Lab1 线程机制实习报告

姓名 姜慧强 学号 1801210840
日期 2019.03.01

目录

内容一：总体概述

本次实验室操作系统高级课程的第一次实验。在 Lab0 调试 Env 的基础上，主要完成了 Thread 部分代码阅读及逻辑梳理，修改线程数据成员，增加对线程数量机制，增加打印 ThreadInfo 的功能。基本完成了实验说要求的任务，加强了对线程状态切换等机制的认识，熟悉了 Nachos 开发环境。

内容二：任务完成情况

任务完成列表（Y/N）

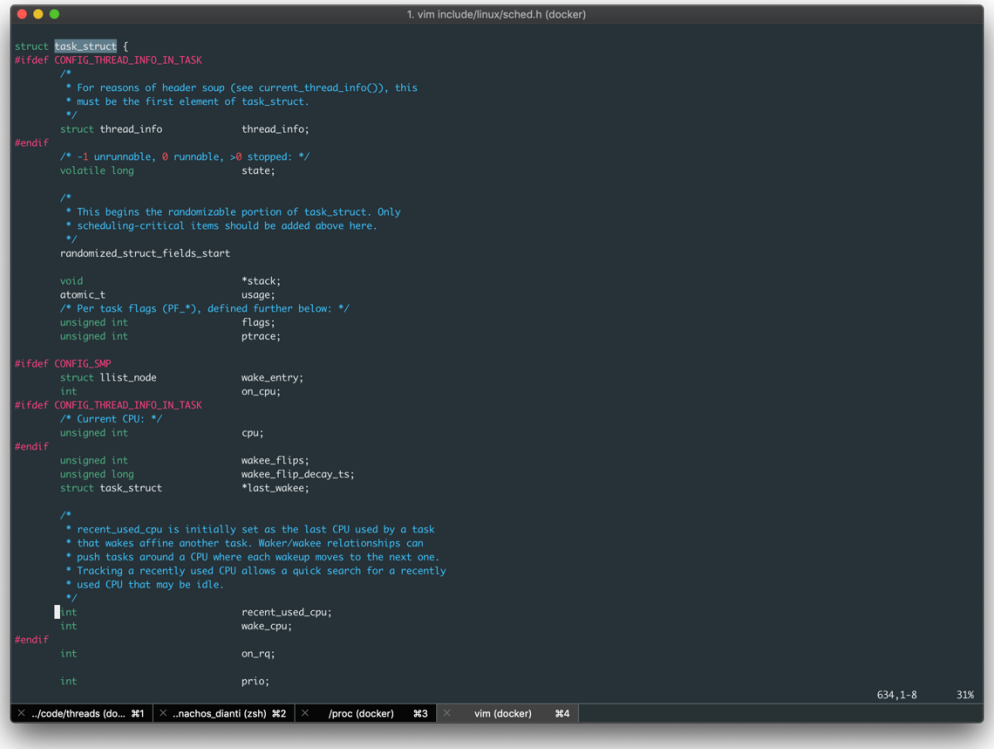
EXERCISE1	EXERCISE2	EXERCISE3	EXERCISE4
Y	Y	Y	Y

具体 Exercise 的完成情况

Exercise1

通过对 Linux kernal 4.19.25 下 include/linux/sched.h 文件中对 task_struct 的源码分析，可知 Linux 下的 PCB 结构复杂。删去所有注释及空行，得到 Linux 的 PCB 属性项多达 248 项，其中不乏 struct 结构，所以其真正对应的参数是一个比较大的数字。这也侧面的反映了 Linux 系统的复杂性。下面列举 Task_Struct 部分参数，其余部分请参见附录部分。

1. 进程状态 区分是否可以运行，是否正在运行。volatile long state;
2. 进程栈信息 void *stack;
3. 进程 Trace 用于记录进程间调用关系及其他信息 unsigned int ptrace;
4. 当前 CPU 状态 unsigned int cpu;
5. 优先级 int prio;
6. 进程标识号 pid_t pid;
7. 父进程号 unsigned int father;
8. 信号 struct signal_struct *signal;
9. 进程信号屏蔽码 sigset_t blocked;
10. 会话号 unsigned int sessionId;
11. Tty 子设备号 unsigned int tty;
12. 用户态运行时间 u64 utime;
13. 内核态运行时间 u64 stime;
14. 子进程用户态运行时间 u64 utimescaled;
15. 子进程内核态运行时间 u64 stimescaled;
16. 进程开始运行时刻 u64 start_time;
17. 任务局部表描述符 struct desc_struct Idt[3];



而 Nachos 的进程信息位于 code/threads/thread.h 文件中。通过阅读源码，可以得到Nachos 的 PCB 主要包含 Thread 信息（保存不在运行态的进程的信息-状态，uid，tid 等等），当前进程信息（状态，uid，tid 等等），当前进程 Stack 信息.

