pid  9538] clone(child_stack=0x7fd285929e70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEART
IDstrace: Process 9540 attached
, parent_tid=[9540], tls=0x7fd28592a700, child_tidptr=0x7fd28592a9d0) = 9540
[pid  9540] set_robust_list(0x7fd28592a9e0, 24 <unfinished ...>
[pid  9538] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid  9540] <... set_robust_list resumed>) = 0
[pid  9538] <... mmap resumed>)        = 0x7fd284929000
[pid  9540] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=2, tv_nsec=0},  <unfinished ...>
[pid  9538] mprotect(0x7fd28492a000, 8388608, PROT_READ|PROT_WRITE) = 0
[pid  9538] clone(child_stack=0x7fd285128e70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEART
IDstrace: Process 9541 attached
, parent_tid=[9541], tls=0x7fd285129700, child_tidptr=0x7fd2851299d0) = 9541
[pid  9541] set_robust_list(0x7fd2851299e0, 24 <unfinished ...>
[pid  9538] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid  9541] <... set_robust_list resumed>) = 0
[pid  9538] <... mmap resumed>)        = 0x7fd284128000
[pid  9541] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=3, tv_nsec=0},  <unfinished ...>
[pid  9538] mprotect(0x7fd284129000, 8388608, PROT_READ|PROT_WRITE) = 0
[pid  9538] clone(child_stack=0x7fd284927e70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEART
IDstrace: Process 9542 attached
, parent_tid=[9542], tls=0x7fd284928700, child_tidptr=0x7fd2849289d0) = 9542
[pid  9542] set_robust_list(0x7fd2849289e0, 24 <unfinished ...>
[pid  9538] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
[pid  9542] <... set_robust_list resumed>) = 0
[pid  9538] <... mmap resumed>)        = 0x7fd283927000
[pid  9542] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=4, tv_nsec=0},  <unfinished ...>
[pid  9538] mprotect(0x7fd283928000, 8388608, PROT_READ|PROT_WRITE) = 0
[pid  9538] clone(child_stack=0x7fd284126e70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEART
IDstrace: Process 9543 attached
, parent_tid=[9543], tls=0x7fd284127700, child_tidptr=0x7fd2841279d0) = 9543
[pid  9543] set_robust_list(0x7fd2841279e0, 24 <unfinished ...>
[pid  9538] futex(0x7fd28612b9d0, FUTEX_WAIT, 9539, NULL <unfinished ...>
[pid  9543] <... set_robust_list resumed>) = 0
[pid  9543] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=5, tv_nsec=0},  <unfinished ...>
[pid  9539] <... clock_nanosleep resumed>0x7fd28612ac70) = 0
[pid  9539] gettid()                   = 9539
[pid  9539] getpid()                   = 9538
[pid  9539] write(1, "thread 1:\n", 10thread 1:
```

```
) = 10
[pid  9539] write(1, "pid: 9538, tid: 9539, &printf: 7"..., 44pid: 9538, tid: 9539, &printf: 7fd286193e10
) = 44
[pid  9539] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fd27b927000
[pid  9539] munmap(0x7fd27b927000, 7180288) = 0
[pid  9539] munmap(0x7fd280000000, 59928576) = 0
[pid  9539] mprotect(0x7fd27c000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid  9539] syscall_0x1bb(0xd, 0x7fd28612b620, 0x7fd28612b690, 0x7fd27c000b60, 0, 0x100002543) = 0
[pid  9539] write(1, "code         \tvaddr: 55a03e83f"..., 54code           vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
[pid  9539] write(1, "data         \tvaddr: 55a03e842"..., 54data           vaddr: 55a03e842010, paddr: 1ba532010
) = 54
[pid  9539] write(1, "bss          \tvaddr: 55a03e842"..., 54bss            vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
[pid  9539] write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main     vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
