به نام خدا

دانشگاه تهران

دانشکده برق و کامپیوتر

گزارش کار آزمایشگاه سوم سیستم عامل

پرنا اسدی

مرتضی بهجت

امید پناکاری

فهرست بندی گزارش کار

بخش اول: سوالات پروژه

- **چرا فراخوانی تابع sched ،منجر به فراخوانی تابع scheduler میشود؟**

در هنگام فراخوانی تابع exit ، در انتهای عملیات تابع sched صدا میشود و وارد عملیات زمان بندی میشود . در این تابع state پردازه فعلی بررسی میشود. همچنین intena که یک ویژگی هسته هست ذخیره و بازیابی میشود. سپس با استفاده از تابع swtch، context switch اتفاق افتاده و وارد پردازه زمابند میشویم. انتخاب پردازه بعدی برای اجرا نیز وظیفه scheduler است. Schedulerمقدار بازگشتی ندارد درواقع حلقه میزند و یک پردازه برای اجرا انتخاب میکند و با swtch آن پردازه را اجرا میکند و در انتها کنترل را به آن منتقل میکند. و با صدا کردن switchuvm درواقع تعویض TSSو h/w page table برای پردازه خواسته شده انجام میشود. در انتها اگر زمان تعیین شده تمام شود، پردازه متوقف شده و وضعیت پردازه نیز باید تغییر کند.

- **صف پردازه هایی که تنها منبعی که برای اجرا کم دارند پردازنده است، صف آماده یا صف اجرا نام دارد. در xv6 صف آماده مجزا وجود نداشته و از صف پردازهها بدین منظور استفاده میگردد 15 .در زمانبند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟**

```
struct runqueue {
        spinlock_t          lock;   /* spin lock that protects this runqueue */
        unsigned long       nr_running;         /* number of runnable tasks */
        unsigned long       nr_switches;        /* context switch count */
        unsigned long       expired_timestamp;    /* time of last array swap */
        unsigned long       nr_uninterruptible;   /* uninterruptible tasks */
        unsigned long long  timestamp_last_tick;  /* last scheduler tick */
        struct task_struct  *curr;                /* currently running task */
        struct task_struct  *idle;          /* this processor's idle task */
        struct mm_struct    *prev_mm;       /* mm_struct of last ran task */
        struct prio_array   *active;        /* active priority array */
        struct prio_array   *expired;       /* the expired priority array */
        struct prio_array   arrays[2];      /* the actual priority arrays */
        struct task_struct  *migration_thread; /* migration thread */
        struct list_head    migration_queue;   /* migration queue*/
        atomic_t            nr_iowait; /* number of tasks waiting on I/O */
};
```

ساختار صف اجرا به صورت بالا در لینوکس وجود دارد.دارای یک lock برای حفاظت صف اجرا میباشد.تعداد تسک های قابل اجرا و تعداد تعویض متن و زمان جا به جایی آرایه پیشین و زمان اخرین tick زمانبند نیز به صورت long در این ساختار وجود دارد. همچنین تعداد تسک های منتظر برای I/O نیز ذخیره میشود. این داده ساختار از ساده ترین داده ساختار ها در زمابند میباشد. در مسیر kernel/sched.c قرار دارد.صف اجرا دارای لیستی از پردازه های قابل اجرا در یک پردازه است.برای هر پردازه یک صف وجود دارد و هر پردازه فقط در یک صف قرار میگیرد. نحوه ی عملکرد صف اجرا در لینوکس نیز به این صورت است که از یک درخت قرمز سیاه استفاده می کند که کلید های مقدار virtutime یا زمان مجازی هر پردازه هست که این مقدار در داده ساختار task_struct ذخیره شده است. سمت چپ ترین برگ این درخت به عنوان پردازه برگزیده انتخاب می شود.

