به نام خدا

گزارش پروژه درس بی‌درنگ

گروه ۱۶

توضیحات پیاده‌سازی

وظیفه ها:

برای ایجاد هر وظیفه، کلاس های زیر پیاده سازی شده‌اند؛

- کلاس base_task: کلاس پایه وظیفه هایمان می باشد و شامل اطلاعاتی درباره تناوب و سطح بحرانیت هر وظیفه است. همچنین برای آن یک سری متود ها به صورت انتزاعی تعریف شده‌اند که در کلاس های یکه از آن ارث‌بری می‌کنند، در ادامه توضیح داده خواهند شد. تابع get_utilization، بهره وری وظیفه که حال تقسیم زمان اجرا بر تناوب است را برمی‌گرداند.
- کلاس high_critically_task: از کلاس base_task ارث‌بری می‌کند و علاوه بر اطلاعات کلاس پدر، شامل اطلاعاتی نظیر زمان اجرای بزرگ و کوچک و ددلاین مجازی می باشد. برای راحتی پیاده سازی کد، تابع get_computation_time پیاده سازی شده و بر اساس حالت فعلی، زمان اجرا را برمیگرداند. (در حالت نرمال -> زمان اجرای کوچک، در حالت سرریز -> زمان اجرای بزرگ) همچنین تابع get_deadline نیز ددلاین وظیفه را براساس حالت فعلی برمی‌گرداند.(در حالت نرمال -> ددلاین مجازی، در حالت سرریز -> ددلاین اصلی)
- کلاس low_critically_task: از کلاس base_task ارث بری می‌کند و تابع get_computation_time زمان اجرای آن و تابع get_deadline زمان ددلاین آن را برمی گرداند.

حال برای ساخت این وظایف، از الگوریتم uunifast استفاده شده که به تشریح این الگوریتم می‌پردازیم.
در این الگوریتم مقدار بهره وری کلی وجود دارد و برای هر وظیفه یک بهره وری رندوم تعریف می شود. دقت می‌شود که مجموع بهره وری همه وظایف کمتر مساوی بهره وری کل شود و ددلاین هر وظیفه نیز کمتر مساوی بهره وری هر هسته شود. از آنجایی که همه هسته ها همگن می باشند، بهره وری کل از حاصلضرب تعداد هسته ها در بهره وری هر هسته محاسبه می‌شود.
سپس وظایف تولید شده را به نسبت گفته شده برای وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین تقسیم بندی می‌کنیم. سپس یک عدد رندوم برای تناوب هر وظیفه نیز تولید می‌کنیم. حال با توجه به اینکه بهره وری از حاصل تقسیم زمان اجرا (زمان اجرای کوچک برای وظایف با سطح بحرانیت بالا) بدست می‌آید، این مقدار را محاسبه کرده و برای وظایف با سطح بحرانیت بالا نیز یک عدد رندوم با بازه های مشخص برای زمان اجرای بزرگ تولید می‌کنیم.

منابع:
برای منبع یک مدل به نام resource تعریف شده است و شامل اطلاعاتی درباره ظرفیت منبع(اینکه از آن منبع در کل چند واحد موجود است) و سطح دسترسی آن در هرلحظه(اینکه در حال حاضر آزاد است یا خیر)می‌باشد.
ظرفیت منبع نیز به اندازه تعداد هسته ها در نظر گرفته می شود. این فرآیندها در resource generation انجام می‌شود.

ادغام وظایف تولید شده با منابع درخواستی:
حال برای وظایف شده باید تعیین کنیم که هر وظیفه نیاز به کدام منابع برای انجام دارد و هر منبع را برای چند واحد زمانی نیاز دارد. برای این کار در ماژول resource_usage با تقسیم بندی زمان اجرای هر وظیفه بین منابع موجود، این امر را محقق کرده‌ایم. دقت شود که لزوما هر وظیفه، همه منابع را نیاز نخواهد داشت و واحد زمانی مورد نیاز برای منابعی که از آن استفاده نمی‌کند، برابر با صفر خواهد بود.

هسته ها:
برای هسته ها، یک مدل به نام core تعریف شده است و شامل اطلاعاتی درباره وظایفی که انجام داده، وظایفی که به آن نگاشت شده است،ازدحام منابع برروی آن و بهره‌وری هسته می‌باشد. نحوه مقدار دهی به این فیلد ها در ادامه در ماژول simulator توضیح داده شده است.

