سوالات تئوری:

1-الف)

Firmware: برنامه ای که در حالت عادی تغییر نمیکند و در هاردور لحاظ میشود. این برنامه به شناسایی و اولویت دهی (برای مثال روشن کردن سیستم) و ... اجزای مختلف سیستم میپردازد.

Software: برنامه ای که برای یک سیستم نهفته کار اصلی را مشخص میکند و توسط برنامه‌نویس نوشته و تغییر میابد.

Memory: حافظه های مختلف مانند رم و رجیستر ها که در لایه های گوناگون اعم از ذخیره سیگنال ها، ذخیره موقت برای پردازش های بعدی و همچنین ذخیره دستور ها.

UI: شامل LEDها و GUI های متصل به LCD که با کاربر تعامل دارد.

Sensor: جهت دریافت ورودی از محیط اطراف استفاده میگردد.

Actuator: اغلب موتور ها هستند و به جهت واکنش به محیط بیرون با توجه ب داده ها و پردازش ها استفاده میشود.

Emulation and Diagnosis: رابط های مختلفی مانند JTAG که به صورت استاندارد از پیش برنامه ریزی شده داده هایی را از وضعیت عملکرد و وضعیت برنامه در حال اجرا بر روی سیستم میدهد. این رابط ها باید در درون متن Software باید تعبیه و از پیش برنامه ریزی گردد.

Analog I/O: A2D یا D2A یا فیلتر ها و تثویت کننده هایی که ممکن است به صورت مجتمع داخل IC یا به صورت یک IPدر اختیار ما قرار بگیرد.

Application Specific Gate: پورت های Serial, I2c, SPI و ... که برای مصارفی که جلو تر بررسی میشوند، با پین هایی از پیش مشخص شده د اختیار ما قرار میگیرند.

Processor: این بخش شامل واحدهای پردازش مختلفی است که اطالعات ورودی را پردازش میکنند و تصمیمات الزم را اتخاذ میکنند. این واحدها ممکن است شامل میکروکنترلرها، پردازنده های دیجیتال، FPGAو غیره باشند.

1-ب)

مدیریت حافظه: سیستم تا جای ممکن بدون پایمال کردن deadline ها از کمترین منابع استفاده کند. همچنین ضروری است که در job های مختلف به صورت مدیریت شده ای از منابع مشترک استفاده گردد تا داده های هر تسک که در اثر سوییچ کردن بین تسک ها از بین نرود.

مصرف انرژی: با تجه به اینکه ممکن است سیستم های نهفته در ماک نهایی با منابع انرژی محدود (باتری) استفاده شوند از این رو لازم است مضرف انرژی به مقدار خوبی بهینه باشد.

قابلیت اطمینان: هندل کردن Worst Caseهای سیستم در کنار Corner Case ها از جمله موارد افزایش قابلیت اطمینان سیستم میباشد و همچنین هندل کردن \الت های بحرانی ولتاژ به جهت حفظ داده ها و عملکرد صحیح سیستم ضروری است.

1-ج)

برای ارتباط سیستم ها با یکدیگر و یا دستگاه های دیگر و همچنین ماژول ها و IP ها آماده، از پروتکل های ارتباطی استفاده میگردد. که میتوان به I2C, SPI, UART, CAN یاد کرد.

I2C: ارتباط حسگر ها یا دو سیستم در درون یک IC با این پروتکل انجام میگیرد.

SPI: باری اتصال سریع میکرو ها استفاده میگردد. پهنای باند انتقال به کلاک مشترک وابسته است. این اتصال به صورت دو طرفه انجام میگردد و رجیستر خاصی به این پروتکل اختصاص داده میشود.

UART: برای اتباط میکرو ها با سیستم های کامپیوتر قابل استفاده است که بین دو دستگاه استفاده کننده باید پورت سریال ساپورت شود. IC هایی برای سهولت استفاده از این پروتکل وجود دارد.

