سند بررسی کد سوال اول تمرین۴

امیرمهدی دارائی - ۹۹۱۰۵۴۳۱

تحلیل زمان‌بندی وظایف

این سند به تحلیل زمان‌بندی وظایف با استفاده از زمان‌بندی **Earliest Deadline First** و سرور پراکنده **(Sporadic Server)** می‌پردازد.
کد زیر شامل بخش‌های مختلفی است که در ادامه توضیح مختصری از هر کلاس و تابع ارائه شده است.

کلاس Task

کلاس **Task** نمایانگر یک وظیفه منفرد با مشخصات زیر است:
name: نام وظیفه.
period: دوره‌ی وظیفه.
execution_time: زمان اجرای وظیفه.
remaining_time: زمان باقی‌مانده برای اجرای وظیفه.
arrival_time: زمان ورود وظیفه.
deadline: مهلت انجام وظیفه.
task_type: نوع وظیفه (سخت یا نرم).
server: سرور مرتبط با وظیفه (در صورت وجود).
start_time: زمان شروع وظیفه.

```python
class Task:
    """
    A class to represent a task in the scheduling system.

    Attributes:
    ----------
    name : str
        The name of the task.
    period : int
        The period of the task.
    execution_time : int
        The execution time of the task.
    remaining_time : int
        The remaining execution time of the task.
    arrival_time : int
        The arrival time of the task.
    deadline : int
        The deadline of the task.
    task_type : str
        The type of the task ('hard' or 'soft').
    server : SporadicServer
        The server associated with the task.
    start_time : int
        The start time of the task.
    """

    def __init__(self, task_name, task_period, exec_time, arrival=0, task_kind='hard', server_instance=None):
        self.name = task_name
        self.period = task_period
        self.execution_time = exec_time
        self.remaining_time = exec_time
        self.arrival_time = arrival
        self.deadline = task_period + arrival if task_kind == 'hard' else arrival + server_instance.period
        self.task_type = task_kind
        self.server = server_instance
        self.start_time = None

    def __lt__(self, other):
        if self.task_type == 'soft' and self.server.remaining_capacity == 0:
            return False
        if other.task_type == 'soft' and other.server.remaining_capacity == 0:
            return True
        if self.deadline == other.deadline:
            return self.task_type == 'soft' and other.task_type == 'hard'
        return self.deadline < other.deadline
```

این کلاس شامل توابع زیر است:

- `__init__`: وظیفه جدیدی را با نام، دوره، زمان اجرا، زمان ورود، نوع وظیفه و سرور مرتبط (در صورت وجود) ایجاد می‌کند.
- `__lt__`: دو وظیفه را بر اساس مهلت انجام و نوع آنها مقایسه می‌کند تا مشخص کند کدام وظیفه اولویت بالاتری دارد.

کلاس SporadicServer

کلاس SporadicServer سرور پراکنده را در سیستم زمان‌بندی نمایان می‌کند.

period: دوره‌ی تجدید سرور.

capacity: ظرفیت کل سرور.

remaining_capacity: ظرفیت باقی‌مانده‌ی سرور.

replenishment_queue: صف تجدید ظرفیت.

```
class SporadicServer:
    """
    A class to represent a sporadic server in the scheduling system.

    Attributes:
    ----------
    period : int
        The replenishment period of the server.
    capacity : int
        The total capacity of the server.
    remaining_capacity : int
        The remaining capacity of the server.
    replenishment_queue : list
        The queue to handle capacity replenishment.
    """

    def __init__(self, server_period, server_capacity):
        self.period = server_period
        self.capacity = server_capacity
        self.remaining_capacity = server_capacity
        self.replenishment_queue = []
```

این کلاس شامل وظایف زیر است:

