import re

from typing import List

import heapq

import numpy as np

def SJF(processes: List[List]):

"""

processes: [[process, arrival\_time, burst\_time]]

"""

# Variables to keep track of current time, turnaround times and completion times

current\_time = 0

turnaround\_times = {}

completion\_times = {}

# Ready queue to store processes that are ready to be executed. It is a min heap based on burst time, then arrival time, then process id

ready\_queue = []

def execute\_process():

"""inner function to get the shortest job from ready queue, simulate its execution,

update current time and store completion and turnaround times.

If ready queue is empty, just increments current time and returns.

"""

nonlocal current\_time

if not ready\_queue:

current\_time += 1

return

burst, arrival, process\_id = heapq.heappop(ready\_queue)

print(f"Process {process\_id} started at {current\_time}")

current\_time += burst

print(f"Process {process\_id} finished at {current\_time}")

completion\_times[process\_id] = current\_time

turnaround\_times[process\_id] = current\_time - arrival

# get all processes that have arrived by current time and add them to ready queue, then execute the process with shortest burst time

while processes:

for i in range(len(processes) - 1, -1, -1):

(process\_id, arrival\_time, burst\_time) = processes[i]

if arrival\_time <= current\_time:

heapq.heappush(ready\_queue, (burst\_time, arrival\_time, process\_id))

del processes[i]

execute\_process()

# execute remaining processes in ready queue

while ready\_queue:

execute\_process()

return turnaround\_times, completion\_times

def SRT(origprocesses: List[List]):

'''

pre-emptive version of SJF

processes: [[process, arrival\_time, burst\_time]]

'''

# Variables to keep track of current time, turnaround times and completion times

processes = origprocesses.copy()

current\_time = 0

turnaround\_times = {}

completion\_times = {}

# Ready queue to store processes that are ready to be executed. It is a min heap based on burst time, then arrival time, then process id

ready\_queue = []

def execute\_process():

"""inner function to get the shortest job from ready queue, simulate its execution,

update current time and store completion and turnaround times.

If ready queue is empty, just increments current time and returns.

"""

nonlocal current\_time

if not ready\_queue:

current\_time += 1

return

burst, arrival, process\_id = heapq.heappop(ready\_queue)

if process\_id not in completion\_times:

print(f"Process {process\_id} started at {current\_time}")

completion\_times[process\_id] = current\_time

else:

print(f"Process {process\_id} resumed at {current\_time}")

current\_time += 1

burst -= 1

completion\_times[process\_id] += 1

if burst == 0:

print(f"Process {process\_id} finished at {current\_time}")

turnaround\_times[process\_id] = current\_time - arrival

else:

heapq.heappush(ready\_queue, (burst, arrival, process\_id))

# get all processes that have arrived by current time and add them to ready queue, then execute the process with shortest burst time

while processes:

for i in range(len(processes) - 1, -1, -1):

(process\_id, arrival\_time, burst\_time) = processes[i]

if arrival\_time <= current\_time:

heapq.heappush(ready\_queue, (burst\_time, arrival\_time, process\_id))

del processes[i]

execute\_process()

# execute remaining processes in ready queue

while ready\_queue:

execute\_process()

return turnaround\_times, completion\_times

Text

Description automatically generated with low confidence

For both algorithms, I used a minheap to keep track of which processes are in the queue, and ensure that the shortest job in the queue is always executed. The general steps for each algorithm were:

1. If there are still processes, add all processes that have arrived by the current time.
2. “execute” the first process in the queue, if there are processes in the queue.
3. Increment the program elapsed time by the process time of the executed process, or by one if no programs were in the queue.
4. Once all processes have been added to the queue, keep executing items in the queue until there are no more processes

The only difference between SJF and SRT is that in the SRT algorithm, only 1 second at a time is removed from processes in step 2, then they are added by to the queue and the loop continues. This allows the algorithm to be pre-emptive—cutting up processes in chunks.

The main insight I had was the use of a minheap to keep track of the shortest process left to be executed—this kept me from having to sort the processes and both sped up and simplified the algorithm.

SOURCE CODE:

https://github.com/rilesc555/CS-431---Intro-to-OS.git