PiazzaHut

Group 2

Emily Liu, Syaam Khandaker, Eric Ma, Edward Chong, Thanh Ngo

Project Introduction

Problem Statement:

Students often struggle to get a quick response on Ed Discussion & Piazza due to limited availability from TAs and instructors who struggle to keep up with the high volume and complexity of student questions, resulting in a backlog of unanswered posts.

Solution:

We introduce PiazzaHut, an autonomous AI-powered assistant which generates instant responses to student questions by learning from the context of past Piazza/Ed Discussion threads. Instructors and TAs can review and approve these answers, ensuring reliability while also reducing their workload.

Project Design

Course-specific Data
Scrape and utilize course-specific data to assist in question-answering

Performance-based model selection
Evaluate and prioritize responses from AI models based on performance metrics (endorsements, feedback, etc.)

Course Specific Application
Ensure that responses use nuanced language, terminology, and recurring challenges for specific class

Related Work:

- Yale’s Bots++, an Ed Discussion bot (GPT-4 based) for answering student questions
- CS 3790 uses an LLM trained on class specific resources to assist students with assignments
- Socratic Mind: an AI-powered language model designed to reinforce student understanding via Socratic questioning

Implementation

Data Pipeline

Sources: Ed Discussion, Piazza
Collection: Python (Beautifulsoup / Selenium)

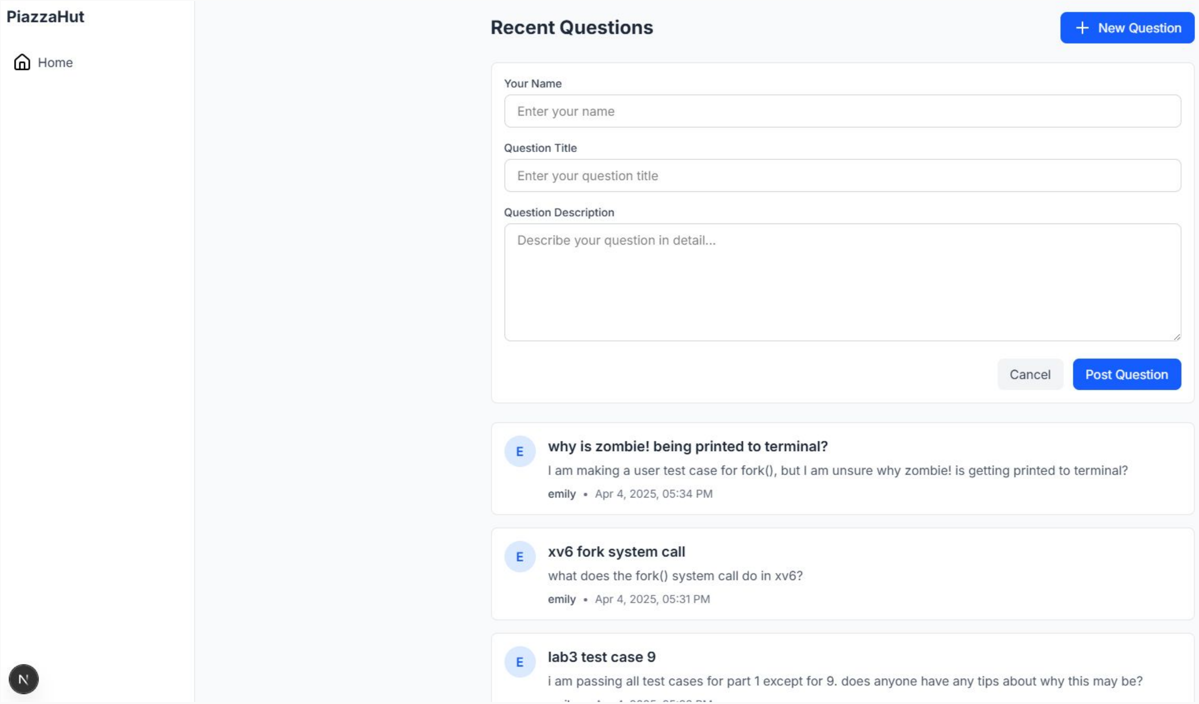
Processing: Chunk information into meaningful sentences before passing the data through an E5 embedding model for dense retrieval. Store on a Pinecone vector database for low-latency similarity search

LLM/Response Generation ("AI-generated response")

Models: DeepSeek, ChatGPT-o1, and Anthropic Claude

Retrieval-Augmented Generation (RAG): As a student posts a question on Ed Discussion, query the Pinecone database for relevant content to submit, along with the question, as context into each LLM for answer generation. Rank and summarize response outputs from all 3 LLMs to produce a final response forwarded to frontend.

