به نام خدا

گزارش کار پروژه سیستم های بی درنگ

امید آزادی

۹۸۱۰۹۶۶۷

عطا رحیم زاده

۹۸۱۷۰۸۰۵

فاز یک:

در فاز نخست پروژه سامانه های بی‌درنگ به نوشتن و تولید تست ها پرداختیم. برای تولید قسمت Utilization هر وظیفه، از الگوریتم UUniFast استفاده کردیم. الگوریتم UUniFast را Bini برای حل مسئله‌ی تولید وظیفه‌های مصنوعی منصفانه و مستقل برای زمانبندی روی تک‌پردازنده مطرح کرده‌است.

قطعه کد زیر نحوه پیاده سازی الگوریتم UUniFast را نشان می دهد:

```python
import random

def generate_uunifastdiscard(nsets: int, u: float, n: int):
    sets = []
    while len(sets) < nsets:
        utilizations = []
        sumU = u
        for i in range(1, n):
            nextSumU = sumU * random.random() ** (1.0 / (n - i))
            utilizations.append(sumU - nextSumU)
            sumU = nextSumU
        utilizations.append(sumU)

        if all(ut <= 1 for ut in utilizations):
            sets.append(utilizations)

    return sets
```

برای تولید Period های هر وظیفه از توزیع نرمال گاوسی بهره گرفتیم، به دلیل اینکه هم وظیفه‌هایی با Period های خیلی کم و خیلی زیاد داشته باشیم، و هم بیشتری وظیفه‌ها Period ای در حدود میانگین داشته باشند.

چالش اصلی در پیاده‌سازی یک تست، دسترسی‌هایش به منابع بود. از آنجایی که باید آن دسترسی سریال و هم دسترسی موازی پشتیبانی شود، کد تولید تست ما اینگونه عمل می‌کند:

- برای یک تست ابتدا همه‌ی دسترسی‌هایی که باید انجام بدهد را می‌سازد.
- دسترسی‌هایش به را به گروه‌های سریال قسمت می‌کند.
- در هر گروه سریال، دسترسی‌ها را تودرتو می‌چیند.

فاز دو :

این فاز بخش نهایی پروژه است. از نظر معنایی، کد و کارهای مربوط به این بخش به این دسته‌بندی زیر قابل تقسیم‌اند:

۱. بخش EDF-VD: دستگاه پمپ دارو در خون نیاز دارد تا طیف بزرگی از وظیفه‌ها را در سخت‌افزاری کوچک بگنجاند، که این طیف کارها هم کارهای حیاتی را شامل می‌شوند که انجامشان به مرگ و زندگی وابسته است (پمپ سر زمان مناسب)، و هم کارهای کم‌اهمیت‌تر دیگر را شامل می‌شود (گزارش اطلاعات به بیرون). بدین منظور سیستم از یک طراحی Multi-Criticality (به اختصار MC) استفاده می‌کند و راه‌حل پروژه‌ی ما از الگوریتم شناخته‌شده‌ی EDF-VD برای حل سیستم‌های MC بهره می‌برد.

الگوریتم EDF-VD به‌صورت مجازی ددلاین‌های وظیفه‌های HC (با درجه‌ی اهمیت بالا) را کم می‌کند، تا در زمانبندی EDF زودتر اجرا شوند. یک عدد ثابت x کمتر از یک محاسبه می‌شود تا در ددلاین‌های عادی وظیفه‌های HC ضرب شود، و الگوریتم EDF-VD با اینکار زمانبندی درست همه‌ی وظیفه‌ها را تضمین می‌کند.

Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on a unit-speed preemptive processor.

1) Compute x as follows:

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

2) If $\left(x\, U_{\mathrm{LO}}^{\mathrm{LO}}(\tau) + U_{\mathrm{HI}}^{\mathrm{HI}}(\tau) \leq 1\right)$ **then**

$\hat{T}_i \leftarrow x\, T_i$ for each HI-criticality task τ_i
declare success and **return**

else declare failure and **return**

Figure 1. EDF-VD: The preprocessing phase.

