# fork()定义

fork()函数在glibc的如下位置进行声明：

glibc： posix/unistd.h + 756

1. /\* Clone the calling process, creating an exact copy.
2. Return -1 for errors, 0 to the new process,
3. and the process ID of the new process to the old process.  \*/
4. **extern** \_\_pid\_t fork (**void**) \_\_THROWNL;

但是，在glibc中没有直接搜索到fork()函数的实现。

尝试搜索fork，发现如下位置有一个alias的声明：

glibc-2.29/sysdeps/nptl/fork.c + 157

1. weak\_alias (\_\_libc\_fork, fork)

而weak\_alias在glibc-2.29/include/libc-symbols.h中定义如下：

1. /\* Define ALIASNAME as a weak alias for NAME.
2. If weak aliases are not available, this defines a strong alias.  \*/
3. # define weak\_alias(name, aliasname) \_weak\_alias (name, aliasname)
4. # define \_weak\_alias(name, aliasname) \
5. **extern** \_\_typeof (name) aliasname \_\_attribute\_\_ ((weak, alias (#name))) \
6. \_\_attribute\_copy\_\_ (name);

最终weak\_alias展开如下：

1. **extern** \_\_pid\_t fork \_\_attribute\_\_(weak, alias(\_\_libc\_fork)) \_\_attirbute\_copy\_\_(\_\_libc\_fork);

由于fork()没有定义，因此最终会跳转到对应的弱符号\_\_libc\_fork()进行执行(这里使用弱符号是为了防止fork()已经有定义导致链接冲突出现错误)。

# **\_\_libc\_fork()**

\_\_libc\_fork()调用arch\_fork()跳转到对应arch的定义处进行下一步执行。

Glibc到kernel的跳转 待续。。。。

系统调用的name为clone。

# **Arm64异常向量表**

Arm64在arch/arm64/kernel/entry.S中配置异常向量表：

1. /\*
2. \* Exception vectors.
3. \*/
4. .align  11
5. ENTRY(vectors)
6. ventry  el1\_sync\_invalid        // Synchronous EL1t
7. ventry  el1\_irq\_invalid         // IRQ EL1t
8. ventry  el1\_fiq\_invalid         // FIQ EL1t
9. ventry  el1\_error\_invalid       // Error EL1t
11. ventry  el1\_sync            // Synchronous EL1h
12. ventry  el1\_irq             // IRQ EL1h
13. ventry  el1\_fiq\_invalid         // FIQ EL1h
14. ventry  el1\_error\_invalid       // Error EL1h
16. ventry  el0\_sync            // Synchronous 64-bit EL0
17. ventry  el0\_irq             // IRQ 64-bit EL0
18. ventry  el0\_fiq\_invalid         // FIQ 64-bit EL0
19. ventry  el0\_error\_invalid       // Error 64-bit EL0
21. #ifdef CONFIG\_COMPAT
22. ventry  el0\_sync\_compat         // Synchronous 32-bit EL0
23. ventry  el0\_irq\_compat          // IRQ 32-bit EL0
24. ventry  el0\_fiq\_invalid\_compat      // FIQ 32-bit EL0
25. ventry  el0\_error\_invalid\_compat    // Error 32-bit EL0
26. #else
27. ventry  el0\_sync\_invalid        // Synchronous 32-bit EL0
28. ventry  el0\_irq\_invalid         // IRQ 32-bit EL0
29. ventry  el0\_fiq\_invalid         // FIQ 32-bit EL0
30. ventry  el0\_error\_invalid       // Error 32-bit EL0
31. #endif
32. END(vectors)

glibc中的fork()函数执行svc指令完成从el0到el1的跳转属于el0\_sync异常，el0\_sync异常中对于el0\_svc的处理如下：

1. /\*
2. \* SVC handler.
3. \*/
4. .align  6
5. el0\_svc:
6. adrp    stbl, sys\_call\_table        // load syscall table pointer(加载系统调用表)
7. uxtw    scno, w8            // syscall number in w8
8. mov sc\_nr, #\_\_NR\_syscalls
9. el0\_svc\_naked:                  // compat entry point
10. stp x0, scno, [sp, #S\_ORIG\_X0]  // save the original x0 and syscall number
11. enable\_dbg\_and\_irq
12. ct\_user\_exit 1
14. ldr x16, [tsk, #TI\_FLAGS]       // check for syscall hooks
15. tst x16, #\_TIF\_SYSCALL\_WORK
16. b.ne    \_\_sys\_trace
17. cmp     scno, sc\_nr                     // check upper syscall limit
18. b.hs    ni\_sys
19. ldr x16, [stbl, scno, lsl #3]   // address in the syscall table(根据系统调用号scno计算在系统调用表中的偏移)
20. blr x16             // call sys\_\* routine(完成跳转)
21. b   ret\_fast\_syscall

