به نام خدا

دانشکده مهندسی کامپیوتر

گزارش تمرین عملی سوم

درس: سیستم‌های نهفته و بی‌درنگ

دانشجو: فرشید نوشی – ۹۸۳۱۰۶۸

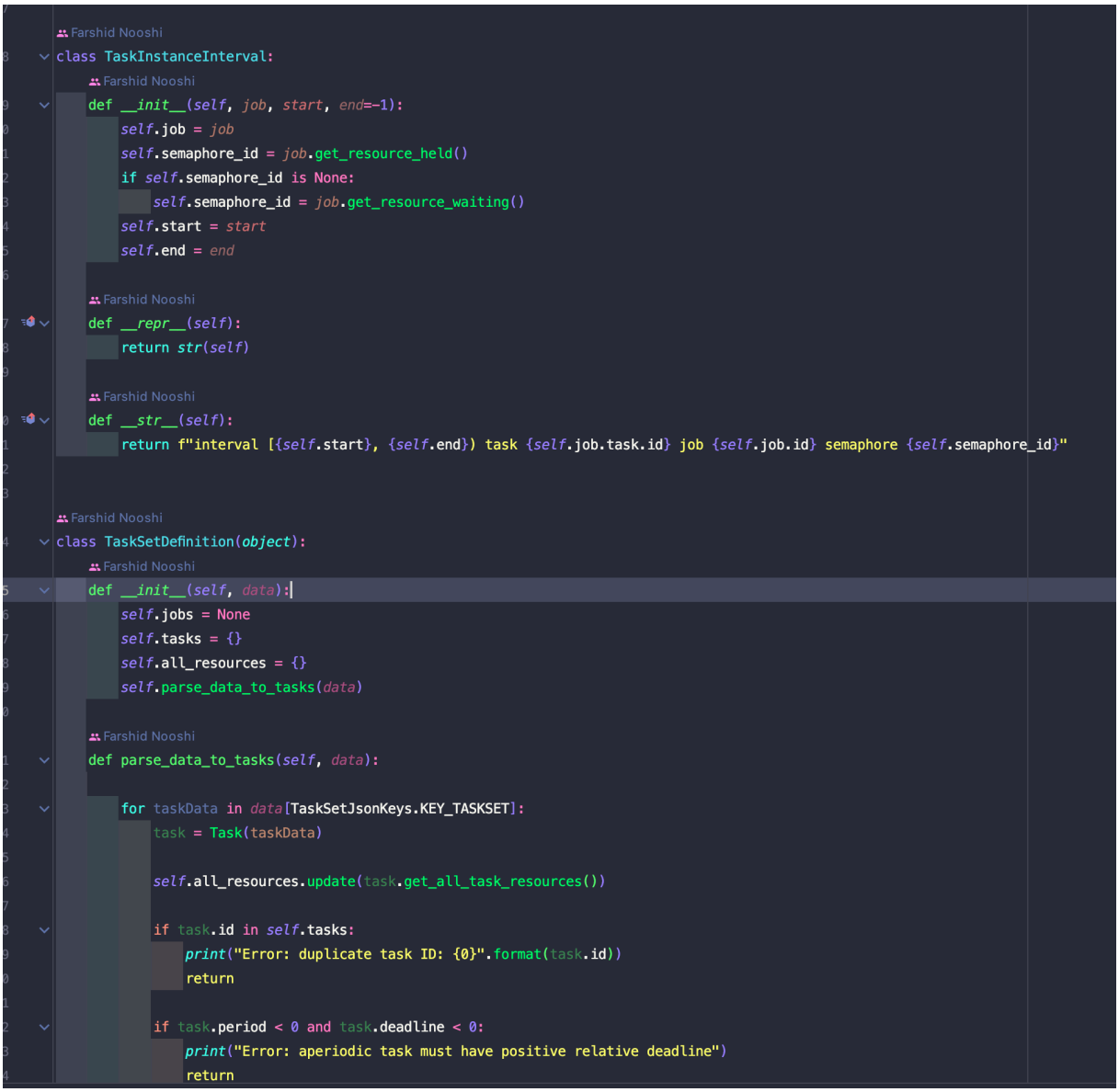
توضیح کد

```python
#!/usr/bin/env python

"""
taskSet.py - parser for task set from JSON file
"""

import json
import random
from time import sleep

import numpy as np
from matplotlib import pyplot as plt

READY = 0
RUNNING = 1
BLOCKED = 2
FINISHED = 3
MISSED = 4

NP_PRIORITY = -2
NPP_PRIORITY = -1

NPP = 0
HLP = 1

RM = 0
DM = 1


Farshid Nooshi
def plot_tasks(scheduler):
    tasks = scheduler.task_set.tasks
    tasks = {k: v for k, v in reversed(sorted(tasks.items(), key=lambda item: item[1].period))}
    fig, ax = plt.subplots()
    y_label = np.array(list(tasks.keys()))[::-1]
    c_width = y_label.size / (y_label.size + 1)
    color_map = scheduler.task_set.all_resources.copy()
    color_map.pop(0)
    for k in color_map.keys():
        color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
        color_map[k] = tuple([x / 255 for x in color])
    for task_id in y_label:
        color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
        color_map[f"task {task_id}"] = tuple([x / 255 for x in color])
```