DEMO



Recent Questions

Your Name: Enter your name

Question Title: Enter your question title

Question Description: Describe your question in detail...

why is zombie! being printed to terminal?
I am making a user test case for fork(), but I am unsure why zombie! is getting printed to terminal?
emily • Apr 4, 2025, 05:34 PM

xv6 fork system call
what does the fork() system call do in xv6?
emily • Apr 4, 2025, 05:31 PM

lab3 test case 9
i am passing all test cases for part 1 except for 9. does anyone have any tips about why this may be?

Demo - Dataset

Using Selenium and Beautiful Soup, we scraped Piazza posts from CS3210, representing each post as a text file with the following structure:

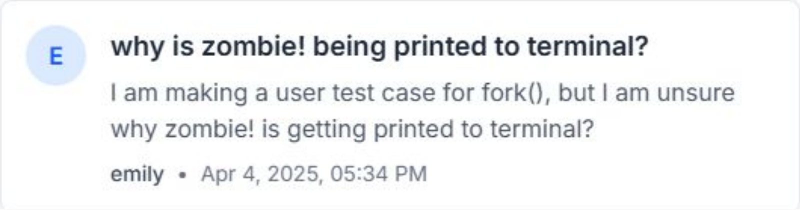
```
Question: …

Student Answer: …

Instructor Answer: …

Followup Discussions: …
```

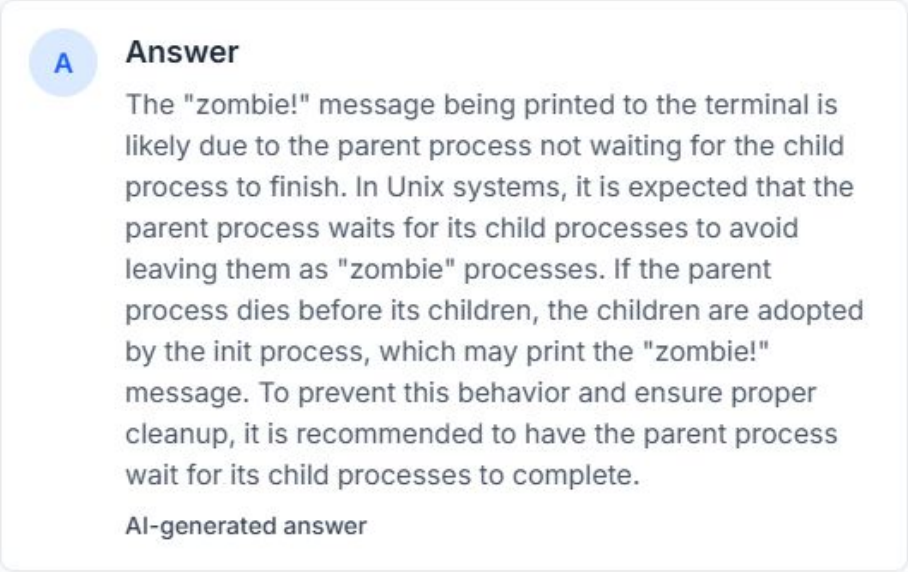
Demo Cases – Easy (straight-forward)



why is zombie! being printed to terminal?

I am making a user test case for fork(), but I am unsure why zombie! is getting printed to terminal?

emily • Apr 4, 2025, 05:34 PM

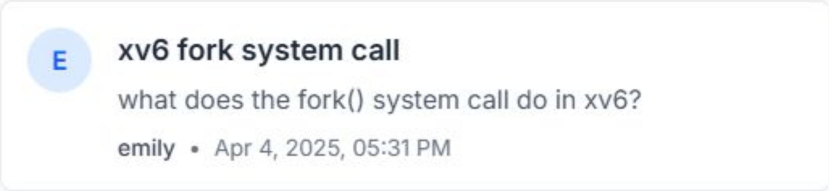


Answer

The "zombie!" message being printed to the terminal is likely due to the parent process not waiting for the child process to finish. In Unix systems, it is expected that the parent process waits for its child processes to avoid leaving them as "zombie" processes. If the parent process dies before its children, the children are adopted by the init process, which may print the "zombie!" message. To prevent this behavior and ensure proper cleanup, it is recommended to have the parent process wait for its child processes to complete.

AI-generated answer

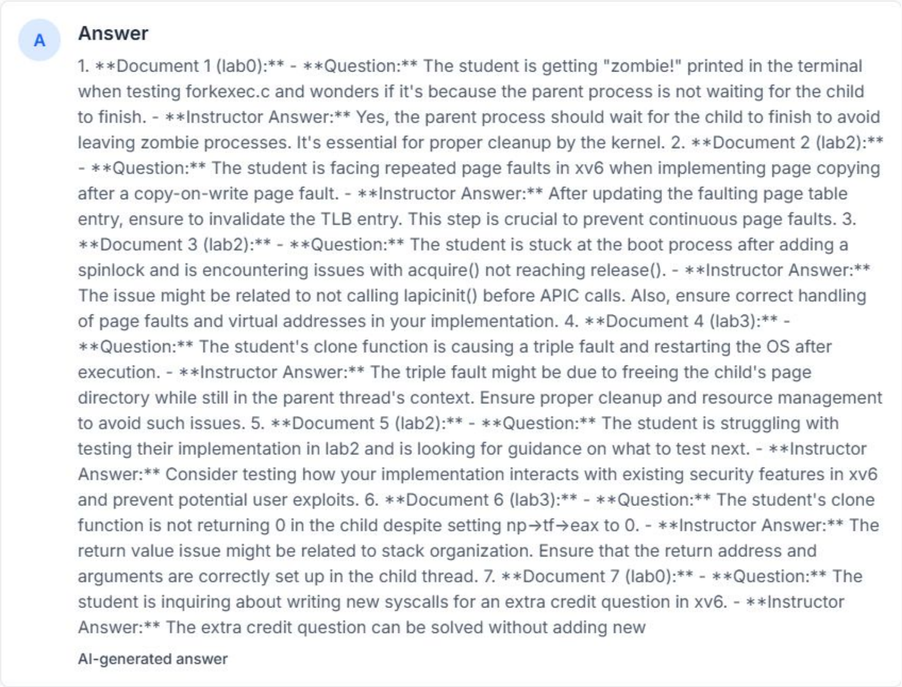
Demo Cases – Average (more conceptual)



xv6 fork system call

what does the fork() system call do in xv6?