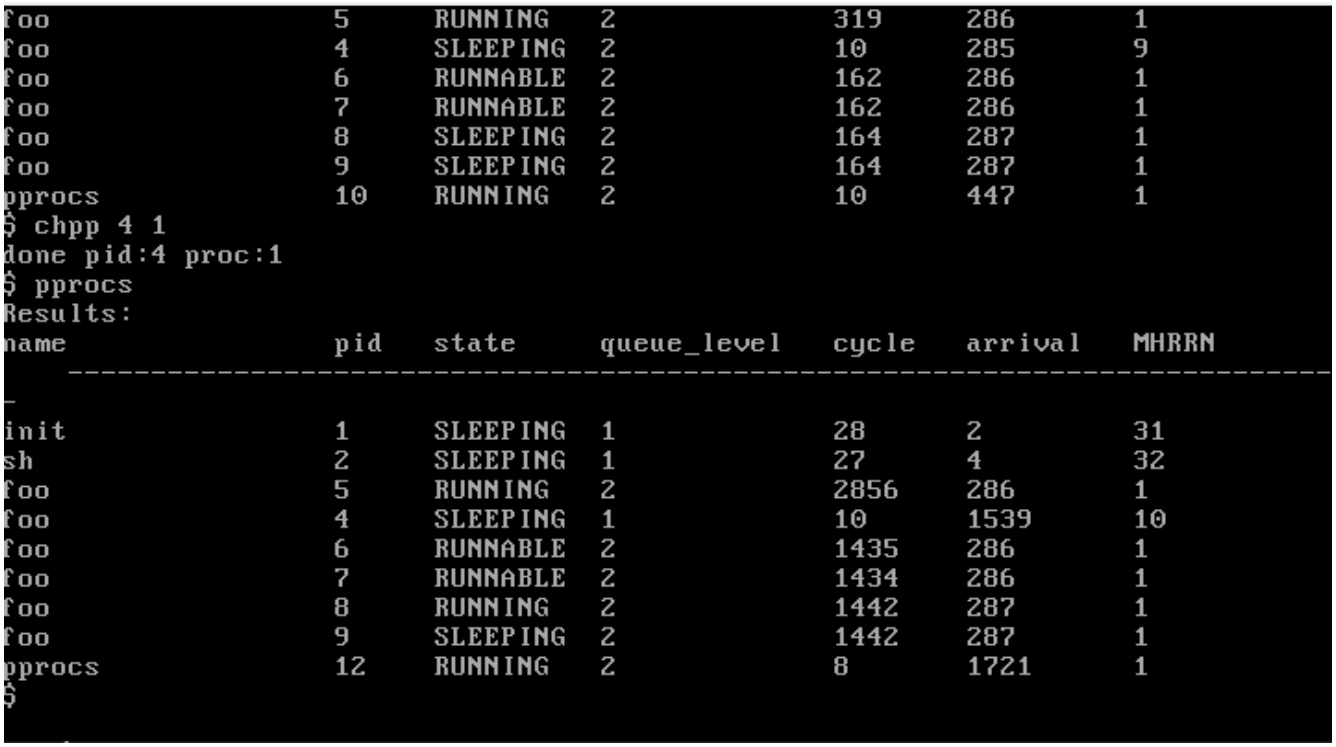
- **هر هسته پردازنده در xv6 یک زمانبند دارد. در لینوکس نیز به همین گونه است. این دو سیستم عامل را از منظر مشترک یا مجزا بودن صف های زمانبندی بررسی نمایید. و یک مزیت و یک نقص صف مشترک نسبت به صف مجزا را بیان کنید.**
  - **لینوکس : صف های زمانبندی در لینوکس برای هر پردازه به صورت مجزا قرار دارد. به طوری که هر پردازه صف مربوط به خودش را دارد.**
  - **Xv6 : صف های زمانبندی در xv6به صورت مشترک هستند و از صف پردازه ها برای صف آماده استفاده میکند.**
  - ✓ **مزیت: برای utilization پردازه مناسب است و برای تمام پردازه نیز منصفانه قرار میگیرد.(load imbalance برخلاف صف مجزا ندارد.)**
  - ✓ **عیب: scalable نیست و cache locality ضعیفی دارد.صف های جدا ساده پیاده سازی میشوند و scalable هستند. همچنین در این شرایط race condition میان پردازنده ها به وجود می آید.**
- **در هر اجرای حلقه، ابتدا برای مدتی وقفه فعال میگردد. علت چیست؟ آیا در سیستم تک هسته ای به آن نیاز است؟**

تابع sti برای فعال کردن وقفه ها در این حلقه صدا زده میشود. علت این موضوع آن است که ممکن است در شرایطی هیچ کدام از پردازه ها آماده ی اجرا نباشند(RUNNABLE) و بعضی از آن ها در انتظار عمل I/O باشند که در این حالت اگر وقفه ها فعال نباشند این امکان وجود ندارد که با پس از اتمام عملیات I/O وضعیت پردازه به درحالت اجرا تغییر کند، پس در نتیجه زمان بند تا ابد در یک حلقه گیر می کند. این حالت هم در سیستم های چندحلقه ای و هم تک حلقه ای ممکن است اتفاق بیافتدو از این نظر تفاوتی ندارند.

