مستندات پروژه درس سیستم‌های نهفته و بی‌درنگ

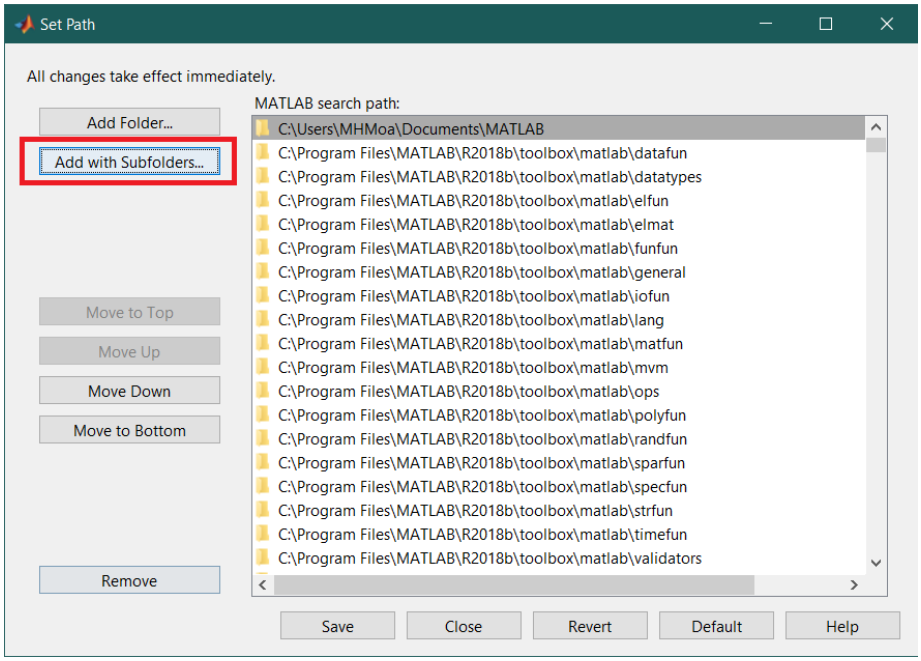
استاد امین عنایت زارع

اعضای گروه:
محمدحسین موذن‌نیا
زینب واعظزاده
صادق همدانی‌پور

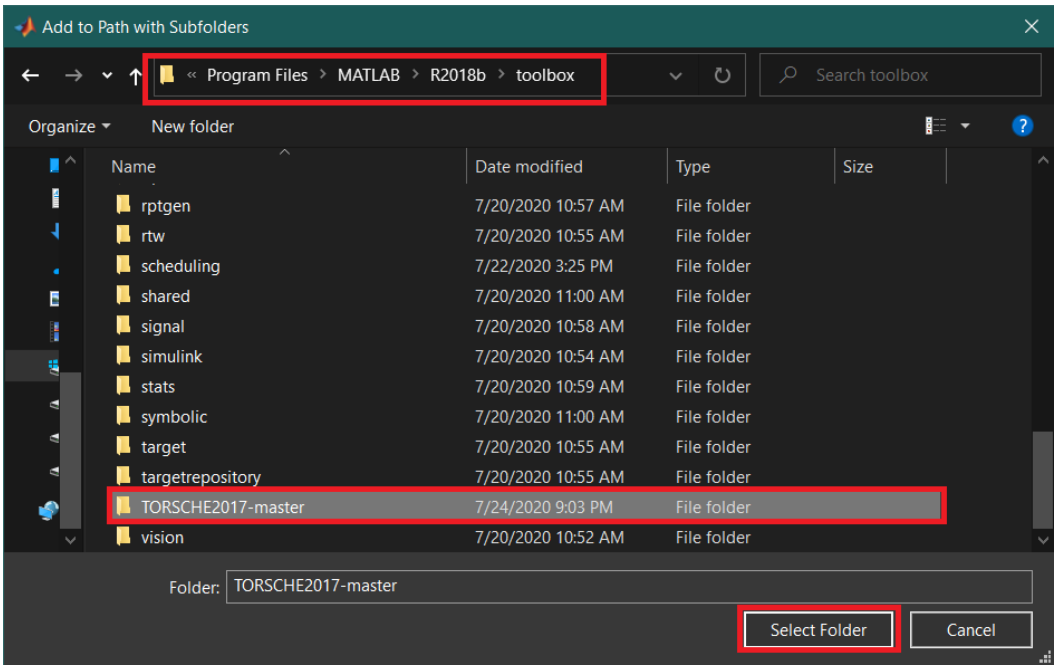
شرح پروژه

قبل از اجرای کد کتابخانه torsche را باید به متلب اضافه کنیم. برای انجام اینکار ابتدا فایل کتابخانه را از صفحه گیت‌هاب این پروژه به آدرس https://github.com/CTU-IIG/TORSCHE2017 دانلود کرده، سپس پوشه TORSCHE2017 موجود در فایل فشرده دریافت شده را به پوشه toolbox محل نصب متلب انتقال می‌دهیم.