- `use_capacity`: مقدار مشخصی از ظرفیت سرور را استفاده می‌کند و زمان تجدید آن را برنامه‌ریزی می‌کند.

```python
    def use_capacity(self, amount, current_time):
        """
        Uses the server's capacity and schedules replenishment.

        Parameters:
        ----------
        amount : int
            The amount of capacity to use.
        current_time : int
            The current time in the scheduling system.
        """
        self.remaining_capacity -= amount
        replenishment_time = current_time + self.period
        heapq.heappush(self.replenishment_queue, (replenishment_time, amount))
```

- `replenish_capacity`: اگر زمان تجدید سرور فرا رسیده باشد، ظرفیت سرور را تجدید می‌کند.

```python
def replenish_capacity(self, current_time):
    """
    Replenishes the server's capacity if it's time to do so.

    Parameters:
    ----------
    current_time : int
        The current time in the scheduling system.
    """
    while self.replenishment_queue and self.replenishment_queue[0][0] <= current_time:
        _, amount = heapq.heappop(self.replenishment_queue)
        self.remaining_capacity += amount
        task_scheduler.reorganize_queue()
```

کلاس EDFScheduler

کلاس EDFScheduler زمان‌بند EDF را نمایان می‌کند که وظایف را بر اساس مهلت انجامشان مدیریت می‌کند.

current_time: زمان کنونی در سیستم زمان‌بندی.

ready_tasks: صف وظایف آماده.

execution_log: گزارش وظایف اجرا شده.

server_capacity_log: گزارش ظرفیت سرور در طول زمان.

hyper_period: دوره هایپر سیستم.

periodic_tasks: لیست وظایف دوره‌ای در سیستم.

```python
class EDFScheduler:
    """
    A class to represent the EDF Scheduler.

    Attributes:
    ----------
    current_time : int
        The current time in the scheduling system.
    ready_tasks : list
        The queue of ready tasks.
    execution_log : list
        The log of executed tasks.
    server_capacity_log : list
        The log of server capacities over time.
    hyper_period : int
        The hyper period of the system.
    periodic_tasks : list
        The list of periodic tasks in the system.
    """

    def __init__(self):
        self.current_time = 0
        self.ready_tasks = []
        self.execution_log = []
        self.server_capacity_log = []
        self.hyper_period = 24
        self.periodic_tasks = []
```

این کلاس شامل وظایف زیر است:

- `add_task_to_queue`: وظیفه‌ای را به صف وظایف آماده اضافه می‌کند.

```python
    def add_task_to_queue(self, task):
        """
        Adds a task to the ready queue.

        Parameters:
        ----------
        task : Task
            The task to be added to the queue.
        """
        heapq.heappush(self.ready_tasks, task)
```

- `advance_scheduler_time`: زمان زمان‌بند را به مدت مشخصی جلو می‌برد.

```
    def advance_scheduler_time(self, skip_duration):
        """
        Advances the scheduler's time by a specified duration.

        Parameters:
        ----------
        skip_duration : int
            The duration to advance the scheduler's time.
        """
        for _ in range(skip_duration):
            self.execute_schedule()
```

- `reorganize_queue`: صف وظایف آماده را برای حفظ خصوصیات heap دوباره سازماندهی می‌کند.

```
def reorganize_queue(self):
    """
    Reorganizes the ready queue to maintain heap properties.
    """
    self.ready_tasks = [heapq.heappop(self.ready_tasks) for _ in range(len(self.ready_tasks))]
    heapq.heapify(self.ready_tasks)
```

- `log_execution`: اجرای یک وظیفه را در گزارش ثبت می‌کند.

```
    def log_execution(self, task_name, task_kind):
        """
        Logs the execution of a task.

        Parameters:
        ----------
        task_name : str
            The name of the task.
        task_kind : str
            The type of the task ('hard' or 'soft').
        """
        self.execution_log.append((self.current_time, task_name, task_kind))
```