[pid  9539] write(1, "heap_thread   \tvaddr: 7fd27c000"..., 54heap_thread       vaddr: 7fd27c000b60, paddr: 1ba352b60
) = 54
[pid  9539] write(1, "libc         \tvaddr: 7fd286193"..., 54libc           vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
[pid  9539] write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
[pid  9539] write(1, "stack_thread  \tvaddr: 7fd28612a"..., 54stack_thread     vaddr: 7fd28612acc4, paddr: 1bab39cc4
) = 54
[pid  9539] write(1, "stack_thread_2 \tvaddr: 7fd28612a"..., 54stack_thread_2   vaddr: 7fd28612ac94, paddr: 1bab39c94
) = 54
[pid  9539] write(1, "data_thread   \tvaddr: 7fd28612b"..., 54data_thread       vaddr: 7fd28612b5a0, paddr: 1bab385a0
) = 54
[pid  9539] write(1, "data_thread_2  \tvaddr: 7fd28612b"..., 54data_thread_2     vaddr: 7fd28612b5a4, paddr: 1bab385a4
) = 54
[pid  9539] write(1, "bss_thread    \tvaddr: 7fd28612b"..., 54bss_thread        vaddr: 7fd28612b5b0, paddr: 1bab385b0
) = 54
[pid  9539] write(1, "bss_thread_2   \tvaddr: 7fd28612b"..., 54bss_thread_2      vaddr: 7fd28612b5b4, paddr: 1bab385b4
) = 54
[pid  9539] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=5, tv_nsec=0},  <unfinished ...>
[pid  9540] <... clock_nanosleep resumed>0x7fd285929c70) = 0
[pid  9540] gettid()                = 9540
[pid  9540] getpid()                = 9538
[pid  9540] write(1, "thread 2:\n", 10thread 2:
) = 10
[pid  9540] write(1, "pid: 9538, tid: 9540, &printf: 7"..., 44pid: 9538, tid: 9540, &printf: 7fd286193e10
) = 44
[pid  9540] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fd274000000
[pid  9540] munmap(0x7fd278000000, 67108864) = 0
[pid  9540] mprotect(0x7fd274000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid  9540] syscall_0x1bb(0xd, 0x7fd28592a620, 0x7fd28592a690, 0x7fd274000b60, 0, 0x100002544) = 0
[pid  9540] write(1, "code         \tvaddr: 55a03e83f"..., 54code           vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
[pid  9540] write(1, "data         \tvaddr: 55a03e842"..., 54data           vaddr: 55a03e842010, paddr: 1ba532010
) = 54
[pid  9540] write(1, "bss          \tvaddr: 55a03e842"..., 54bss            vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
[pid  9540] write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main     vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
[pid  9540] write(1, "heap_thread   \tvaddr: 7fd274000"..., 54heap_thread       vaddr: 7fd274000b60, paddr: 173b2cb60
) = 54
[pid  9540] write(1, "libc         \tvaddr: 7fd286193"..., 54libc           vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
[pid  9540] write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
[pid  9540] write(1, "stack_thread  \tvaddr: 7fd285929"..., 54stack_thread     vaddr: 7fd285929cc4, paddr: 1bab3bcc4
) = 54
[pid  9540] write(1, "stack_thread_2 \tvaddr: 7fd285929"..., 54stack_thread_2   vaddr: 7fd285929c94, paddr: 1bab3bc94
) = 54
[pid  9540] write(1, "data_thread   \tvaddr: 7fd28592a"..., 54data_thread       vaddr: 7fd28592a5a0, paddr: 1bab3a5a0
) = 54
[pid  9540] write(1, "data_thread_2  \tvaddr: 7fd28592a"..., 54data_thread_2     vaddr: 7fd28592a5a4, paddr: 1bab3a5a4
) = 54
[pid  9540] write(1, "bss_thread    \tvaddr: 7fd28592a"..., 54bss_thread        vaddr: 7fd28592a5b0, paddr: 1bab3a5b0
) = 54
[pid  9540] write(1, "bss_thread_2   \tvaddr: 7fd28592a"..., 54bss_thread_2      vaddr: 7fd28592a5b4, paddr: 1bab3a5b4
) = 54
[pid  9540] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=4, tv_nsec=0},  <unfinished ...>
[pid  9541] <... clock_nanosleep resumed>0x7fd285128c70) = 0
