به نام خدا

دانشکده مهندسی کامپیوتر

درس سیستم‌های عامل
نیم‌سال اول ۰۲-۰۳
استاد: دکتر حسین اسدی

پاسخ تمرین عملی سری چهارم

امیرحسین رحمتی

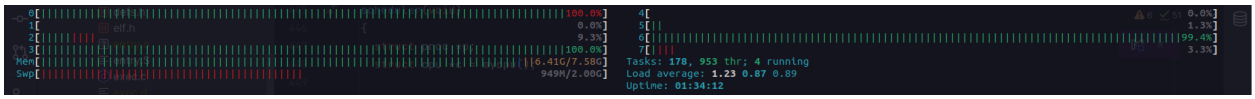
۱.

```
// wait for interrupt
static inline void
wfi()
{
    asm volatile("wfi");
}
```

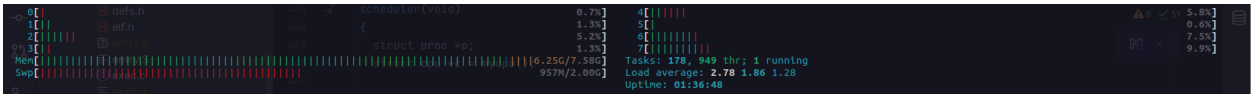
ابتدا تابع wfi را در riscv.h اضافه کردیم.

```
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();

  c->proc = 0;
  for(;;){
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();

    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        // Switch to chosen process.  It is the process's job
        // to release its lock and then reacquire it
        // before jumping back to us.
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);

        // Process is done running for now.
        // It should have changed its p->state before coming back.
        c->proc = 0;
      }
      release(&p->lock);
    }
    wfi();
  }
}
```

و در ادامه آن را در تابع اسکجولر صدا زدیم.



تصویر بالا وضعیت هسته های پردازشی را بدون استفاده از wfi نشان می‌دهد.



تصویر بالا وضعیت هسته های پردازشی را با استفاده از wfi نشان می‌دهد. که مشخص می‌کند استفاده از این دستور باعث کاهش مصرف CPU شده است.

با استفاده از دستور usertests سعی کردم تا سرعت انجام پردازش هارا بررسی کنم. واضحا در زمان استفاده از wfi اجرای این دستور چندین برابر یعنی نزدیک به چند دقیقه طول می‌کشد.(من خودم صبر نکردم تا تمام شود دستور حتی)

۲.

```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  int is_scheduled = 0;

  c->proc = 0;
  for(;;){
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();
    is_scheduled = 0;
    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        // Switch to chosen process.  It is the process's job
        // to release its lock and then reacquire it
        // before jumping back to us.
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);
        is_scheduled = 1;

        // Process is done running for now.
        // It should have changed its p->state before coming back.
        c->proc = 0;
      }
      release(&p->lock);
    }
    if(!is_scheduled) {wfi();}
  }
```

قطعه کد نوشته شده برای سوال ۲ را می‌توانید در تصویر بالا مشاهده کنید. متغیر is scheduled نشان می‌دهد آیا در آخرین اجرای الگوریتم اسکجولینگ پردازه ای اسکجول شده است یا خیر. در صورتی که اسکجول نشده بود، دستور wfi را اجرا می‌کنیم.

افزودن این قطعه کد سرعت سیستم عامل را نسبت به حالت قبل افزایش می‌دهد اما باز هم تفاوت چشم گیری با حالت اولیه دارد.

۳.

```
int
devintr()
{
  uint64 scause = r_scause();
    struct cpu *c;
    int return_value 0;
    if((scause & 0x8000000000000000L) &&
     (scause & 0xff) == 9){
    // this is a supervisor external interrupt, via PLIC.

    // irq indicates which device interrupted.
    int irq = plic_claim();

    if(irq == UART0_IRQ){
      uartintr();
    } else if(irq == VIRTIO0_IRQ){
      virtio_disk_intr();
    } else if(irq){
      printf("unexpected interrupt irq=%d\n", irq);
    }

    // the PLIC allows each device to raise at most one
    // interrupt at a time; tell the PLIC the device is
    // now allowed to interrupt again.
    if(irq)
      plic_complete(irq);

    return_value = 1;
  } else if(scause == 0x8000000000000001L){
    // software interrupt from a machine-mode timer interrupt,
    // forwarded by timervec in kernelvec.S.

    if(cpuid() == 0){
      clockintr();
    }

    // acknowledge the software interrupt by clearing
    // the SSIP bit in sip.
    w_sip( x: r_sip() & ~2);

    return_value =2
  }
  if (return_value != 0){
      for(c = cpus; c < &cpus[NCPU]; c++) {
          __atomic_store_4(&c->wfi_not_needed,1,__ATOMIC_RELAXED);
      }
  }
    return return_value;
}
```

تابع devintr را به شکلی که مشاهده می‌کنید تغییر دادیم تا در زمانی که یک اینتراپت valid شناسایی می‌شود به تمام CPU ها بگوید که نیازی به اجرای دستور WFI نمی‌باشد.

```
443     //    via swtch back to the scheduler.
444     void
445     scheduler(void)
446     {
447       struct proc *p;
448       struct cpu *c = mycpu();
449       int wfi_not_needed = 0;
450
451       c->proc = 0;
452       for(;;){
453         // Avoid deadlock by ensuring that devices can interrupt.
454         intr_on();
455         __atomic_store_4(&c->wfi_not_needed,0,__ATOMIC_RELAXED);
456         for(p = proc; p < &proc[NPROC]; p++) {
457           acquire(&p->lock);
458           if(p->state == RUNNABLE) {
459             // Switch to chosen process.  It is the process's job
460             // to release its lock and then reacquire it
461             // before jumping back to us.
462             p->state = RUNNING;
463             c->proc = p;
464             swtch(&c->context, &p->context);
465
466             // Process is done running for now.
467             // It should have changed its p->state before coming back.
468             c->proc = 0;
469           }
470           release(&p->lock);
471         }
472         wfi_not_needed = __atomic_load_4(&c->wfi_not_needed,__ATOMIC_RELAXED);
473         if(!wfi_not_needed) {wfi();}
474       }
475     }
```

همچنین تابع scheduler را نیز چنین تغییر دادیم تا تصمیم گیری برای اجرای دستور wfi را از روی متغیر نوشته شده در struct cpu به نام wfi_not_needed انجام دهد.

پس از اعمال تغییرات مشاهده می‌شود که سرعت انجام پردازش ها به مراتب از حالت دوم بهتر شده و دستور usertests را سریع تر اجرا می‌کند. و تقریبا مشابه حالتی است که از دستور wfi استفاده نمی‌کردیم.