Exercise2

threads/main.cc

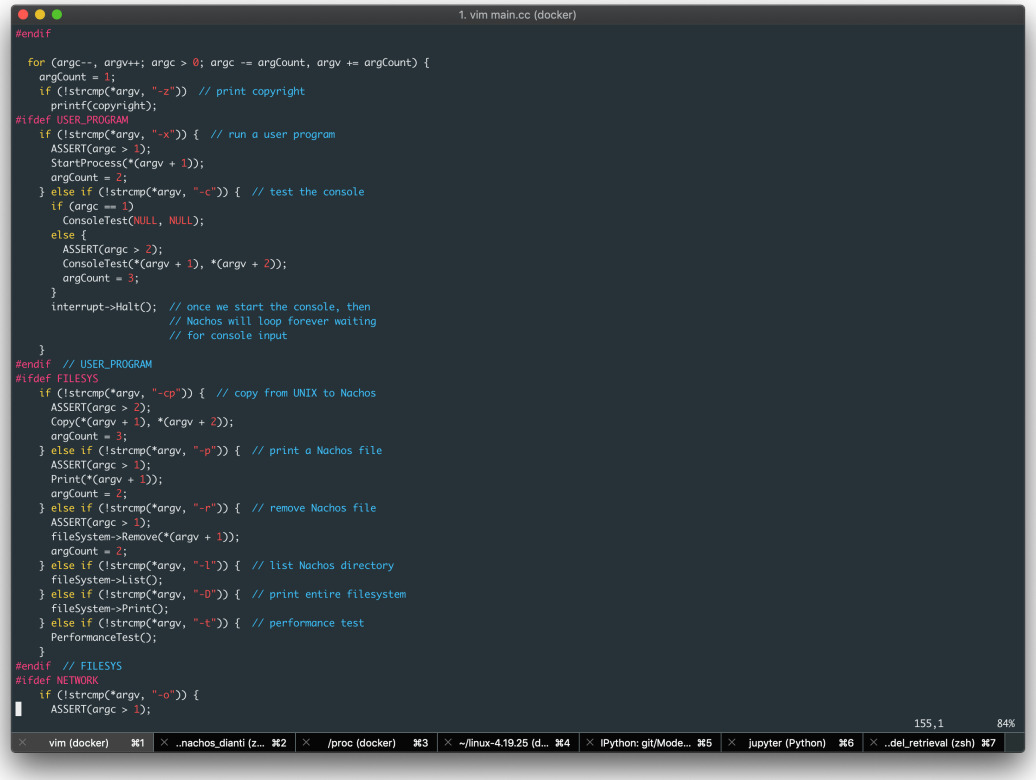
主要做的工作是进程初始化，函数声明，shell 命令解析分发。

进程初始化，主要是调用 system.h 中实现好的 Initialize 函数。

可以看到在 main.cc 中声明了如下的函数，这些函数在具体文件中完成实现。

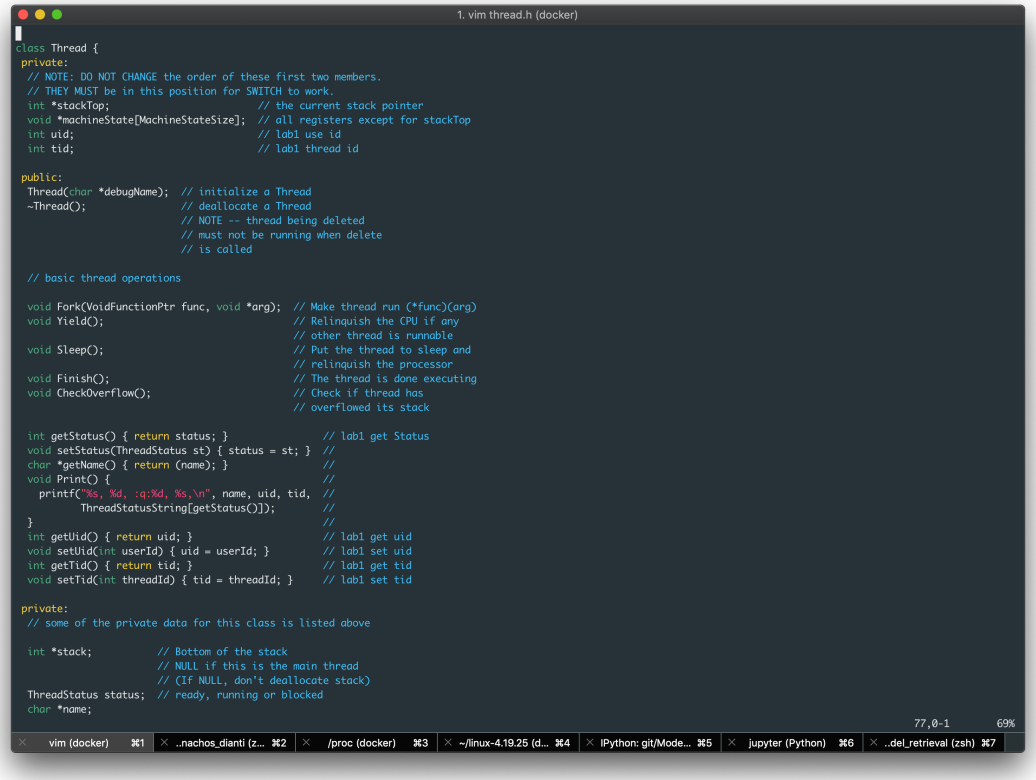
```
extern void ThreadTest(void), Copy(char *unixFile, char *nachosFile);
extern void Print(char *file), PerformanceTest(void);
extern void StartProcess(char *file), ConsoleTest(char *in, char *out);
extern void MailTest(int networkID);
```

此外通过控制 int argc 和 char **argv 两个参数，实现对 shell 命令的解析和具体函数调用的分发。



threads/thread.h

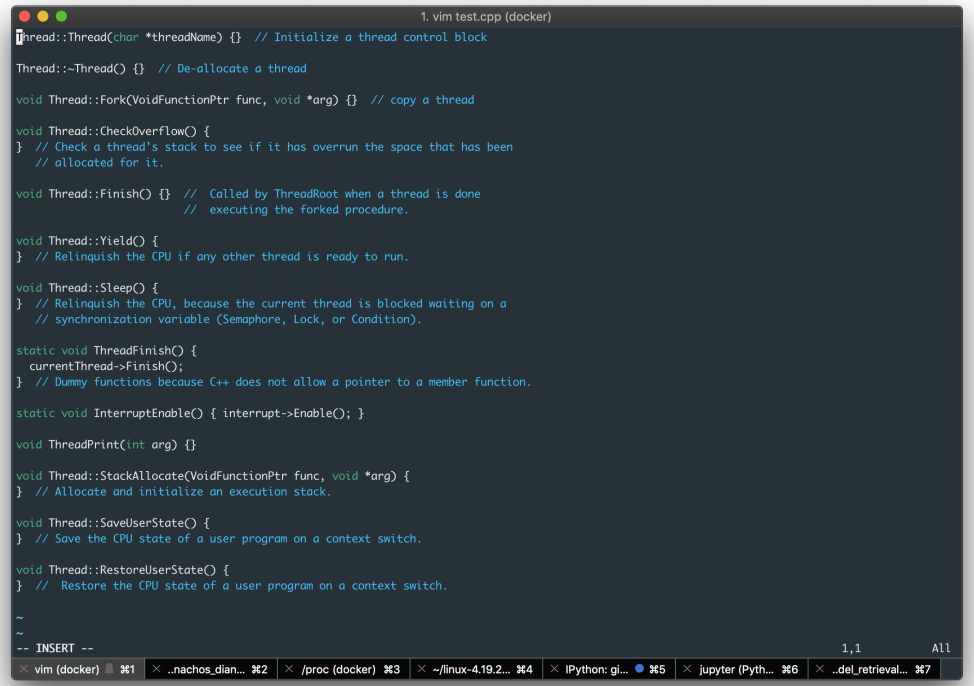
主要实现了一些进程相关的枚举类、常量的定义，定义并声明了 Thread 类(部分实现)。