[pid  9541] gettid()                = 9541
[pid  9541] getpid()                = 9538
[pid  9541] write(1, "thread 3:\n", 10thread 3:
) = 10
[pid  9541] write(1, "pid: 9538, tid: 9541, &printf: 7"..., 44pid: 9538, tid: 9541, &printf: 7fd286193e10
) = 44
[pid  9541] mmap(0x7fd278000000, 67108864, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fd278000000
[pid  9541] mprotect(0x7fd278000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid  9541] syscall_0x1bb(0xd, 0x7fd285129620, 0x7fd285129690, 0x7fd278000b60, 0, 0x100002545) = 0
[pid  9541] write(1, "code         \tvaddr: 55a03e83f"..., 54code           vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
[pid  9541] write(1, "data         \tvaddr: 55a03e842"..., 54data           vaddr: 55a03e842010, paddr: 1ba532010
) = 54
[pid  9541] write(1, "bss          \tvaddr: 55a03e842"..., 54bss            vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
[pid  9541] write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main     vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
[pid  9541] write(1, "heap_thread   \tvaddr: 7fd278000"..., 54heap_thread       vaddr: 7fd278000b60, paddr: 1b7170b60
) = 54
[pid  9541] write(1, "libc         \tvaddr: 7fd286193"..., 54libc           vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
[pid  9541] write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
[pid  9541] write(1, "stack_thread  \tvaddr: 7fd285128"..., 54stack_thread     vaddr: 7fd285128cc4, paddr: 1bab3dcc4
) = 54
```

```
[pid  9541] write(1, "stack_thread_2 \tvaddr: 7fd285128"..., 54stack_thread_2   vaddr: 7fd285128c94, paddr: 1bab3dc94
) = 54
[pid  9541] write(1, "data_thread   \tvaddr: 7fd285129"..., 54data_thread      vaddr: 7fd2851295a0, paddr: 1bab3c5a0
) = 54
[pid  9541] write(1, "data_thread_2  \tvaddr: 7fd285129"..., 54data_thread_2    vaddr: 7fd2851295a4, paddr: 1bab3c5a4
) = 54
[pid  9541] write(1, "bss_thread    \tvaddr: 7fd285129"..., 54bss_thread        vaddr: 7fd2851295b0, paddr: 1bab3c5b0
) = 54
[pid  9541] write(1, "bss_thread_2  \tvaddr: 7fd285129"..., 54bss_thread_2      vaddr: 7fd2851295b4, paddr: 1bab3c5b4
) = 54
[pid  9541] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=3, tv_nsec=0},  <unfinished ...>
[pid  9542] <... clock_nanosleep resumed>0x7fd284927c70) = 0
[pid  9542] gettid()                = 9542
[pid  9542] getpid()                = 9538
[pid  9542] write(1, "thread 4:\n", 10thread 4:
) = 10
[pid  9542] write(1, "pid: 9538, tid: 9542, &printf: 7"..., 44pid: 9538, tid: 9542, &printf: 7fd286193e10
) = 44
[pid  9542] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fd26c000000
[pid  9542] munmap(0x7fd270000000, 67108864) = 0
[pid  9542] mprotect(0x7fd26c000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid  9542] syscall_0x1bb(0xd, 0x7fd284928620, 0x7fd284928690, 0x7fd26c000b60, 0, 0x100002546) = 0
[pid  9542] write(1, "code         \tvaddr: 55a03e83f"..., 54code           vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
[pid  9542] write(1, "data         \tvaddr: 55a03e842"..., 54data           vaddr: 55a03e842010, paddr: 1ba532010
) = 54
[pid  9542] write(1, "bss          \tvaddr: 55a03e842"..., 54bss            vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
[pid  9542] write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main     vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
[pid  9542] write(1, "heap_thread   \tvaddr: 7fd26c000"..., 54heap_thread       vaddr: 7fd26c000b60, paddr: 176b47b60
) = 54