اکنون که وظایف تولید شده اند و منابع و هسته ها نیز مشخص هستند، نوبت به شبیه سازی می رسد. برای این امر، ماژول simulator نوشته شده است و وظایف و منابع و هسته ها به آن داده می‌شود تا کار شبیه سازی را شروع کند. این ماژول در ادامه تشریح می‌شود. همچنین برای تست این ماژول و گزارش دهی نتایج، ورودی هایی به برنامه داده می‌شود که این ورودی ها به فرمت json در دایرکتوری inputs قابل مشاهده هستند.

ماژول Simulator

پیاده‌سازی اصلی الگوریتم زمان‌بندی و نگاشت به هسته‌ها را در اینجا پیاده‌سازی کردیم. در ابتدا در مقدار دهی این تسک‌های با بحرانیت بالا، تسک‌ها با بحرانیت پایین، منابع و هسته‌هارا میگیریم. همینطور برای کل شبیه‌ساز یک mode در نظر میگیریم که نشان‌دهنده وضعیت بحرانیت سیستم هست. و برای جلو بردن همه تغییرات و نگه‌داشتن زمان از current_time استفاده میکنیم.
در ابتدا در مقدار دهی این کلاس مقدار x برای EDF-VD را محاسبه میکنیم. برای مصاحبه این مقدار Utitlization تسک‌های low critical و Utilization تسک‌های high critical را محاسبه میکنیم. سپس از رابطه زیر برای محاسبه x استفاده میکنیم:

$$x \leftarrow \frac{U_{\mathrm{HI}}^{\mathrm{LO}}(\tau)}{1 - U_{\mathrm{LO}}^{\mathrm{LO}}(\tau)}$$

در قدم بعدی ددلاین تسک‌های با بحرانیت بالا را محاسبه و مقدار دهی میکنیم. در تابع update_high_critical_tasks_ این کار را انجام میدهیم.

همینطور یک مپ نگه میداریم که نشان‌دهنده تسک‌های اساین شده به هر هسته میباشد. در هر مرحله شبیه‌سازی اگر تسکی را به یک هسته اساین کردیم این مپ را آپدیت نگه میداریم.

در قدم بعدی جدول مربوط به MSRP را تشکیل میدهیم. در ایجاد این جدول که یک جدول دو بعدی هست تابع CR برای برای هر منبع و با n_k های مختلف محاسبه میکنیم. در این محاسبه ابتدا ماکسیمم پریود را محاسبه میکنیم. در پیاده‌سازی چون از توابع max و min استفاده کردیم در صورت لزوم ممکن هست نیاز باشد تا یک دیفالت برای این تابع در نظر بگیریم و چون پریود بیشتر یعنی اولویت کمتر(۰) در نتیجه از ماکسیمم پریود برای دیفالت توابع max استفاده میکنیم.

در ادامه برای محاسبه این جدول حساب میکنیم اگر تسکی از این منبع به تعداد بیشتر از i استفاده مینمود مینیمم همه پریودهای این جنس تسک‌هارا میگیریم و برابر با آن خانه منبع قرار میدهیم. در واقع در اینجا اولویتی برای هر تسک تعریف نکردیم به صورت جدا بنابراین پریود کمتر معنی اولویت بالاتر میدهد. بدین شکل جدول مربوط به MSRP را تشکیل میدهیم.

همینطور یک system_preemption_level تعیین میکنیم که در ابتدا math.inf هست. بدلیل استفاده از پریود برای نشان دادن اولویت هر تسک این فیلد ماکسیمم بودنش به معنی کمترین اولویت هست.

در ادامه شبیه‌سازی را با تابع execute شروع میکنیم. ابتدا در این تابع _assigne_to_core را صدا میزنیم که در آن به ترتیب تسک‌هارا به هسته‌ها اساین میکنیم. در این اساین کردن ابتدا لیست کل کورهارا میگیریم. سپس آن هارا بر اساس محسابه congestion مرتب میکنیم. در قدم بعدی روی کل کورها فیلتر میزنیم تا کورهایی بمانند که یوتیلیزیشن باقی‌مانده آن‌ها از یوتلیزیشن تسک کمتر باشد. سپس یک While current_time < time_limit داریم که درواقع به جای time_limit اندازه پنجره شبیه‌سازی ما قرار میگیرد. در ابتدای شبیه سازی system_preemption_level را با تابع update_system_preemption_level_ آپدیت مینماییم. در این تابع preemption_level هر تسک را میگیریم و بین همه آن‌ها مینیمم میگیریم. که در واقع بالاترین اولویت را بدست میاوریم.