در تصویر بالا ابتدا مقادیر ثابتی برای الگوریتم‌ها و پروتکل‌های منابع مشترک بین تسک ها نوشته شده اند. در ادامه نیز تابعی برای نمایش گانت چارت تسک ها آورده شده است. در این تمرین پروتکل‌های NPP, HLP پیاده شده اند و از DM استفاده شده است.

```
Farshid Nooshi
class TaskSetJsonKeys(object):
    # Task set
    KEY_TASKSET = "taskset"

    # Task
    KEY_TASK_ID = "taskId"
    KEY_TASK_PERIOD = "period"
    KEY_TASK_WCET = "wcet"
    KEY_TASK_DEADLINE = "deadline"
    KEY_TASK_OFFSET = "offset"
    KEY_TASK_SECTIONS = "sections"

    # Schedule
    KEY_SCHEDULE_START = "startTime"
    KEY_SCHEDULE_END = "endTime"

    # Release times
    KEY_RELEASETIMES = "releaseTimes"
    KEY_RELEASETIMES_JOBRELEASE = "timeInstant"
    KEY_RELEASETIMES_TASKID = "taskId"


Farshid Nooshi
class TaskSetIterator:
    Farshid Nooshi
    def __init__(self, taskSet):
        self.taskSet = taskSet
        self.index = 0
        self.keys = iter(taskSet.tasks)

    Farshid Nooshi
    def __next__(self):
        key = next(self.keys)
        return self.taskSet.tasks[key]


new *
class TaskInstanceInterval:
    new *
    def __init__(self, job, start, end=-1):
        self.job = job
```

در تصویر بالا نیز متغیرهای ثابت برای خواندن فایل جیسون ورودی تسک ها در تمرین آمده است به همراه یک کلاس Iterator برای حرکت کردن بر روی TaskSet.



```
Farshid Nooshi
class TaskInstanceInterval:
    Farshid Nooshi
    def __init__(self, job, start, end=-1):
        self.job = job
        self.semaphore_id = job.get_resource_held()
        if self.semaphore_id is None:
            self.semaphore_id = job.get_resource_waiting()
        self.start = start
        self.end = end

    Farshid Nooshi
    def __repr__(self):
        return str(self)

    Farshid Nooshi
    def __str__(self):
        return f"interval [{self.start}, {self.end}) task {self.job.task.id} job {self.job.id} semaphore {self.semaphore_id}"


Farshid Nooshi
class TaskSetDefinition(object):
    Farshid Nooshi
    def __init__(self, data):
        self.jobs = None
        self.tasks = {}
        self.all_resources = {}
        self.parse_data_to_tasks(data)

    Farshid Nooshi
    def parse_data_to_tasks(self, data):

        for taskData in data[TaskSetJsonKeys.KEY_TASKSET]:
            task = Task(taskData)

            self.all_resources.update(task.get_all_task_resources())

            if task.id in self.tasks:
                print("Error: duplicate task ID: {0}".format(task.id))
                return

            if task.period < 0 and task.deadline < 0:
                print("Error: aperiodic task must have positive relative deadline")
                return
```

