Código VH-   
  
Abaixo está um exemplo de código Python que implementa **escalonamento de múltiplas filas (Multilevel Queue Scheduling)** com suporte a diferentes políticas de escalonamento por fila. Este código é modular, pensado para integração com uma interface gráfica (GUI), onde a professora poderá escolher:

* Quantas filas deseja.
* A política de escalonamento de cada fila (por exemplo, FCFS, Round Robin).
* O quantum (se aplicável).
* Os processos e suas filas.

### **✅ FUNCIONALIDADES DO CÓDIGO:**

* Criação dinâmica de filas com diferentes políticas.
* Suporte a políticas:  
  + FCFS (First-Come, First-Served)
  + Round Robin (com quantum configurável)
* Simulação de execução com ordem final.

✅ ESTRUTURA DO CÓDIGO:  
  
class Process:

def \_\_init\_\_(self, pid, arrival\_time, burst\_time, queue\_level):

self.pid = pid

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

self.remaining\_time = burst\_time

self.queue\_level = queue\_level

self.completion\_time = 0

def fcfs\_scheduling(processes):

processes.sort(key=lambda p: p.arrival\_time)

time = 0

result = []

for process in processes:

if time < process.arrival\_time:

time = process.arrival\_time

time += process.burst\_time

process.completion\_time = time

result.append(process)

return result

def round\_robin\_scheduling(processes, quantum):

queue = sorted(processes, key=lambda p: p.arrival\_time)

time = 0

result = []

ready\_queue = []

index = 0

while queue or ready\_queue:

while index < len(queue) and queue[index].arrival\_time <= time:

ready\_queue.append(queue[index])

index += 1

if not ready\_queue:

time = queue[index].arrival\_time

continue

current = ready\_queue.pop(0)

exec\_time = min(current.remaining\_time, quantum)

current.remaining\_time -= exec\_time

time += exec\_time

while index < len(queue) and queue[index].arrival\_time <= time:

ready\_queue.append(queue[index])

index += 1

if current.remaining\_time > 0:

ready\_queue.append(current)

else:

current.completion\_time = time

result.append(current)

return result

def multilevel\_queue\_scheduler(all\_processes, queues\_config):

# queues\_config = [{'policy': 'FCFS'}, {'policy': 'RR', 'quantum': 2}]

scheduled\_processes = []

for level, config in enumerate(queues\_config):

level\_processes = [p for p in all\_processes if p.queue\_level == level]

if config['policy'] == 'FCFS':

result = fcfs\_scheduling(level\_processes)

elif config['policy'] == 'RR':

result = round\_robin\_scheduling(level\_processes, config['quantum'])

else:

raise ValueError(f"Política desconhecida: {config['policy']}")

scheduled\_processes.extend(result)

scheduled\_processes.sort(key=lambda p: p.completion\_time)

return scheduled\_processes

# Exemplo de uso (para ser adaptado na GUI):

if \_\_name\_\_ == "\_\_main\_\_":

processes = [

Process("P1", 0, 5, 0),

Process("P2", 1, 3, 1),

Process("P3", 2, 8, 0),

Process("P4", 3, 6, 1)

]

queues = [

{"policy": "FCFS"},

{"policy": "RR", "quantum": 2}

]

result = multilevel\_queue\_scheduler(processes, queues)

print("Processo | Fila | Tempo de Chegada | Tempo de Execução | Tempo de Conclusão")

for p in result:

print(f"{p.pid}\t\t{p.queue\_level}\t\t{p.arrival\_time}\t\t{p.burst\_time}\t\t{p.completion\_time}")