[pid  9542] write(1, "libc         \tvaddr: 7fd286193"..., 54libc           vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
[pid  9542] write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
[pid  9542] write(1, "stack_thread   \tvaddr: 7fd284927"..., 54stack_thread      vaddr: 7fd284927cc4, paddr: 1bab3fcc4
) = 54
[pid  9542] write(1, "stack_thread_2 \tvaddr: 7fd284927"..., 54stack_thread_2   vaddr: 7fd284927c94, paddr: 1bab3fc94
) = 54
[pid  9542] write(1, "data_thread   \tvaddr: 7fd284928"..., 54data_thread       vaddr: 7fd2849285a0, paddr: 1bab3e5a0
) = 54
[pid  9542] write(1, "data_thread_2  \tvaddr: 7fd284928"..., 54data_thread_2    vaddr: 7fd2849285a4, paddr: 1bab3e5a4
) = 54
[pid  9542] write(1, "bss_thread    \tvaddr: 7fd284928"..., 54bss_thread        vaddr: 7fd2849285b0, paddr: 1bab3e5b0
) = 54
[pid  9542] write(1, "bss_thread_2  \tvaddr: 7fd284928"..., 54bss_thread_2      vaddr: 7fd2849285b4, paddr: 1bab3e5b4
) = 54
[pid  9542] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=2, tv_nsec=0},  <unfinished ...>
[pid  9543] <... clock_nanosleep resumed>0x7fd284126c70) = 0
[pid  9543] gettid()                = 9543
[pid  9543] getpid()                = 9538
[pid  9543] write(1, "thread 5:\n", 10thread 5:
) = 10
[pid  9543] write(1, "pid: 9538, tid: 9543, &printf: 7"..., 44pid: 9538, tid: 9543, &printf: 7fd286193e10
) = 44
[pid  9543] mmap(0x7fd270000000, 67108864, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fd270000000
[pid  9543] mprotect(0x7fd270000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid  9543] syscall_0x1bb(0xd, 0x7fd284127620, 0x7fd284127690, 0x7fd270000b60, 0, 0x100002547) = 0
[pid  9543] write(1, "code         \tvaddr: 55a03e83f"..., 54code           vaddr: 55a03e83f41d, paddr: 168d0a41d
) = 54
[pid  9543] write(1, "data         \tvaddr: 55a03e842"..., 54data           vaddr: 55a03e842010, paddr: 1ba532010
) = 54
[pid  9543] write(1, "bss          \tvaddr: 55a03e842"..., 54bss            vaddr: 55a03e84202c, paddr: 1ba53202c
) = 54
[pid  9543] write(1, "heap_main     \tvaddr: 55a03f61a"..., 54heap_main     vaddr: 55a03f61a6b0, paddr: 1bab4f6b0
) = 54
[pid  9543] write(1, "heap_thread   \tvaddr: 7fd270000"..., 54heap_thread       vaddr: 7fd270000b60, paddr: 1b771cb60
) = 54
[pid  9543] write(1, "libc         \tvaddr: 7fd286193"..., 54libc           vaddr: 7fd286193e10, paddr: 11f749e10
) = 54
[pid  9543] write(1, "stack_main    \tvaddr: 7fffa48cc"..., 54stack_main        vaddr: 7fffa48cce6c, paddr: 1ba412e6c
) = 54
[pid  9543] write(1, "stack_thread   \tvaddr: 7fd284126"..., 54stack_thread      vaddr: 7fd284126cc4, paddr: 1bab23cc4
) = 54
[pid  9543] write(1, "stack_thread_2 \tvaddr: 7fd284126"..., 54stack_thread_2   vaddr: 7fd284126c94, paddr: 1bab23c94
) = 54
[pid  9543] write(1, "data_thread   \tvaddr: 7fd284127"..., 54data_thread       vaddr: 7fd2841275a0, paddr: 1bab405a0
) = 54
[pid  9543] write(1, "data_thread_2  \tvaddr: 7fd284127"..., 54data_thread_2    vaddr: 7fd2841275a4, paddr: 1bab405a4
) = 54
[pid  9543] write(1, "bss_thread    \tvaddr: 7fd284127"..., 54bss_thread        vaddr: 7fd2841275b0, paddr: 1bab405b0
) = 54
[pid  9543] write(1, "bss_thread_2  \tvaddr: 7fd284127"..., 54bss_thread_2      vaddr: 7fd2841275b4, paddr: 1bab405b4
) = 54
[pid  9543] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=1, tv_nsec=0},  <unfinished ...>
[pid  9541] <... clock_nanosleep resumed>0x7fd285128c70) = 0
[pid  9540] <... clock_nanosleep resumed>0x7fd285929c70) = 0
[pid  9541] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC <unfinished ...>