در بالا نیز کلاسی برای محدوده هر TaskSetInstance آمده است در ادامه نیز کلاسی برای توضیح TaskSet آمده است که در آن یک مجموعه تسک آورده شده اند. در این کلاس جاب‌ها به همراه تسک‌ها تعریف میشوند.

```
    def generate_gantt_chart(self):
        fig, ax = plt.subplots()
        ax.set_xlabel("Time")
        ax.set_ylabel("Tasks")
        ax.set_yticks([(i + 1) for i in range(len(self.tasks))])
        ax.set_yticklabels([f"Task {task.id}" for (id, task) in self.tasks.items()])
        ax.grid(True, which='both', axis='x', linestyle='--', linewidth=1)
        # set the x-splitter to be the 5
        ax.set_xticks(np.arange(0, 100, 5))

        for (id, task) in self.tasks.items():  # Plotting the jobs
            for job in task.get_task_instance_to_run():
                release_time = job.releaseTime
                deadline = job.deadline
                execution_time = task.sections[0][1]  # Assuming all sections have the same execution time
                ax.broken_barh([(release_time, execution_time)], (task.id - 0.4, 0.8), facecolors="tab:blue")
                ax.plot([deadline, deadline], [task.id - 0.2, task.id + 0.2], color="red")

        plt.show()


    Farshid Nooshi
class Task(object):
    Farshid Nooshi
    def __init__(self, task_dict):
        self.id = int(task_dict[TaskSetJsonKeys.KEY_TASK_ID])
        self.period = float(task_dict[TaskSetJsonKeys.KEY_TASK_PERIOD])
        self.wcet = float(task_dict[TaskSetJsonKeys.KEY_TASK_WCET])
        self.deadline = float(task_dict.get(TaskSetJsonKeys.KEY_TASK_DEADLINE, self.period))
        self.offset = float(task_dict.get(TaskSetJsonKeys.KEY_TASK_OFFSET, 0.0))
        self.sections = task_dict[TaskSetJsonKeys.KEY_TASK_SECTIONS]

        self.job_count = 0
        self.spawn = self.offset

    Farshid Nooshi
    def get_all_task_resources(self):
        resources = {}
        for section in self.sections:
            if section[0] != 0:
                resources[section[0]] = None
        return resources

    Farshid Nooshi
```

تابع اول تصویر بالا مربوط به کشیدن گانت چارت می‌باشد. در ادامه نیز تعریف خود کلاس تسک آمده است. در این کلاس یک فیلد id برای شناسایی تسک، و فیلدهای period, wcet, deadline, offset همانند تعاریف درس آورده شده اند. فیلد sections مربوط به section های هر تسک براساس فایل ورودی json می‌باشد.

متد پایینی در تصویر بالا نیز مربوط به گرفتن تمامی منابع گرفته شده در سیستم می‌باشد که یک دیکشنری برمیگرداند.

```python
Farshid Nooshi
class TaskInstance(object):

    Farshid Nooshi
    def __init__(self, task, jobId, arrive):
        self.task = task
        self.id = jobId
        self.arrive = arrive
        self.deadline = task.deadline + arrive
        self.state = READY

        self.total_runtime = 0

        self.it = iter(task.sections)
        section = next(self.it, None)
        self.semaphore_id = section[0]
        self.section_time = section[1]
        self.section_runtime = 0

        self.original_priority = None
        self.priority = None

    Farshid Nooshi
    def get_resource_held(self):
        if self.semaphore_id != 0 and self.section_runtime != 0:
            return self.semaphore_id
        return None

    Farshid Nooshi
    def get_resource_waiting(self):
        if self.semaphore_id != 0 and self.section_runtime == 0:
            return self.semaphore_id
        return None

    Farshid Nooshi
    def get_remaining_section_time(self):
        return self.section_time - self.section_runtime

    Farshid Nooshi
    def execute(self, resources: dict):
        print(f"[{self.task.id}:{self.id}:{self.semaphore_id}] run {self.total_runtime}:{self.section_runtime}")
        if self.is_completed():
            return False
```