- `execute_schedule`: الگوریتم زمان‌بندی را برای یک مرحله زمانی فعلی اجرا می‌کند.

```
def execute_schedule(self):
    """
    Executes the scheduling algorithm for the current time step.
    """
    sporadic_server.replenish_capacity(self.current_time)
    self.server_capacity_log.append(sporadic_server.remaining_capacity)

    for task in self.periodic_tasks:
        if self.current_time > task.arrival_time and (self.current_time - task.arrival_time) % task.period == 0:
            self.add_task_to_queue(Task(task.name, task.period, task.execution_time, self.current_time, task.task_type))

    if self.ready_tasks:
        current_task = heapq.heappop(self.ready_tasks)
        if current_task.start_time is None:
            current_task.start_time = self.current_time

        if current_task.task_type == 'soft' and sporadic_server.remaining_capacity > 0:
            sporadic_server.use_capacity(1, self.current_time)
            current_task.remaining_time -= 1
            self.log_execution(current_task.name, 'soft')
        elif current_task.task_type == 'hard':
            current_task.remaining_time -= 1
            self.log_execution(current_task.name, 'hard')

        if current_task.remaining_time > 0:
            heapq.heappush(self.ready_tasks, current_task)
        else:
            print(f"Time {self.current_time}: Task {current_task.name} completed")

    self.current_time += 1
```

- `run_scheduler`: زمان‌بند را تا رسیدن به دوره هایپر یا تکمیل تمام وظایف اجرا می‌کند.

```
def run_scheduler(self):
    """
    Runs the scheduler until the hyper period is reached or all tasks are completed.
    """
    while self.current_time < self.hyper_period or self.ready_tasks:
        self.execute_schedule()
    self.display_schedule()
```

- `display_schedule`: زمان‌بندی وظایف و ظرفیت سرور را با استفاده از **matplotlib** نمایش می‌دهد.

تابع handle_user_input

handle_user_input: ورودی‌های کاربر را برای افزودن وظایف، جلو بردن زمان، یا پایان شبیه‌سازی مدیریت می‌کند.

```
def handle_user_input(server_instance):
    """
    Handles user input to add tasks, skip time, or end the simulation.

    Parameters:
    ----------
    server_instance : SporadicServer
        The instance of the sporadic server.
    """
    while True:
        command = input("Enter command (soft, skip, end): ").strip()
        if command == 'soft':
            duration = int(input("Enter duration of the soft task: ").strip())
            soft_task = Task(task_name=f"Soft{len(task_scheduler.execution_log) + 1}", task_period=float('inf'),
                             exec_time=duration, arrival=task_scheduler.current_time,
                             task_kind='soft', server_instance=server_instance)
            task_scheduler.add_task_to_queue(soft_task)
        elif command == 'skip':
            skip_duration = int(input("Enter time to skip: ").strip())
            task_scheduler.advance_scheduler_time(skip_duration)
        elif command == 'end':
            task_scheduler.run_scheduler()
            break
        else:
            print("Invalid command")
```

تابع main

در این کد، ابتدا یک سرور پراکنده با دوره تجدید ۶ و ظرفیت ۳ ایجاد می‌شود. این سرور وظیفه مدیریت و تجدید ظرفیت برای اجرای وظایف نرم را بر عهده دارد. سپس یک زمان‌بند EDF (Earliest Deadline First) ایجاد می‌شود که وظیفه مدیریت وظایف بر اساس مهلت انجام آنها را بر عهده دارد. در ادامه دو وظیفه سخت ایجاد می‌شود: وظیفه اول با نام τ۱، دوره ۸، زمان اجرای ۲، زمان ورود ۰ و وظیفه دوم با نام τ۲، دوره ۱۲، زمان اجرای ۳، زمان ورود ۰. این وظایف به صف وظایف آماده زمان‌بند اضافه می‌شوند. سپس این وظایف به لیست وظایف دوره‌ای زمان‌بند نیز اضافه می‌شوند. در نهایت، تابعی برای مدیریت ورودی‌های کاربر فراخوانی می‌شود تا به کاربر اجازه دهد وظایف نرم جدید اضافه کند، زمان زمان‌بند را جلو ببرد یا شبیه‌سازی را به پایان برساند. وظایف نرم جدید به سرور پراکنده مرتبط می‌شوند.