可以看到 `class Thread` 描述的是 Nachos 中的 PCB 信息,包括栈信息，进程状态等等。

threads/thread.cc

主要完成 `thread.h` 中声明的函数的实现。



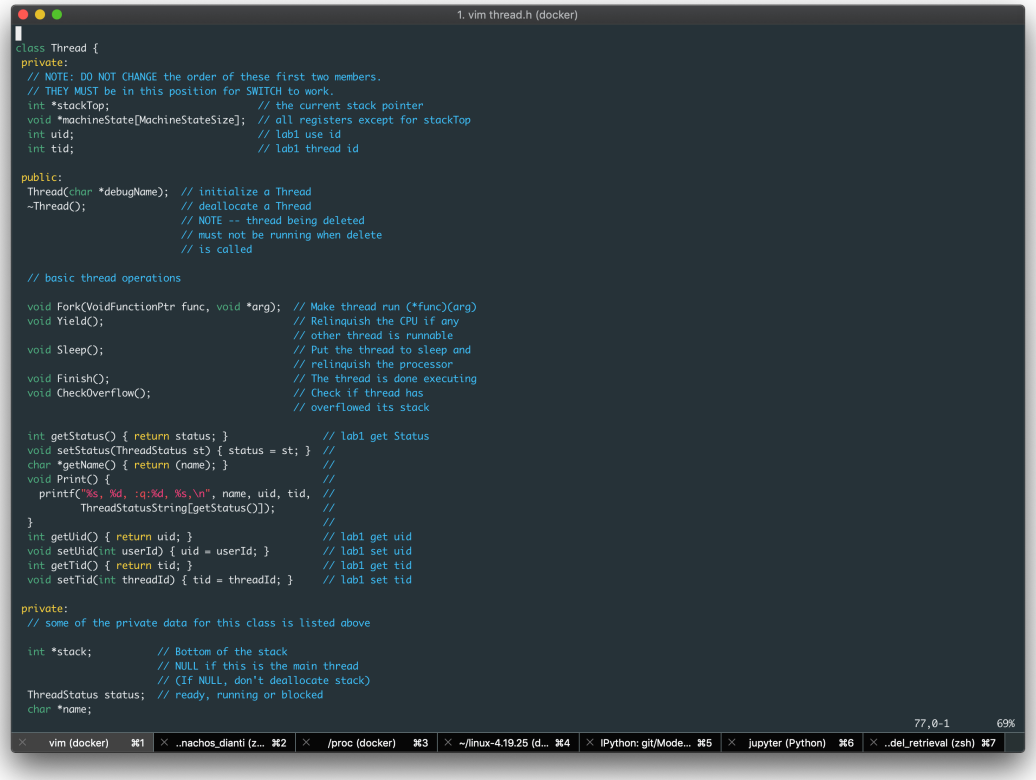
threads/threadtest.cc

threadtest.cc 主要是实现一些单测函数，通过 Main.cc 中的 ThreadTest()实现调用。

Exercise3

经过前面的源码阅读，可以发现如果要在进程的数据结构中增加用户 Id，线程 Id 两个属性，则需要修改 thread.h 中 class Thread 类中参数即可。

再添加完参数之后，还需要添加相应的 get，set 函数。



实现完数据结构的修改，还需要进行对添加的数据进行初始化和回收的工作。

在 system.cc 中， 增加对变量的声明，对声明的变量进行初始化。

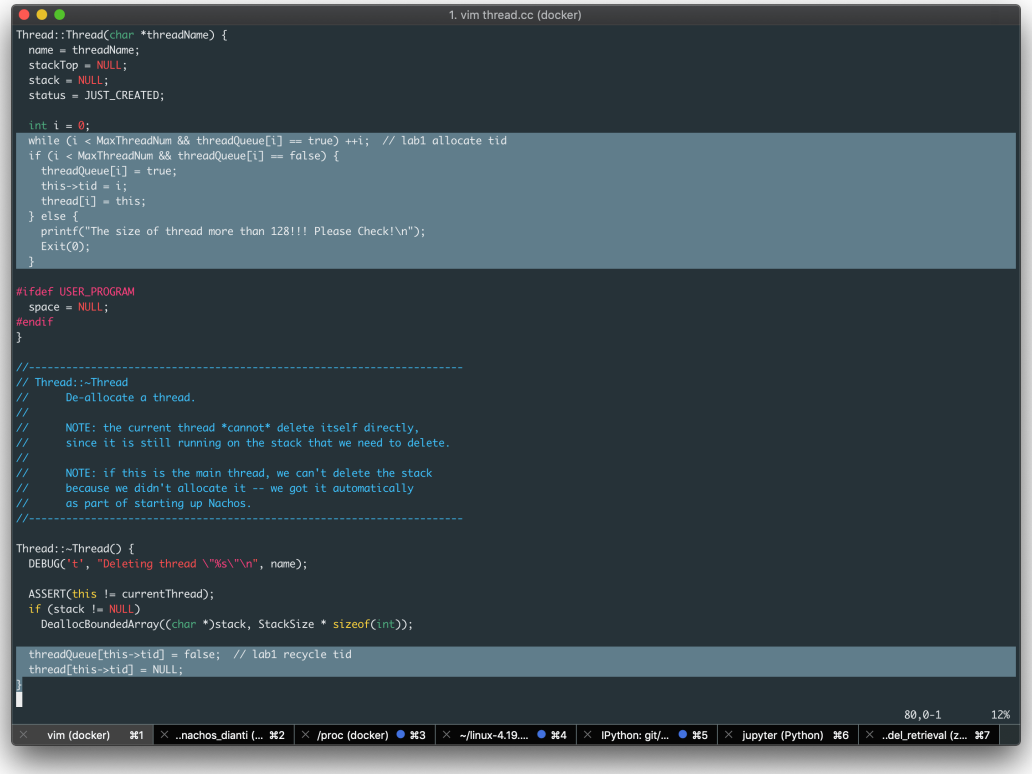
```
#define MaxThreadNum 128
bool threadQueue[MaxThreadNum]; // lab1 thread queue
Thread *thread[MaxThreadNum];
void Initialize(int argc, char **argv) {
…
  memset(threadQueue, 0, sizeof(threadQueue)); // lab1 init thread queue
…
}
```

