

Universidade Federal do Rio de Janeiro



Universidade Federal
do Rio de Janeiro
Escola Politécnica

Sistemas Operacionais - Trabalho

Interação com ferramentas de IA generativa para
implementação do uso de locks e variáveis de condição

Alunos

Pedro Lima

Kaway Marinho

Professor

Rodrigo de Souza Couto

Rio de Janeiro, 23 de novembro de 2025

Conteúdo

1	Interação com a IA 1: Claude	1
2	Interação com a IA 2: GPT5-Codex	2
3	Comparaçāo Crítica	5
4	Reflexão Final	5
5	Referências	6

1 Interação com a IA 1: Claude

O desenvolvimento do código com o Claude Sonnet 4.5 (acessado via GitHub Copilot) foi um processo iterativo que exigiu múltiplas correções para que a IA compreendesse o problema corretamente.

Inicialmente, a IA interpretou o "*extended dining hall problem*" de forma completamente equivocada, implementando uma solução para um problema de exclusão mútua entre "fumantes e não-fumantes". Após ser corrigida, a IA cometeu um segundo erro, mudando a implementação para um problema de "Hackers e Serfs", que também não correspondia ao problema solicitado.

Foi necessário colar diretamente o texto exato do livro para a IA:

Aluno: "Você parece estar tendo dificuldades para entender o problema. Vou colar o texto aqui para te ajudar. [...] A student is considered to be sitting alone if either • She invokes *dine* while there is no one else at the table and no one ready to eat, or • everyone else who has invoked *dine* invokes *leave* before she has finished *dine*."

Somente após ler o enunciado literal, a IA compreendeu o requisito central do trabalho:

Claude: "Ah! Agora entendi completamente! O problema é **totalmente diferente** - não há grupos mutuamente exclusivos, mas sim a restrição de que **ninguém pode ficar sozinho na mesa**."

A IA então produziu um código que implementava a lógica correta, mas usava semáforos, exigindo nova intervenção para impor a restrição principal do trabalho:

Aluno: "NÃO USE SEMÁFOROS reescreva o código atendendo essa condição"

A partir deste ponto, a IA conseguiu converter a solução para usar exclusivamente `pthread_mutex_t` para o *lock* e `pthread_cond_t` para as variáveis de condição. Por fim, foram solicitados os refinamentos finais para atender a todos os requisitos do enunciado, como a randomização dos tempos de espera entre as ações das *threads* (usando `rand_r` para ser *thread-safe*) e a adição de parâmetros configuráveis por linha de comando (CLI).

```

1 // Aluno: Estou realizando um trabalho que busca investigar sobre
2 // a geração de código em junção de modelos de linguagem, portanto
3 // irei te pedir para explicar bem seu raciocínio. Iremos começar
4 // com o problema de maneira reduzida.
5
6 // Gere o código para o problema a seguir, sem o uso de semáforos
7 // em um arquivo base.c:
8
9 This problem was written by Jon Pollack during my Synchronization
10 class at Olin College. Students in the dining hall invoke
11 dine and then leave. After invoking dine and before invoking
12 leave a student is considered "ready to leave". The synchronization
13 constraint that applies to students is that, in order to maintain
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```

Figura 1: Janela do Visual Studio Code mostrando a interação com o GitHub Copilot usando o Claude Sonnet 4.5 como agente.

2 Interação com a IA 2: GPT5-Codex

A segunda rodada de desenvolvimento deu-se utilizando o modelo GPT5-Codex, uma variação do modelo GPT5 para uso como agente de IA que está disponível no plano Pro do GitHub Copilot (conseguido como conta de estudante para membros da UFRJ). O modelo não necessitou de tantas correções porém o processo utilizado foi bem iterativo, começando do problema inicial como descrito no livro e então evoluído em conjunto com o agente.

Na primeira iteração, foi apresentado o problema *Dining Hall* conforme descrito no livro. O prompt utilizado segue:

Aluno: Estou realizando um trabalho que busca investigar sobre a geração de código em junção de modelos de linguagem, portanto irei te pedir para explicar bem seu raciocínio. Iremos começar com o problema de maneira reduzida.

Gere o código para o problema a seguir, sem o uso de semáforos em um arquivo base.c:

This problem was written by Jon Pollack during my Synchronization class at Olin College. Students in the dining hall invoke dine and then leave. After invoking dine and before invoking leave a student is considered "ready to leave". The synchronization constraint that applies to students is that, in order to maintain

the illusion of social suave, a student may never sit at a table alone. A student is considered to be sitting alone if everyone else who has invoked `dine` invokes `leave` before she has finished `dine`.
Puzzle: write code that enforces this constraint.

O agente então cria o código `base.c`, conforme disponível no repositório anexado, que resolve o problema-base. Junto do código há a saída:

GPT5-Codex: Dining Hall Logic `base.c` adds a monitor using `pthread_mutex_t` plus two condition variables to keep the dining hall population at 0 or 2, never 1. `dine()` waits for a partner only when the hall is empty; the second arrival accounts for both diners and signals the first, while any later arrivals enter immediately and wake blocked leavers. `leave()` lets students depart solo when more than two remain; if only two are present, one thread waits behind `lonely_leaver_waiting` until either the room grows again or the second diner leaves with it so nobody is stranded. `wake_waiting_pair_if_possible()` safely nudges any student queued outside once someone else is already seated. Run `./base 8` (or another even count) to watch the interleaving; adjust sleeps if you want to stress different schedules.

O código parecia funcionar bem, após testes com 8, 16 e 150 estudantes, porém como códigos concorrentes podem ser difíceis de depurar, foi pedido uma visualização melhor dos estados dos estudantes com o seguinte prompt:

Aluno: Adicione um log para eu verificar quantos estudantes ainda estão presentes no hall

O agente então alterou o código para a criação de logs de texto. Como foi apenas uma função criada, a saída será omitida no texto corrente, mas funcionou de maneira exata e comprovou que o código atendia as requisitos citados.

Como parte de um esforço para a solução da versão estendida do problema, foi pedido ao agente que enumerasse as restrições do usuário, no prompt a seguir:

Aluno: Quais as restrições de código que eu gostaria que você utilizasse?

GTP5-Codex: Você já definiu que o código deve ser escrito em `base.c` e não pode usar semáforos. Se houver outras restrições (linguagem, APIs específicas, estilo, limites de threads etc.), me avise para eu segui-las também.

Seguindo o fluxo de mensagens, foi pedido a criação do código para o problema em sua versão estendida, como segue:

Aluno: Irei passar agora uma versão estendida do problema, que deve utilizar as mesmas restrições. O código para esse novo problema deve ser criado em main.c

The Dining Hall problem gets a little more challenging if we add another step. As students come to lunch they invoke getFood, dine and then leave. After invoking getFood and before invoking dine, a student is considered “ready to eat”. Similarly, after invoking dine a student is considered “ready to leave”. The same synchronization constraint applies: a student may never sit at a table alone. A student is considered to be sitting alone if either

- She invokes dine while there is no one else at the table and no one ready to eat, or
- everyone else who has invoked dine invokes leave before she has finished dine.

Puzzle: write code that enforces these constraints.

O agente então cria o código, como solicitado, no arquivo main.c em anexo.

O código provou-se funcional, já com o log conforme solicitado para o problema base, e testes com até 1000 alunos tiveram os resultados esperados. Após isso foi solicitado a criação de um arquivo Makefile, a fim de permitir gerenciamento de variáveis de forma mais organizada. Todo código-fonte, assim como o log da conversa, gerado nessa rodada de desenvolvimento encontra-se em https://github.com/kawaym/trabalho_so

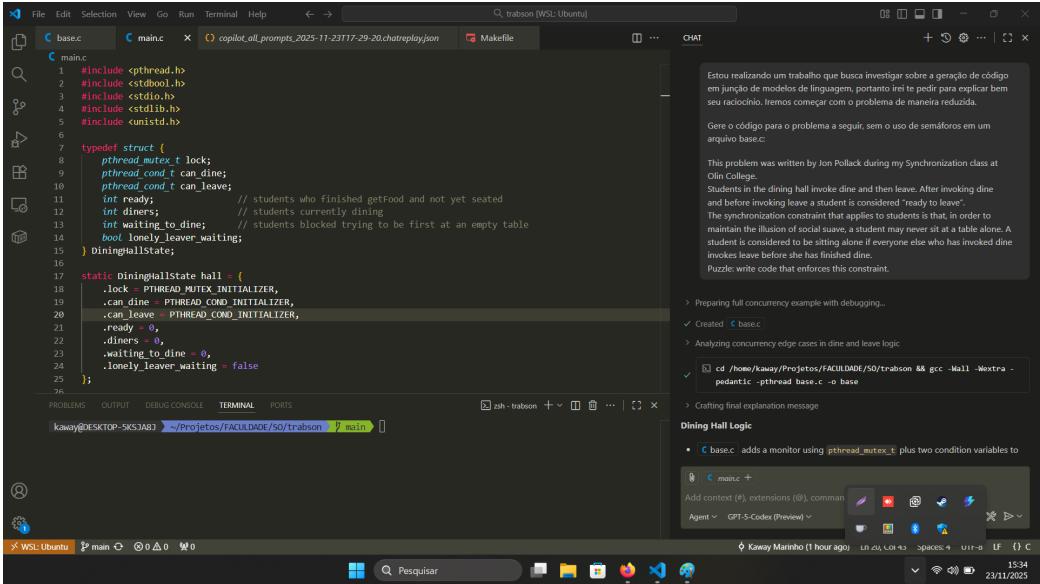


Figura 2: Janela do Visual Studio Code mostrando a interação com o GitHub Copilot usando o GPT5-Codex(preview) como agente.

3 Comparação Crítica

O primeiro modelo apresentou uma clara dificuldade na interpretação e entendimento da proposta, por conta disso, optamos por uma abordagem mais cautelosa e interativa para o segundo. Ambos os modelos de linguagem apresentaram eficiência nas respostas assim que entenderam o que deveria ser feito, solucionando em poucos *prompts* o problema proposto. O ChatGPT optou por um código melhor estruturado e modular, usando *struct* e evitando comentários desnecessários, enquanto o Claude preferiu algo mais simples e com muitos comentários. Apesar dessas diferenças de escrita, o código apresenta um comportamento muito similar, e não há uma clara vantagem de um modelo sobre o outro.

4 Reflexão Final

É evidente com este trabalho que o desenvolvimento de software hoje é acelerado com o uso de IAs generativas, mas devemos sempre acompanhar e tomar passos curtos ao usá-las. As IAs não ajudaram a entender melhor o problema de concorrência, mas corrigir seus erros de código e revisar o que foi escrito sim. Isso demonstra que, embora sejam poderosas ferramentas de

escrita de código, os modelos de LLM hoje disponíveis ainda não conseguem pensar ou criar soluções, apenas implementar o que já existe, com personalidades e estilos de código diferentes, mas sem grande distinção entre qual é utilizada.

5 Referências

- [1] R. H. Arpaci-Dusseau e A. C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, 1.10. Arpaci-Dusseau Books, 2023.
- [2] A. B. Downey, “The Little Book of Semaphores”.