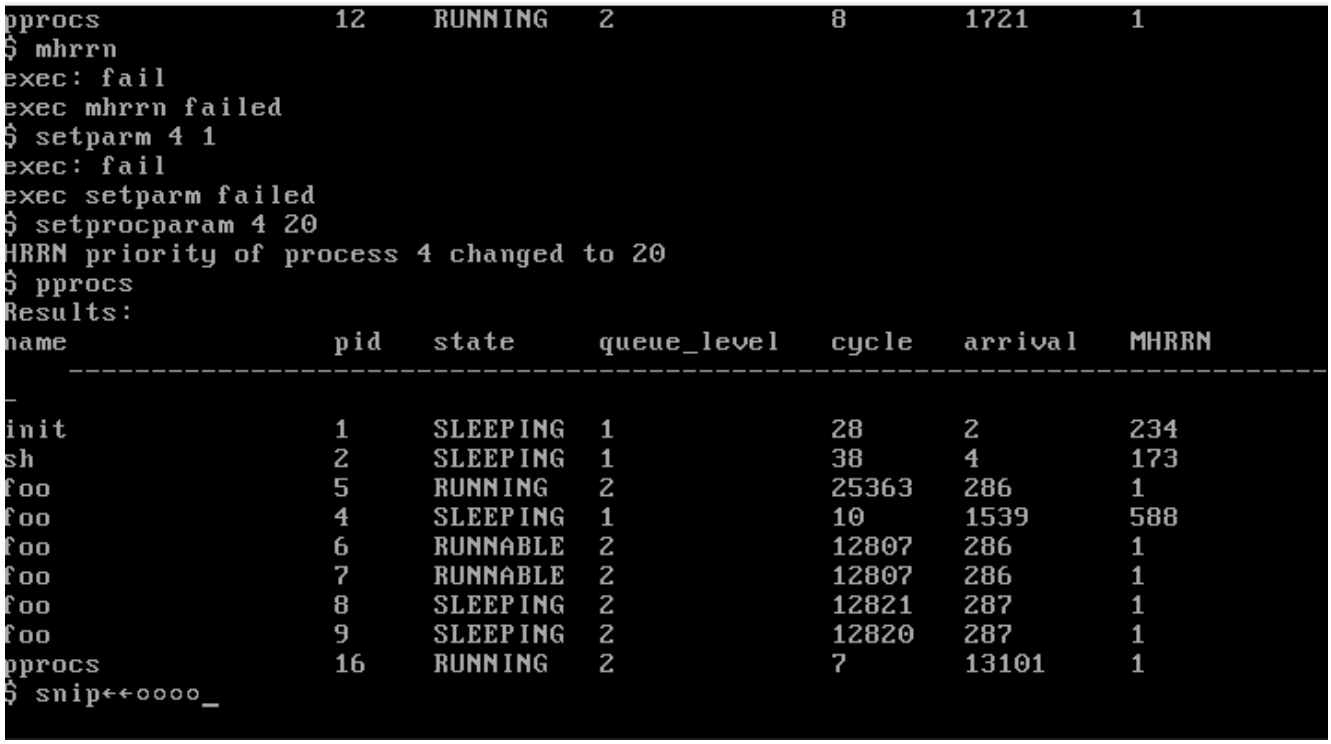
- **وقفه ها اولویت بالاتری نسبت به پردازهها دارند. به طور کلی مدیریت وقفهها در لینوکس در دو سطح صورت میگیرد. آنها را نام برده و به اختصار توضیح دهید. اولویت این دو سطح مدیریت نسبت به هم و نسبت به پردازهها چگونه است؟ مدیریت وقفه ها در صورتی که بیش از حد زمان بر شود، می تواند منجر به گرسنگی پردازه ها گردد. این موضوع می تواند منجر به گرسنگی پردازه ها گردد. این می تواند به خصوص در سیستم های بی درنگ دردسر مشکل ساز باشد. چگونه این مشکل حل شده است؟**

