.1

- تابع pinit: این تابع که نه ورودی میگیرد و نه چیزی برمیگرداند. در این تابع initlock داریم که در spinlock.c تعریف شده. این تابع یک اشاره گر از نوع struct spinlock و یک اسم میگیرد.
- تابع cpuid: این تابع آیدی سی پی یو (مقدار برگردانده شده از فراخونی تابع mycpu منهای cpus) را برمیگرداند. cupus از نوع struct cpu در mp.c تعریف شده است. در xv6 حدکثر تعداد سی پی یو 8 است. این تابع زمانی باید فراخوانی شود که وقفه ها غیر فعال باشند.
- تابع mycpu: این تابع یک اشاره گر به struct cpu سی پی یو فعلی برمیگرداند. در این تابع یک شرط وجود دارد تا زمانی که وقفه ها غیر فعال نباشند اجرا نشود. در این تابع یک متغیر عدد صحیح apicid تعریف می‌شود و مقدار برگردانده شده از تابع lapicid که در lapic.c تعریف شده، به آن داده میشود. در این تابع یک حلقه وجود دارد تا بتواند سی پی یو فعلی را به درستی برگرداند.
- تابع myproc: این تابع اشاره گر struct pointer را برای پردازه ای که روی cpu فعلی اجرا میشود، برمیگرداند. این تابع وقفه ها را غیر فعال می کند، تابع mycpu را فراخوانی می کند، اشاره گر پردازه فعلی را میگیرد و وقفه را دوباره فعال می کند.
- تابع allocproc: این تابع برای تخصیص پردازه است. در جدول پردازه ها به دنبال پردازه ای که در حالت unused است می‌گردد (برای اینکه در زمان گشتن، پردازه های دیگر به این جدول دسترسی نداشته باشند و در حالت waiting بمانند، در ابتدا ptable.lock را acquire میکند و در انتهای جستجو، آن را آزاد میکند). اگر پیدا کرد وضعیت آن را به EMBRYO تغییر می‌دهد و pid آن را مقداردهی می کند. این تابع یک اسلات (struct proc) را در جدول پردازه ها تخصیص می‌دهد. برای این کار kernel stack را تخصیص میدهد و فضا برای trap frame آزاد میگذارد و یک context راه اندازی میکند تا در forkret اجرا شود.
- تابع userinit: این تابع اولین پردازه کاربر را راه اندازی می کند. در ابتدا با تابع allocproc پردازه تخصیص داده میشود. State تابع RUNNABLE می‌شود تا بقیه هسته ها بتوانند این پردازه را اجرا کنند. در این تابع دستور lock ضروری است چون که ممکن است این assignment به صورت اتمیک نباشد.
- تابع growproc: اندازه حافظه پردازه را n بایت تغییر می دهد. اگر با موفقیت انجام شد 0 و در غیر این صورت 1- را برمیگرداند. اشاره گر curproc از نوع struct proc تعریف می شود. سایز آن از نوع عدد صحیح مثبت تعریف و مقداردهی می شود. بر اساس مثبت یا منفی بودن n که ورودی تابع است، با تابع های allocuvm و deallocuvm اندازه حافظه پردازه تغییر می کند.
- تابع fork: یک پردازه جدید ایجاد می‌کند که از پردازه والد کپی میکند. در ابتدا np و curproc از نوع اشاره گر struct proc تعریف می‌شوند و برای curproc تابع myproc فراخوانی میشود.

سپس به کمک تابع allocproc در صورت موفقیت آمیز بودن، پردازه تخصیص داده میشود. سپس به کمک copyuvm، اسیت پردازه کپی می شود. مقدار eax صفر می‌شود تا این تابع در فرزند مقدار صفر را برگرداند و state پردازه در حالت lock به دلیل اینکه ممکن است اتمیک نباشد، به صورت RUNNABLE در می آید.