在 system.h 中对变量进行相应的声明

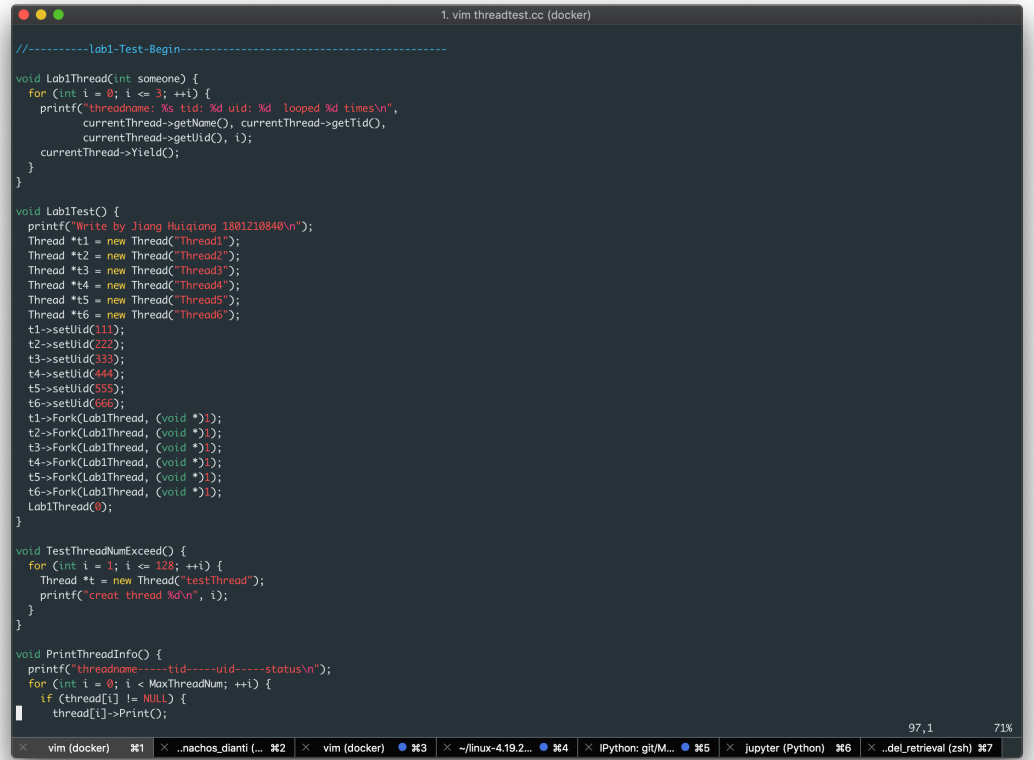
```
#define MaxThreadNum 128
bool threadQueue[MaxThreadNum]; // lab1 thread queue
Thread *thread[MaxThreadNum];
```

除了初始化之外，我们在创建 Thread 的时候需要分配给 Thread 一个 uid，tid。在 thread.cc 中对分配 uid 的逻辑进行设计。

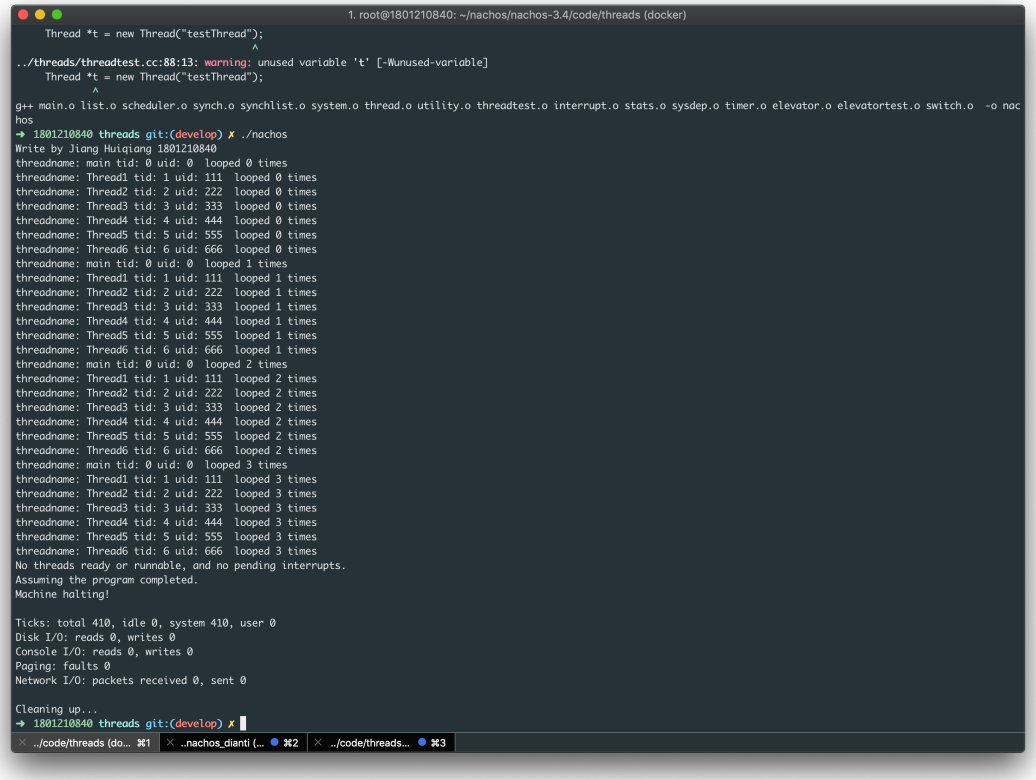
从 threadQueue 中寻找未被分配的 uid，分配其 uid&tid。待进程变成终止态的时候，对 uid 和 tid 进行回收。