CAN: یک پروتکل صنعتی است که برای ارتباط بین دستگاه های صنعتی استفاده میگردد. از این پروتکل در شبکه داخلی خودرو ها نیز استفاده میگردد

1-د)

تهدیدات امنیتی رایجی که سیستم های نهفته با آن مواجه می شوند، شامل دسترسی غیرمجاز، تغییر داده (data tampering) و حملات DoS (Service of Denial) می‌باشند. این تهدیدات در سیستم‌های نهفته می‌توانند منجر به وقوع حوادث جدیدی شوند. برخی از نمونه‌های واقعی از این تهدیدات که در سیستم‌های نهفته به کار رفته‌اند عبارتند از:

1. دسترسی غیرمجاز (Unauthorized Access):

- در این نوع حمله، اشخاص غیرمجاز به سیستم یا داده‌های مهم دسترسی پیدا می‌کنند و ممکن است از این دسترسی برای اهداف خود سوءاستفاده کنند.

2. تغییر داده (Data Tampering):

- حملات تغییر داده شامل دستکاری غیرمجاز اطلاعات می‌شوند که می‌تواند به تخریب یا تغییر نتایج مورد انتظار سیستم منجر شود.

3. حملات DoS (Service of Denial):

- در این نوع حمله، حمله‌کننده سعی دارد با اشغال منابع سیستم، سرویس مورد نظر را برای کاربران معمولی غیرقابل دسترسی کند.

این تهدیدات امنیتی رایج در سیستم‌های نهفته می‌توانند منجر به وقوع حوادث جدی شوند و نیازمند برنامه‌ها و راهکارهای امنیتی مناسب برای پیشگیری از آنها هستند.

1. حمله Stuxnet:

- در سال 2010، ویروس Stuxnet یک حمله سایبر جهانی راه‌اندازی کرد که هدف آن تجهیزات هسته‌ای ایران بود.

- این حمله توانست سیستم‌های نهفته مربوط به کنترل واحد پردازش آن تجهیزات را مورد دستکاری قرار داده و تاثیر بسزایی بر فعالیت آنها داشته باشد.

2. حملات به خودروهای هوشمند:

- دسترسی غیرمجاز به سیستم‌های نهفته خودروهای هوشمند می‌تواند به دسترسی به اطلاعات حساس، کنترل‌کننده‌های مهم و حتی کنترل خودرو منجر شود.

- حوادثی چون حملات به سیستم‌های رانندگی خودروهای تجاری و تست‌های امنیتی نشان دهنده شکست امنیت در این حوزه می‌باشد.

3. حملات DoS بر علیه تجهیزات شبکه‌ای:

- حملات سرویس نقض منابع سرورها و تجهیزات شبکه‌ای می‌تواند به تخریب کارایی سیستم‌های نهفته منجر شود.

- حملات DoS بر روی تجهیزات اینترنت اشیاء و سیستم‌های هوشمند می‌توانند نقض سیستم‌ها را به خطر اندازند.

این نمونه‌های واقعی از حوادث نشان می‌دهند که تهدیدات امنیتی مطرح شده، چقدر می‌توانند سیستم‌های نهفته را تحت تاثیر قرار دهند و نیازمندی به راهکارهای امنیتی قوی برای پیشگیری از آنها هستند.

1-ه) رای مقایسه کاربردها و عملکرد پردازنده‌های مختلف در سیستم‌های نهفته، می‌توانید به موارد زیر توجه کنید:

کاربردها:

Arm: پردازنده‌های ARM به عنوان یکی از محبوب‌ترین پردازنده‌های نهفته، در دستگاه‌های مختلف از تلفن‌همراه تا دستگاه‌های خانگی به کار می‌روند. آنها برای کاربردهایی مانند اینترنت اشیاء(IoT) ، دستگاه‌های پزشکی، خودروها، و دستگاه‌های صنعتی مناسب هستند.

MIPS: پردازنده‌های MIPS نیز در سیستم‌های نهفته مورد استفاده قرار می‌گیرند. آنها در دستگاه‌های شبکه، دستگاه‌های ذخیره‌سازی، و دستگاه‌های صنعتی کاربرد دارند.

Intel Quark: این پردازنده‌های کوچک و انرژی‌کم توسط Intel برای سیستم‌های نهفته طراحی شده‌اند. آنها در دستگاه‌های اینترنت اشیاء (IoT) و دستگاه‌های پزشکی مورد استفاده قرار می‌گیرند.