در سیستم‌های MC، وظیفه‌های HC دارای دو WCET متفاوت هستند، که یکی بزرگ‌تر و بدبینانه‌تر است. اگر اجرای یک وظیفه‌ی HC در مدت WCET کمترش تمام نشود، کل سیستم وارد حالت overrun می‌شود و دیگر تمام وظیفه‌های LC را دور می‌ریزد و فقط HC اجرا می‌کند. ما در تست‌هایمان حالت overrun هم در نظر گرفته‌ایم.

۲. بخش محاسبه‌ی Deadline Floor: زمانبند از الگوریتم DFP برای کار با منابع مشترک بهره می‌برد. الگوریتم DFP یک فاز Pre-Proccessing دارد که در آن برای یک منبع مانند x، و هر مقدار ممکن از ۰ تا تعداد کل مقدار موجود منبع، مانند y، عدد DF(x,y) محاسبه می‌شود. عدد DF(x,y) برابر مینیمم ددلاین نسبی وظیفه‌ای است که به مقدار بیشتری از y عدد از منبع x نیاز دارد و در صورت وجود y عدد از منبع x، بلاک می‌شود. این مقدار با یک لوپ ساده روی وظیفه‌ها محاسبه می‌شود.

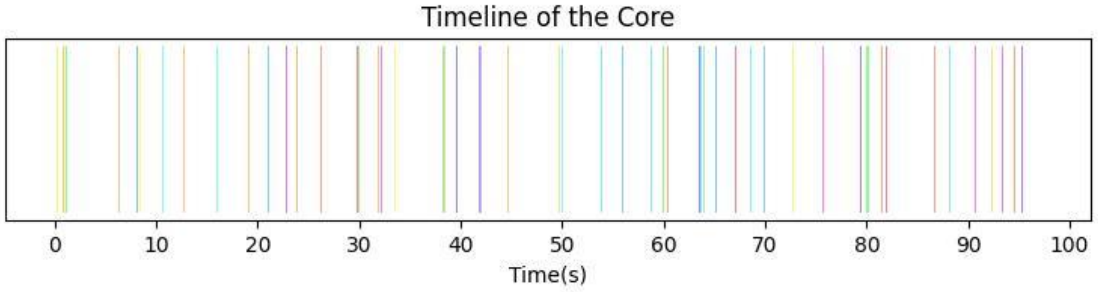
۳. بخش اجرای الگوریتم: به اجرای اصلی می‌رسیم. همه‌ی کارها را از روی وظیفه‌ها استنتاج می‌کنیم و مرتب می‌کنیم. به ترتیب زمان را جلو می‌بریم. در مرحله یکی از این چند اتفاق می‌تواند بیافتد:

- یک کار جدید ظاهر شود.
- کار در حال اجرا وارد بخش بحرانی شود.
- کار در حال اجرا از بخش بحرانی خارج شود.
- کار در حال اجرا تمام شود.

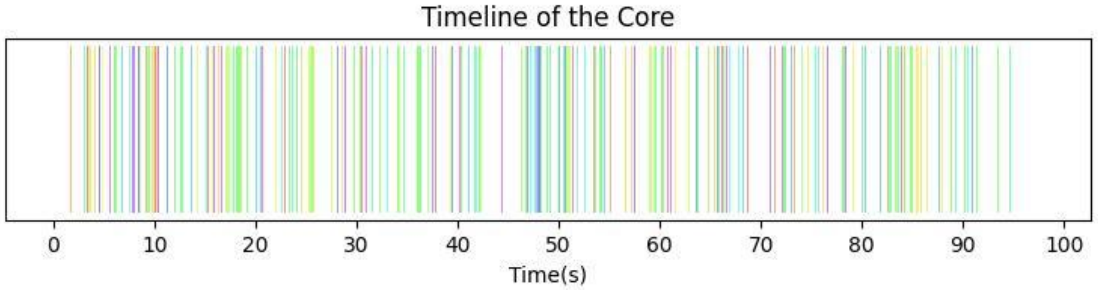
وقتی کار وارد یک بخش بحرانی می‌شود، مطابق پروتکل DFP، ددلاین نسبی پویای آن می‌تواند کمتر شود.

