**第0002讲 进程原理及系统调用**

**1、进程有两种特殊的形式：没有用户虚拟地址空间的进程叫内核线程，共享用户虚拟地址空间的进程叫用户线程。共享同一个用户虚拟地址空间的所有用户线程叫线程组。**

**C语言标准库进程 Linux内核进程**

包括多个线程的进程 线程组

只有一个线程的进程 任务或进程

线程 共享用户虚拟地址空间的进程

**2、Linux内核提供API函数来设置进程的状态：**

1. TASK\_RUNNING（可运行状态或者可就绪状态）；
2. TASK\_INTERRUPTIBLE（可中断睡眠状态，又叫浅睡眠状态）；
3. TASK\_UNINTERRUPTIBLE（不可中断状态，又叫深度睡眠状态）；我们可以通过ps命令查看被标记为D状态的进程。
4. \_\_TASK\_STOPPED（终止状态）；
5. EXIT\_ZOMBIE（僵尸状态）；

3、Linux内核进程描述符task\_struct数据结构，源码分析如下：

// 进程描述符 task\_struct

struct task\_struct {

#ifdef CONFIG\_THREAD\_INFO\_IN\_TASK

    /\*

     \* For reasons of header soup (see current\_thread\_info()), this

     \* must be the first element of task\_struct.

     \*/

    struct thread\_info      thread\_info;

#endif

    /\* -1 unrunnable, 0 runnable, >0 stopped: \*/

    volatile long           state; // 进程的状态标志

    /\*

     \* This begins the randomizable portion of task\_struct. Only

     \* scheduling-critical items should be added above here.

     \*/

    randomized\_struct\_fields\_start

    void                \*stack; // 指向内核栈

    refcount\_t          usage;

    /\* Per task flags (PF\_\*), defined further below: \*/

    unsigned int            flags;

    unsigned int            ptrace;

#ifdef CONFIG\_SMP

    struct llist\_node       wake\_entry;

    int             on\_cpu;

#ifdef CONFIG\_THREAD\_INFO\_IN\_TASK

    /\* Current CPU: \*/

    unsigned int            cpu;

#endif

    unsigned int            wakee\_flips;

    unsigned long           wakee\_flip\_decay\_ts;

    struct task\_struct      \*last\_wakee;

    /\*

     \* recent\_used\_cpu is initially set as the last CPU used by a task

     \* that wakes affine another task. Waker/wakee relationships can

     \* push tasks around a CPU where each wakeup moves to the next one.

     \* Tracking a recently used CPU allows a quick search for a recently

     \* used CPU that may be idle.

     \*/

    int             recent\_used\_cpu;

    int             wake\_cpu;

#endif

    int             on\_rq;

    /\* 下面这个四是进程调度策略和优先级\*/

    int             prio;

    int             static\_prio;

    int             normal\_prio;

    unsigned int            rt\_priority;

    const struct sched\_class    \*sched\_class;

    struct sched\_entity     se;

    struct sched\_rt\_entity      rt;

#ifdef CONFIG\_CGROUP\_SCHED

    struct task\_group       \*sched\_task\_group;

#endif

    struct sched\_dl\_entity      dl;

#ifdef CONFIG\_UCLAMP\_TASK

    /\* Clamp values requested for a scheduling entity \*/

    struct uclamp\_se        uclamp\_req[UCLAMP\_CNT];

    /\* Effective clamp values used for a scheduling entity \*/

    struct uclamp\_se        uclamp[UCLAMP\_CNT];

#endif

#ifdef CONFIG\_PREEMPT\_NOTIFIERS

    /\* List of struct preempt\_notifier: \*/

    struct hlist\_head       preempt\_notifiers;

#endif

#ifdef CONFIG\_BLK\_DEV\_IO\_TRACE

    unsigned int            btrace\_seq;

#endif

    unsigned int            policy;

    int             nr\_cpus\_allowed;

    const cpumask\_t         \*cpus\_ptr; // 此成员允许进程在那个CPU上运行

    cpumask\_t           cpus\_mask;

#ifdef CONFIG\_PREEMPT\_RCU

    int             rcu\_read\_lock\_nesting;

    union rcu\_special       rcu\_read\_unlock\_special;

    struct list\_head        rcu\_node\_entry;

    struct rcu\_node         \*rcu\_blocked\_node;

#endif /\* #ifdef CONFIG\_PREEMPT\_RCU \*/

#ifdef CONFIG\_TASKS\_RCU

    unsigned long           rcu\_tasks\_nvcsw;

    u8              rcu\_tasks\_holdout;

    u8              rcu\_tasks\_idx;

    int             rcu\_tasks\_idle\_cpu;

    struct list\_head        rcu\_tasks\_holdout\_list;