[pid  9540] futex(0x7fd286383968, FUTEX_WAIT_PRIVATE, 2, NULL <unfinished ...>
[pid  9541] <... openat resumed>)      = 3
[pid  9541] fstat(3, {st_mode=S_IFREG|0644, st_size=61202, ...}) = 0
[pid  9541] mmap(NULL, 61202, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd286346000
[pid  9542] <... clock_nanosleep resumed>0x7fd284927c70) = 0
[pid  9541] close(3 <unfinished ...>
[pid  9542] futex(0x7fd286383968, FUTEX_WAIT_PRIVATE, 2, NULL <unfinished ...>
[pid  9541] <... close resumed>)       = 0
[pid  9541] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
[pid  9541] read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\3405\0\0\0\0\0\0"..., 832) = 832
```

```
[pid  9541] fstat(3, {st_mode=S_IFREG|0644, st_size=104984, ...}) = 0
[pid  9541] mmap(NULL, 107592, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd28390c000
[pid  9541] mmap(0x7fd28390f000, 73728, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7fd28390f000
[pid  9541] mmap(0x7fd283921000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15000) = 0x7fd283921000
[pid  9541] mmap(0x7fd283925000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7fd283925000
[pid  9541] close(3)                    = 0
[pid  9541] mprotect(0x7fd283925000, 4096, PROT_READ <unfinished ...>
[pid  9543] <... clock_nanosleep resumed>0x7fd284126c70) = 0
[pid  9541] <... mprotect resumed>)     = 0
[pid  9543] futex(0x7fd286383968, FUTEX_WAIT_PRIVATE, 2, NULL <unfinished ...>
[pid  9541] munmap(0x7fd286346000, 61202) = 0
[pid  9541] futex(0x7fd286383968, FUTEX_WAKE_PRIVATE, 1) = 1
[pid  9541] futex(0x7fd2839261e0, FUTEX_WAKE_PRIVATE, 2147483647 <unfinished ...>
[pid  9540] <... futex resumed>)        = 0
[pid  9541] <... futex resumed>)        = 0
[pid  9540] futex(0x7fd286383968, FUTEX_WAKE_PRIVATE, 1 <unfinished ...>
[pid  9542] <... futex resumed>)        = 0
[pid  9541] madvise(0x7fd284929000, 8368128, MADV_DONTNEED <unfinished ...>
[pid  9540] <... futex resumed>)        = 1
[pid  9542] futex(0x7fd286383968, FUTEX_WAKE_PRIVATE, 1 <unfinished ...>
[pid  9541] <... madvise resumed>)      = 0
[pid  9540] madvise(0x7fd28512a000, 8368128, MADV_DONTNEED <unfinished ...>
[pid  9542] <... futex resumed>)        = 1
[pid  9541] exit(0 <unfinished ...>
[pid  9543] <... futex resumed>)        = 0
[pid  9541] <... exit resumed>)         = ?
[pid  9540] <... madvise resumed>)      = 0
[pid  9542] madvise(0x7fd284128000, 8368128, MADV_DONTNEED <unfinished ...>
[pid  9543] futex(0x7fd286383968, FUTEX_WAKE_PRIVATE, 1 <unfinished ...>
[pid  9542] <... madvise resumed>)      = 0
[pid  9541] +++ exited with 0 +++
[pid  9540] exit(0 <unfinished ...>
[pid  9543] <... futex resumed>)        = 0
[pid  9542] exit(0 <unfinished ...>
[pid  9540] <... exit resumed>)         = ?
[pid  9543] madvise(0x7fd283927000, 8368128, MADV_DONTNEED <unfinished ...>
[pid  9542] <... exit resumed>)         = ?
[pid  9540] +++ exited with 0 +++
[pid  9543] <... madvise resumed>)      = 0
[pid  9542] +++ exited with 0 +++
[pid  9543] exit(0)                     = ?
[pid  9543] +++ exited with 0 +++
[pid  9539] <... clock_nanosleep resumed>0x7fd28612ac70) = 0
[pid  9539] madvise(0x7fd28592b000, 8368128, MADV_DONTNEED) = 0
[pid  9539] exit(0)                     = ?
[pid  9539] +++ exited with 0 +++
<... futex resumed>)                    = 0
munmap(0x7fd28592b000, 8392704)         = 0
write(1, "Press Enter to end.\n", 20Press Enter to end.
)   = 20
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0x55a03f61acc0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0,
```