OPERATING SYSTEM

CONTENT:

01. concept of the scheduling and the project

02. Introduction to each algorithm.

03. Explanation of the code to each algorithm.

04. The final output.

concept of scheduling

- Scheduling is the process of arranging, controlling, and optimizing work and workloads in a production process or other operation.
- It involves allocating resources and time to tasks to ensure efficient execution and timely completion.
- In the project we used the scheduling to try to find the best suitable way to schedule the processes which enter the **CPU** to get the highest efficient use of it.

The algorithms of the scheduling

01

Round Robin(RR):

Each process is assigned a fixed time slice (quantum) in a cyclic order.

02

Earliest deadline firts(EDF):

Used in real-time systems, processes with the earliest deadlines are scheduled first.

03

First come first serve(FCFS):

Processes are executed in the order they arrive in the ready queue.

04

Shortest Job First (SJF):

The process with the shortest execution time is selected next(with non-preempitive).

05

Shortest Remaining Time First (SRTF):

A preemptive version of SJF where the process with the shortest remaining time is selected next.

The processes list which is used:

- [["A", 0, 5, 20],
  ["B", 3, 10, 30],
  ["c", 4, 12, 40]]

Coding in Python:

- At first create a parent class called SchedulingCLS containing a group of methods some of it are static and common between all algorithms and others are abstract which are uniqe for each algorithm.

- **Init method:** have the main variables and lists with its primary values.
- **Finish method**: contain what will happen when the process end its work and leave the cpu and the ready queue.
- **unFinished method:** it is used when the process hasn't end its work yet, but has left the CPU into the ready queue for any reason depend on the algorithm.

- **to_string:** contain any text we want to print at the end of the project.
- **fit:** (have our processes list as a parameter) it fetches the process list and set a list of the burst time of each process.
- **ready_queue:** check if there is an available process or no, if there isn't it is being idle.
- **coming_after:** check the arrival time list to determine if there any another process will come next.
- **calc:** make any calculations like the average.

-then we will inherit from it the other classes.

```
class RR(Scheduling):
    self.quantam = quantam
    1 usage
    def run(self):
        while (self.c):
            self.readyQueue()
            if self.ready_queue == []:
                continue

            self.process = self.ready_queue.pop(0)
            self.gantt.append(self.process[0])

            # setting the Start Time
            if self.process[0] not in self.start_t:
                self.start_t[self.process[0]] = self.t

            # setting the Response Time
            if self.process[0] not in self.RT:
                self.RT[self.process[0]] = self.t - self.process[1]

            self.rem_bt = self.process[2]
            if self.rem_bt <=self.quantam:
                self.finish()
            else:
                #if any one come when this quantum
                coming = self.coming_after(self.quantam)
                self.unFinish(self.quantam)
                if coming>0:
                    self.readyQueue()
                self.ready_queue.append(self.process)
        self.calc()
```

Round Robin

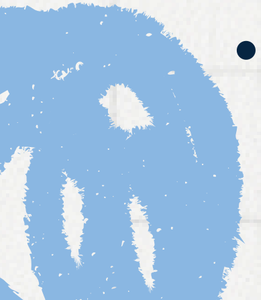
- Define the variable of the quantum.
- **run method:** contains the specific code of the RR.
- First we call the function of ready queue to check if there any process arrived or no.
- if there, we set the process and its time.
- then we compare the burst remaining time of the process with our quantum to determine it’s end or no.
- if yes, we call the finish function.
- if no, we put it to the ready queue and call the unFinish function.

```
order of executing the tasks:  ['A', 'B', 'C', 'A', 'B', 'C', 'B', 'C']
The total cpu time: 27
The turnaround time of each process is:  {'A': 13, 'B': 20, 'C': 23}
The average of turnaround time: 18.667
The waiting time of each process is:  {'A': 8, 'B': 10, 'C': 11}
The average of waiting time: 9.667
The response time of each process is:  {'A': 0, 'B': 1, 'C': 4}
The average of response time: 1.667
The throughput:  0.111  per unit
the cpu utilaization: 100.0  %
The Proportionality:  {'A': 2.6, 'B': 2.0, 'C': 1.9166666666666667}
the States: {'A': 'Succed', 'B': 'Succed', 'C': 'Succed'}
```