为了测试 uid,tid 分配情况，改写了测试样例，如下图所示，分配了 6 个进程，每个进程循环三次，分别打印进程的 name，uid，tid 等信息。



输出结果如下图所示：



Exercise4

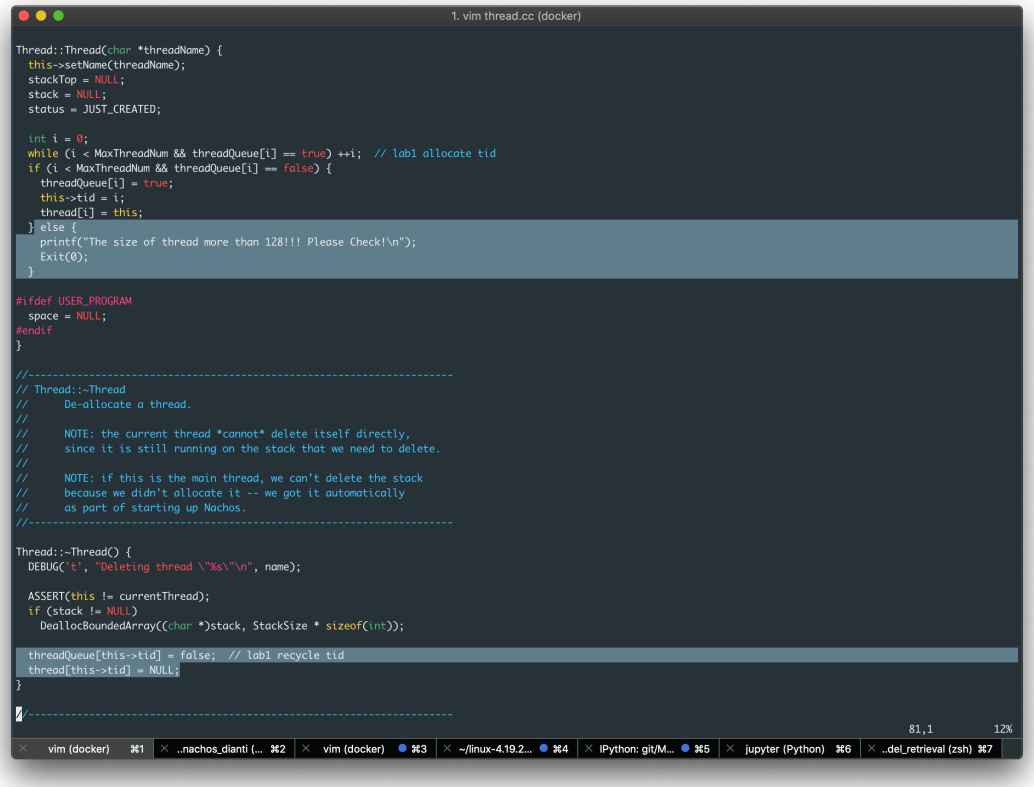
需要增加对进程总数量的限制，则利用开的 ThreadQueue 数组大小的限制完成这个功能。

每次 new Thread()的时候先去检查 ThreadQueue 有没有空位，若有空位则分配进程，若无则，打印 log 日志。

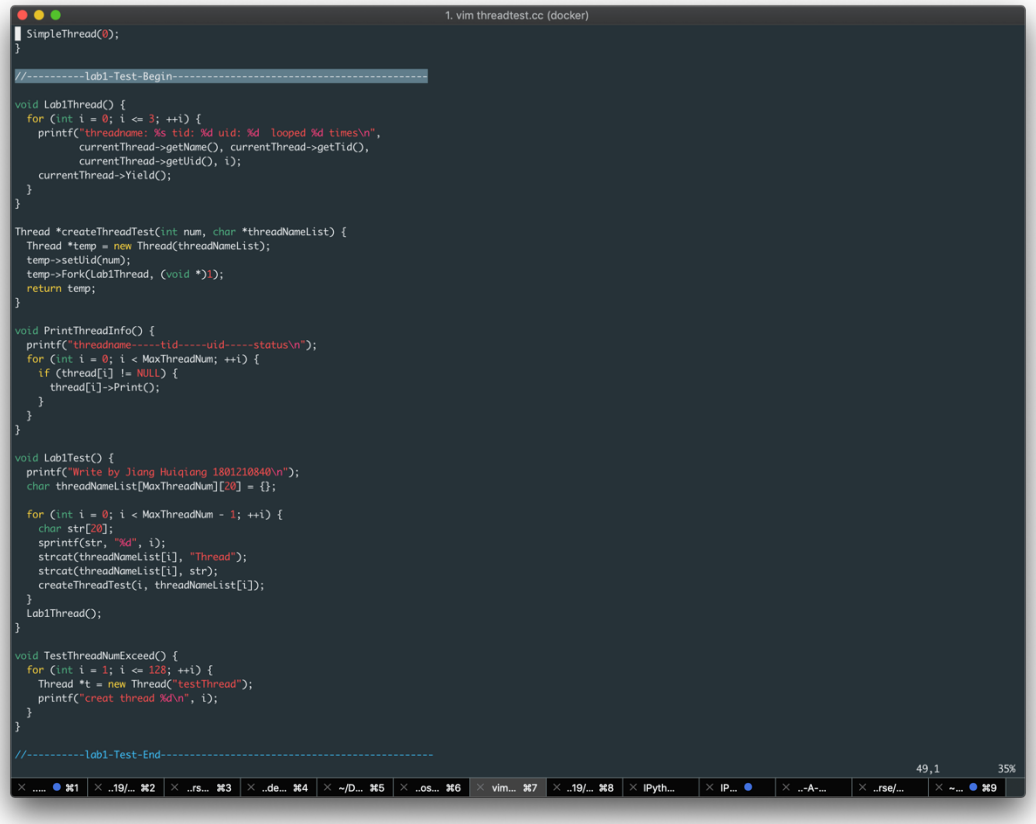
具体来说，修改 thread.cc 中 Thread::Thread()方法

PS 的功能是打印出进程中的 Info，在这里 TS 是想打印出进程（线程 Nachos 中为同一概念）的 info。在 main.cc 文件中声明 PrintThreadInfo()函数，然后在 threadtest.cc 中实现。