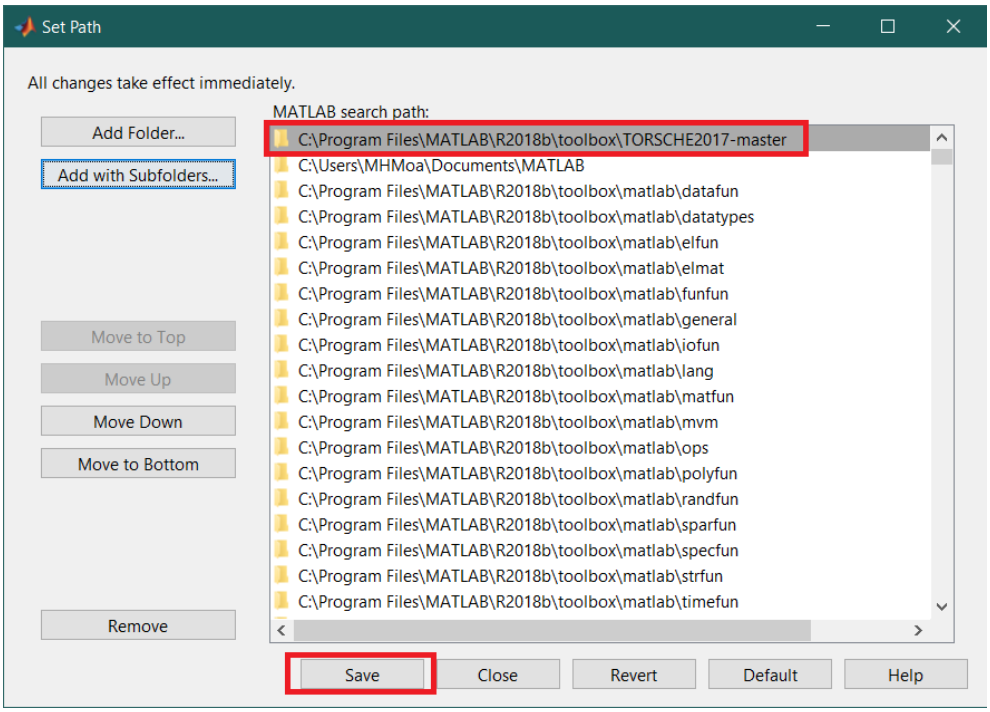
برای اضافه کردن این کتابخانه باید قبل از اجرای اسکریپت آن را از طریق گزینه set path موجود در تب Home به آن اضافه کنیم. برای اینکار طبق تصاویر زیر پیش می‌رویم.



بعد از انتخاب گزینه Add with Subfolders باید به محلی که پوشه کتابخانه را در آن قرار دادیم رفته و آن را انتخاب کنیم.



بعد از انتخاب پوشه حاوی کتابخانه می‌بینیم که کتابخانه با موفقیت اضافه شده است. برای ذخیره گزینه save را انتخاب کرده و پنجره را می‌بندیم.