The gantt char and the output

Earliest Deadline First

- **dead_method:** to check if the deadline of the process has been exceeded, if yes it’s failed and we kill it.
- **run method:** contains the specific code of EDF.
- we call the function of ready queue to check if there any process arrived or no.
- if there, we first sort the processes by its burst time, then set the smallest process and its time.
- check If there is any proc hasn’t came yet and determine when it will come.
- compare between the remaining burst time and the deadline, if burst exceeded the deadline we run it only till the deadline and kill it after that.



```python
class EDF(Scheduling):
    def __init__(self):
        super().__init__()

    1 usage
    def dead(self,rem):
        if self.process[0] not in self.start_t:
            self.start_t[self.process[0]] = self.t

        # setting the Response Time
        if self.process[0] not in self.RT:
            self.RT[self.process[0]] = self.t - self.process[1]


        self.t+= rem
        self.end_t[self.process[0]] = self.t
        for p in self.ready_queue:
            self.WT[p[0]] +=rem
        #setting the Turnaround Time
        self.TT[self.process[0]] = self.t - self.process[1]
        #setting Proportionality
        self.Proportionality[self.process[0]]=0
        self.c-=1


    1 usage
    def run(self):
        while(self.c):
            self.readyQueue()
            if self.ready_queue ==[] :
                continue
```

```python
        self.process = self.ready_queue[0]
        self.gantt.append(self.process[0])
        self.ready_queue.remove(self.process)
        #📌if deadline over , just  off
        if self.t>= self.process[3]:
            self.dead(self.t-self.process[3])
            continue
        #setting the Start Time
        if self.process[0] not in self.start_t:
            self.start_t[self.process[0]] = self.t
        #setting the Response Time
        if self.process[0] not in self.RT:
            self.RT[self.process[0]] = self.t - self.process[1]
        #📌If there is any proc hasnt came yet I wanna know when next process come
        if (len(self.ready_queue) + 1) == self.c:
            pass
        else:
            rem = self.coming_after(self.process[2])
            if rem > 0:
                self.unFinish(rem)
                self.ready_queue.append(self.process)
                continue
        #📌If deadline will over before brust , off too
        if ((self.t+self.process[2])>self.process[3]):
            rem= self.process[3]-self.t
            self.unFinish(rem)
            self.ready_queue.append(self.process)
            continue
```

```
order of executing the tasks:  ['A', 'A', 'A', 'B', 'C']
The total cpu time: 27
The turnaround time of each process is:  {'A': 5, 'B': 12, 'C': 23}
The average of turnaround time: 13.333
The waiting time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of waiting time: 4.333
The response time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of response time: 4.333
The throughput:  0.111  per unit
the cpu utilaization: 100.0  %
The Proportionality:  {'A': 1.0, 'B': 1.2, 'C': 1.9166666666666667}
the States: {'A': 'Succed', 'B': 'Succed', 'C': 'Succed'}
```

The gantt char and final output

First Come First Serve

```
1 usage
class FCFS(Scheduling):

    def __init__(self):
        Scheduling.__init__(self)

    1 usage
    def run(self):
        while(self.c):
            self.readyQueue()
            if self.ready_queue ==[] :
                continue
            self.process = self.ready_queue.pop(0)
            self.gantt.append(self.process[0])

            #setting the Start Time
            if self.process[0] not in self.start_t:
                self.start_t[self.process[0]] = self.t

            #setting the Response Time
            if self.process[0] not in self.RT:
                self.RT[self.process[0]] = self.t - self.process[1]

            self.finish()
        self.calc()
```

- **run method:** contains the specific code of the FCFS.
- First we call the function of ready queue to check if there any process arrived or no.
- if there, we set the process and its time.
- because it’s non_preempitive, the process don’t leave the cpu until it ends its work.
- we only call the finish function.

```
order of executing the tasks:  ['A', 'B', 'C']
The total cpu time: 27
The turnaround time of each process is:  {'A': 5, 'B': 12, 'C': 23}
The average of turnaround time: 13.333
The waiting time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of waiting time: 4.333
The response time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of response time: 4.333
The throughput:  0.111  per unit
the cpu utilaization: 100.0  %
The Proportionality:  {'A': 1.0, 'B': 1.2, 'C': 1.9166666666666667}
the States: {'A': 'Succed', 'B': 'Succed', 'C': 'Succed'}
```

The gantt char and final output

Shortest Job First

- **run method:** contains the specific code of the SJF.
- First we call the function of ready queue to check if there any process arrived or no.
- if there, we first sort the processes by its burst time, then set the smallest process and its time.
- because it’s non_preempitive, the process don’t leave the cpu until it ends its work.
- we only call the finish function.

```python
from SchedulingCLS import Scheduling


1 usage
class SJF(Scheduling):
    1 usage
    def run(self):
        while(self.c):
            self.readyQueue()
            if self.ready_queue == []:
                continue
            self.ready_queue = sorted(self.ready_queue, key=lambda x: x[2])
            self.process = self.ready_queue[0]
            self.gantt.append(self.process[0])
            self.ready_queue.remove(self.process)

            # setting the Start Time
            if self.process[0] not in self.start_t:
                self.start_t[self.process[0]] = self.t

            # setting the Response Time
            if self.process[0] not in self.RT:
                self.RT[self.process[0]] = self.t - self.process[1]

            self.finish()
        self.calc()
```

```
order of executing the tasks:  ['A', 'B', 'C']
The total cpu time: 27
The turnaround time of each process is:  {'A': 5, 'B': 12, 'C': 23}
The average of turnaround time: 13.333
The waiting time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of waiting time: 4.333
The response time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of response time: 4.333
The throughput:  0.111  per unit
the cpu utilaization: 100.0  %
The Proportionality:  {'A': 1.0, 'B': 1.2, 'C': 1.9166666666666667}
the States: {'A': 'Succed', 'B': 'Succed', 'C': 'Succed'}
```