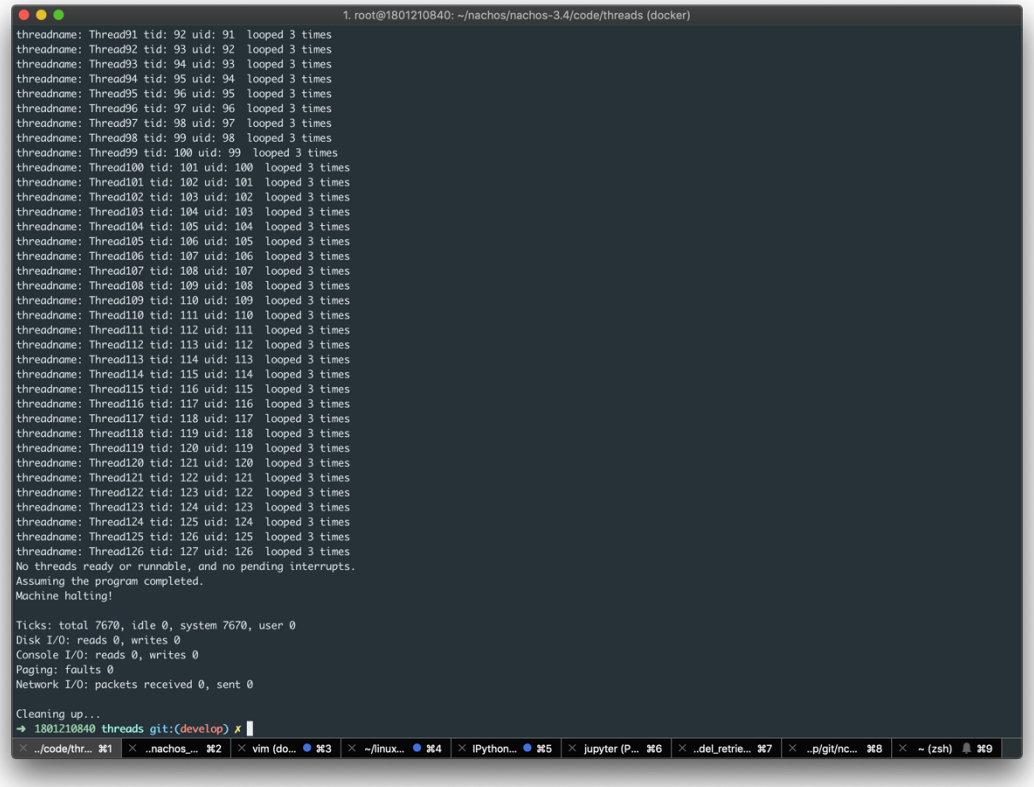
```
extern void PrintThreadInfo();  // lab1 TS
```



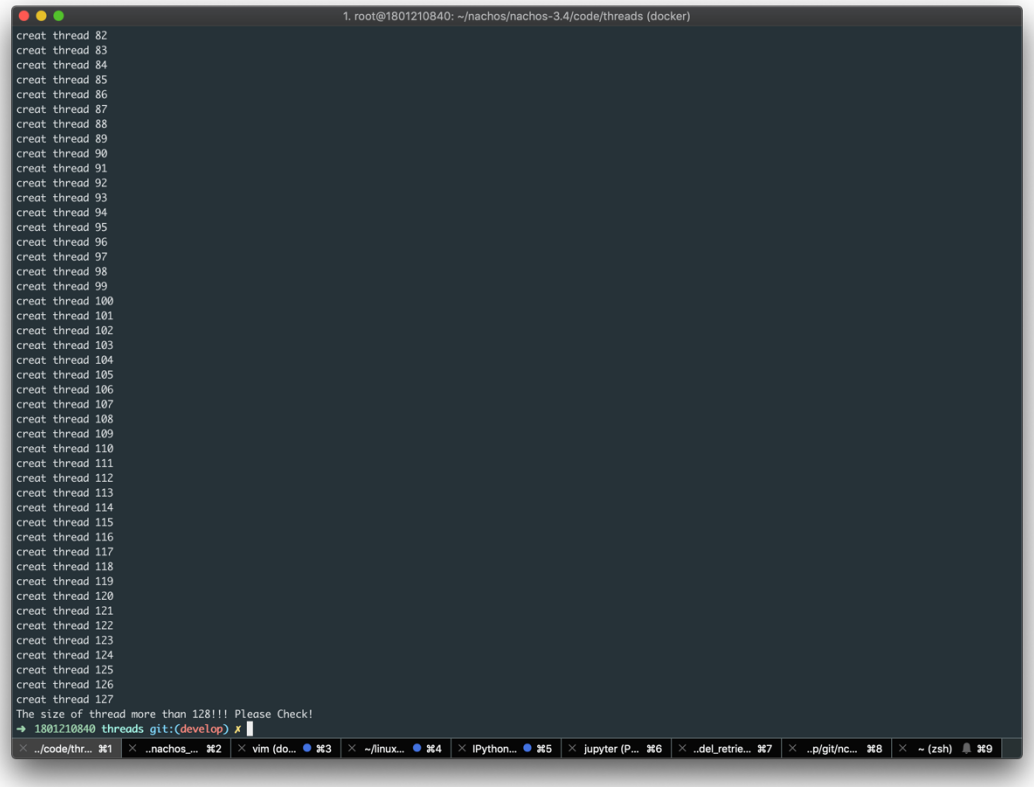
为了测试 128 个进程这个限制，测试单次开进程数量超过 128，和不超过 128 两种情况。



