- فراخوانی سیستم getParentID: در فایل proc.h داخل struct proc یک parent داریم که از جنس struct proc است که خود یک آیدی دارد. در این فراخوانی سیستم ما به parent می‌رویم و آیدی آن را دریافت می‌کنیم. برای اضافه کردن این فراخوانی سیستم ما این فراخوانی سیستم را با سینتکس مناسب به syscall.h, sysproc.c, syscall.c, usys.S, user.h, defs.h تعریف کردیم.
- فراخوانی سیستم getChildren یک ورودی می‌گیرد که pid است که ما می‌خواهیم چک کند فرزندی دارد یا خیر (و اگر دارد، چه فرزندانی). این فراخوانی سیستم با جستجو در جدول پردازه‌ها و چک می‌کند که شناسه والد پردازه با شناسه‌ای که از ورودی گرفتیم یکسان است یا خیر. اگر یکسان باشد، آن پردازه، پردازه فرزند است و شناسه آن را برمی‌گردانیم.
- برای فراخوانی سیستم getSyscallCount، یک آرایه به طول 64 درون struct proc تعریف شده است که شماره فراخوانی های سیستم را ذخیره کند. زمانی که یک پردازه یک فراخوانی سیستم انجام می‌دهد، با اضافه کردن دستور مناسب در تابع syscall در فایل syscall.c شمارنده افزایش می‌یابد. همچنین موقع اجرای یک پردازه، با دستور مناسب در proc.c همه خانه‌های این آرایه را صفر می‌شود تا با اجراهای بعد تداخل ایجاد نشود.
- برای تغییر کوانتوم زمانی الگوریتم Round Robin، یک متغیر صحیح درون struct proc ایجاد شده است به نام ticksPassed. این متغیر تیک‌هایی که به سی پی یو تا آن واحد زمانی خورده است را نگه می‌دارد. درون تابع yield در فایل proc.c یک عدد به این متغیر برای پردازه افزوده می‌شود و زمانی که این متغیر از پردازه فعلی از مقدار QUANTUM که 10 تعریف کرده‌ایم، بیشتر باشد، تعویض زمینه (Context Switch) صورت می‌گیرد و مقدار این متغیر بعد از تعویض زمینه صفر می‌شود.
- برای الگوریتم Priority Scheduling، درون struct proc دو متغیر تعریف می‌کنیم. متغیر priority که مقدار اولویت در آن ذخیره می‌شود و هر چه کمتر باشد به معنای بیشتر بودن اولویت است و متغیر priorityModule که برای انتخاب پردازه با اولویت بیشتر است. در تابع allocproc در proc.c اولویت پردازه ۳ در نظر گرفته می‌شود و به کمک متغیرهای minPriority و متغیر flag که صفر یا یک مقدار دهی می‌شود، priorityModule آپدیت می‌شود. در تابع scheduler در همین فایل، یک اشاره گر struct proc داریم برای تعیین پردازه‌های با اولویت بیشتر. بعد از هر کوانتوم زمانی، الگوریتم priorityModule را آپدیت می‌کند. به این صورت که مقدار این متغیر با متغیر priority پردازه‌های در حال اجرا آپدیت می‌شود. الگوریتم بیشترین priorityModule یعنی کوچکترین مقدار را بین پردازه‌ها دارد را به وضعیت اجرا در می‌آورد که این پردازه می‌تواند یک پردازه جدید باشد یا از بین پردازه‌های قبلی باشد.
- در فراخوانی سیستم setPriority یک ورودی داریم که با توجه به آن ورودی، متغیر priority از struct proc مقداردهی می‌شود. در این فراخوانی سیستم، اگر ورودی عددی خارج از بازه ۱ تا ۶ باشد، اولویت ۵ در نظر گرفته می‌شود.

- در فراخوانی سیستم changePolicy، چهار الگوریتم زمانبندی define کردیم با عدد های ۰ تا ۳. اگر ورودی این تابع خارج از این بازه می‌شود، الگوریتم زمانبندی همان الگوریتم پیش‌فرض که با عدد صفر در نظر گرفته شده‌است، تعیین می‌شود. در فایل proc.c یک متغیر صحیح SCHEDULING_POLICY تعریف کردیم که در ابتدا همان DEFAULT_SCHEDULING مقداردهی شده است. در تابع scheduler همین فایل، با دستور شرطی این فراخوانی سیستم اعمال می‌شود.
- برای قابلیت اندازه‌گیری زمان، در ابتدا در struct proc در فایل proc.h متغیرهای صحیح creationTime, terminationTime, runningTime, readyTime و sleepingTime تعریف می‌شود. در تابع allocproc در فایل proc.c متغیر creationTime پردازه را برابر تیک سی پی یو که متغیر ticks است، قرار می‌دهیم و بقیه متغیرها را صفر مقداردهی می‌کنیم. در تابع exit نیز متغیر terminationTime را برابر ticks قرار می‌دهیم. در این فایل تابع updateProcessTimes را تعریف می‌کنیم که نه ورودی دارد و نه چیزی برمی‌گرداند. در این تابع با گشتن در جدول پردازه‌ها، برای هر پردازه به کمک دستورهای شرطی، اگر وضعیت پردازه RUNNABLE باشد، متغیر readyTime افزایش می‌یابد، اگر وضعیت RUNNING باشد، متغیر runningTime افزایش می‌یابد و اگر وضعیت SLEEPING باشد، متغیر sleepingTime. از این تابع در فایل trap.c در تابع trap استفاده می‌شود و بعد از هر تیک سی پی یو، این تابع صدا زده می‌شود تا مقادیر متغیرها آپدیت شوند. برای حالتی که وضعیت پردازه ZOMBIE است، نباید زمان چرخش و زمان انتظار تغییر کند. به این منظور که ما بتوانیم در فضای کاربر نیز از این تابع استفاده کنیم، فراخوانی سیستم waiting تعریف کردیم که یک struct processTime دارد که این struct در proc.h تعریف کردیم و همان متغیرهایی دارد که به struct proc اضافه کردیم. در این فراخوانی سیستم تابع wait_for_child صدا زده می‌شود که در proc.c تعریف کردیم. این تابع یک struct processTime می‌گیرد و خروجی آن شناسه پردازه است. این تابع با بررسی تمام پردازه‌ها و دستور شرطی برای وضعیت ZOMBIE این مسئله را حل می‌کند.
- برای الگوریتم زمانبندی طبق صف چند لایه‌ای، تابع mlq درون proc.c تعریف می‌شود. این تابع ورودی نمی‌گیرد و شناسه پردازه‌ای که قرار است اجرا شود را برمی‌گرداند. در این تابع به جای تعریف ساختار صف، ما با ذخیره در ptable و جستجو برای اجرای پردازه ۴ اولویت متفاوت تعیین کردیم. اولویت اول همان زمانبندی پیشفرض، اولویت دوم زمانبندی بر اساس اولویت، اولویت سوم زمانبندی برعکس اولویت قبل و اولویت چهارم زمانبندی Round Robin است. این چهار اولویت با بازه‌های مناسب define شده اند. در تابع scheduler اگر SCHEDULING_POLICY برابر MLQ_SCHEDULING باشد، این تابع صدا زده می‌شود و پردازه‌ای که شناسه آن از تابع گرفته شده است، به حالت اجرا در می‌آید. همچنین تابع changeQueue تعریف شده است که با صدا زدن آن، اولویت صف یکی کمتر می‌شود.