در تصویر بالا نیز پیاده سازی یک instance از task آورده شده است. در این تصویر متد سازنده این کلاس در آغاز آمده است در ادامه نیز متدهایی برای گرفتن منابع گرفته شده تسک، منابع درخواست داده تسک و منابع باقی مانده تسک آمده اند همچنین در این کلاس متدی برای اجرای تسک نیز آمده است که در پایین تصویر قابل مشاهده است.

```python
300
    Farshid Nooshi
301 def execute(self, resources: dict):
302     print(f"[{self.task.id}:{self.id}:{self.semaphore_id}] run {self.total_runtime}:{self.section_runtime}")
303     if self.is_completed():
304         return False
305
306     if self.semaphore_id != 0 and resources.get(self.semaphore_id) != self:
307         return True
308
309     self.total_runtime += 1
310     self.section_runtime += 1
311
312     if self.get_remaining_section_time() == 0:
313         self.section_runtime = 0
314         resources[self.semaphore_id] = None
315         if self.priority != NP_PRIORITY:
316             self.priority = self.original_priority
317
318         section = next(self.it, None)
319         if section is None:
320             return False
321         self.semaphore_id = section[0]
322         self.section_time = section[1]
323
324     return False
325
    Farshid Nooshi
326 def execute_to_completion(self):
327     return self.task.wcet - self.total_runtime
328
    Farshid Nooshi
329 def is_completed(self):
330     return self.total_runtime == self.task.wcet
331
    Farshid Nooshi
332 def __str__(self):
333     return f"[{self.task.id}:{self.id}] released at {self.arrive} -> deadline at {self.deadline}"
334
```

در تصویر بالا تابع اجرای تسک به طور کامل آورده شده است. در این تابع براساس اینکه اجرای کامل شده است یا خیر تصمیم گیری میشود و در صورت اینکه تسک منتظر یک منبع هست یا خیر نیز شرط ها در ادامه اش آمده اند.

```python
Farshid Nooshi
class Scheduler(object):
    Farshid Nooshi
    def __init__(self, data, algorithm=DM, resource_access_protocol=NPP, preemptive=True):
        self.task_set = TaskSetDefinition(data)
        self.job_queue = TaskInstanceQueue()
        self.intervals = []
        self.algorithm = algorithm
        self.resource_access_protocol = resource_access_protocol
        self.preemptive = preemptive
        self.timer = 0
        self.start_time = float(data[TaskSetJsonKeys.KEY_SCHEDULE_START])
        self.end_time = float(data[TaskSetJsonKeys.KEY_SCHEDULE_END])

    Farshid Nooshi
    def print_interval(self):
        for interval in self.intervals:
            print(interval)

    Farshid Nooshi
    def get_highest_priority_task_instance_resource(self, hp_job, sem_id):
        tasks = list(self.task_set.tasks.values())
        highest_priority = hp_job.original_priority

        for job in self.job_queue.running_set:
            tasks.remove(job.task)
            if sem_id in job.task.get_all_task_resources().keys():
                highest_priority = min(highest_priority, self.get_task_priority(job))

        for task in tasks:
            if sem_id in task.get_all_task_resources().keys():
                job = TaskInstance(task, -1, task.spawn)
                highest_priority = min(highest_priority, self.get_task_priority(job))

        return highest_priority

    Farshid Nooshi
    def get_running_task_instance(self):
        if self.job_queue.running_set == set():
            return None

        running_jobs = list(filter(lambda job: job.state == RUNNING, self.job_queue.running_set))
        hp_running_job = list(filter(lambda job: job.priority != job.original_priority, running_jobs))
```

در تصویر بالا پیاده سازی زمانبند آمده است که در آن فیلدهای مربوط به زمان آغاز و پایان به همراه تایمر و الگوریتم و پروتکل منابع مشترکی و دیگر کانفیگ ها هستند. همچنین یک متد دیگر برای گرفتن تسکی که بالاترین اولویت را دارند نیز هست.