#endif /\* #ifdef CONFIG\_TASKS\_RCU \*/

    struct sched\_info       sched\_info;

    struct list\_head        tasks;

#ifdef CONFIG\_SMP

    struct plist\_node       pushable\_tasks;

    struct rb\_node          pushable\_dl\_tasks;

#endif

    // 这个两指针指向内存描述符。进程：mm和active\_mm指向同一个内存描述符。内核线程：mm是空指针，

    // 当内核线程运行时，active\_mm指向从进程借用内存描述符

    struct mm\_struct        \*mm;

    struct mm\_struct        \*active\_mm;

    /\* Per-thread vma caching: \*/

    struct vmacache         vmacache;

#ifdef SPLIT\_RSS\_COUNTING

    struct task\_rss\_stat        rss\_stat;

#endif

    int             exit\_state;

    int             exit\_code;

    int             exit\_signal;

    /\* The signal sent when the parent dies: \*/

    int             pdeath\_signal;

    /\* JOBCTL\_\*, siglock protected: \*/

    unsigned long           jobctl;

    /\* Used for emulating ABI behavior of previous Linux versions: \*/

    unsigned int            personality;

    /\* Scheduler bits, serialized by scheduler locks: \*/

    unsigned            sched\_reset\_on\_fork:1;

    unsigned            sched\_contributes\_to\_load:1;

    unsigned            sched\_migrated:1;

    unsigned            sched\_remote\_wakeup:1;

#ifdef CONFIG\_PSI

    unsigned            sched\_psi\_wake\_requeue:1;

#endif

    /\* Force alignment to the next boundary: \*/

    unsigned            :0;

    /\* Unserialized, strictly 'current' \*/

    /\* Bit to tell LSMs we're in execve(): \*/

    unsigned            in\_execve:1;

    unsigned            in\_iowait:1;

#ifndef TIF\_RESTORE\_SIGMASK

    unsigned            restore\_sigmask:1;

#endif

#ifdef CONFIG\_MEMCG

    unsigned            in\_user\_fault:1;

#endif

#ifdef CONFIG\_COMPAT\_BRK

    unsigned            brk\_randomized:1;

#endif

#ifdef CONFIG\_CGROUPS

    /\* disallow userland-initiated cgroup migration \*/

    unsigned            no\_cgroup\_migration:1;

    /\* task is frozen/stopped (used by the cgroup freezer) \*/

    unsigned            frozen:1;

#endif

#ifdef CONFIG\_BLK\_CGROUP

    /\* to be used once the psi infrastructure lands upstream. \*/

    unsigned            use\_memdelay:1;

#endif

    unsigned long           atomic\_flags; /\* Flags requiring atomic access. \*/

    struct restart\_block        restart\_block;

    pid\_t               pid;  // 全局的进程号

    pid\_t               tgid; // 全局的线程组标识符

#ifdef CONFIG\_STACKPROTECTOR

    /\* Canary value for the -fstack-protector GCC feature: \*/

    unsigned long           stack\_canary;

#endif

    /\*

     \* Pointers to the (original) parent process, youngest child, younger sibling,

     \* older sibling, respectively.  (p->father can be replaced with

     \* p->real\_parent->pid)

     \*/

    /\* Real parent process: \*/

    struct task\_struct \_\_rcu    \*real\_parent; // 指向真实的父进程

    /\* Recipient of SIGCHLD, wait4() reports: \*/

    struct task\_struct \_\_rcu    \*parent; // 指向父进程

    /\*

     \* Children/sibling form the list of natural children:

     \*/

    struct list\_head        children;

    struct list\_head        sibling;

    struct task\_struct      \*group\_leader; // 指向线程组的组长

    /\*

     \* 'ptraced' is the list of tasks this task is using ptrace() on.

     \*

     \* This includes both natural children and PTRACE\_ATTACH targets.

     \* 'ptrace\_entry' is this task's link on the p->parent->ptraced list.