۴. نمودارها: ما در از کار دو نوع نمودار استفاده کرده‌ایم: نمودار زمانبندپذیری و نمودار کیفیت خدمات. همچنین بخشی از تست‌ها را به حالت overrun در آورده‌ایم که در آن، وقتی سیستم از زمان نصف می‌گذرد، وارد بخش overrun می‌شود. بخشی از نمودارهای ساخته‌شده را در زیر ضمیمه‌ی گزارش کار می‌کنیم. نمودارها در منبع کد پروژه در سایت Github قابل دسترسی‌اند.

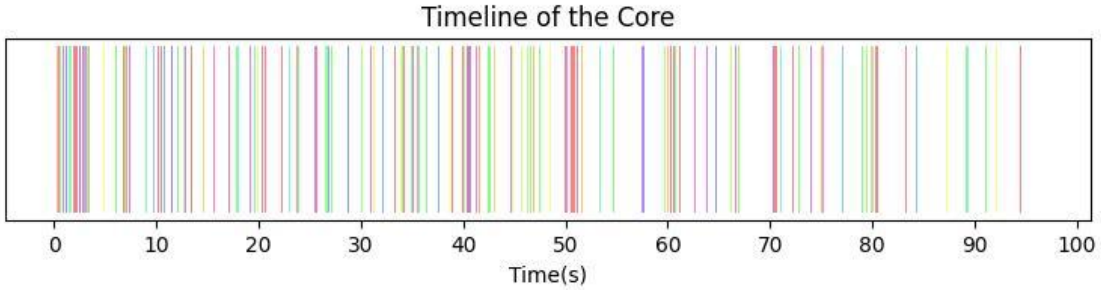
- نمودار زمانبندپذیری در Utilization=0.3:



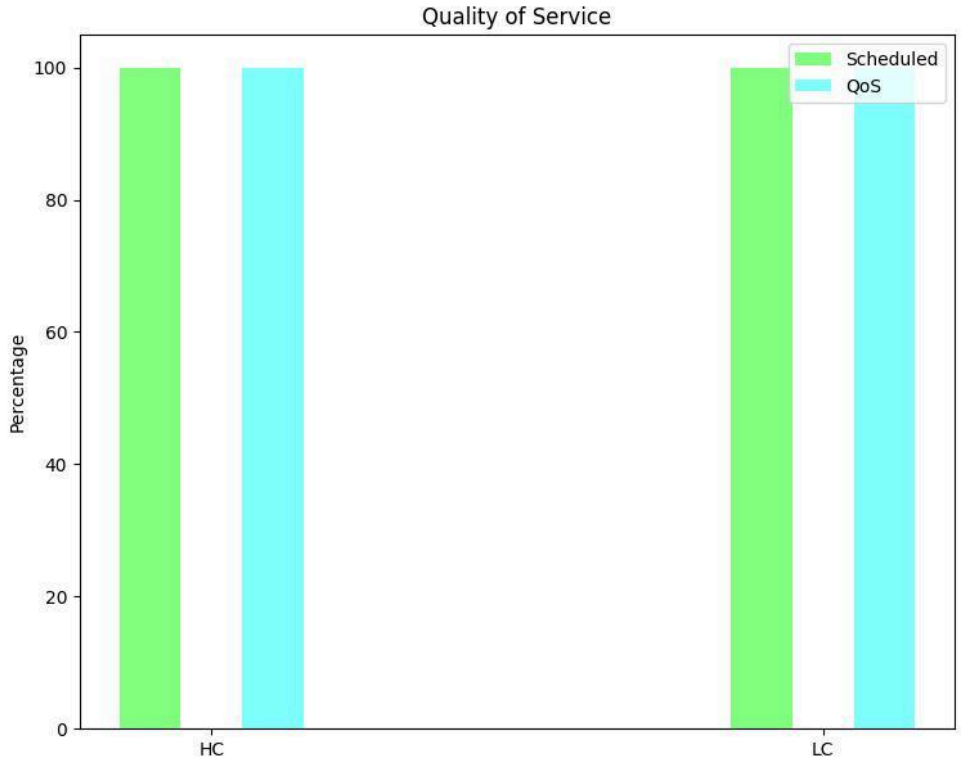
- نمودار زمانبندپذیری در Utilization=0.9:



- نمودار زمانبندپذیری در Utilization=0.5 و overrun:



- نمودار کیفیت خدمات در Utilization=0.9:



- نمودار کیفیت خدمات در Utilization=0.9 و overrun: