PA1 - The ChatGPT Scheduler, Group 31

Steven Grady, Braden Voyles, Jovanny Rebollar, Dallas Moody, Kiyoshi Iwasaki

NOTE: Our github can be found here:
https://github.com/SSGrady/cop4600-pa1-group31/tree/main

Jovanny and Kiyoshi had ChatGPT conversations that could not be shared for some reason (we think because of file uploads). Their .txt forms can be found in the github, along with the code we had for each individual algorithm and the final .py.

Code links:

- Steven: https://chatgpt.com/share/67b8e78c-72e8-8006-9e55-e27113628724
- Braden: https://chatgpt.com/share/67b81d87-7cf8-8003-8551-a2eb8423c451 and https://chatgpt.com/share/67b94268-7f20-8003-8fa7-5b1565d1b252
- Jovanny:https://chatgpt.com/share/67b92b56-7de0-8003-89bc-87f4e3952819 and https://chatgpt.com/share/67b94243-c088-8003-8342-38fd71aff0e0 and Conversation_Jovanny.txt
- Dallas: https://chatgpt.com/share/67b905fb-f0e4-8010-bba3-ca7f0b9d436d
- Kiyoshi: https://chatgpt.com/share/67b91d80-94bc-8013-b80d-7c9e77b67bc3 and Conversation_Kiyoshi.txt

Analysis:

1. The implementations from ChatGPT of Shortest Job First (sjf), First-Come, First Serve (fcfs, a.k.a. First-In First-Out), and Round Robin (rr) are clear. To reach the best result we chose to combine the working schedulers from each group member's committed code into one master folder, ***finalSolution/***, into **scheduler-gpt.py**
    a. We broke the project into bite-sized tasks—sjf, rr, fcfs—split them up, and then collaborated to finalize the solution.
    b. ***Why?*** To maintain efficiency while working asynchronously. ***How?*** Honor system and code check-ins through Discord.
    c. Braden handled FCFS, Steven did SJF, Kiyoshi did RR, Dallas looked for optimizations in RR, and Braden and Jovanny worked together to combine all these results into one final python program.
    d. **Anecdote from an author:** "I took on SJF and began my prompt engineering. I assigned ChatGPT a role, goal, taboos/must-haves, and a long context dump of what I gathered from pa1's requirements. After a lot of back and forth with the AI, I pivoted to manual debugging with stack tracing and print statements. Switching to a "driver" approach in our (ChatGPT and I) pair programming was key, and I was able to validate and push SJF code into group 31's repo. Fun assignment!"
    e. **Anecdote from another author:** "I was responsible for getting FCFS working with ChatGPT. I provided the relevant portions of the assignment for this part, along with sample input and output files, and defined all the variables in the input files. From this, ChatGPT immediately created a correct FCFS scheduler from just my initial prompt alone. The rest of my conversation was spent adjusting the way its output was formatted so that it would match the given expected output

files exactly (no differences are returned when running diff with Ubuntu between its output and the provided expected output). ChatGPT was able to correctly update itself every time I asked it to match a different part of the output, such as the whitespace and the order the tasks are printed in at the bottom. Anecdotally, I know that LLMs actually tend to perform very slightly better when the user is kind to it, such as using “please” and “thank you”, so I was sure to do so (an interesting Forbes article about this can be found here: https://www.forbes.com/sites/bernardmarr/2024/11/05/why-you-should-be-polite-to-chatgpt-and-other-ais/). Because my initial prompt produced the desired outcome, my following iterations focused only on making slight adjustments to its output. Overall, I found this assignment to be a fun and unique way to understand LLM prompting better, in a way even my AI classes don’t allow for. However, when trying to help combine all files together, the results were less optimal. I had to constantly baby ChatGPT and tell it its code wasn’t working, and slowly began feeding it more and more output files until it was able to get somewhere.”

f. **Anecdote from another author**: “I was responsible for getting the RR working with ChatGPT. I was able to properly give it all the guidelines of the project and successfully within a few attempts and some debugging it was able to properly create the RR although at first it had an issue with properly executing the processes. ChatGPT was able to successfully modify the prompt once given enough tries and I pointed out the errors that it was making. Overall, I found this assignment to provide a lot of insight towards the power of using AI within the software engineering role towards solving actual problems that people have “

g. **Anecdote from another author**: “I was in charge of combining the 3 above files into 1 working final file. Despite having 3 working files, ChatGPT seemed to struggle combining them, and all 3 broke in some way. SJF was the hardest, and I needed to spend a long time working on it alone. My prompts were focused on giving ChatGPT the output of the code it made vs the expected output and seeing if it could solve the problem on its own. It worked great at first, but ChatGPT hit a wall when working with writing the output in SJF, and I had to help guide it better. I even tried using Deepseek and Claude at one point, but neither were as effective as ChatGPT. Overall, I think this assignment was a fun way to get experience with LLMs.”

h. **Anecdote from another author**: “I attempted to get ChatGPT to optimize the RR code that it had previously written. This process had mixed results. I fed it the RR code and asked it to find optimizations, and it did so, but without the context of the assignment itself. It did properly fix the data structure, implementing a heap to reduce the waiting time of waiting_list and replacing the process_count variable with simple checks to len(processes), but also had issues with some requirements of the assignment such as running into timeout even if everything has finished by then and changing how events are logged. I also checked its ability to run sample inputs and generate outputs, which it did quite well. Overall, I would say I learned a lot about working with LLMs from this assignment.”

2. **Correctness:** Our final code has tested and validated several edge cases:

a. Multiple processes arriving at the same time
b. Processes finishing with remaining idle CPU time
c. An accompanying .in file not being provided
d. A quantum of None in RR
e. Other edge cases, such as zero values in the input

In all these edge cases and in the provided cases given in the assignment, our output files match exactly, even in white space, to what we were given to expect. So, our code does indeed perform the intended task correctly. Additionally, seeing as the output is always correct, there are no known bugs or errors that we would need to spend time correcting, as we already handled all of those during the prompting process.

3. **Efficiency:** Sorting processes by their arrival times happen multiple times, with an average runtime of ***O(nlogn)***. We have verified that all 3 scheduling algorithms run in ***O(nlogn)*** time, and all other functions run in shorter time (***O(n)*** to read and parse the input, write the output, and run through main). If we had more time the group could have experimented with optimizations that perhaps avoid redundant operations, i.e., ensure the process list gets sorted once during parsing or afterwards.
    a. In ***preemptive_sjf_scheduling()***, pushing and popping from the ***ready_queue*** with a heap is a logical and efficient way that GPT identified when implementing preemptive SJF

4. **Readability:** We believe the code is readable, though not great - namely due to the clear function names and descriptive class fields. Readability could've improved if we took each larger variable name, like ***next_process_tuple***, and assisted them with a previous inline comment. Most of the code is actually uncommented, which greatly reduces the readability of it, and asking ChatGPT to add comments would likely help with this.
    a. Taking time to ensure consistency in formatting/logging style would have also polished up our final code
    b. Due to the way we chose to go about this, 3 different files were created for each of the scheduling algorithms, which were then combined into our final scheduler-gpt.py file and optimized. Had we worked on the entire thing in one conversation, rather than having to combine 3, it would likely have had more consistent stylizing.
    c. ChatGPT notably leaves comments in SJF and RR, but not FCFS. Additionally, when combining the 3 files into the one final one, the comments in RR were actually removed. This was interesting to us, as we hadn't seen ChatGPT ever not leave a comment in any code it creates, so it seems that sometimes you may have to request it to do so, as comments would greatly improve the readability of the code.
    d. Code duplication is avoided, as all 3 processes share a function to write output and take input. Variable names are meaningful and there aren't any useless variables.
    e. The coding style sees a few changes, but nothing major. All indentation stays the same, the largest differences are the lack of comments in large parts of the code

and how sometimes if statement chunks are followed by a blank line, but sometimes they are not.

Overall, we do believe the code is readable, but it leaves much to be desired that could be improved with prompting specifically for matching styles and adding comments.

5. **Completeness:** Parsing function assumes that the input is strictly formatted and correct, but further unit testing on unexpected or missing keywords would have benefited our final submission. Likewise, making our input validation and error handling in **main()** more robust would have improved developer usability - but these ideas were outside scope. In the event of a race condition, where two processes arrive at the same time in FCFS or have the same time to completion in SJF, our program can handle them like so:
    a. In FCFS, whichever one is first in the input is run first when 2 run at the same time (P1 runs before P2 before P3 etc.)
    b. In SJF, the same applies. Additionally, if a process with the same time to completion as the one currently being worked on arrives, it is ignored, as we use less than and not less than or equal to for preempting.

    In the event of a malformed or missing input file, the code simply gives you an error message and ends before actually starting. This can be seen in RR files lacking a quantum, attempts to invoke a scheduling type that does not exist in the code, and having no input file at all included in the run command. Process_count, run_for, scheduling_type, and quantum are also loaded with dummy variables at first, so if they somehow did not get updated, the code could still work without entirely breaking.

6. **Innovation:** The AI wrote the code including “heapq” for preemptive sjf - it’s a solid implementation of Python’s standard libraries. Unfortunately, we do not think that the most recent PEP guidelines were completely followed.
    a. Outside assignment scope, we could have also added a timeline graph (visualization) to visually represent process scheduling beyond the CLI
    b. Outside of this, we did not see anything particularly surprising from ChatGPT, though this is likely because schedulers are an old enough concept that most of the data it is drawing from would be from before any new exciting concepts by today’s standards existed.

7. **Overall Quality:** We would give our architecture a 8/10. It’s well-structured and separates input parsing, scheduling logic, and output handlers into functions. Since our code submission is modular it is easier for onboarded users to read, write, and test.
    a. We think adding docstrings to the process class would have aided in code maintenance and extendability.
    b. Additionally, getting ChatGPT to create comments and normalize its style would greatly improve its readability.
    c. This process was easier at times and harder at times than we expected. The creation of the actual algorithms themselves ended up being incredibly easy, and the hard part was combining them together. This was completely the opposite of how our group expected this to go, and with retrospect, we would likely try to

form it all in 1 file now to begin with in order to remove the process of making it all work together. This would also help make sure it has good programming practices, such as reusing code instead of repeating yourself, but in our case ChatGPT was actually able to solve that on its own in the combining of the files.

d. Our team learned a lot during the process. Notably, we feel we have learned that the first message you send the LLM is by far the most important, and can often dictate how it learns. It's also crucial to correct any mistakes immediately and potentially edit or remove messages to get rid of them altogether. Even if ChatGPT knows something is wrong, if it is in its memory, it can get attached to it, which we saw plenty of in the combining phase.
e. If we had to write code with AI again (which we will in assignment 2), we would not attempt to have smaller pieces come together at the end, but instead all work together, picking up where the previous person left off, in one continuous effort. This would let us use the LLM's strengths to our advantage, as we can use its memory and processing to continue updating the same code instead of forcing it to re-learn the code when going to combine everything together.