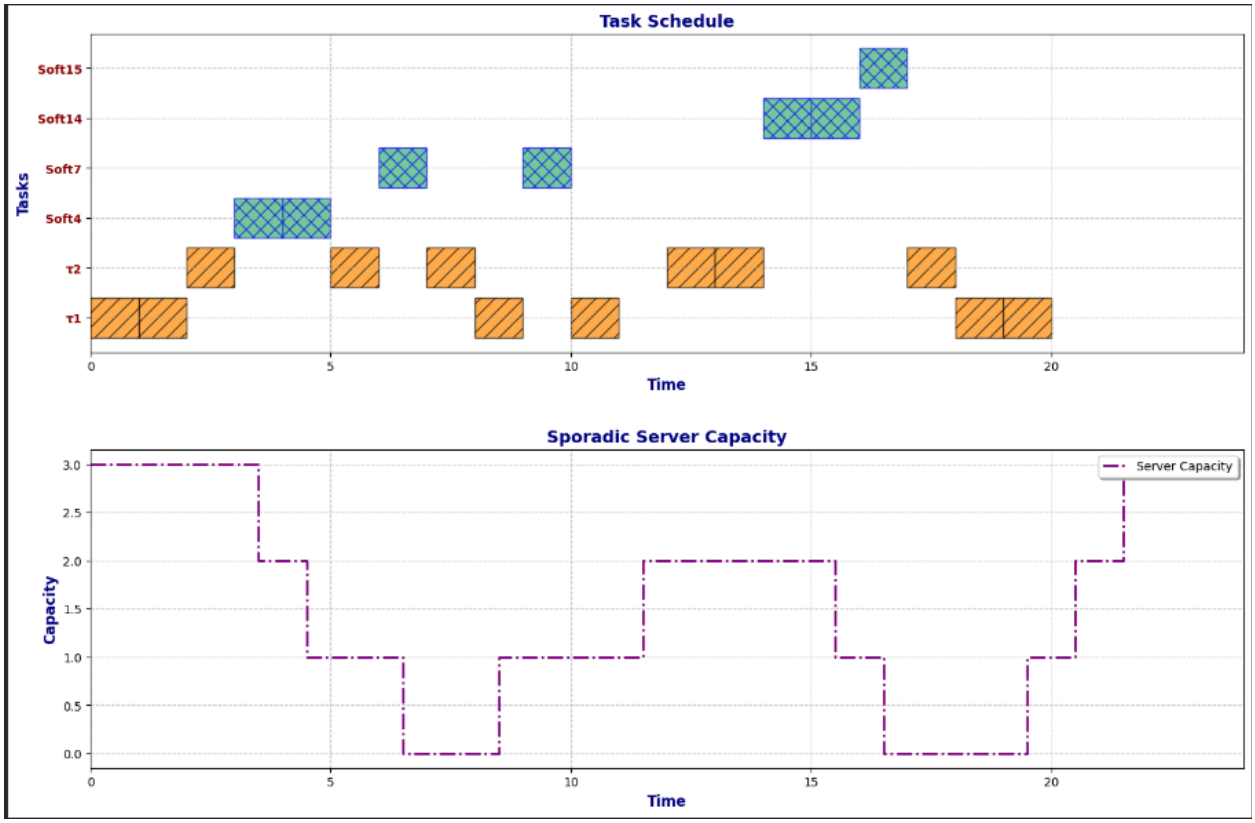
```
sporadic_server = SporadicServer(server_period=6, server_capacity=3)

task_scheduler = EDFScheduler()
task1 = Task('τ1', 8, 2, 0, 'hard', sporadic_server)
task2 = Task('τ2', 12, 3, 0, 'hard', sporadic_server)
task_scheduler.add_task_to_queue(task1)
task_scheduler.add_task_to_queue(task2)
task_scheduler.periodic_tasks.append(task1)
task_scheduler.periodic_tasks.append(task2)

handle_user_input(sporadic_server)
```