     \*/

    struct list\_head        ptraced;

    struct list\_head        ptrace\_entry;

    /\* PID/PID hash table linkage. \*/

    struct pid          \*thread\_pid;

    struct hlist\_node       pid\_links[PIDTYPE\_MAX]; // 进程号，进程组标识符和会话标识符

    struct list\_head        thread\_group;

    struct list\_head        thread\_node;

    struct completion       \*vfork\_done;

    /\* CLONE\_CHILD\_SETTID: \*/

    int \_\_user          \*set\_child\_tid;

    /\* CLONE\_CHILD\_CLEARTID: \*/

    int \_\_user          \*clear\_child\_tid;

    u64             utime;

    u64             stime;

#ifdef CONFIG\_ARCH\_HAS\_SCALED\_CPUTIME

    u64             utimescaled;

    u64             stimescaled;

#endif

    u64             gtime;

    struct prev\_cputime     prev\_cputime;

#ifdef CONFIG\_VIRT\_CPU\_ACCOUNTING\_GEN

    struct vtime            vtime;

#endif

#ifdef CONFIG\_NO\_HZ\_FULL

    atomic\_t            tick\_dep\_mask;

#endif

    /\* Context switch counts: \*/

    unsigned long           nvcsw;

    unsigned long           nivcsw;

    /\* Monotonic time in nsecs: \*/

    u64             start\_time;

    /\* Boot based time in nsecs: \*/

    u64             start\_boottime;

    /\* MM fault and swap info: this can arguably be seen as either mm-specific or thread-specific: \*/

    unsigned long           min\_flt;

    unsigned long           maj\_flt;

    /\* Empty if CONFIG\_POSIX\_CPUTIMERS=n \*/

    struct posix\_cputimers      posix\_cputimers;

    /\* Process credentials: \*/

    /\* Tracer's credentials at attach: \*/

    const struct cred \_\_rcu     \*ptracer\_cred;

    /\* Objective and real subjective task credentials (COW): \*/

    const struct cred \_\_rcu     \*real\_cred; // 此成员指向主体和真实客体证书

    /\* Effective (overridable) subjective task credentials (COW): \*/

    const struct cred \_\_rcu     \*cred; // 指向有效客体证书

#ifdef CONFIG\_KEYS

    /\* Cached requested key. \*/

    struct key          \*cached\_requested\_key;

#endif

    /\*

     \* executable name, excluding path.

     \*

     \* - normally initialized setup\_new\_exec()

     \* - access it with [gs]et\_task\_comm()

     \* - lock it with task\_lock()

     \*/

    char                comm[TASK\_COMM\_LEN]; // 进程名称

    struct nameidata        \*nameidata;

// 下面这两个成员用于UNIX系统：信号量和共享内存

#ifdef CONFIG\_SYSVIPC

    struct sysv\_sem         sysvsem;

    struct sysv\_shm         sysvshm;

#endif

#ifdef CONFIG\_DETECT\_HUNG\_TASK

    unsigned long           last\_switch\_count;

    unsigned long           last\_switch\_time;

#endif

    /\* Filesystem information: \*/

    struct fs\_struct        \*fs; // 此成员文件系统信息，主要是进程的根目录和当前工作目录

    /\* Open file information: \*/

    struct files\_struct     \*files;  //  打开文件表

    /\* Namespaces: \*/

    struct nsproxy          \*nsproxy; // 命名空间

    // 下面这个块的成员主要是用于：信号处理

    /\* Signal handlers: \*/

    struct signal\_struct        \*signal;

    struct sighand\_struct \_\_rcu     \*sighand;

    sigset\_t            blocked;

    sigset\_t            real\_blocked;

    /\* Restored if set\_restore\_sigmask() was used: \*/

    sigset\_t            saved\_sigmask;

    struct sigpending       pending;

    unsigned long           sas\_ss\_sp;

    size\_t              sas\_ss\_size;

    unsigned int            sas\_ss\_flags;

    struct callback\_head        \*task\_works;

#ifdef CONFIG\_AUDIT

#ifdef CONFIG\_AUDITSYSCALL

    struct audit\_context        \*audit\_context;

#endif

    kuid\_t              loginuid;

    unsigned int            sessionid;

#endif

    struct seccomp          seccomp;

    /\* Thread group tracking: \*/

    u64             parent\_exec\_id;

    u64             self\_exec\_id;

    /\* Protection against (de-)allocation: mm, files, fs, tty, keyrings, mems\_allowed, mempolicy: \*/

    spinlock\_t          alloc\_lock;

    /\* Protection of the PI data structures: \*/

    raw\_spinlock\_t          pi\_lock;

    struct wake\_q\_node      wake\_q;

#ifdef CONFIG\_RT\_MUTEXES

    /\* PI waiters blocked on a rt\_mutex held by this task: \*/

    struct rb\_root\_cached       pi\_waiters;

    /\* Updated under owner's pi\_lock and rq lock \*/

    struct task\_struct      \*pi\_top\_task;

    /\* Deadlock detection and priority inheritance handling: \*/

    struct rt\_mutex\_waiter      \*pi\_blocked\_on;

#endif

#ifdef CONFIG\_DEBUG\_MUTEXES

    /\* Mutex deadlock detection: \*/

    struct mutex\_waiter     \*blocked\_on;