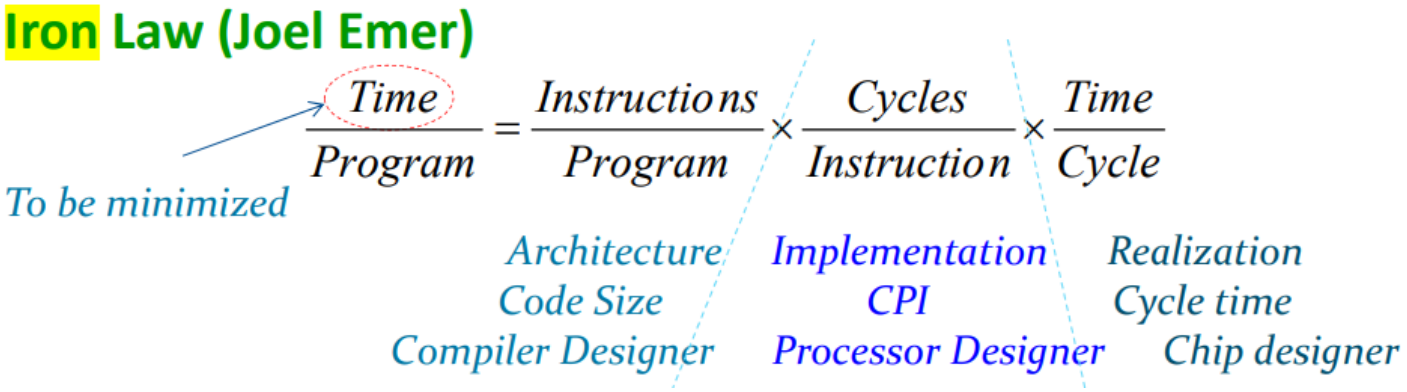
عملکرد:

ARM: پردازنده‌های ARM به دلیل معماری پیچیده و کارایی بالا، در کاربردهایی که نیاز به مصرف انرژی کم و عملکرد قوی دارند، عالی عمل می‌کنند.

MIPS: پردازنده‌های MIPS نیز در کاربردهایی که نیاز به پردازش داده‌های عددی و عملیات محاسباتی دارند، مؤثر هستند.

Intel Quark: این پردازنده‌ها به دلیل اندازه کوچک و مصرف انرژی پایین، برای دستگاه‌های اینترنت اشیاء (IoT) و دستگاه‌های پزشکی مناسب هستند.

با توجه به این مقایسه، هر پردازنده ویژگی‌ها و کاربردهای خاص خود را دارد و انتخاب مناسب بستگی به نیاز های خاص پروژه دارد.



بر این اساس در طراحی پروسسور ها، میتوان $\frac{\text{Cycles}}{\text{Instruction}}$ را کاهش داد که میتوان با تمرکز بر روی استفاده خاص در طراحی پردازشگر ها این اتفاق را رقم زد که در قسمت قبل بررسی شد.

2)

ابتدا برای هر حالت ارائه شده CPI را محاسبه میکنیم:

حالت اول:

$$\overline{\text{CPI}} = 45\% \times 3 + 25\% \times 5 + 30\% \times 2 = 3.2$$

حالت دوم:

$$\overline{\text{CPI}} = 70\% \times 3 + 18\% \times 1 + 12\% \times 6 = 3$$

اکنون تعداد کلاک هر برنامه را حساب میکنیم:

$$\text{Clock} = \text{Intruction} \times \text{CPI}$$

و زمانی که در هر پردازنده طول میکشد برابر $\frac{\text{Clock}}{\text{Procerssor Frequency}}$ خواهد شد بنابر این:

	پردازنده A با برنامه A	پردازنده B با برنامه B
حالت اول	0.016	0.016
حالت دوم	0.015	0.015

واحد های زمانی ثانیه هستند.