- تابع exit: این تابع چیزی برنمیگرداند. پردازه فعلی را خارج می کند. پردازه خارج شده تا زمانی که والد آن تابع wait را صدا بزند و متوجه شود که این پردازه خارج شده است، در حالت ZOMBIE میماند. در این تابع تمام فایل‌های باز، بسته میشوند. وضعیت به تابع wait گزارش می شود. این تابع منابع را آزاد می کند، والد را اگر در حالت wait باشد، بیدار می کند. فرزند باید $p \rightarrow lock$ خود را نگه دارد چون که در غیر این صورت والد ممکن است آن را در حالت ZOMBIE ببیند و آن را آزاد کند، در حالی که ممکن است در حال اجرا باشد. (به دلیل غیر اتمیک بودن دستورها).
- تابع wait: این تابع منتظر میماند تا پردازه فرزند تابع exit را صدا بزند و pid خود را برگرداند. این تابع 1- را برمیگرداند اگر پردازه فرزندی نداشته باشد. این تابع در جدول پردازه ها دنبال فرزندی که در وضعیت ZOMBIE باشد، میگردد، سپس منابع و struct proc فرزند را آزاد میکند و exit status آن را کپی میکند و ID پردازه فرزند را برمیگرداند. اگر والد قبل از فرزند خارج شده باشد، فرزند را به پردازه init میدهد. اگر wait فرزندی که exit کرده باشد را پیدا نکند، sleep را صدا میزند. این تابع برای همه پردازه ها $np \rightarrow parent$ را چک میکند تا فرزندهای آن را پیدا کند. اگر پردازه فرزندی نداشته باشد دستور release(&ptable.lock) انجام می‌شود چرا که وقتی فرزندی نباشد نیازی به منتظر ماندن نیست.
- تابع scheduler: وظیفه این تابع این است که انتخاب کند چه پردازه ای بار بعد اجرا شود. این تابع چیزی برنمیگرداند. در ابتدا اشاره گر c از نوع struct cpu تعریف می‌شود و از طریق فراخوانی mycpu یک cpu گرفته می‌شود و در ابتدا هیچ فرآیندی روی آن cpu گذاشته نمیشود. در این تابع یک حلقه بی‌انتها وجود دارد. در این حلقه، ابتدا ptable.lock را acquire میکنیم و در آخر release میکنیم تا در این فاصله پردازه دیگری به این جدول دسترسی نداشته باشد. در جدول پردازه ها جستجو می‌شود تا پردازه RUNNABLE پیدا شود. از طریق switchuvm اطلاعات آن پردازه لود می‌شود و وضعیت آن به RUNNING تغییر می‌کند و بعد از آن context switch به پردازه جدید انجام می‌شود و اطلاعات کرنل لود میشود. پردازه باید خودش ptable.lock را آزاد کند و دوباره بگیرد و بعد از اینکه کارش را انجام داد، وضعیت خودش را تغییر دهد. در آخر $c \rightarrow proc$ صفر می‌شود و ptable.lock آزاد میشود.
- تابع sched: این تابع نه ورودی میگیرد و نه چیزی برمیگرداند. این تابع ()myproc را فراخوانی میکند و چک میکند که ptable.lock نگه داشته شده باشد و اگر $mycpu() \rightarrow ncli$ برابر یک باشد. ncli در proc.h تعریف شده و برابر عمق pushcli nesting است. و چک میکند که وضعیت پردازه

RUNNING نباشد. متغیر intena برای این است که چک کند که آیا وقفه قبل از pushcli فعال شده است یا نه. این تابع مقدار intena را ذخیره می‌کند و بازمی‌یابد چون که intena متعلق به ریسه هسته است و نه سی پی یو فعلی.

- تابع yield: نه ورودی می‌گیرد و نه چیزی برمیگرداند. این تابع سی پی یو را برای یک دور زمانبندی میدهد. Ptable.lock را acquire میکند، وضعیت پردازه فعلی را RUNNING میکند، تابع schded را صدا میزند و سپس ptable.lock را آزاد می کند.
- تابع forkret: این تابع نیز نه ورودی میگیرد و نه چیزی برمیگرداند. در صورتی که یک پردازه جدید اولین زمانبندی شده باشد، به جای ()sched، این تابع صدا زده می‌شود (در ()allocproc). این تابع وجود دارد تا ptable.lock را که قبل تر نگه داشته شده بود، آزاد کند. هم چنین ROOTDEV که در param.h تعریف شده و برابر یک است، مقدار دهی اولیه میشود.
- تابع sleep: این تابع ptable.lock را acquire میکند. هر پردازه ای که به خواب رود، هم ptable.lock و هم lk را میگیرد. نگه داشتن lk نیاز است چون اگر تضمین میکند که هیچ پردازه دیگری نتواند wakeup را صدا بزند. وقتی که ptable.lock را نگه دارد، میتوان lk را آزاد کرد چون اگر پردازه دیگری wakeup را صدا زند، wakeup منتظر میماند تا ptable.lock را بتواند acquire کند و در نتیجه منتظر می‌ماند تا ()sleep بتواند پردازه را sleep کند. اشاره گر chan که در struct proc تعریف شده است، برای پردازه مقداردهی می‌شود و وضعیت پردازه SLEEPING می‌شود و ()sched صدا زده می‌شود. در آخر p → chan خالی می‌شود و ptable.lock آزاد شده و lk دوباره acquire می‌شود.
- تابع wakeup1: همه پردازه هایی که روی chan، sleep کرده بودند را بیدار می‌کند و وضعیت پردازه را RUNNING می‌کند. Ptable.lock باید نگه داشته شود.
- تابع wakeup: در این تابع چون ممکن است wakeup(chan) غیراتمیک اجرا شود، قبل از آن ptable.lock را acquire می‌کند و بعد از آن آزاد میکند. این تابع همه پردازه ها روی chan، sleep کرده بودند را بیدار میکند.
- تابع kill: یک pid میگیرد و آن پردازه را kill میکند. اگر موفق نشد 1- را برمیگرداند. این تابع اجازه میدهد تا یک پردازه درخواست کند یک پردازه دیگر terminate شود. این تابع صرفاً p →killed آن پردازه را برابر یک قرار میدهد(متغیر killed در struct proc در proc.h تعریف شده است) و آن را بیدار میکند (وضعیت آن را از SLEEPING به RUNNABLE تغییر میدهد). قبل از این کار ptable.lock را acquire میکند و در انتها آزاد میکند. اگر p → killed برابر یک شده باشد، کد درون usertrap تابع exit را صدا میزند.
- تابع procdump: این تابع ورودی نمیگیرد و چیزی برنمیگرداند. کار این تابع این است که لیست پردازه ها را روی کنسول چاپ کند. این تابع هیچ قفلی را acquire نمیکند. در ابتدا آرایه ای از وضعیت‌های ممکن پردازه تعریف میشود. حلقه به اندازه تعداد پردازه ها اجرا میشود، اگر