emily • Apr 4, 2025, 05:31 PM

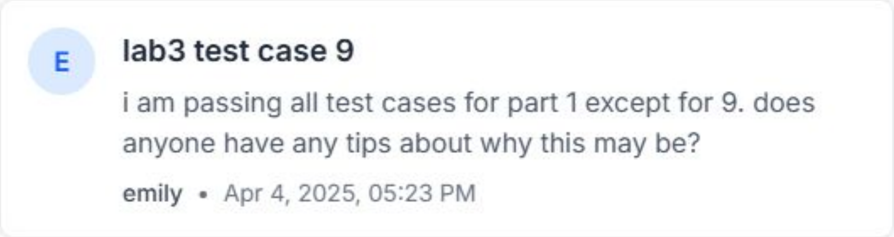


Answer

1. **Document 1 (lab0):** - **Question:** The student is getting "zombie!" printed in the terminal when testing forkexec.c and wonders if it's because the parent process is not waiting for the child to finish. - **Instructor Answer:** Yes, the parent process should wait for the child to finish to avoid leaving zombie processes. It's essential for proper cleanup by the kernel. 2. **Document 2 (lab2):** - **Question:** The student is facing repeated page faults in xv6 when implementing page copying after a copy-on-write page fault. - **Instructor Answer:** After updating the faulting page table entry, ensure to invalidate the TLB entry. This step is crucial to prevent continuous page faults. 3. **Document 3 (lab2):** - **Question:** The student is stuck at the boot process after adding a spinlock and is encountering issues with acquire() not reaching release(). - **Instructor Answer:** The issue might be related to not calling lapicinit() before APIC calls. Also, ensure correct handling of page faults and virtual addresses in your implementation. 4. **Document 4 (lab3):** - **Question:** The student's clone function is causing a triple fault and restarting the OS after execution. - **Instructor Answer:** The triple fault might be due to freeing the child's page directory while still in the parent thread's context. Ensure proper cleanup and resource management to avoid such issues. 5. **Document 5 (lab2):** - **Question:** The student is struggling with testing their implementation in lab2 and is looking for guidance on what to test next. - **Instructor Answer:** Consider testing how your implementation interacts with existing security features in xv6 and prevent potential user exploits. 6. **Document 6 (lab3):** - **Question:** The student's clone function is not returning 0 in the child despite setting np→tf→eax to 0. - **Instructor Answer:** The return value issue might be related to stack organization. Ensure that the return address and arguments are correctly set up in the child thread. 7. **Document 7 (lab0):** - **Question:** The student is inquiring about writing new syscalls for an extra credit question in xv6. - **Instructor Answer:** The extra credit question can be solved without adding new

AI-generated answer

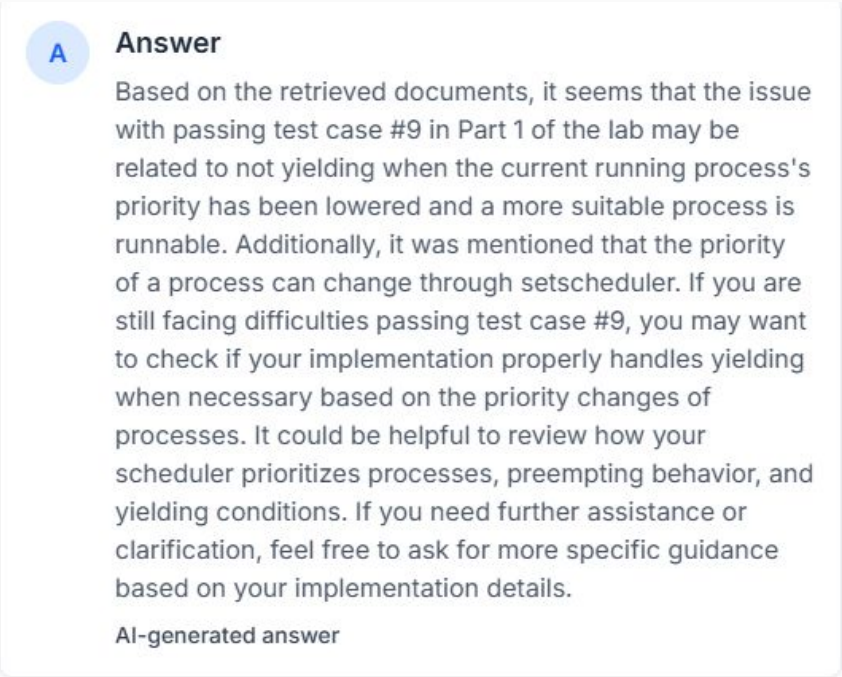
Demo Cases – Hard (course-specific)



lab3 test case 9

i am passing all test cases for part 1 except for 9. does anyone have any tips about why this may be?