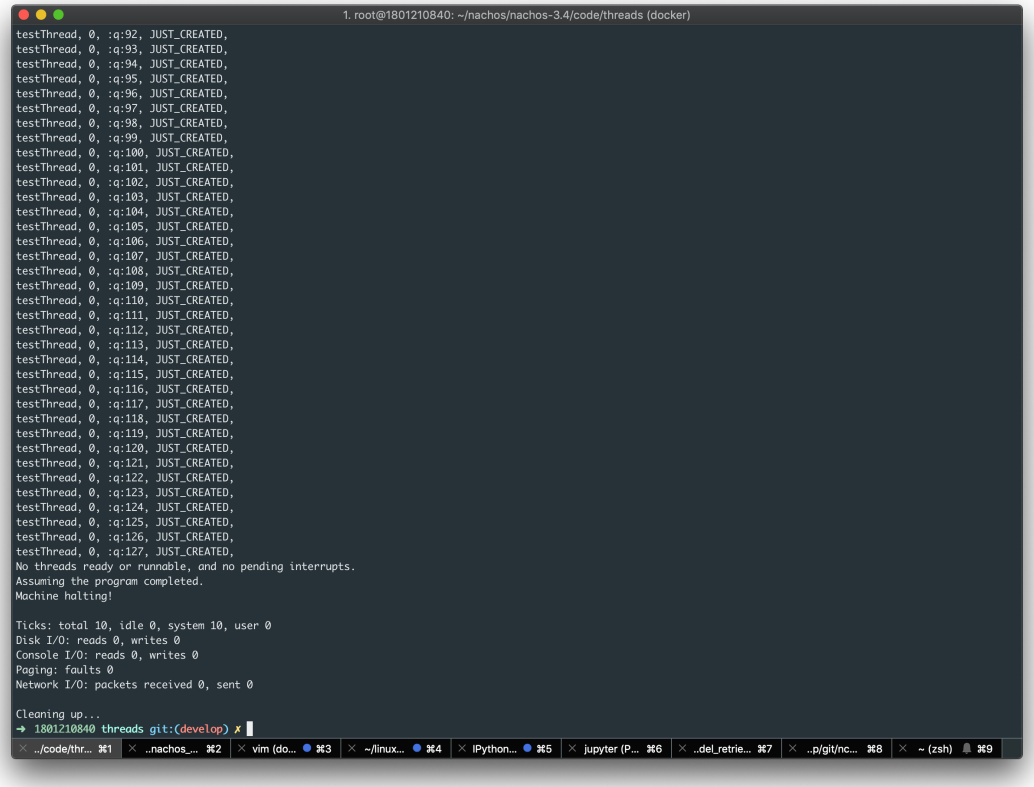
Thread 创建 127 个进程



Thread 创建 128 个进程



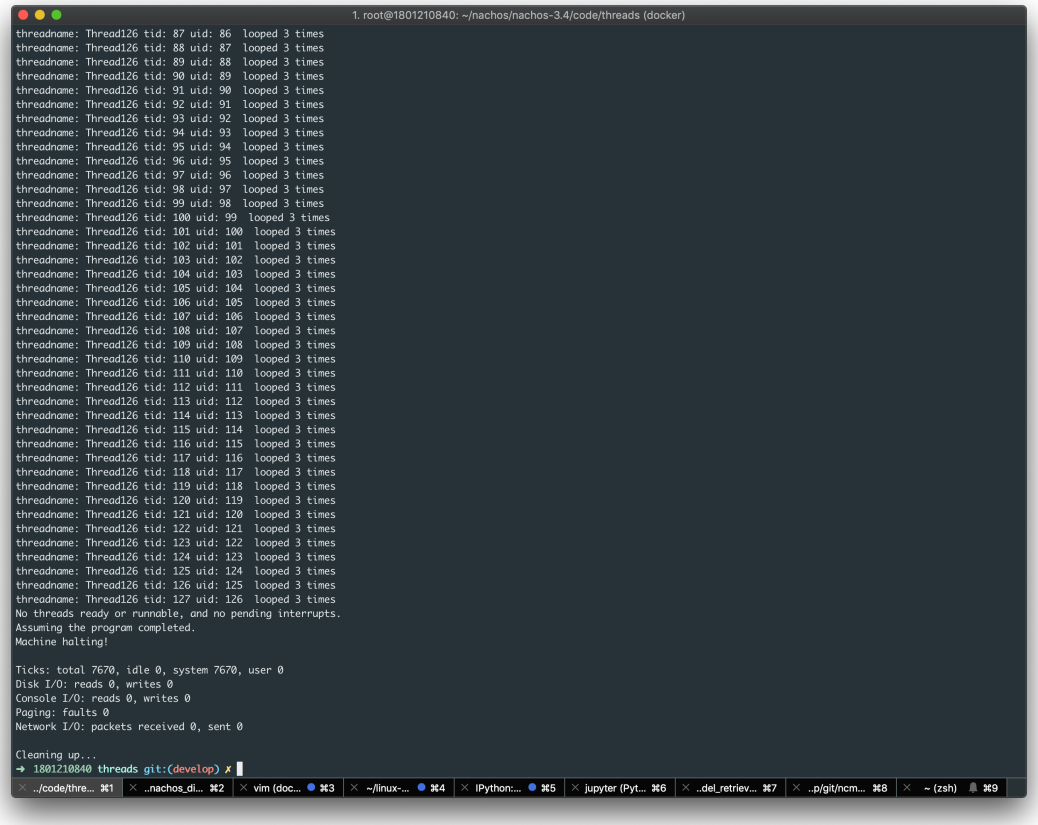
打印 Thread Status



内容三：遇到的困难以及解决方法

在测试超过 128 个进程限制的时候，为了方便测试，把进程新建，分配 uid，Fork 等写到一个函数里批量操作。

但在实际测试过程中发现 ThreadName 全部输出最后一个 Thread 的 name。



分析代码，一开始怀疑是 Thread Init 的时候 setName()函数出了问题，可能是一个公用变量，但其他 uid，tid 等等没有问题。修改 Thread Init()之后也不能解决问题。

之后怀疑是不是因为 Thread 执行顺序的问题，把原来的函数拆成几个 for 循环。多次尝试之后，发现实际上是因为 setName 的时候传进去的是一个 char 指针，在我之前的方案中，这个 char 指针在 for 循环的每一轮中都重新赋值，重新计算，导致了写到 Thread 中的 name 实际上是这个字符串的指针。而到第 127 轮，内存里这个指针指向的就是”Thread 126”。每轮循环，这个地址都被分配给 name 变量，当一轮结束之后这个变量就被回收。解决方法就是维护一个对循环全局的 char list，保证存放 name 的指针不会被复用。

内容四：收获及感想

首先，自己很久没写 c 了，这次实验是一次很好的提升自己 c 编程能力的机会。其次通过对问题的分析结果，增加了对指正，进程状态间转换等机制的认识。提到了对操作系统整体的理解。另外加强了自己在 linux 下开发能力。

内容五：对课程的意见和建议

第一次作业对于新手来说任务量其实还是比较大的，如果老师能说明下大概任务量，会对学生预留安排作业时间有所帮助。

内容六：参考文献

[1] Stevens, W. R. (2002). UNIX 环境高级编程：英文版. 机械工业出版社.