وضعیت پردازه UNUSED باشد کاری نمی کند، برای وضعیت‌های دیگر، وضعیت را در متغیر state مقداردهی می‌کند و آن را به همراه pid و اسم پردازه چاپ می‌کند. اگر وضعیت پردازه SLEEPING باشد، تابع getcallerpcs صدا زده می‌شود که در spinlock.c تعریف شده است و مقدار ebp که در struct context در proc.h تعریف شده به علاوه ۲ را در pc قرار میدهد و در انتها مقادیری از آرایه pc که صفر نیستند را چاپ می‌کند.

2. الگوریتم زمانبندی پیشفرض xv6 الگوریتم Round Robin است چون پردازه دایما اجرا نمیشود و وقتی وسط کارش وقفه تایمر رخ میدهد، مجبور است از حالت RUNNING به RUNNABLE تغییر وضعیت دهد (Preemptive).
برای تغییر در الگوریتم زمانبندی باید تابع در proc.c تغییراتی انجام دهیم. همچنین به proc.h نیاز داریم و ممکن است برای برخی define کردن‌ها به param.h نیاز داشته باشیم. همچنین ممکن است به exec.c نیاز داشته باشیم. برای تغییر در الگوریتم زمانبندی ممکن است به اضافه کردن یک فراخوانی سیستم هم نیاز داشته باشیم که برای آن به فایل‌های syscall.h ,syscall.c,sysproc.c, usys.S, user.h, defs.h نیاز داریم.
برای تغییر در الگوریتم زمانبندی مهم‌ترین تغییر ها در proc.c انجام میشود. چون اولویت دهی به پردازه ها در الگوریتم های زمانبندی با هم متفاوت است، برای تعیین اولویت باید ()allocproc را تغییر دهیم. مثلاً برای زمانبندی priority، عمل‌کرد مدنظرمان را به آن تابع اضافه میکنیم. هم چنین در ()scheduler نحوه زمانبندی را جایگزین کدهای درون این تابع می کنیم.

3. فراخوانی های سیستم برای سرویس های سخت افزار، ساخت یا اجرای پردازه و ارتباط با سرویس های هسته مانند زمانبندی پردازه، استفاده میشوند. برای تعریف یک فراخوانی سیستم، ابتدا باید در syscall.h تعریف شود و یک عدد به آن تخصیص یابد، سپس باید در syscall.c یک اشاره گر به فراخوانی سیستم اضافه شود. در این فایل یک آرایه از اشاره گر های تابع وجود دارد که از عدد های تعریف شده برای سیستم کال ها به عنوان اشاره گر به آن‌ها استفاده می کند. در این تابع باید prototype تابع نیز تعریف شود. در sysproc.c تابع های سیستم کال ها تعریف میشوند. توابع درون این فایل ورودی نمیگیرند و به عنوان مثال برای فراخوانی سیستمی مانند fork صرفاً ()fork برگردانده می‌شود و خود ()fork در proc.c پیاده‌سازی می‌شود. فراخوانی های سیستم برای رابط با کاربر در usys.S تعریف می‌شوند و همچنین در user.h پروتوتایپ توابع آن‌ها تعریف میشود. همچنین پروتوتایپ توابع هر کدوم از آن‌ها در defs.h نیز باید تعریف شود.

تابع syscall وظیفه مدیریت سیستم کال ها را دارد. در syscall.h هر سیستم کال یک شماره دارد که برای هر سیستم کال در رجیستر eax ذخیره میشود. در تابع syscall شماره مختص آن سیستم کال در

متغیر num مقدار دهی میشود. در این تابع یک شرط داریم که اگر num از صفر بزرگ‌تر باشد و از تعداد المنت های سیستم کال ها کمتر باشد و syscalls[num] موجود باشد، آن سیستم کال را اجرا کند و در غیر این صورت پیغام مناسب چاپ می‌شود و در eax مقدار 1- قرار داده میشود.