|  |
| --- |
| 北 京 邮 电 大 学  实 验 报 告  课程名称： 操作系统原理  院系：计算机学院（国家示范性软件工程学院）  班级：2021211318  姓名：唐子潇  学号：2021211460  教师： 赵方 金昕  成绩：  2023年春季学期 |
| 一、实验目的 1. 理解 Linux 管理进程所用到的数据结构。  2. 理解 Linux 的进程调度算法的处理逻辑及其实现所用到的数据结构。 二、实验环境 VirtualBox虚拟机下安装的Ubuntu-15.04.6系统。  **三、实验任务及内容**  1. 通过查阅参考书或者上网找资料，熟悉/usr/src/linux（注意：这里最后一级目录名可能是个含有具体内核版本号和“linux”字符串的名字）下各子目录的内容，即所含 Linux源代码的情况。  2. 分析 Linux 进程调度有关函数的源代码，主要是 schedule()函数，并且要对它们引用的头文件等一并分析。  3. 实现 Linux 的进程调度算法及理解其实现所用的主要数据结构。  其中，我主要负责任务三的CFS公平调度算法：  include/linux/sched.h  struct task\_struct {  volatile long state;  void \*stack;  atomic\_t usage;  unsigned int flags;  unsigned int ptrace;  #ifdef CONFIG\_SMP  struct llist\_node wake\_entry;  int on\_cpu;  struct task\_struct \*last\_wakee;  unsigned long wakee\_flips;  unsigned long wakee\_flip\_decay\_ts;  int wake\_cpu;  #endif    int on\_rq;  int prio, static\_prio, normal\_prio;  unsigned int rt\_priority;  const struct sched\_class \*sched\_class;  struct sched\_entity se;  struct sched\_rt\_entity rt; } struct sched\_entity {  struct load\_weight load;  struct rb\_node run\_node;  struct list\_head group\_node;  unsigned int on\_rq;  u64 exec\_start;  u64 sum\_exec\_runtime;  u64 vruntime;  u64 prev\_sum\_exec\_runtime;  u64 nr\_migrations;  #ifdef CONFIG\_SCHEDSTATS  struct sched\_statistics statistics;  #endif  #ifdef CONFIG\_FAIR\_GROUP\_SCHED  int depth;  struct sched\_entity \*parent;  /\* rq on which this entity is (to be) queued: \*/  struct cfs\_rq \*cfs\_rq;  /\* rq "owned" by this entity/group: \*/  struct cfs\_rq \*my\_q;  #endif  #ifdef CONFIG\_SMP  /\* Per-entity load-tracking \*/  struct sched\_avg avg;  #endif  };  static void tick\_periodic(int cpu){ if (tick\_do\_timer\_cpu == cpu) {write\_seqlock(&jiffies\_lock);tick\_next\_period = ktime\_add(tick\_next\_period, tick\_period);do\_timer(1);write\_sequnlock(&jiffies\_lock);update\_wall\_time();}update\_process\_times(user\_mode(get\_irq\_regs()));profile\_tick(CPU\_PROFILING);}void update\_process\_times(int user\_tick){struct task\_struct \*p = current;account\_process\_tick(p, user\_tick);run\_local\_timers();rcu\_check\_callbacks(user\_tick);#ifdef CONFIG\_IRQ\_WORKif (in\_irq())irq\_work\_tick();#endifscheduler\_tick();run\_posix\_cpu\_timers(p);}void scheduler\_tick(void){int cpu = smp\_processor\_id();struct rq \*rq = cpu\_rq(cpu);struct task\_struct \*curr = rq->curr;sched\_clock\_tick();raw\_spin\_lock(&rq->lock);update\_rq\_clock(rq);curr->sched\_class->task\_tick(rq, curr, 0);update\_cpu\_load\_active(rq);calc\_global\_load\_tick(rq);raw\_spin\_unlock(&rq->lock);perf\_event\_task\_tick();#ifdef CONFIG\_SMPrq->idle\_balance = idle\_cpu(cpu);trigger\_load\_balance(rq);#endifrq\_last\_tick\_reset(rq);}static void task\_tick\_fair(struct rq \*rq, struct task\_struct \*curr, int queued){struct cfs\_rq \*cfs\_rq;struct sched\_entity \*se = &curr->se;for\_each\_sched\_entity(se) {cfs\_rq = cfs\_rq\_of(se);entity\_tick(cfs\_rq, se, queued);}if (numabalancing\_enabled)task\_tick\_numa(rq, curr);update\_rq\_runnable\_avg(rq, 1);}static void entity\_tick(struct cfs\_rq \*cfs\_rq, struct sched\_entity \*curr, int queued){update\_curr(cfs\_rq);update\_entity\_load\_avg(curr, 1);update\_cfs\_rq\_blocked\_load(cfs\_rq, 