نتایج

من برای تست از همان مثال اسلاید، و اعداد آن استفاده کردم که بعضا در بالا، اعداد برخی از پارامتر های آن ذکر شد. در زیر نیز نحوه ورودی دادن به سیستم را برای این مثال مشاهده می‌کنیم:

```
Enter command (soft, skip, end):  skip
Enter time to skip:  3
Time 1: Task τ1 completed
Enter command (soft, skip, end):  soft
Enter duration of the soft task:  2
Enter command (soft, skip, end):  skip
Enter time to skip:  3
Time 4: Task Soft4 completed
Enter command (soft, skip, end):  soft
Enter duration of the soft task:  2
Enter command (soft, skip, end):  skip
Enter time to skip:  8
Time 7: Task τ2 completed
Time 9: Task Soft7 completed
Time 10: Task τ1 completed
Enter command (soft, skip, end):  soft
Enter duration of the soft task:  2
Enter command (soft, skip, end):  skip
Enter time to skip:  1
Enter command (soft, skip, end):  soft
Enter duration of the soft task:  1
Enter command (soft, skip, end):  end'
Invalid command
Enter command (soft, skip, end):  end
Time 15: Task Soft14 completed
Time 16: Task Soft15 completed
Time 17: Task τ2 completed
Time 19: Task τ1 completed
```

و نهایتا نتیجه‌ی زمان‌بندی برای این مثال به صورت زیر است:



شرح خروجی

در این بخش، خروجی برنامه به همراه ورودی‌های کاربر توضیح داده می‌شود. خروجی شامل دو بخش است: گراف زمان‌بندی وظایف و ظرفیت سرور پراکنده و لاگ ورودی‌های کاربر و وضعیت وظایف.

گراف زمان‌بندی وظایف و ظرفیت سرور پراکنده

در گراف بالا، زمان‌بندی وظایف مختلف نشان داده شده است. وظایف سخت (τ_1 و τ_2) با رنگ نارنجی و وظایف نرم با رنگ سبز نمایش داده شده‌اند. محور افقی زمان و محور عمودی نام وظایف است. وظایف مختلف در بازه‌های زمانی مشخص اجرا شده‌اند و تکمیل شده‌اند.

در گراف پایین، ظرفیت سرور پراکنده در طول زمان نشان داده شده است. محور افقی زمان و محور عمودی ظرفیت سرور است. تغییرات ظرفیت سرور با خطوط بنفش نمایش داده شده‌اند که نشان‌دهنده تجدید ظرفیت در زمان‌های مختلف است.

دستور: skip، **زمان:** ۳

- **وضعیت:** زمان‌بند به مدت ۳ واحد زمانی جلو برده شد.

دستور: soft، **مدت زمان:** ۲

- **وضعیت:** وظیفه نرم با نام Soft1 و مدت زمان ۲ واحد زمانی اضافه شد.

دستور: skip، **زمان:** ۳

- **وضعیت:** زمان‌بند به مدت ۳ واحد زمانی جلو برده شد.
- **وضعیت وظیفه:** وظیفه τ۱ در زمان ۶ تکمیل شد.

دستور: soft، **مدت زمان:** ۲

- **وضعیت:** وظیفه نرم با نام Soft4 و مدت زمان ۲ واحد زمانی اضافه شد.

دستور: skip، **زمان:** ۳

- **وضعیت:** زمان‌بند به مدت ۳ واحد زمانی جلو برده شد.

وضعیت وظیفه: وظیفه τ۲ در زمان ۸ تکمیل شد.
دستور: soft، **مدت زمان:** ۲

- **وضعیت:** وظیفه نرم با نام Soft7 و مدت زمان ۲ واحد زمانی اضافه شد.

دستور: skip، **زمان:** ۸

- **وضعیت وظیفه:** وظیفه τ۱ در زمان ۱۰ تکمیل شد.

دستور: soft، **مدت زمان:** ۲

- **وضعیت:** وظیفه نرم با نام Soft10 و مدت زمان ۲ واحد زمانی اضافه شد.