شرح کد

کد نوشته شده از دو تابع که هر کدام در یک فایل جداگانه قرار دارند و یک فایل دیگر که با استفاده از این دو تابع زمان‌بندی RM و EDF را روی یک مجموعه وظیفه اجرا می‌کند، تشکیل شده است.

ابتدا کدی که با استفاده از توابع الگوریتم ها را اجرا می‌کند، شرح می‌دهیم.

شرح کد اصلی

در ادامه کد نوشته شده در فایل project.m شرح می‌دهیم. این کد با استفاده از توابع RM و EDF الگوریتم ها را اجرا می‌کند.

```
1 -    clear
2 -    clc
```

ابتدا با دو دستور بالا متغیرهای قبلی تعریف شده در محیط کار را حذف کرده و صفحه را پاکسازی می کنیم. دلیل انجام اینکار جلوگیری از تداخل متغیرهای کد فعلی با متغیرهای پیشین است.

```
4 -     t1 = torsche.ptask('t1', 3,20,0,7);
5 -     t2 = torsche.ptask('t2', 2,5,0,4);
6 -     t3 = torsche.ptask('t3', 1,10,0,8);
```

در سه خط بعدی با استفاده از تابع ptask سه وظیفه دوره‌ای موردنیاز را تعریف می‌کنیم. ورودی های این تابع به ترتیب شامل موارد زیر است.

- نام وظیفه
- زمان اجرا
- دوره تناوب
- زمان ورود
- مهلت تکمیل

```
8 -     TS1 = torsche.taskset([t1 t2 t3]);
```

در خط بالا از تابع taskset موجود در کتابخانه torsche برای تشکیل یک مجموعه وظیفه به نام TS1 از وظیفه‌های تعریف شده استفاده کرده‌ایم.

```
9 -     maxAxis1 = max(get(TS1,'Period'));
```

با استفاده از دستور بالا لیست تمام دوره تناوب های موجود در مجموعه وظیفه TS1 را گرفته و بیشینه این لیست را پیدا می‌کنیم. این عدد بیشینه را در متغیر maxAxis1 قرار داده و از این مقدار جهت مشخص کردن طول محور افقی نمودار خود استفاده می‌کنیم.

```
11 -    t4 = torsche.ptask('t4', 3, 10, 0, 10);
12 -    t5 = torsche.ptask('t5', 2, 8, 0, 10);
13 -    t6 = torsche.ptask('t6', 3, 12, 0, 6);
14
15 -    TS2 = torsche.taskset([t4 t5 t6]);
16 -    maxAxis2 = max(get(TS2,'Period'));
```

با دستورات بالا سه وظیفه دیگر هم درست کرده و همان کارهایی که برای سه وظیفه قبلی انجام دادیم برای این وظایف هم انجام می‌دهیم.

اکنون دو مجموعه وظیفه TS1 و TS2 و دو متغیر maxAxis1 و maxAsix2 را داریم.

در ادامه قصد داریم این دو مجموعه وظیفه را هرکدام یکبار با RM و یکبار با EDF زمان‌بندی کنیم. برای اینکار ادامه کد را به چهار قسمت که هر قسمت نشان دهنده یک پردازنده است تقسیم کرده‌ایم.

در زیر کد پردازنده اول که از الگوریتم RM استفاده می‌کند را شرح می‌دهیم.

```
19      % P1
20      % RM
21 -    RM_S1 = RM(TS1);
```

در دستور بالا مجموعه وظیفه TS1 را به تابع RM می‌دهیم. این تابع یک مجموعه وظیفه زمان‌بندی شده را به عنوان خروجی بر می‌گرداند. سپس مجموعه وظیفه زمان‌بندی شده را که به عنوان خروجی از تابع RM گرفته‌ایم در متغیر RM_S1 قرار می‌دهیم.

```
22 -    subplot(2,2,1);
23 -    title('RM1');
24 -    plot(RM_S1,'axis',[0,maxAxis1],'Proc',0)
```