taskset.py

```
Farshid Nooshi
def get_task_priority(self, job):
    if self.algorithm == RM:
        return job.task.period
    elif self.algorithm == DM:
        return job.task.deadline
    elif self.algorithm == EDF:
        return job.deadline
    else:
        raise Exception("Invalid algorithm")

Farshid Nooshi
def schedule_tasks(self):
    print("Scheduling tasks...")
    self.job_queue.update_jobs(self.start_time + self.timer, self.intervals)

    for task in self.task_set.tasks.values():
        print("Task: ", task.id, " has ", task.get_all_task_resources().keys(), " resources")
        job = task.spawn_job(self.start_time + self.timer)
        if job:
            print("Adding job: ", job.id, " to queue")
            job.original_priority = self.get_task_priority(job)
            job.priority = job.original_priority
            self.job_queue.add_job(job)

    hp_job = self.get_task_instance_to_run()

    if self.start_time + self.timer == self.end_time:
        print("End of schedule")
        if self.intervals and self.intervals[-1].end == -1:
            self.intervals[-1].end = self.start_time + self.timer
        return

    if not hp_job:
        return

    is_blocked = hp_job.execute(self.task_set.all_resources)

    if is_blocked:
        print("Job: ", hp_job.id, " is blocked")
        semaphore_id = hp_job.get_resource_waiting()
        print("Job: ", hp_job.id, " is waiting for resource: ", semaphore_id)
        self.check_condition(hp_job, semaphore_id)
```

در ادامه نیز تابعی برای برنامه زمانبندی کردن تسک ها آورده شده است. توضیحات مربوط به هربخش که چه اتفاقی در حال رخ دادن میباشد همانند آغاز زمانبندی و یا زمانبندی تسکی خاص یا اضافه کردن یک تسک به صف تسک‌های اجرا شونده یا پایان زمانبندی در بخش های مختلفی آمده اند و به صورت print نوشته شده اند.

```python
Farshid Nooshi *
def main():
    DATA_FILE = "data/json/HLP_taskset.json"

    with open(DATA_FILE) as json_data:
        data = json.load(json_data)

    scheduler = Scheduler(data, resource_access_protocol=HLP)
    print("Scheduler: ", scheduler.algorithm, " ", scheduler.resource_access_protocol)

    # Scheduler loop - runs until end time
    while scheduler.start_time + scheduler.timer <= scheduler.end_time:
        scheduler.schedule_tasks()
        scheduler.timer += 1
        sleep(0.1)

    scheduler.print_interval()

    # Declaring a figure "gnt"
    plot_tasks(scheduler)
    print("Utilization: ", scheduler.get_utilization())


if __name__ == "__main__":
    main()
```

در تصویر بالا نیز تابع اصلی آغاز برنامه آمده است که در آن فایل Json خوانده میشود و به زمانبند تسک ها داده میشوند و در ادامه نیز در یک حلقه تسک ها زمانبندی میشوند.

برای بررسی الگوریتم و پیاده‌سازی ها نیز مثال‌های خود اسلایدهای درس حدالامکان سعی در گذاشته شدن داشتند.

نتایج

برای این تمرین از NPP, HIP استفاده شده است که پیاده سازی شدند. در تصاویر گانت چارت نیز زمانی که از منبعی استفاده میشده است رنگ آن تغییر کرده است که این موضو ع نمایان باشد که در حال گذراندن ناحیه بحرانی هستیم.

