به نام خدا

درس:

سیستم‌های بی‌درنگ

موضوع:

گزارش پروژه

تاریخ:

۱۷ دی ۱۴۰۲

مشخصات افراد:

محمدعلی محمدخانی 98102251

احسان موفق 98106101

سوال اول

پیش از آن که به توضیحات فایل‌های پروژه و کدها بپردازیم، در مورد پروتکل SRP یک توضیح ارائه می‌دهیم:

Stack Resource Policy (SRP)

SRP یک پروتکل یا استراتژی استفاده شده در الگوریتم‌های زمانبندی برای مدیریت منابع در یک سیستم کامپیوتری به صورت کارآمد است. این پروتکل به طور خاص برای سیستم‌های بی‌درنگ طراحی شده است که وظایف یا فرآیندها نیازهای زمانی دقیق دارند.

در SRP، وظایف بر اساس اهمیت یا بحرانی بودنشان، یک اولویت می‌گیرند. وظایف در یک پشته یا یک صف اولویت‌ها سازماندهی می‌شوند، جایی که وظیفه با بالاترین اولویت اول اجرا می‌شود. هنگامی که یک وظیفه وارد می‌شود یا به یک منبع نیاز دارد، سیستم اولویت آن را با اولویت‌های وظایفی که در حال حاضر از منبع استفاده می‌کنند، مقایسه می‌کند.

این پروتکل اصول کلیدی زیر را دنبال می‌کند:

1. مدیریت اولویت: وظایف بر اساس ددلاین‌هایشان، اهمیت یا معیارهای دیگر اولویت‌بندی می‌شوند. وظایف با اولویت بالاتر اولویت بیشتری نسبت به وظایف با اولویت پایین‌تر دارند.

2. تخصیص منابع: هنگامی که یک وظیفه به یک منبع نیاز دارد، سیستم اطمینان حاصل می‌کند که وظیفه با بالاترین اولویت که درخواست منبع را دارد، دسترسی به منبع را داشته باشد. وظایف با اولویت پایین‌تر ممکن است تعلیق شوند یا به تعویق بیفتند اگر وظیفه با اولویت بالاتری به همان منبع نیاز داشته باشد.

3. مدیریت پشته:اولویت‌های وظایف در یک پشته یا یک صف اولویت‌ها مدیریت می‌شوند. هنگام ورود یا اتمام وظایف، پشته به‌روزرسانی می‌شود و زمان‌بند بر این اطمینان حاصل می‌شود که وظیفه با بالاترین اولویت موجود همیشه اجرا می‌شود.

4. مداخله: اگر وظیفه‌ای با اولویت بالاتر وارد شود یا به یک منبع نیاز داشته باشد که وظیفه با اولویت پایین‌تری در حال استفاده است، ممکن است وظیفه با اولویت پایین‌تر به‌طور موقت متوقف شود تا وظیفه با اولویت بالاتر اجرا شود.

5. رعایت ددلاین: SRP به هدف دارد که با اولویت دادن به وظایف با اولویت بالاتر، اطمینان حاصل کند که وظایف موفق به رعایت ددلاین‌های خود باشند و با مدیریت کارآمد منابع، این موضوع را تضمین کند.

با پیاده‌سازی SRP، سیستم‌ها می‌توانند وظایف حیاتی را به‌طور موثر مدیریت کنند و اطمینان حاصل کنند که آن‌ها از منابع مورد نیاز برخوردار شده و ددلاین‌های خود را رعایت می‌کنند. این پروتکل در سیستم‌های بی‌درنگ بسیار حیاتی است و نقض ددلاین‌ها می‌تواند منجر به شکست یا خطرات ایمنی شود.

توضیحات فایل er_edf.py که حاوی پیاده‌سازی الگوریتم ER-EDF است

فایل er_edf.py بخش کلیدی از پروژه است که الگوریتم زمان‌بندی ER-EDF یا همان EDF با منابع پیشرفته را برای سیستم‌های بی‌درنگ پیاده‌سازی می‌کند. این فایل شامل توابع زیر است:

resources_that_task_can_acquire(task, resources)

این تابع بررسی می‌کند که آیا یک وظیفه می‌تواند یک منبع را از منابع موجود تهیه کند یا خیر. این تابع بر روی منابع یک لوپ می‌زند و اولین منبعی را که وظیفه می‌تواند بر اساس واحدهای بخش بحرانی آن تهیه کند را برمی‌گرداند. اگر چنین منبعی یافت نشد، None برمی‌گرداند، به این معنا که هیچ منبعی نمی‌تواند به تسک اختصاص یابد. این تابع یک وظیفه و یک لیست منابع را به عنوان پارامترهای ورودی می‌گیرد.

er_edf_stack_resource(tasks, resources)

این تابع اصلی در فایل است که الگوریتم زمان‌بندی ER-EDF را با پروتکل SRP پیاده‌سازی می‌کند. این تابع یک لیست وظایف و منابع را به عنوان ورودی می‌گیرد و با ایجاد یک صف از

وظایف مرتب شده بر اساس ددلاین‌های آن‌ها شروع می‌شود. سپس وارد حلقه‌ای می‌شود که تا زمانی که تمام وظایف پردازش نشده‌اند ادامه دارد.