در این قسمت از کد یک زیر نمودار در سطر یک و ستون یک ایجاد می‌کنیم. نام این نمودار را RM گذاشته و در آن مجموعه وظیفه RM_S1 را رسم می‌کنیم. از متغیر axis برای مشخص کردن طول محور افقی نمودار استفاده کرده، و از متغیر proc برای تعیین اینکه هر وظیفه در یک سطر رسم شود استفاده کرده‌ایم.

```
27      % P2
28      % RM
29 -    RM_S2 = RM(TS2);
30 -    subplot(2,2,2);
31 -    title('RM2');
32 -    plot(RM_S2,'axis',[0,maxAxis2],'Proc',0)
```

دستورات بالا کد پردازنده دوم است. این کد تمام کارهایی که برای اجرای الگوریتم RM روی مجموعه اول انجام دادیم این‌بار برای مجموعه دوم تکرار می‌کند.

```
35      % P3
36      % EDF
37 -    EDF_S1 = EDF(TS1);
38 -    subplot(2,2,3);
39 -    title('EDF1');
40 -    plot(EDF_S1,'axis',[0,maxAxis1],'Proc',0)
41
42      % P4
43      % EDF
44 -    EDF_S2 = EDF(TS2);
45 -    subplot(2,2,4);
46 -    title('EDF2');
47 -    plot(EDF_S2,'axis',[0,maxAxis2],'Proc',0)
```

با استفاده از دستورات بالا نیز تمام کارهایی که برای دو بار اجرای الگوریتم RM روی دو مجموعه وظیفه TS1 و TS2 انجام دادیم، این بار برای اجرای الگوریتم EDF روی این دو مجموعه تکرار می‌کنیم.

این قسمت از کد معرف پردازنده سوم و چهارم است.

```
clear
clc

t1 = torsche.ptask('t1', 3, 20, 0, 7);
t2 = torsche.ptask('t2', 2, 5, 0, 4);
t3 = torsche.ptask('t3', 1, 10, 0, 8);

TS1 = torsche.taskset([t1 t2 t3]);
maxAxis1 = max(get(TS1,'Period'));

t4 = torsche.ptask('t4', 3, 10, 0, 10);
t5 = torsche.ptask('t5', 2, 8, 0, 10);
t6 = torsche.ptask('t6', 3, 12, 0, 6);

TS2 = torsche.taskset([t4 t5 t6]);
maxAxis2 = max(get(TS2,'Period'));


% P1
% RM
RM_S1 = RM(TS1);
subplot(2,2,1);
title('RM1');
plot(RM_S1,'axis',[0,maxAxis1],'Proc',0)


% P2
% RM
RM_S2 = RM(TS2);
subplot(2,2,2);
title('RM2');
plot(RM_S2,'axis',[0,maxAxis2],'Proc',0)
```

```
33
34
35      % P3
36      % EDF
37 -    EDF_S1 = EDF(TS1);
38 -    subplot(2,2,3);
39 -    title('EDF1');
40 -    plot(EDF_S1,'axis',[0,maxAxis1],'Proc',0)
41
42      % P4
43      % EDF
44 -    EDF_S2 = EDF(TS2);
45 -    subplot(2,2,4);
46 -    title('EDF2');
47 -    plot(EDF_S2,'axis',[0,maxAxis2],'Proc',0)
48
49      %end of file
```

تمام کد فایل project.m در تصاویر بالا آورده شده است.