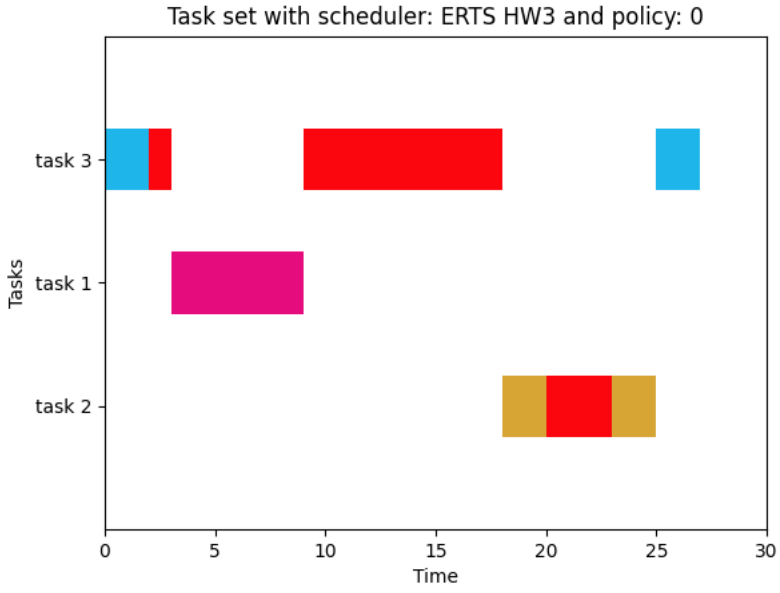
NPP

تصویر بالا نمونه خروجی کدهای پیاده‌سازی شده برای پروتکل NPP میباشد که در آن تسک ها فرض میشود که Non-preemptive هستند تا منابعی را که میخواهند را به اشتراک نگذارند. صحت این پیاده سازی نیز برقرار میباشد زیرا که تسک ست داده شده برای این مثال همان تسک ست اسلاید درس است.

در مثال های زیر نیز offset تغییر داده شده است و همانگونه که از تصاویر مشخص است تغییر میزان offset میتواند در مشخص شدن اولین تسکی که وارد ناحیه بحرانی میشود موثر باشد.

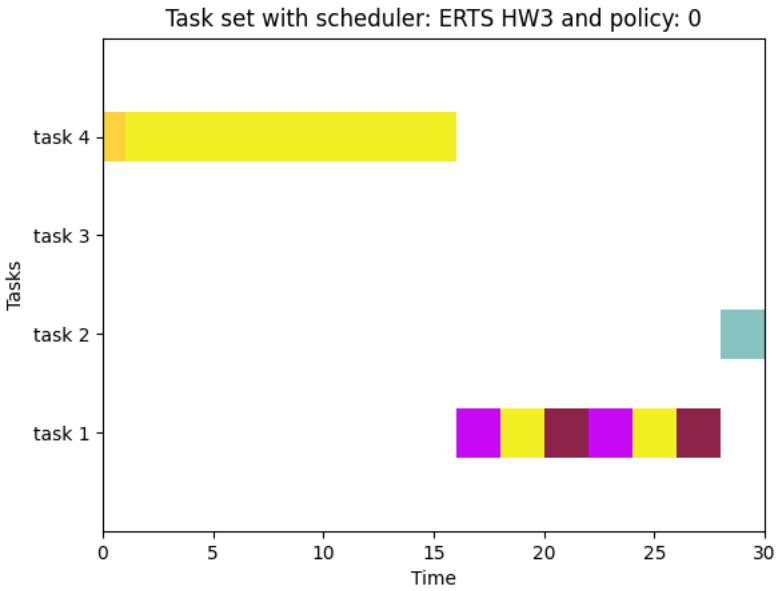
در تسک ست دیگر نیز به خروجی زیر رسیدیم که به اصطلاح آن را تسک ست Hard نامیدیم.

```
interval [15.0, 16.0) task 3 job 1 semaphore 1
interval [16.0, 17.0) task 3 job 1 semaphore 1
interval [17.0, 18.0) task 3 job 1 semaphore 1
interval [18.0, 20.0) task 2 job 1 semaphore None
interval [20.0, 23.0) task 2 job 1 semaphore 1
interval [23.0, 25.0) task 2 job 1 semaphore None
interval [25.0, 27.0) task 3 job 1 semaphore None
Calculating utilization...
Utilization:  0.9
Is feasible:  True
```



با توجه به خروجی بالا قابل درک می‌باشد که برای ایجاد تاخیرهای بیشتر برای تسک های مهم تر میشود که ناحیه بحرانی را طولانی تر نمود. همانطور که در اسلایدهای درس نیز گفته شد این که تسک با ناحیه بحرانی طولانی داشته باشیم میتواند ددلاین تسکهای دیگر را رد بکند.

