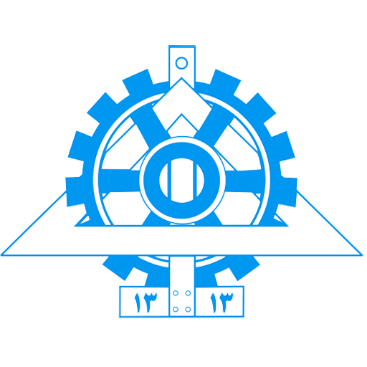
**به نام خدا**

** 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 ......................................................................................3**
* **بخش دوم: زمانبندی بازخوردی چند سطحی ........................................................8**
* **بخش سوم: سازوکار افزایش سن .......................................................................11**
* **بخش چهارم: فراخوانی های سیستمی مورد نیاز....................................................13**

**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.زمانبند در تابع swtch استفاده میشود که دو ورودی میگیرد و حالت پردازه توسط این تابع از حالت مربوط به زمانبند به حالت پردازه برگزینده در انتها تغییر میکند. انتخاب پردازه بعدی برای اجرا نیز وظیفه scheduler است.Scheduler مقدار 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 TSS و 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 context switch 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.

* 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 xv6 به 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.
* 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.اما عدم دسترسی به قفل جدول پردازه ها، وقفه ها را غیرفعال میکند.با صدا زدن این تابع، زمانبند هرآنچه که وضعیت push/popcli هست را بازنویسی میکند که اجازه رخداد یه وقفه IO را میدهد.یه بازه زمانی بین صدا زدن تابع sti و cli وجود دارد. ( در تابع acquire,pushcli,cli).درواقع sti صریحا وقفه ها را فعال نمیکند.بلکه از تابع release ، تابع popcli صدا زده میشود که باعث فعال شدن وقفه میشود.هرکجا که popcli وقفه ای را فعال کند، pushcli قطعا وقفه را غیرفعال خواهد کرد(به طور خلاصه برای تعویض متن صورت میگیرد). در سیستم های تک هسته نیز به دلیل نیاز به تعویض متن در ابتدای هر حلقه نیاز به فعال کردن وقفه داریم.**