دستور: skip، **زمان:** ۱

- **وضعیت:** زمان‌بند به مدت ۱ واحد زمانی جلو برده شد.
- **وضعیت وظیفه:** وظیفه τ۲ در زمان ۱۲ تکمیل شد.

دستور: soft، **مدت زمان:** ۱

- **وضعیت:** وظیفه نرم با نام Soft13 و مدت زمان ۱ واحد زمانی اضافه شد.

دستور: end

- **وضعیت:** دستور پایان داده شد اما نادرست وارد شد.

دستور: end

- **وضعیت:** شبیه‌سازی به پایان رسید.
- **وضعیت وظیفه:** وظایف نرم Soft14، Soft15 و وظایف سخت τ۱ و τ۲ در زمان‌های مختلف تکمیل شدند

شرح تغییرات نمودار زمان‌بندی وظایف

نمودار زمان‌بندی وظایف نشان‌دهنده اجرای وظایف سخت و نرم در طول زمان است. در این نمودار، وظایف سخت با رنگ نارنجی و وظایف نرم با رنگ سبز نمایش داده شده‌اند. در ادامه، تغییرات دقیق نمودار زمان‌بندی وظایف توضیح داده شده است:

وظایف سخت

1. **وظیفه τ۱**:
   - شروع: زمان 0
   - پایان: زمان 6
   - وظیفه τ۱ در بازه‌های زمانی 0 تا 2، 4 تا 6، 8 تا 10، 16 تا 18، و 19 تا 21 اجرا شده است.
   - این وظیفه در مجموع 2 واحد زمان اجرا شده و در نهایت در زمان 6 تکمیل شده است.
2. **وظیفه τ۲**:
   - شروع: زمان 0
   - پایان: زمان 7
   - وظیفه τ۲ در بازه‌های زمانی 0 تا 3، 5 تا 8، 12 تا 15، و 18 تا 21 اجرا شده است.
   - این وظیفه در مجموع 3 واحد زمان اجرا شده و در نهایت در زمان 8 تکمیل شده است.

وظایف نرم

1. **وظیفه Soft1**:
   - شروع: زمان 3
   - پایان: زمان 4
   - این وظیفه در بازه زمانی 3 تا 4 اجرا شده و در نهایت در زمان 4 تکمیل شده است.
2. **وظیفه Soft4**:
   - شروع: زمان 6
   - پایان: زمان 8
   - این وظیفه در بازه‌های زمانی 6 تا 8 اجرا شده و در نهایت در زمان 8 تکمیل شده است.
3. **وظیفه Soft7**:
   - شروع: زمان 10
   - پایان: زمان 12
   - این وظیفه در بازه زمانی 10 تا 12 اجرا شده و در نهایت در زمان 12 تکمیل شده است.
4. **وظیفه Soft10**:
   - شروع: زمان 12
   - پایان: زمان 13
   - این وظیفه در بازه زمانی 12 تا 13 اجرا شده و در نهایت در زمان 13 تکمیل شده است.
5. **وظیفه Soft13**:
   - شروع: زمان 13
   - پایان: زمان 14
   - این وظیفه در بازه زمانی 13 تا 14 اجرا شده و در نهایت در زمان 14 تکمیل شده است.
6. **وظیفه Soft14**:
   - شروع: زمان 15
   - پایان: زمان 16
   - این وظیفه در بازه زمانی 15 تا 16 اجرا شده و در نهایت در زمان 16 تکمیل شده است.
7. **وظیفه Soft15**:
   - شروع: زمان 17
   - پایان: زمان 19
   - این وظیفه در بازه‌های زمانی 17 تا 19 اجرا شده و در نهایت در زمان 19 تکمیل شده است.