#endif

#ifdef CONFIG\_DEBUG\_ATOMIC\_SLEEP

    int             non\_block\_count;

#endif

#ifdef CONFIG\_TRACE\_IRQFLAGS

    unsigned int            irq\_events;

    unsigned long           hardirq\_enable\_ip;

    unsigned long           hardirq\_disable\_ip;

    unsigned int            hardirq\_enable\_event;

    unsigned int            hardirq\_disable\_event;

    int             hardirqs\_enabled;

    int             hardirq\_context;

    unsigned long           softirq\_disable\_ip;

    unsigned long           softirq\_enable\_ip;

    unsigned int            softirq\_disable\_event;

    unsigned int            softirq\_enable\_event;

    int             softirqs\_enabled;

    int             softirq\_context;

#endif

#ifdef CONFIG\_LOCKDEP

# define MAX\_LOCK\_DEPTH         48UL

    u64             curr\_chain\_key;

    int             lockdep\_depth;

    unsigned int            lockdep\_recursion;

    struct held\_lock        held\_locks[MAX\_LOCK\_DEPTH];

#endif

#ifdef CONFIG\_UBSAN

    unsigned int            in\_ubsan;

#endif

    /\* Journalling filesystem info: \*/

    void                \*journal\_info;

    /\* Stacked block device info: \*/

    struct bio\_list         \*bio\_list;

#ifdef CONFIG\_BLOCK

    /\* Stack plugging: \*/

    struct blk\_plug         \*plug;

#endif

    /\* VM state: \*/

    struct reclaim\_state        \*reclaim\_state;

    struct backing\_dev\_info     \*backing\_dev\_info;

    struct io\_context       \*io\_context;

#ifdef CONFIG\_COMPACTION

    struct capture\_control      \*capture\_control;

#endif

    /\* Ptrace state: \*/

    unsigned long           ptrace\_message;

    kernel\_siginfo\_t        \*last\_siginfo;

    struct task\_io\_accounting   ioac;

#ifdef CONFIG\_PSI

    /\* Pressure stall state \*/

    unsigned int            psi\_flags;

#endif

#ifdef CONFIG\_TASK\_XACCT

    /\* Accumulated RSS usage: \*/

    u64             acct\_rss\_mem1;

    /\* Accumulated virtual memory usage: \*/

    u64             acct\_vm\_mem1;

    /\* stime + utime since last update: \*/

    u64             acct\_timexpd;

#endif

#ifdef CONFIG\_CPUSETS

    /\* Protected by ->alloc\_lock: \*/

    nodemask\_t          mems\_allowed;

    /\* Seqence number to catch updates: \*/

    seqcount\_t          mems\_allowed\_seq;

    int             cpuset\_mem\_spread\_rotor;

    int             cpuset\_slab\_spread\_rotor;

#endif

#ifdef CONFIG\_CGROUPS

    /\* Control Group info protected by css\_set\_lock: \*/

    struct css\_set \_\_rcu        \*cgroups;

    /\* cg\_list protected by css\_set\_lock and tsk->alloc\_lock: \*/

    struct list\_head        cg\_list;

#endif

#ifdef CONFIG\_X86\_CPU\_RESCTRL

    u32             closid;

    u32             rmid;

#endif

#ifdef CONFIG\_FUTEX

    struct robust\_list\_head \_\_user  \*robust\_list;

#ifdef CONFIG\_COMPAT

    struct compat\_robust\_list\_head \_\_user \*compat\_robust\_list;

#endif

    struct list\_head        pi\_state\_list;

    struct futex\_pi\_state       \*pi\_state\_cache;

    struct mutex            futex\_exit\_mutex;

    unsigned int            futex\_state;

#endif

#ifdef CONFIG\_PERF\_EVENTS

    struct perf\_event\_context   \*perf\_event\_ctxp[perf\_nr\_task\_contexts];

    struct mutex            perf\_event\_mutex;

    struct list\_head        perf\_event\_list;

#endif

#ifdef CONFIG\_DEBUG\_PREEMPT

    unsigned long           preempt\_disable\_ip;

#endif

#ifdef CONFIG\_NUMA

    /\* Protected by alloc\_lock: \*/

    struct mempolicy        \*mempolicy;

    short               il\_prev;

    short               pref\_node\_fork;

#endif

#ifdef CONFIG\_NUMA\_BALANCING

    int             numa\_scan\_seq;

    unsigned int            numa\_scan\_period;

    unsigned int            numa\_scan\_period\_max;

    int             numa\_preferred\_nid;

    unsigned long           numa\_migrate\_retry;