در قدم بعدی update_mode را کال میکنیم. در واقع در این تابع بررسی میکنیم که اگر تسکی از جنس بحرانیت بالا وجود داشت و اکتیو بود(جاب فعالی از آن در دست اجرا بود) و همینطور ددلاین آن گذشته بود به حالت overrun در میاییم. اگر هم این شرایط برقرار نبود به حالت normal میبریم سیستم را.

سپس تابع handle_done_tasks صدا زده میشود که برای مدیریت تسک‌های پایان یافته از قبل هست. در این تابع تسک‌هایی که در currently_assigned_tasks هستند را بررسی میکنیم و اگر نتیجه متد is_finished آن

تسک مثبت بود، job بعدی آن را میسازیم(این بدان معنی نیست که این جاب لزوما وارد پروسه میشود. بلکه صرفا مشخص میشود وقت انجام جاب بعدی هست و اگر به release time مانده باشد تابع is_active منفی برخواهد گرداند و استفاده نمیشود تا زمان release time).

همینطور اگر تسک از جنس بحرانیت بالا بود و ددلاین آن گذشته بود، سیستم به حالت panic mode میرود و کل شبیه‌سازی به اتمام میرسد.در غیر اینصورت اگر ددلاین یک تسک با بحرانیت پایین گذشته شده بود آن تسک نیز advanced_forward میشود تا جاب بعدی آن آماده شود.

در قدم بعدی تابع _disable_low_critical_tasks_in_overrun صدا زده میشود که در ادامه این تابع ابتدا بررسی میشود اگر در مود overrun بودیم همه تسک‌ها با بحرانیت پایین preempt میشوند.

همه تابع‌ها تا این مرحله برای بررسی صحت عملکرد سیستم میباشد. در ادامه تابع _scheduled_tasks تسک را صدا میزنیم. در آن ابتدا تابع _get_tasks_by_deadline_ordering را صدا میزنیم که در آن کل تسک‌های فعلی کور را به شرط is_active بودن به ترتیب ددلاینشان مرتب میکنیم. دقت کنید is_active بودن یعنی جاب فعال داشته باشد آن تسک. و ددلاین بر حسب اینکه mode چه باشد برای تسک‌های با بحرانیت بالا فرق خواهد کرد. در نهایت در صورتی که سیستم در مد نرمال باشد تسک‌های مرتب شده با بحرانیت کم و زیاد را باهم جمع کرده و اگر در مد overrun باشد صرفا تسک‌های با بحرانیت بالا را برای زمان‌بندی انتخاب میکنیم. در قدم بعدی یک لیستی از تسک‌های آن کور با شروط اعمال شده سیستم به دست می‌آیند. در این مرحله یک فیلتر روی کل تسک‌های موجود میزنیم و تسک‌هایی را که preemption_level آن‌ها از preemption_level سیستم بیشتر باشد(چون ما از period برای لول استفاده کردیم از عبارت کوچکتر در کد استفاده کردیم.) را انتخاب مینماییم. در قدم بعدی روی تسک‌های به دست‌آمده که به ترتیب از ددلاین کمتر به ددلاین بیشتر مرتب‌شده‌اند. در این مرحله اولین تسک این لیست را برمیداریم و به آن هسته اساین میکنیم.

در قدم آخر advanced_forward_tasksرا صدا میزنیم که تابع calculate تسک‌هارا صدا میزنم تا یک واحد محاسبه روی تسک‌هایی که در حال حاضر به هسته اساین شده‌اند شکل گیرد.

نتایج

در کل برای ارزیابی از ۶ سناریو مختلف استفاده کردیم:

1. تعداد وظایف: ۱۰۰، تعداد هسته ها: ۲، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۲۵
2. تعداد وظایف: ۴۰۰، تعداد هسته ها: ۲، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۵
3. تعداد وظایف: ۱۰۰، تعداد هسته ها: ۴، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۲۵
4. تعداد وظایف: ۱۰۰، تعداد هسته ها: ۴، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۵

5. تعداد وظایف: ۱۰۰، تعداد هسته ها: ۸، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۲۵
6. تعداد وظایف: ۱۰۰، تعداد هسته ها: ۸، نسبت وظایف با سطح بحرانیت بالا به وظایف با سطح بحرانیت پایین: برابر، بهره وری هسته:۰.۵