The gantt char and final output

Shortest Remaining Job First

- **run method:** contains the specific code of the SJF.
- First we call the function of ready queue to check if there any process arrived or no.
- if there, we first sort the processes by its burst time, then set the smallest process and its time.
- because it’s non_preempitive, the process don’t leave the cpu until it ends its work.
- we only call the finish function.

```
class SRF(Scheduling):
    def run(self):
        while(self.c):
            self.readyQueue()
            if self.ready_queue ==[] :
                continue

            self.ready_queue = sorted(self.ready_queue, key=lambda x: x[2])
            self.process = self.ready_queue[0]
            self.gantt.append(self.process[0])
            self.ready_queue.remove(self.process)

            #setting the Start Time
            if self.process[0] not in self.start_t:
                self.start_t[self.process[0]] = self.t

            #setting the Response Time
            if self.process[0] not in self.RT:
                self.RT[self.process[0]] = self.t - self.process[1]

            #📌If there is any proc hasnt came yet I wanna know when next process come
            if (len(self.ready_queue) + 1) == self.c:
                pass
            else:
                rem = self.coming_after(self.process[2])
                if rem > 0:
                    self.unFinish(rem)
                    self.ready_queue.append(self.process)
                    continue

            self.finish()
```

```
order of executing the tasks:  ['A', 'A', 'A', 'B', 'C']
The total cpu time: 27
The turnaround time of each process is:  {'A': 5, 'B': 12, 'C': 23}
The average of turnaround time: 13.333
The waiting time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of waiting time: 4.333
The response time of each process is:  {'A': 0, 'B': 2, 'C': 11}
The average of response time: 4.333
The throughput:  0.111  per unit
the cpu utilaization: 100.0  %
The Proportionality:  {'A': 1.0, 'B': 1.2, 'C': 1.9166666666666667}
the States: {'A': 'Succed', 'B': 'Succed', 'C': 'Succed'}
```

The gantt char and final output

The final file output

```
AWT             FCFS(4.333)     RR(9.667)       SJF(4.333)      SRTF(4.333)     EDF(4.333)
ART             FCFS(4.333)     RR(1.667)       SJF(4.333)      SRTF(4.333)     EDF(4.333)
ATT             FCFS(13.333)    RR(18.667)      SJF(13.333)     SRTF(13.333)    EDF(13.333)
Throughput      FCFS(0.111)     RR(0.111)       SJF(0.111)      SRTF(0.111)     EDF(0.111)
Utilizationuti  FCFS(100.0)     RR(100.0)       SJF(100.0)      SRTF(100.0)     EDF(100.0)
Proportionality FCFS(1.481)     RR(2.074)       SJF(1.481)      SRTF(1.481)     EDF(1.481)

================================================================================================

Policy: FCFS
-----------------------------------------------------
  Task   Start Time   End Time   Duration   Status
-----------------------------------------------------
   A         0           5          5       Succed
   B         5           15         12      Succed
   C         15          27         23      Succed
-----------------------------------------------------

Policy: RR
-----------------------------------------------------
  Task   Start Time   End Time   Duration   Status
-----------------------------------------------------
   A         0           13         13      Succed
   B         4           23         20      Succed
   C         8           27         23      Succed
-----------------------------------------------------
```

```
Policy: SJF
-----------------------------------------------------
  Task   Start Time   End Time   Duration   Status
-----------------------------------------------------
   A         0           5          5       Succed
   B         5           15         12      Succed
   C         15          27         23      Succed
-----------------------------------------------------

Policy: SRF
-----------------------------------------------------
  Task   Start Time   End Time   Duration   Status
-----------------------------------------------------
   A         0           5          5       Succed
   B         5           15         12      Succed
   C         15          27         23      Succed
-----------------------------------------------------

Policy: EDF
-----------------------------------------------------
  Task   Start Time   End Time   Duration   Status
-----------------------------------------------------
   A         0           5          5       Succed
   B         5           15         12      Succed
   C         15          27         23      Succed
-----------------------------------------------------
```

Thank you very much!