emily • Apr 4, 2025, 05:23 PM

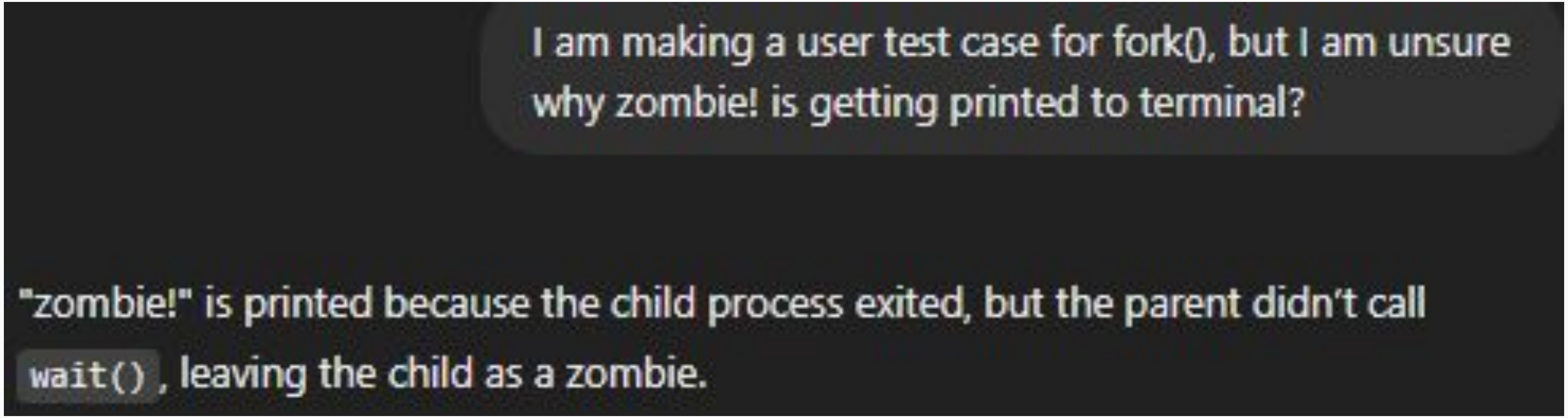


Answer

Based on the retrieved documents, it seems that the issue with passing test case #9 in Part 1 of the lab may be related to not yielding when the current running process's priority has been lowered and a more suitable process is runnable. Additionally, it was mentioned that the priority of a process can change through setscheduler. If you are still facing difficulties passing test case #9, you may want to check if your implementation properly handles yielding when necessary based on the priority changes of processes. It could be helpful to review how your scheduler prioritizes processes, preempting behavior, and yielding conditions. If you need further assistance or clarification, feel free to ask for more specific guidance based on your implementation details.

AI-generated answer

Evaluation – Easy Case

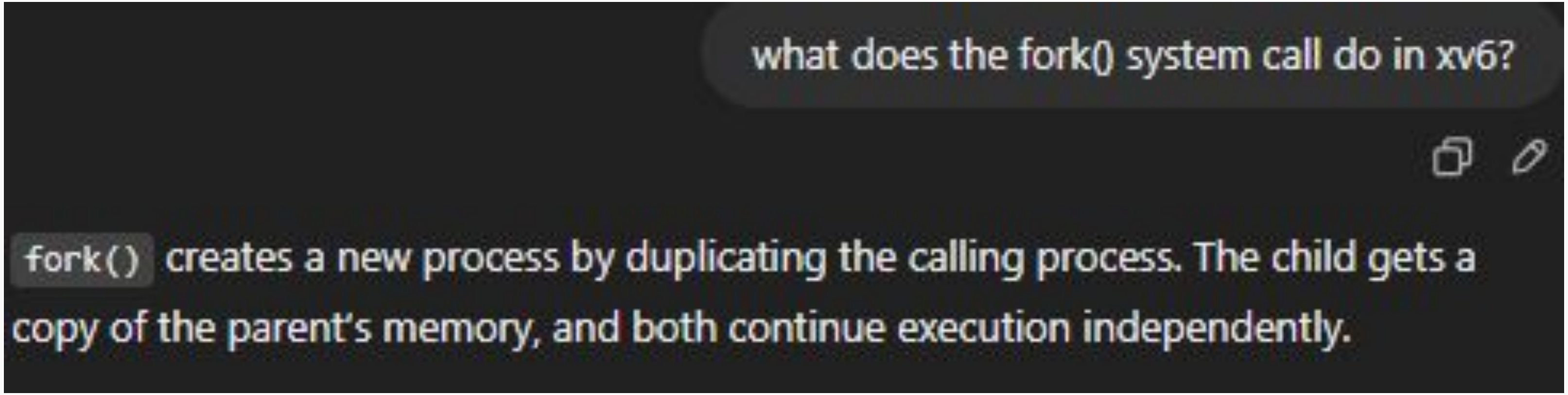


I am making a user test case for fork(), but I am unsure why zombie! is getting printed to terminal?