در هر تکرار از حلقه، تابع:

- وظیفه با نزدیک‌ترین ددلاین را بازیابی می‌کند و بررسی می‌کند که آیا وظیفه نیاز به یک منبع دارد و آیا منبع در دسترس است یا خیر. اگر منبع در دسترس نباشد، بررسی می‌کند که آیا هر وظیفه‌ای که آن منبع را نگه می‌دارد تمام شده است یا نه. اگر شده بود، منبع را به صف برمی‌گرداند. اگر منبع در دسترس باشد، آن را تهیه و قبضه می‌کند. پس از تخصیص منبع، وظیفه را به عنوان در حال اجرا علامت‌گذاری کرده و زمان اجرای وظیفه را به‌روزرسانی می‌کند.
- سپس، چک می‌کند که آیا وظیفه به زمان اجرای کوتاه خود رسیده است و اگر وظیفه با اهمیت بالا باشد، به زمان اجرای بلند switch می‌کند.
- در ادامه، بررسی می‌کند که آیا وظیفه به ددلاین خود رسیده است یا خیر. اگر رسیده باشد، وظیفه را اگر هنوز آزاد نشده باشد آزاد می‌کند و چک می‌کند که آیا وظایفی با ددلاین‌های دیرتر قابل برنامه‌ریزی هستند یا خیر.
- چک می‌کند که آیا وظیفه به تمامی منابعی که نیاز دارد دسترسی دارد یا خیر. اگر وظیفه، دسترسی به تمام منابع را دارد، از آن‌ها استفاده می‌کند. اما اگر وظیفه به تمامی منابع دسترسی نداشته باشد، آن را رها می‌کند تا بعدا برای بار دیگر تلاش کند.
- بررسی می‌کند که آیا وظیفه اجرای خود را کامل کرده است یا خیر. اگر کامل شده باشد، تمام منابع نگهداری شده توسط وظیفه را آزاد می‌کند تا بقیه بتوانند استفاده کنند.
- زمان را به‌روزرسانی می‌کند.

این تابع تمام رویدادهای کلیدی را در حین اجرای خود ثبت می‌کند، مانند پردازش وظایف، قبضه کردن و آزادسازی منابع، تعویض به زمان اجرای بلند، رسیدن به ددلاین‌ها و اتمام اجراها.

این تابع لیست وظایف را پس از پردازش تمامی آن‌ها برمی‌گرداند.

کلاس Task

کلاس وظیفه یک وظیفه در سیستم را نمایندگی می‌کند. هر وظیفه دارای نام، ددلاین، زمان اجرا، منبع، سطح بحرانیت و ویژگی‌های دیگری است. این کلاس در فایل task.py تعریف شده است.

کلاس Resource

کلاس منبع یک منبع در سیستم را نمایندگی می‌کند. هر منبع دارای تعداد کل واحدها و واحدهای تخصیص یافته است. این کلاس در فایل resource.py تعریف شده است.

تابع uunifast

تابع uunifast برای تولید مجموعه‌ای از وظایف با استفاده از یک استفاده کلی مشخص استفاده می‌شود. این تابع در فایل uunifast.py تعریف شده است. این تابع تعداد وظایف، استفاده کلی، سطوح بحرانیت و دوره را به عنوان پارامترهای ورودی می‌گیرد. یک لیست اشیاء Task را برمی‌گرداند. در ادامه، مراحل الگوریتم را مختصرا توضیح می‌دهیم:

ورودی‌ها:

- تعداد وظایف (n)
- مجموع utilization مورد نظر (U)

مقداردهی اولیه:

- تولید n-1 عدد تصادفی به صورت یکنواخت در بازه 0 تا U.
- مرتب کردن این اعداد به ترتیب صعودی و ساختن آرایه U_sorted.

محاسبه utilization:

محاسبه استفاده برای هر وظیفه (u_i) به عنوان تفاوت بین عناصر متوالی در U_sorted:

```
u_1 = U_sorted[0]
u_2 = U_sorted[1] - U_sorted[0]
...
```

u_n = U - U_sorted[n-2]

خروجی:
یک مجموعه از n تا وظیفه utilize شده (u_1، u_2، ...، u_n) با utilization کلی U.

کلاس Runner

کلاس Runner برای اجرای شبیه‌سازی استفاده می‌شود. این کلاس در فایل main.py تعریف شده است. این کلاس یک متد run دارد که یک مجموعه وظایف با استفاده از تابع uunifast تولید کرده و سپس شبیه‌سازی را با استفاده از تابع er_edf_stack_resource اجرا می‌کند.

ثبت رویدادها (Logging)

این پروژه از ماژول logging داخلی Python برای ثبت اجرای الگوریتم زمان‌بندی استفاده می‌کند. پیام‌های لاگ اطلاعاتی درباره پردازش وظایف، تخصیص منابع و سایر رویدادهای کلیدی فراهم می‌کنند. لاگر در فایل logger.py پیکربندی شده است.

در فایل‌های جانبی این پروژه که به همراه کد آن فرستاده شده است، لاگ‌های دو مورد از تست‌کیس‌های داده شده در دستور کار پروژه قرار داده شده است و تمامی گزارشات اجرای الگوریتم در آنها قرار دارد.