    /\* Migration stamp: \*/

    u64             node\_stamp;

    u64             last\_task\_numa\_placement;

    u64             last\_sum\_exec\_runtime;

    struct callback\_head        numa\_work;

    /\*

     \* This pointer is only modified for current in syscall and

     \* pagefault context (and for tasks being destroyed), so it can be read

     \* from any of the following contexts:

     \*  - RCU read-side critical section

     \*  - current->numa\_group from everywhere

     \*  - task's runqueue locked, task not running

     \*/

    struct numa\_group \_\_rcu     \*numa\_group;

    /\*

     \* numa\_faults is an array split into four regions:

     \* faults\_memory, faults\_cpu, faults\_memory\_buffer, faults\_cpu\_buffer

     \* in this precise order.

     \*

     \* faults\_memory: Exponential decaying average of faults on a per-node

     \* basis. Scheduling placement decisions are made based on these

     \* counts. The values remain static for the duration of a PTE scan.

     \* faults\_cpu: Track the nodes the process was running on when a NUMA

     \* hinting fault was incurred.

     \* faults\_memory\_buffer and faults\_cpu\_buffer: Record faults per node

     \* during the current scan window. When the scan completes, the counts

     \* in faults\_memory and faults\_cpu decay and these values are copied.

     \*/

    unsigned long           \*numa\_faults;

    unsigned long           total\_numa\_faults;

    /\*

     \* numa\_faults\_locality tracks if faults recorded during the last

     \* scan window were remote/local or failed to migrate. The task scan

     \* period is adapted based on the locality of the faults with different

     \* weights depending on whether they were shared or private faults

     \*/

    unsigned long           numa\_faults\_locality[3];

    unsigned long           numa\_pages\_migrated;

#endif /\* CONFIG\_NUMA\_BALANCING \*/

#ifdef CONFIG\_RSEQ

    struct rseq \_\_user \*rseq;

    u32 rseq\_sig;

    /\*

     \* RmW on rseq\_event\_mask must be performed atomically

     \* with respect to preemption.

     \*/

    unsigned long rseq\_event\_mask;

#endif

    struct tlbflush\_unmap\_batch tlb\_ubc;

    union {

        refcount\_t      rcu\_users;

        struct rcu\_head     rcu;

    };

    /\* Cache last used pipe for splice(): \*/

    struct pipe\_inode\_info      \*splice\_pipe;

    struct page\_frag        task\_frag;

#ifdef CONFIG\_TASK\_DELAY\_ACCT

    struct task\_delay\_info      \*delays;

#endif

#ifdef CONFIG\_FAULT\_INJECTION

    int             make\_it\_fail;

    unsigned int            fail\_nth;

#endif

    /\*

     \* When (nr\_dirtied >= nr\_dirtied\_pause), it's time to call

     \* balance\_dirty\_pages() for a dirty throttling pause:

     \*/

    int             nr\_dirtied;

    int             nr\_dirtied\_pause;

    /\* Start of a write-and-pause period: \*/

    unsigned long           dirty\_paused\_when;

#ifdef CONFIG\_LATENCYTOP

    int             latency\_record\_count;

    struct latency\_record       latency\_record[LT\_SAVECOUNT];

#endif

    /\*

     \* Time slack values; these are used to round up poll() and

     \* select() etc timeout values. These are in nanoseconds.

     \*/

    u64             timer\_slack\_ns;

    u64             default\_timer\_slack\_ns;

#ifdef CONFIG\_KASAN

    unsigned int            kasan\_depth;

#endif

#ifdef CONFIG\_FUNCTION\_GRAPH\_TRACER

    /\* Index of current stored address in ret\_stack: \*/

    int             curr\_ret\_stack;

    int             curr\_ret\_depth;

    /\* Stack of return addresses for return function tracing: \*/

    struct ftrace\_ret\_stack     \*ret\_stack;

    /\* Timestamp for last schedule: \*/

    unsigned long long      ftrace\_timestamp;

    /\*

     \* Number of functions that haven't been traced

     \* because of depth overrun:

     \*/

    atomic\_t            trace\_overrun;

    /\* Pause tracing: \*/

    atomic\_t            tracing\_graph\_pause;

#endif

#ifdef CONFIG\_TRACING

    /\* State flags for use by tracers: \*/

    unsigned long           trace;

    /\* Bitmask and counter of trace recursion: \*/

    unsigned long           trace\_recursion;

#endif /\* CONFIG\_TRACING \*/

#ifdef CONFIG\_KCOV

    /\* See kernel/kcov.c for more details. \*/

    /\* Coverage collection mode enabled for this task (0 if disabled): \*/

    unsigned int            kcov\_mode;