شرح تابع RM

```
1      % rate-monotonic scheduling
2    function TS = RM(T)
```

ابتدا این تابع را تعریف می‌کنیم. ورودی این تابع متغیر T و خروجی آن متغیر TS است.

```
3 -    temp_TS = T;
```

با دستور بالا مجموعه وظیفه ورودی را در یک متغیر موقت به نام temp_TS می‌ریزیم.

```
4 -    setprio(temp_TS,'rm');
```

در خط بعدی از تابع setprio استفاده کرده‌ایم، این تابع به هر وظیفه در مجموعه وظیفه ما یک اولویت نسبت می‌دهد. با دادن رشته rm به عنوان ورودی دوم به این تابع مشخص می‌کنیم که این اولویت ها را براساس الگوریتم rm می‌خواهیم.

```
5 -     TS = torsche.fps(temp_TS);
```

در انتها متغیر موقت temp_TS را به تابع fps داده تا زمان‌بندی انجام شود. نتیجه حاصل که یک مجموعه وظیفه زمان‌بندی شده است را در متغیر TS قرار می‌دهیم تا به عنوان خروجی برگردانده شود.

```
6 - end
```

در انتهای دستور end را برای اعلان پایان تعریف تابع می‌نویسیم.

در تصویر زیر تمام کد این تابع آورده شده.

```
1     % rate-monotonic scheduling
2     function TS = RM(T)
3 -   temp_TS = T;
4 -   setprio(temp_TS,'rm');
5 -   TS = torsche.fps(temp_TS);
6 -   end
```

شرح تابع EDF

```
1     % EDF Scheduling
2     function TS = EDF(T)
```

ابتدا تابع EDF را تعریف می‌کنیم. ورودی و خروجی این تابع همانند تابع RM متغیر T و متغیر TS است.

```
3 -       temp_TS = assigning(T);
```

متغیر ورودی (T) را به تابع دیگری به نام assigning که خودمان تعریف کرده‌ایم می‌دهیم، روند اجرا و دستورات این تابع جلوتر توضیح داده خواهدشد. خروجی که از این تابع می‌گیریم را در متغیر موقت temp_TS می‌ریزیم.

```
4 -       TS = torsche.fps(temp_TS);
5 -       end
```

در اینجا با استفاده از تابع fps زمان‌بندی را انجام داده و مجموعه وظیفه زمان‌بندی شده را در متغیر TS ریخته و به عنوان خروجی برمی‌گردانیم.

سپس با دستور end تعریف تابع را به پایان می‌رسانیم.

در زیر کل کد این تابع را قرار داده‌ایم.

```
1     % EDF Scheduling
2     function TS = EDF(T)
3 -   temp_TS = assigning(T);
4 -   TS = torsche.fps(temp_TS);
5 -   end
```

شرح تابع assigning

```
7     % assign weight to tasks based on deadline
8     function TS = assigning(T)
```

در اینجا یک تابع به اسم assigning تعریف کرده‌ایم.

این تابع یک مجموعه وظیفه را گرفته و به وظایف آن بر اساس مهلت اجرای آن‌ها آن ها یک اولویت تخصیص می‌دهد.

```
9 -   TS = sort(T,'Deadline','dec');
```

وظایف را ابتدا بر اساس مهلت اجرای آن‌ها از بزرگ به کوچک (نزولی) مرتب می‌کنیم.

```
10 -  n = count(TS);
11 -  for i= 1:n
12 -      TS.Weight(i) = i;
13 -  end
14 -  end
```

تعداد وظایف را بدست آورده و در متغیر n قرار می‌دهیم.

سپس با یک حلقه for روی مجموعه به هر یک از وظیفه ها یک عدد تخصیص می‌دهیم.

در نهایت با دستور end تعریف تابع را تمام می‌کنیم.

```
  % assign weight to tasks based on deadline
function TS = assigning(T)
TS = sort(T,'Deadline','dec');
n = count(TS);
for i= 1:n
    TS.Weight(i) = i;
end
end
```

تمام کد این تابع در تصویر بالا مشخص است.

دو تابع EDF و assigning در فایل EDF.m قرار دارند، این دو تابع در بالا شرح داده شدند. تمام کد موجود در این فایل در تصویر زیر آورده شده.

```
  % EDF Scheduling
function TS = EDF(T)
temp_TS = assigning(T);
TS = torsche.fps(temp_TS);
end

  % assign weight to tasks based on deadline
function TS = assigning(T)
TS = sort(T,'Deadline','dec');
n = count(TS);
for i= 1:n
    TS.Weight(i) = i;
end
end
```