主要干了三件事情：

1. 加载系统调用表；
2. 根据传进来的系统调用号计算偏移；
3. 跳转到对应的系统调用进行处理。

系统调用表在include/asm/syscall.h和unistd.h中进行了定义。

待确认：\_\_NR\_vfork、\_\_NR\_clone、\_\_NR\_fork三者的区别。

(include/uapi/asm-generic/unistd.h)

#define \_\_NR\_vfork 1071

\_\_SYSCALL(\_\_NR\_vfork, sys\_vfork)

#define \_\_NR\_clone 220

\_\_SYSCALL(\_\_NR\_clone, sys\_clone)

#define \_\_NR\_fork 1079

#ifdef CONFIG\_MMU

\_\_SYSCALL(\_\_NR\_fork, sys\_fork)

#else

\_\_SYSCALL(\_\_NR\_fork, sys\_ni\_syscall)

#endif /\* CONFIG\_MMU \*/

sys\_fork:

1. #ifdef \_\_ARCH\_WANT\_SYS\_FORK
2. SYSCALL\_DEFINE0(fork)
3. {
4. #ifdef CONFIG\_MMU
5. **return** do\_fork(SIGCHLD, 0, 0, NULL, NULL);
6. #else
7. /\* can not support in nommu mode \*/
8. **return** -EINVAL;
9. #endif
10. }
11. #endif

sys\_vfork:

1. #ifdef \_\_ARCH\_WANT\_SYS\_VFORK
2. SYSCALL\_DEFINE0(vfork)
3. {
4. **return** do\_fork(CLONE\_VFORK | CLONE\_VM | SIGCHLD, 0,  0, NULL, NULL);
5. }
6. #endif

sys\_clone：

1. #ifdef \_\_ARCH\_WANT\_SYS\_CLONE
2. #ifdef CONFIG\_CLONE\_BACKWARDS
3. SYSCALL\_DEFINE5(clone, unsigned **long**, clone\_flags, unsigned **long**, newsp,
4. **int** \_\_user \*, parent\_tidptr,
5. **int**, tls\_val,
6. **int** \_\_user \*, child\_tidptr)
7. #elif defined(CONFIG\_CLONE\_BACKWARDS2)
8. SYSCALL\_DEFINE5(clone, unsigned **long**, newsp, unsigned **long**, clone\_flags,
9. **int** \_\_user \*, parent\_tidptr,
10. **int** \_\_user \*, child\_tidptr,
11. **int**, tls\_val)
12. #elif defined(CONFIG\_CLONE\_BACKWARDS3)
13. SYSCALL\_DEFINE6(clone, unsigned **long**, clone\_flags, unsigned **long**, newsp,
14. **int**, stack\_size,
15. **int** \_\_user \*, parent\_tidptr,
16. **int** \_\_user \*, child\_tidptr,
17. **int**, tls\_val)
18. #else
19. SYSCALL\_DEFINE5(clone, unsigned **long**, clone\_flags, unsigned **long**, newsp,
20. **int** \_\_user \*, parent\_tidptr,
21. **int** \_\_user \*, child\_tidptr,
22. **int**, tls\_val)
23. #endif
24. {
25. **return** do\_fork(clone\_flags, newsp, 0, parent\_tidptr, child\_tidptr);
26. }
27. #endif

fork()函数对应的系统调用是sys\_clone。

这三个函数，最终都会调用do\_fork()函数。

# **do\_fork()**

do\_fork()函数参数说明:

1. /\*
2. \*  Ok, this is the main fork-routine.
3. \*
4. \* It copies the process, and if successful kick-starts
5. \* it and waits for it to finish using the VM if required.
6. \*/
7. **long** do\_fork(unsigned **long** clone\_flags,
8. unsigned **long** stack\_start,
9. unsigned **long** stack\_size,
10. **int** \_\_user \*parent\_tidptr,
11. **int** \_\_user \*child\_tidptr)

clone\_flags: 克隆标志，在 include/uapi/linux/sched.h 中定义

/\* CLONE\_VM：共享内存描述符mm \*/

#define CLONE\_VM 0x00000100 /\* set if VM shared between processes \*/

kernel/fork.c + 966：copy\_mm()，如果flags = CLONE\_VM就不会创建新的mm

1. /\*
2. \* Are we cloning a kernel thread?
3. \*
4. \* We need to steal a active VM for that..
5. \*/
6. oldmm = current->mm;
8. **if** (clone\_flags & CLONE\_VM) {
9. atomic\_inc(&oldmm->mm\_users);
10. mm = oldmm;
11. **goto** good\_mm;
12. }

/\* CLONE\_VFORK：vfork系统调用时使用 \*/

#define CLONE\_VFORK 0x00004000 /\* set if the parent wants the child to wake it up on mm\_release \*/

do\_fork()具体实现：