    /\* Size of the kcov\_area: \*/

    unsigned int            kcov\_size;

    /\* Buffer for coverage collection: \*/

    void                \*kcov\_area;

    /\* KCOV descriptor wired with this task or NULL: \*/

    struct kcov         \*kcov;

    /\* KCOV common handle for remote coverage collection: \*/

    u64             kcov\_handle;

    /\* KCOV sequence number: \*/

    int             kcov\_sequence;

#endif

#ifdef CONFIG\_MEMCG

    struct mem\_cgroup       \*memcg\_in\_oom;

    gfp\_t               memcg\_oom\_gfp\_mask;

    int             memcg\_oom\_order;

    /\* Number of pages to reclaim on returning to userland: \*/

    unsigned int            memcg\_nr\_pages\_over\_high;

    /\* Used by memcontrol for targeted memcg charge: \*/

    struct mem\_cgroup       \*active\_memcg;

#endif

#ifdef CONFIG\_BLK\_CGROUP

    struct request\_queue        \*throttle\_queue;

#endif

#ifdef CONFIG\_UPROBES

    struct uprobe\_task      \*utask;

#endif

#if defined(CONFIG\_BCACHE) || defined(CONFIG\_BCACHE\_MODULE)

    unsigned int            sequential\_io;

    unsigned int            sequential\_io\_avg;

#endif

#ifdef CONFIG\_DEBUG\_ATOMIC\_SLEEP

    unsigned long           task\_state\_change;

#endif

    int             pagefault\_disabled;

#ifdef CONFIG\_MMU

    struct task\_struct      \*oom\_reaper\_list;

#endif

#ifdef CONFIG\_VMAP\_STACK

    struct vm\_struct        \*stack\_vm\_area;

#endif

#ifdef CONFIG\_THREAD\_INFO\_IN\_TASK

    /\* A live task holds one reference: \*/

    refcount\_t          stack\_refcount;

#endif

#ifdef CONFIG\_LIVEPATCH

    int patch\_state;

#endif

#ifdef CONFIG\_SECURITY

    /\* Used by LSM modules for access restriction: \*/

    void                \*security;

#endif

#ifdef CONFIG\_GCC\_PLUGIN\_STACKLEAK

    unsigned long           lowest\_stack;

    unsigned long           prev\_lowest\_stack;

#endif

    /\*

     \* New fields for task\_struct should be added above here, so that

     \* they are included in the randomized portion of task\_struct.

     \*/

    randomized\_struct\_fields\_end

    /\* CPU-specific state of this task: \*/

    struct thread\_struct        thread;

    /\*

     \* WARNING: on x86, 'thread\_struct' contains a variable-sized

     \* structure.  It \*MUST\* be at the end of 'task\_struct'.

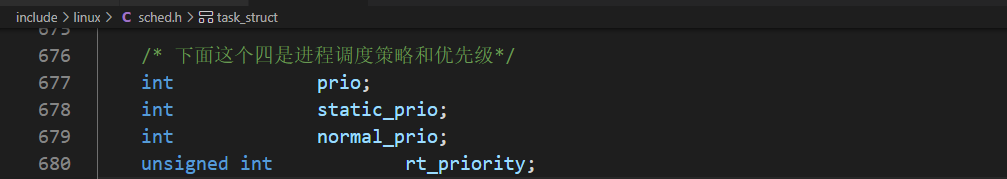
     \*

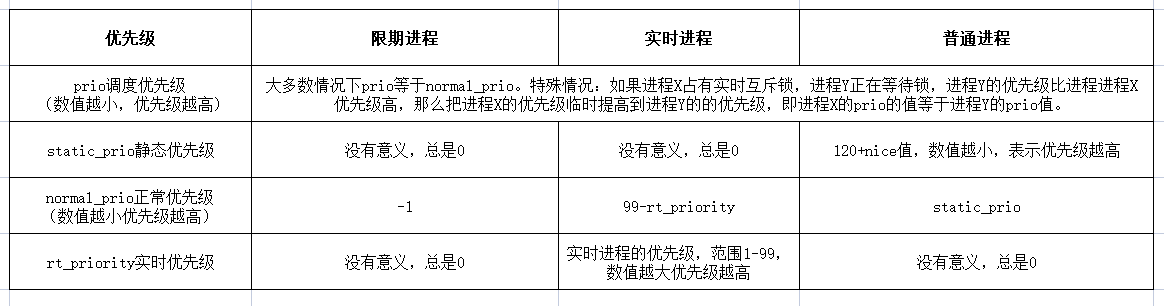
     \* Do not put anything below here!