شرح تغییرات نمودار ظرفیت سرور پراکنده

نمودار ظرفیت سرور پراکنده نشان‌دهنده تغییرات ظرفیت سرور در طول زمان است. این تغییرات ناشی از استفاده از ظرفیت سرور توسط وظایف نرم و تجدید ظرفیت سرور در دوره‌های زمانی مشخص است. در ادامه، تغییرات دقیق نمودار ظرفیت سرور پراکنده توضیح داده شده است:

1. **ابتدای زمان‌بندی** (زمان 0):
   - ظرفیت سرور در ابتدا 3 واحد است.
2. **زمان 1 تا 3:**
   - ظرفیت سرور ثابت می‌ماند و وظیفه‌ای استفاده نمی‌شود.
   - ظرفیت سرور همچنان 3 واحد است.
3. **زمان 4:**
   - وظیفه نرم `Soft1` شروع به استفاده از ظرفیت سرور می‌کند.
   - ظرفیت سرور از 3 واحد به 2 واحد کاهش می‌یابد.
4. **زمان 5:**
   - وظیفه نرم `Soft1` همچنان در حال استفاده از ظرفیت سرور است.
   - ظرفیت سرور از 2 واحد به 1 واحد کاهش می‌یابد.
5. **زمان 6:**
   - وظیفه نرم `Soft1` تکمیل می‌شود.
   - ظرفیت سرور از 1 واحد به 0 واحد کاهش می‌یابد.
   - وظیفه سخت τ_1 نیز در این زمان تکمیل می‌شود.
6. **زمان 7:**
   - ظرفیت سرور تجدید می‌شود.
   - ظرفیت سرور از 0 واحد به 3 واحد افزایش می‌یابد.
7. **زمان 8 تا 10:**
   - وظیفه نرم `Soft4` شروع به استفاده از ظرفیت سرور می‌کند.
   - ظرفیت سرور به ترتیب از 3 واحد به 2 واحد، سپس به 1 واحد و در نهایت به 0 واحد کاهش می‌یابد.
8. **زمان 11:**
   - ظرفیت سرور تجدید می‌شود.
   - ظرفیت سرور از 0 واحد به 3 واحد افزایش می‌یابد.
9. **زمان 12:**
   - وظیفه سخت τ_2 در این زمان تکمیل می‌شود.
   - ظرفیت سرور ثابت می‌ماند (3 واحد).
10. **زمان 13 تا 14:**
    - وظیفه نرم `Soft7` شروع به استفاده از ظرفیت سرور می‌کند.
    - ظرفیت سرور به ترتیب از 3 واحد به 2 واحد و سپس به 1 واحد کاهش می‌یابد.

11. **زمان 15:**
    - ظرفیت سرور تجدید می‌شود.
    - ظرفیت سرور از 1 واحد به 3 واحد افزایش می‌یابد.
12. **زمان 16 تا 18:**
    - وظیفه نرم `Soft10` شروع به استفاده از ظرفیت سرور می‌کند.
    - ظرفیت سرور به ترتیب از 3 واحد به 2 واحد، سپس به 1 واحد و در نهایت به 0 واحد کاهش می‌یابد.
13. **زمان 19:**
    - ظرفیت سرور تجدید می‌شود.
    - ظرفیت سرور از 0 واحد به 3 واحد افزایش می‌یابد.
14. **زمان 20:**
    - وظیفه نرم `Soft14` استفاده از ظرفیت سرور را شروع می‌کند.
    - ظرفیت سرور از 3 واحد به 2 واحد کاهش می‌یابد.
15. **زمان 21:**
    - وظیفه نرم `Soft15` استفاده از ظرفیت سرور را شروع می‌کند.
    - ظرفیت سرور از 2 واحد به 1 واحد کاهش می‌یابد.

جمع‌بندی

این خروجی نشان می‌دهد که چگونه وظایف مختلف (سخت و نرم) توسط زمان‌بند EDF و سرور پراکنده مدیریت شده و در زمان‌های مختلف اجرا و تکمیل شده‌اند. ورودی‌های کاربر برای اضافه کردن وظایف نرم و جلو بردن زمان زمان‌بند تأثیر مستقیمی بر زمان‌بندی و اجرای وظایف داشته‌اند.