و برای قسمت دوم سوال داریم

$$\frac{1}{CPI} \times Processor\ Frequency \times 10^{-6}$$

که برابر

	پردازنده A	پردازنده B
حالت اول	625	375
حالت دوم	666.67	400

خواهد شد. (MIPS)

گزارش بخش عملی:

سوال اول: (Func.py_)

در این بخش برای تمام تست ها از چنین مقادیری استفاده شده است:

$$T = [5, 9, 20]\ C = [2, 2, 5]\ D = [3, 6, 5]\ ap\_task\_time = 3, ap\_task\_jobs = 8$$

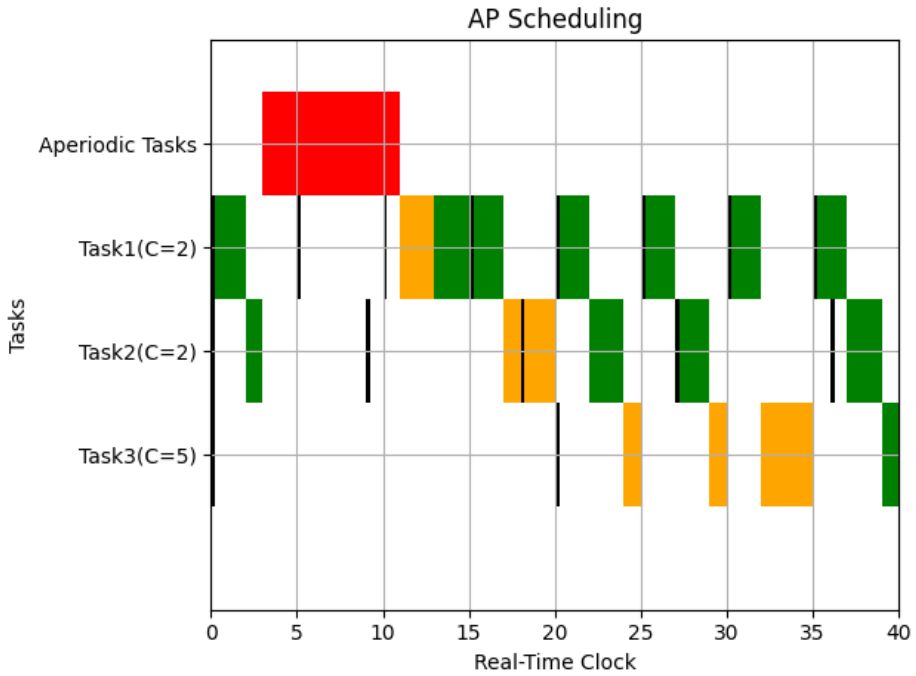
الگوریتم Rate Monotic with Aperiodic Job Handler:

در این الگوریتم اولویت با جابی است که در تناوب های سریع تری تکرار میشود. همچنین تسک غیر متناوب به صورت یک اینتراپت وارد کار شده و مهم ترین اولویت را دارد. در این الگوریتم ددلاین ها زمان رسیدن تناوب بعدی میباشد.

```
def rm_scheduler(examples, ap_task_time, ap_task_jobs,time_limit = 40):
```

در این تابع در بخش ورودی خروجی های تابع Reader را وارد کنید. همچنین این تابع به جهت در نظر گرفتن job های aperiodic نوشته شده که با وارد کردن زمان شروع و طول job آن را لحاظ میکند. (Fixed priority)

به ازای ورودی های پیشفرض خروجی به شکل زیر خواهد بود:



قرمز: تسک aperiodic سبز: تسک های انجام شده پیش از فرا رسیدن ددلاین زردها: تسک های انجام شده پس از ددلاین

خطوط سیاه: تناوب جاب ها

الگوریتم Earliest Deadline First:

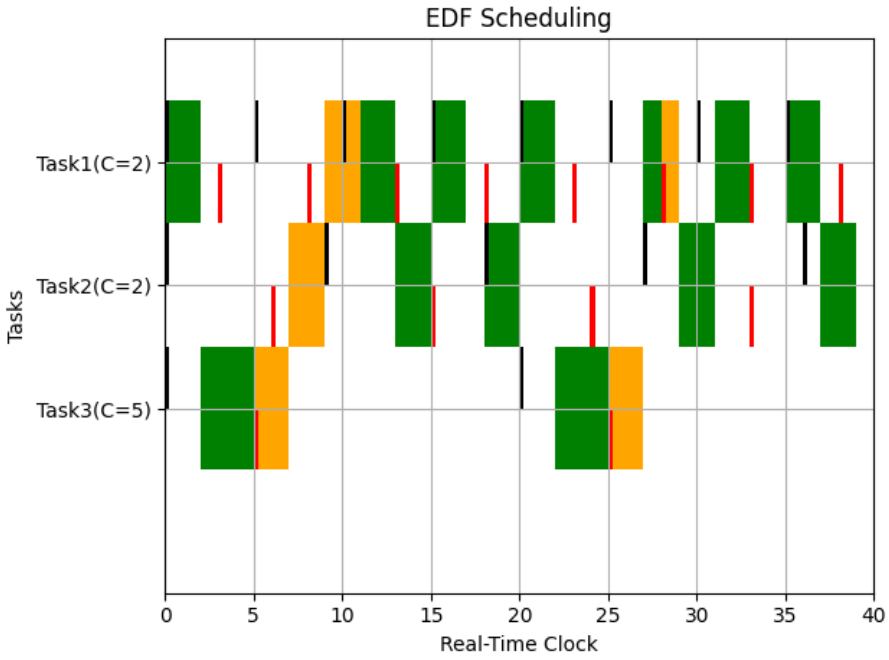
در این الگوریتم از تسک هایی که تا الان ایجاد شده آن تسکی که نزدیک ترین ددلاین را دارد انتخاب شده و پردازش های مربوط به آن انجام میشود. ددلاین های این الگوریتم برخلاف الگوریتم پیشین لزوم ندارد که حتما تناوب باشد یا با آن هماهنگی داشته باشد. در نظر داشته باشید که میتوان ثابت

کرد این الگوریتم بهترین الگوریتم ممکن برای انجام جاب ها در یک سیستم نهفته میباشد اما بدلایلی مانند نیاز به پردازش در هر زمان و اشغال کردن واحد های زمانی از آن استفاده نمیگردد.

```
def ed_scheduler(examples, time_limit = 40):
```

در این تابع صرفا نیاز هست که خروجی های تابع Reader به آن وارد شود.

به ازای خروجی های پیشفرض ورودی به شکل زیر خواهد بود:



سبز: تسک های انجام شده پیش از فرا رسیدن ددلاین زردها: تسک های انجام شده پس از ددلاین خطوط سیاه: تناوب جاب ها خط قرمز: ددلاین جاب ها

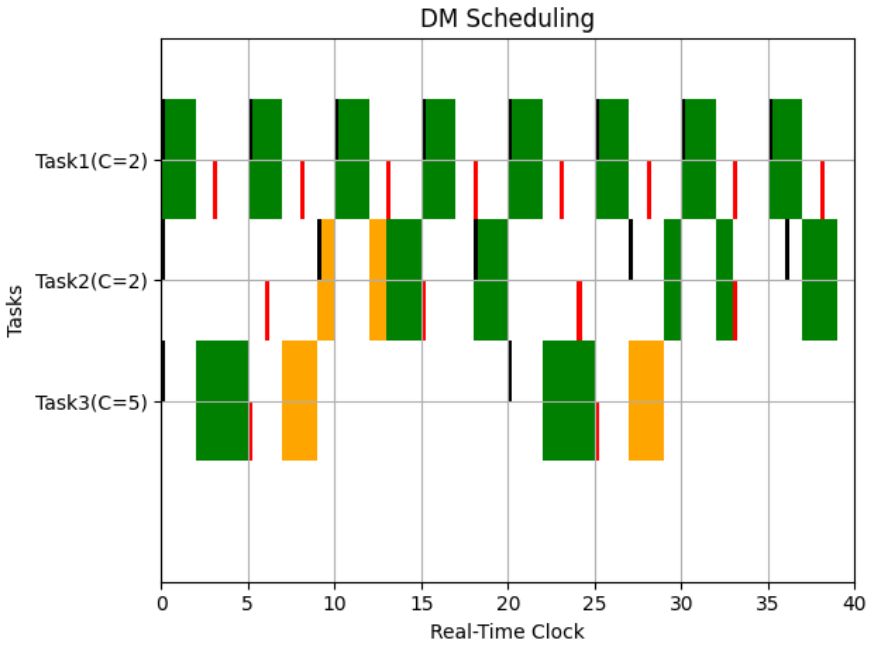
الگوریتم Deadline Monotic:

در این الگوریتم اولویت ها براساس ددلاین ها میباشد. هر تسکی که نسبت به لحظه ایجاد خود ددلاین نزدیک تری داشته باشد از اولیت بالاتری برخودار خواهد شد. در نظر داشته باشید که تفاوت این الگوریتم با Earliest Deadline First آن است که در هر مرحله نیازی به بررسی اولویت ها ندارد صرفا در لحظه ایجاد تسک وضعیت انجام آن مشخص میشود. (Fixed priority)

```
def dm_scheduler(examples, time_limit = 40):
```

در این تابع صرفا نیاز هست که خروجی های تابع Reader به آن وارد شود.

به ازای ورودی های پیشفرض خروجی تابع به صورت زیر خواهد بود:



سبز: تسک های انجام شده پیش از فرا رسیدن ددلاین زردها: تسک های انجام شده پس از ددلاین خطوط سیاه: تناوب جاب ها خط
خطوط قرمز: ددلاین جاب ها

سوال دوم: (Func2.py_)

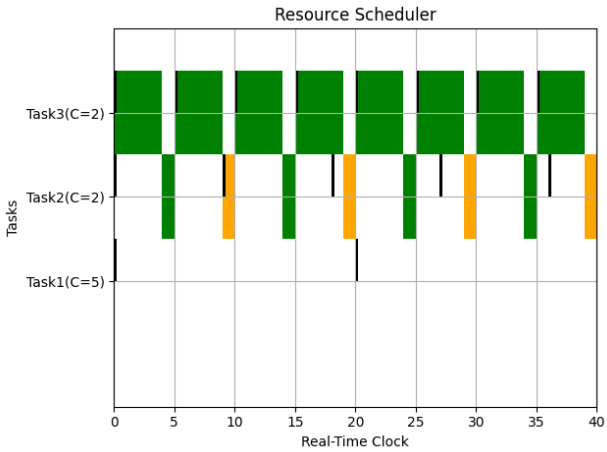
در این بخش برای تمام تست ها از چنین مقادیری استفاده شده است:

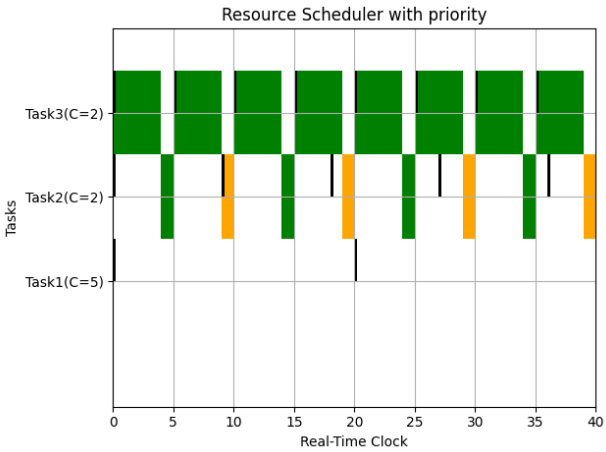
$$T = [5, 9, 20]\ C = [2, 2, 5]\ D = [3, 6, 5]\ R = [2, 0, 4]$$

از یک الگوریتم که قابلیت لاک کردن و بررسی ریسورس ها را داشته باشد و توزیع ریسورس ها را به شکلی انجام دهد که کمترین پر و خالی کردن را شاهد باشیم.

```
def rm_scheduler(examples, R,time_limit = 40):
def rm_schedulerp(examples, R,time_limit = 40):
```

در این دو تابع که تابع اول بدون priority inheritance است و تابع دوم با این ویژگی است. ورودی های این توابع همانند بخش قبل خروجی های تابع Reader و R مدت زمان استفاده از ریسورس ها میباشد. که خروجی های آن ها به ترتیب به شکل زیر خواهد بود:





سبز: تسک های انجام شده پیش از فرا رسیدن ددلاین زردها: تسک های انجام شده پس از ددلاین خطوط سیاه: تناوب جاب ها خط