     \*/

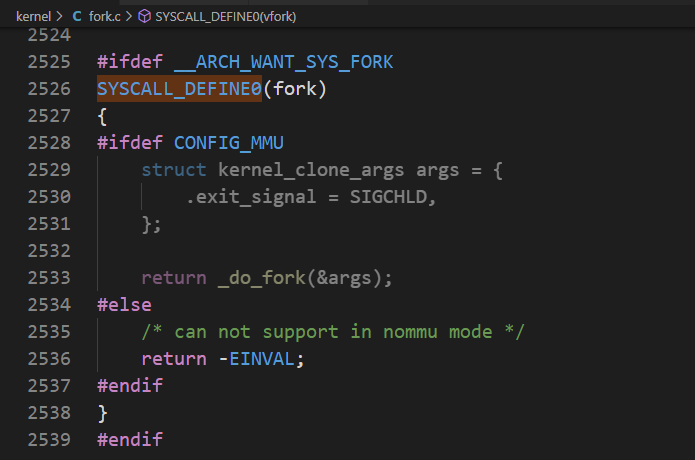
};

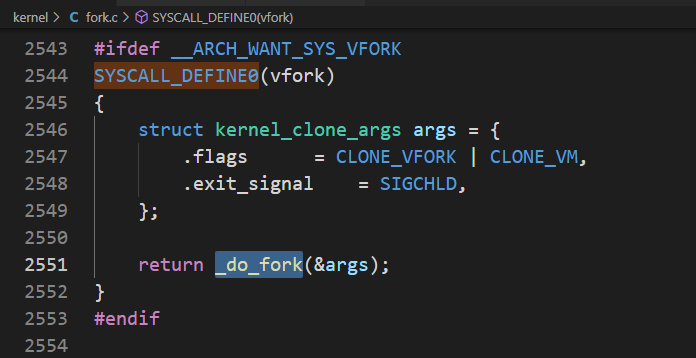
4、进程优先级如下

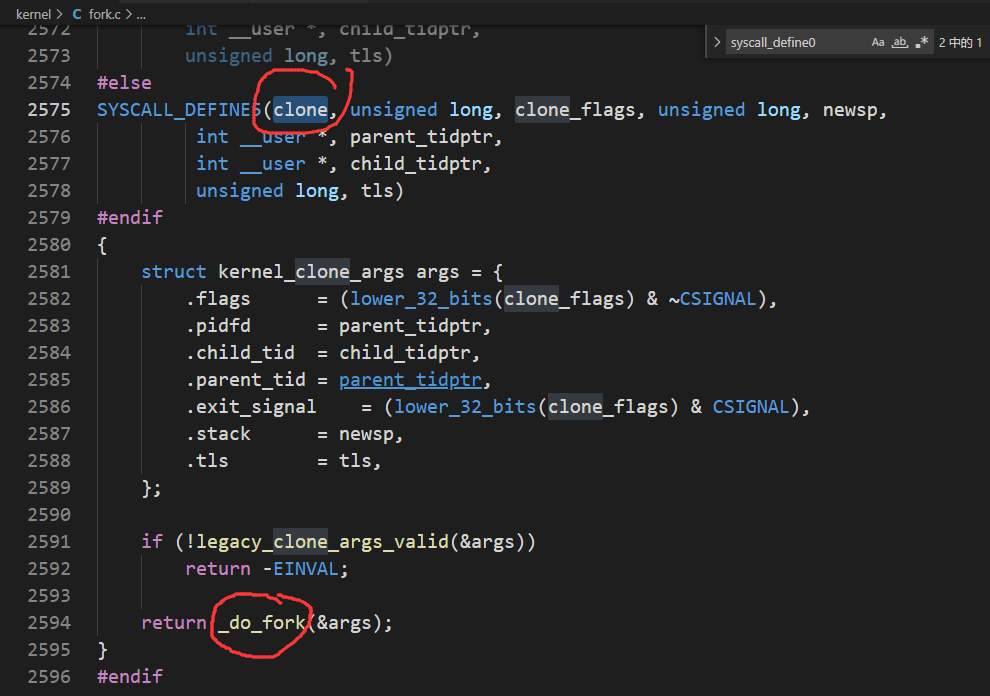


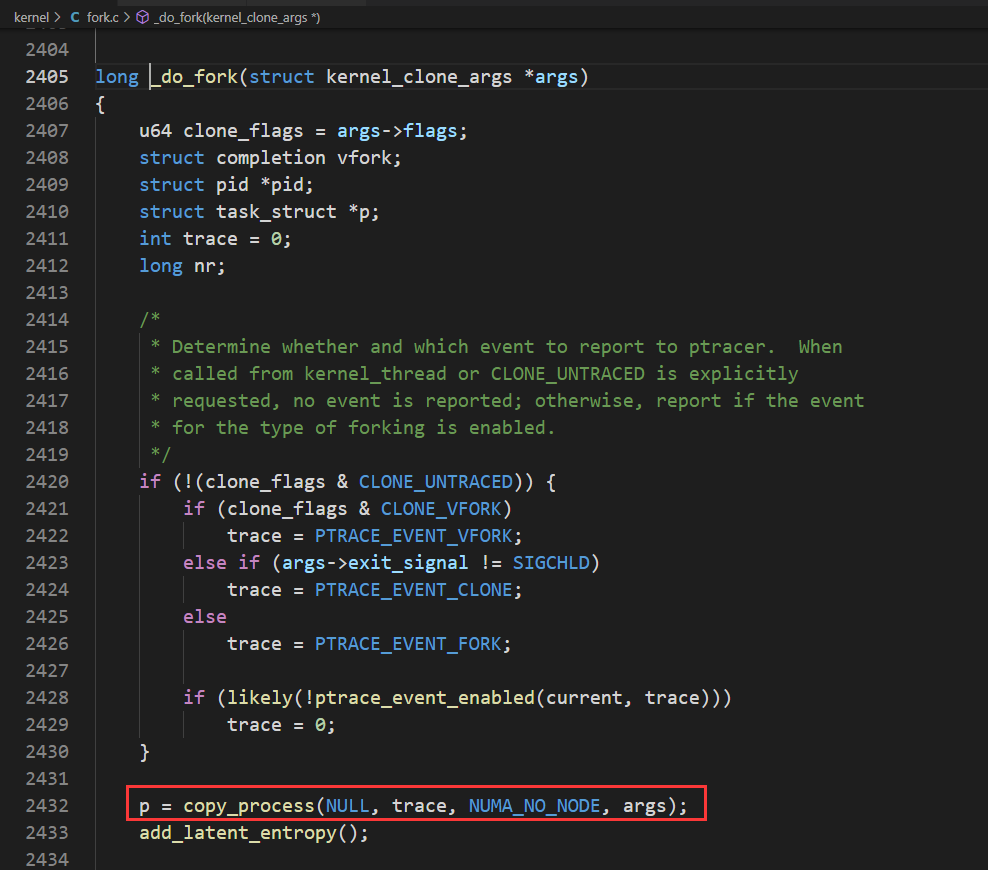


5、系统调用





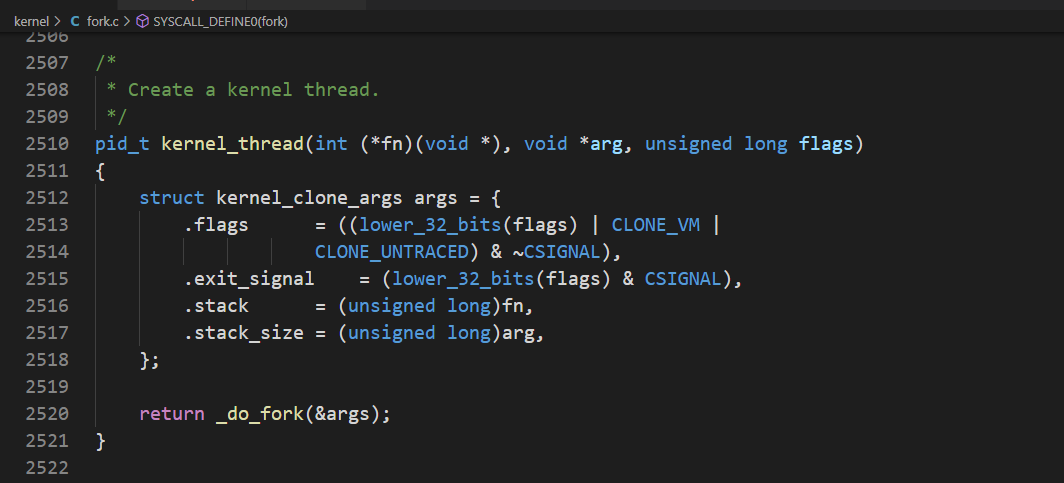






6、内核线程

它是独立运行在内核空间的进程，与普通用户进程区别在于内核线程没有独立的进程地址空间。task\_struct数据结构里面有一个成员指针mm设置为NULL，它只能运行在内核空间。



7、退出进程

进程主动终止：从main()函数返回，链接程序会自动添加到exit()系统调用；主动调用exit()系统函数。

进程被动终止：进程收到一个自己不能处理的信号；进程收到 SIGKILL等终止信息。