附录：

```
struct task_struct {
	/* -1 unrunnable, 0 runnable, >0 stopped: */
	volatile long			state;
	void				*stack;

	unsigned int			flags;
	unsigned int			ptrace;
	struct llist_node		wake_entry;
	int				on_cpu;
	/* Current CPU: */
	unsigned int			cpu;
	unsigned int			wakee_flips;
	unsigned long			wakee_flip_decay_ts;
	struct task_struct		*last_wakee;
	int				recent_used_cpu;
	int				wake_cpu;
	int				on_rq;
	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;
	struct task_group		*sched_task_group;
	struct sched_dl_entity		dl;
	/* List of struct preempt_notifier: */
	struct hlist_head		preempt_notifiers;
	unsigned int			btrace_seq;
	unsigned int			policy;
	int				nr_cpus_allowed;
	cpumask_t			cpus_allowed;
	int				rcu_read_lock_nesting;
	union rcu_special		rcu_read_unlock_special;
	struct list_head		rcu_node_entry;
	struct rcu_node			*rcu_blocked_node;
	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;
	struct sched_info		sched_info;
	struct list_head		tasks;
```

```
        struct plist_node               pushable_tasks;
        struct rb_node                  pushable_dl_tasks;
        struct mm_struct                *mm;
        struct mm_struct                *active_mm;
        /* Per-thread vma caching: */
        struct vmacache                 vmacache;
        struct task_rss_stat            rss_stat;
        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;
        /* Force alignment to the next boundary: */
        unsigned                       :0;
        /* Bit to tell LSMs we're in execve(): */
        unsigned                        in_execve:1;
        unsigned                        in_iowait:1;
        unsigned                        restore_sigmask:1;
        unsigned                        in_user_fault:1;
        unsigned                        memcg_kmem_skip_account:1;
        unsigned                        brk_randomized:1;
        /* disallow userland-initiated cgroup migration */
        unsigned                        no_cgroup_migration:1;
        /* to be used once the psi infrastructure lands upstream. */
        unsigned                        use_memdelay:1;
        unsigned long                   atomic_flags; /* Flags requiring
atomic access. */
        struct restart_block            restart_block;

        pid_t                           pid;
        pid_t                           tgid;

        /* Canary value for the -fstack-protector GCC feature: */
        unsigned long                   stack_canary;
```

```
        /* Real parent process: */
        struct task_struct __rcu        *real_parent;
        /* Recipient of SIGCHLD, wait4() reports: */
        struct task_struct __rcu        *parent;

        struct list_head                children;
        struct list_head                sibling;
        struct task_struct              *group_leader;
        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];
        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;
        u64                             utimescaled;
        u64                             stimescaled;
        u64                             gtime;
        struct prev_cputime             prev_cputime;
        struct vtime                    vtime;
        atomic_t                        tick_dep_mask;
        /* Context switch counts: */
        unsigned long                   nvcsw;
        unsigned long                   nivcsw;
        /* Monotonic time in nsecs: */
        u64                             start_time;
        /* Boot based time in nsecs: */
        u64                             real_start_time;
        unsigned long                   min_flt;
        unsigned long                   maj_flt;
        struct task_cputime             cputime_expires;
        struct list_head                cpu_timers[3];
        /* Tracer's credentials at attach: */
        const struct cred __rcu         *ptracer_cred;
        /* Objective and real subjective task credentials (COW): */
        const struct cred __rcu         *real_cred;
        /* Effective (overridable) subjective task credentials (COW): */
```

```
        const struct cred __rcu         *cred;
        char                            comm[TASK_COMM_LEN];
        struct nameidata                *nameidata;
        struct sysv_sem                 sysvsem;
        struct sysv_shm                 sysvshm;
        unsigned long                   last_switch_count;
        unsigned long                   last_switch_time;
        /* Filesystem information: */
        struct fs_struct                *fs;
        /* Open file information: */
        struct files_struct             *files;
        /* Namespaces: */
        struct nsproxy                  *nsproxy;
        /* Signal handlers: */
        struct signal_struct            *signal;
        struct sighand_struct           *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;
        struct audit_context            *audit_context;
        kuid_t                          loginuid;
        unsigned int                    sessionid;
        struct seccomp                  seccomp;
        /* Thread group tracking: */
        u32                             parent_exec_id;
        u32                             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;
        /* 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;
```

```
	/* Mutex deadlock detection: */
	struct mutex_waiter		*blocked_on;
	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;
	u64				curr_chain_key;
	int				lockdep_depth;
	unsigned int			lockdep_recursion;
	struct held_lock		held_locks[MAX_LOCK_DEPTH];
	unsigned int			in_ubsan;
	/* Journalling filesystem info: */
	void				*journal_info;
	/* Stacked block device info: */
	struct bio_list			*bio_list;
	/* Stack plugging: */
	struct blk_plug			*plug;
	/* VM state: */
	struct reclaim_state		*reclaim_state;
	struct backing_dev_info		*backing_dev_info;
	struct io_context		*io_context;
	/* Ptrace state: */
	unsigned long			ptrace_message;
	siginfo_t			*last_siginfo;
	struct task_io_accounting	ioac;
	/* Accumulated RSS usage: */
	u64				acct_rss_mem1;
	/* Accumulated virtual memory usage: */
	u64				acct_vm_mem1;
	/* stime + utime since last update: */
	u64				acct_timexpd;
	/* 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;
        /* 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;
        /* 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;
        u32                             closid;
        u32                             rmid;
        struct robust_list_head __user  *robust_list;
        struct compat_robust_list_head __user *compat_robust_list;
        struct list_head                pi_state_list;
        struct futex_pi_state           *pi_state_cache;
        struct
perf_event_context      *perf_event_ctxp[perf_nr_task_contexts];
        struct mutex                    perf_event_mutex;
        struct list_head                perf_event_list;
        unsigned long                   preempt_disable_ip;
        /* Protected by alloc_lock: */
        struct mempolicy                *mempolicy;
        short                           il_prev;
        short                           pref_node_fork;
        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;
        struct numa_group               *numa_group;
        unsigned long                   *numa_faults;
        unsigned long                   total_numa_faults;
        unsigned long                   numa_faults_locality[3];
        unsigned long                   numa_pages_migrated;
        struct rseq __user *rseq;
        u32 rseq_len;
        u32 rseq_sig;
        unsigned long rseq_event_mask;
```

```
        struct tlbflush_unmap_batch     tlb_ubc;
        struct rcu_head                 rcu;
        /* Cache last used pipe for splice(): */
        struct pipe_inode_info          *splice_pipe;
        struct page_frag                task_frag;
        struct task_delay_info          *delays;
        int                             make_it_fail;
        unsigned int                    fail_nth;
        int                             nr_dirtied;
        int                             nr_dirtied_pause;
        /* Start of a write-and-pause period: */
        unsigned long                   dirty_paused_when;
        int                             latency_record_count;
        struct latency_record           latency_record[LT_SAVECOUNT];
        u64                             timer_slack_ns;
        u64                             default_timer_slack_ns;
        unsigned int                    kasan_depth;
        /* 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;
        atomic_t                        trace_overrun;
        /* Pause tracing: */
        atomic_t                        tracing_graph_pause;
        /* State flags for use by tracers: */
        unsigned long                   trace;
        /* Bitmask and counter of trace recursion: */
        unsigned long                   trace_recursion;
        /* 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;
        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;
        struct request_queue            *throttle_queue;
        struct uprobe_task              *utask;
        unsigned int                    sequential_io;
        unsigned int                    sequential_io_avg;
        unsigned long                   task_state_change;
        int                             pagefault_disabled;
        struct task_struct              *oom_reaper_list;
        struct vm_struct                *stack_vm_area;
        /* A live task holds one reference: */
        atomic_t                        stack_refcount;
        int patch_state;
        /* Used by LSM modules for access restriction: */
        void                            *security;
}
```