لینوکس وقفه ها را در دو سطح Top-half و Bottom-half مدیریت می کند. در این روش Top-half شامل سرویس روتین اصلی مربوط به interrupt ها می شود که باید بلافاصله اجرا شوند و Bottom-half شامل کارهای مرتبط با interrupt ها می شود که اهمیت کمتری دارند. این بخش از interrupt ها زمانبندی می شوند و می توان مطمئن شد که هیچ کدام از Bottom-half ها هیچ وقت یکدیگر را قطع نمی کنند به همین علت می توان کار های اصلی مربوط به interrupt را انجام داد و سپس بقیه ی کارها را بدون نگرانی از مداخله ی وقفه های دیگر ادامه داد. بنا براین به طور کلی اولویت interrupt ها از تمام پردازه ها بیشتر است و بیشتر کارهای مربوط به آن ها که Top-half ها انجام می شود به صورت فوری اتفاق می افتند اما در interrupt های طولانی برخی از این کارها که اهمیت کمتری دارند به Bottom-half ها سپرده می شوند. همچنین امکانی که می توان برای تکمیل این مکانیزم به برنامه اضافه کرد این است که بتوان Bottom-half ها را در زمان اجرای بخش حیاتی از کد و پردازه های پر اهمیت (به مانند سیستم های بی درنگ) غیر فعال کرد و پس از پایان آن بخش حیاتی دوباره به صف Bottom-half ها بازگشت که این قابلیت می تواند باعث جلوگیری از starvation پردازه ها شود.

بخش دوم: خروجی های پروژه



```
foo                 5     RUNNING   2             319       286       1
foo                 4     SLEEPING  2             10        285       9
foo                 6     RUNNABLE  2             162       286       1
foo                 7     RUNNABLE  2             162       286       1
foo                 8     SLEEPING  2             164       287       1
foo                 9     SLEEPING  2             164       287       1
pprocs              10    RUNNING   2             10        447       1
$ chpp 4 1
done pid:4 proc:1
$ pprocs
Results:
name                pid   state     queue_level   cycle     arrival   MHRRN
    ------------------------------------------------------------------------
_
init                1     SLEEPING  1             28        2         31
sh                  2     SLEEPING  1             27        4         32
foo                 5     RUNNING   2             2856      286       1
foo                 4     SLEEPING  1             10        1539      10
foo                 6     RUNNABLE  2             1435      286       1
foo                 7     RUNNABLE  2             1434      286       1
foo                 8     RUNNING   2             1442      287       1
foo                 9     SLEEPING  2             1442      287       1
pprocs              12    RUNNING   2             8         1721      1
$
```



```
pprocs              12    RUNNING   2             8         1721      1
$ mhrrn
exec: fail
exec mhrrn failed
$ setparm 4 1
exec: fail
exec setparm failed
$ setprocparam 4 20
HRRN priority of process 4 changed to 20
$ pprocs
Results:
name                pid   state     queue_level   cycle     arrival   MHRRN
    ------------------------------------------------------------------------
_
init                1     SLEEPING  1             28        2         234
sh                  2     SLEEPING  1             38        4         173
foo                 5     RUNNING   2             25363     286       1
foo                 4     SLEEPING  1             10        1539      588
foo                 6     RUNNABLE  2             12807     286       1
foo                 7     RUNNABLE  2             12807     286       1
foo                 8     SLEEPING  2             12821     287       1
foo                 9     SLEEPING  2             12820     287       1
pprocs              16    RUNNING   2             7         13101     1
$ snip←←oooo_
```

```
foo                 5     RUNNING   2             25363   286       1
foo                 4     SLEEPING  1             10      1539      588
foo                 6     RUNNABLE  2             12807   286       1
foo                 7     RUNNABLE  2             12807   286       1
foo                 8     SLEEPING  2             12821   287       1
foo                 9     SLEEPING  2             12820   287       1
pprocs              16    RUNNING   2             7       13101     1
$ setsysparam 20
HRRN priority of all processes changed to 20
$ pprocs
Results:
name                pid   state     queue_level   cycle   arrival   MHRRN
    ------------------------------------------------------------------------
_
init                1     SLEEPING  1             28      2         312
sh                  2     SLEEPING  1             42      4         211
foo                 5     RUNNING   2             32790   286       10
foo                 4     SLEEPING  1             10      1539      777
foo                 6     RUNNABLE  2             16586   286       11
foo                 7     RUNNABLE  2             16590   286       11
foo                 8     SLEEPING  2             16606   287       10
foo                 9     SLEEPING  2             16604   287       10
pprocs              18    RUNNING   2             7       16885     1
$ o
```