1);update\_cfs\_shares(cfs\_rq);#ifdef CONFIG\_SCHED\_HRTICKif (queued) {resched\_curr(rq\_of(cfs\_rq));return;}if (!sched\_feat(DOUBLE\_TICK) &&hrtimer\_active(&rq\_of(cfs\_rq)->hrtick\_timer))return;#endifif (cfs\_rq->nr\_running > 1)check\_preempt\_tick(cfs\_rq, curr);//检查当前进程是否需要调度}static void update\_curr(struct cfs\_rq \*cfs\_rq){struct sched\_entity \*curr = cfs\_rq->curr;u64 now = rq\_clock\_task(rq\_of(cfs\_rq));u64 delta\_exec;if (unlikely(!curr))return;delta\_exec = now - curr->exec\_start;if (unlikely((s64)delta\_exec <= 0))return;curr->exec\_start = now;schedstat\_set(curr->statistics.exec\_max,max(delta\_exec, curr->statistics.exec\_max));curr->sum\_exec\_runtime += delta\_exec;schedstat\_add(cfs\_rq, exec\_clock, delta\_exec);curr->vruntime += calc\_delta\_fair(delta\_exec, curr);update\_min\_vruntime(cfs\_rq);if (entity\_is\_task(curr)) {struct task\_struct \*curtask = task\_of(curr);trace\_sched\_stat\_runtime(curtask, delta\_exec, curr->vruntime);cpuacct\_charge(curtask, delta\_exec);account\_group\_exec\_runtime(curtask, delta\_exec);}account\_cfs\_rq\_runtime(cfs\_rq, delta\_exec);}static inline u64 calc\_delta\_fair(u64 delta, struct sched\_entity \*se){if (unlikely(se->load.weight != NICE\_0\_LOAD))delta = \_\_calc\_delta(delta, NICE\_0\_LOAD, &se->load);return delta;}static u64 \_calc\_delta(u64 delta\_exec, unsigned long weight, struct load\_weight \*lw){u64 fact = scale\_load\_down(weight);int shift = WMULT\_SHIFT;unsigned long w;if (likely(lw->inv\_weight))return;w = scale\_load\_down(lw->weight);if (BITS\_PER\_LONG > 32 && unlikely(w >= WMULT\_CONST))lw->inv\_weight = 1;else if (unlikely(!w))lw->inv\_weight = WMULT\_CONST;elselw->inv\_weight = WMULT\_CONST / w;if (unlikely(fact >> 32)) {while (fact >> 32) {fact >>= 1;shift--;}}fact = (u64)(u32)fact \* lw->inv\_weight;while (fact >> 32) {fact >>= 1;shift--;}return mul\_u64\_u32\_shr(delta\_exec, fact, shift);} static void check\_preempt\_tick(struct cfs\_rq \*cfs\_rq, struct sched\_entity \*curr){  unsigned long ideal\_runtime, delta\_exec;  struct sched\_entity \*se;  s64 delta;  ideal\_runtime = sched\_slice(cfs\_rq, curr);  delta\_exec = curr->sum\_exec\_runtime - curr->prev\_sum\_exec\_runtime;  if (delta\_exec > ideal\_runtime) {  resched\_curr(rq\_of(cfs\_rq));  clear\_buddies(cfs\_rq, curr);  return;  }  if (delta\_exec < sysctl\_sched\_min\_granularity)  return;  se = \_\_pick\_first\_entity(cfs\_rq);  delta = curr->vruntime - se->vruntime;  if (delta < 0)  return;  if (delta > ideal\_runtime)  resched\_curr(rq\_of(cfs\_rq));  }  static void \_sched \_schedule(void){  next = pick\_next\_task(rq, prev);  p = fair\_sched\_class.pick\_next\_task(rq, prev);  pick\_next\_task\_fair(struct rq \*rq, struct task\_struct \*prev)  se = pick\_next\_entity(cfs\_rq, curr);  struct sched\_entity \*left = \_pick\_first\_entity(cfs\_rq);  put\_prev\_entity(cfs\_rq, pse);  set\_next\_entity(cfs\_rq, se);  rq = context\_switch(rq, prev, next);  }  struct sched\_entity \*\_pick\_first\_entity(struct cfs\_rq \*cfs\_rq){  struct rb\_node \*left = cfs\_rq->rb\_leftmost;  if (!left)  return NULL;  return rb\_entry(left, struct sched\_entity, run\_node);  } 四、实验心得 通过本次实验，我熟悉了Linux进程调度有关函数的源代码，初步了解了如何通过编程实现 Linux 的进程调度算法，尤其是其中的CFS公平调度算法。 |