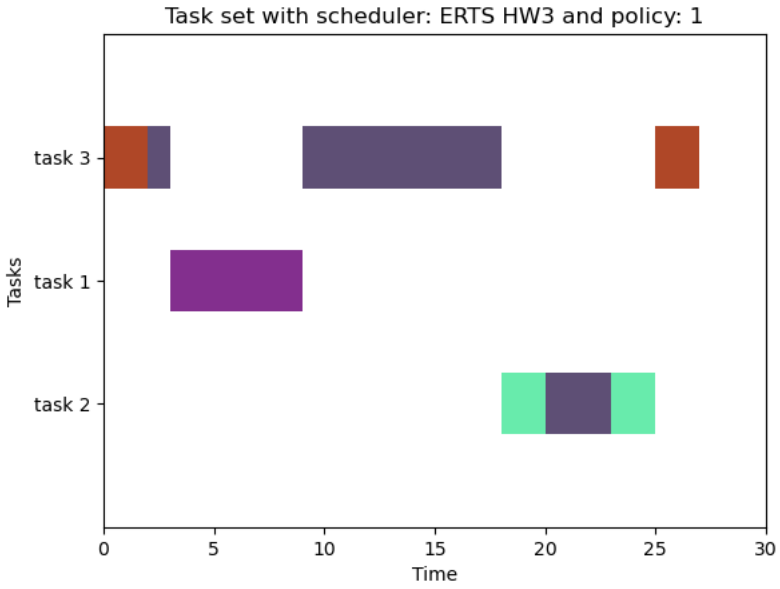
در زیر نیز مثالی برای رد شدن ددلاین با طولانی شدن ناحیه بحرانی امده است.



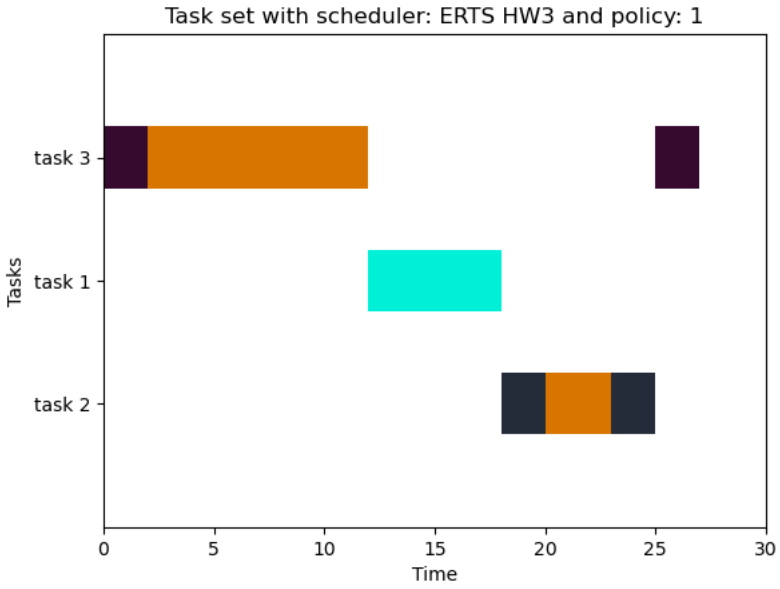

```
interval [16.0, 18.0) task 1 job 1 semaphore None
interval [18.0, 20.0) task 1 job 1 semaphore 1
interval [20.0, 22.0) task 1 job 1 semaphore 2
interval [22.0, 24.0) task 1 job 2 semaphore None
interval [24.0, 26.0) task 1 job 2 semaphore 1
interval [26.0, 28.0) task 1 job 2 semaphore 2
interval [28.0, 30.0) task 2 job 1 semaphore None
Calculating utilization...
Utilization:  1.0
Is feasible:  False
```

HLP

با مثال اسلاید استفاده شده است.



در حالتی که این این تسک ست با پروتکل NPP اجرا بشود داریم:



که همانگونه که مشاهده میشود در حالت ورود تسک ۳ به ناحیه بحرانی جلوی تسک با اولویت بالاتر در این مثال گرفته میشود.(تسک شماره ۱)