"zombie!" is printed because the child process exited, but the parent didn't call `wait()`, leaving the child as a zombie.

ChatGPT provides a clear and concise answer, similar to our model.

Evaluation – Average Case

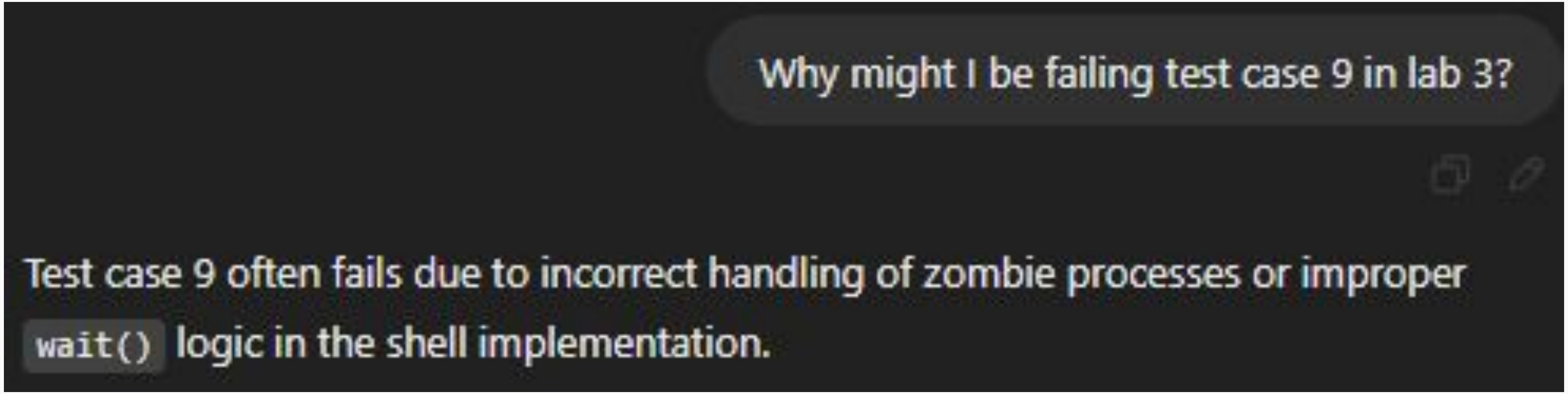


what does the fork() system call do in xv6?

`fork()` creates a new process by duplicating the calling process. The child gets a copy of the parent's memory, and both continue execution independently.

For more conceptual questions that are not course-specific, ChatGPT is still able to provide high quality answers.

Evaluation – Hard Case



Why might I be failing test case 9 in lab 3?

Test case 9 often fails due to incorrect handling of zombie processes or improper `wait()` logic in the shell implementation.

Without any context, ChatGPT hallucinates and makes up a response. Students are unable to use this response to assist them in their projects.

Plans for Additional Evaluation

- **Methods**

  Quantitative Testing: Simulated User Questions
  - Sample past questions from resolved Ed Discussion/Piazza threads for testing (exclude answer responses from LLM queries)
  - Compare PiazzaHut's AI-generated responses against verified answers

  Qualitative Expert Review
  - Collect feedback from TAs on AI-generated responses via ratings for accuracy (incorrect/correct) and relevance (on/off topic) on a 5-point scale
  - Survey question: Would you endorse this answer? Yes/No

- **Metrics/Datasets**
  - Capture data for accuracy (% of responses matching verified answers), efficiency (average response time), and endorsements (% of responses endorsed)

Summary + Lessons Learned

Using data scraped from Piazza, we were able to answer more course-specific questions that the typical LLM could provide. However, there is still much to improve on the quality of answers.

Lessons Learned/Next Steps:

- Experiment with different prompts to see if providing varying instructions can lead to better responses, especially for more straight-forward/average cases
- Evaluate the quality of responses from additional LLMs such as DeepSeek and Claude, potentially combining all three as the final answer
- Continuous testing and debugging with more classes

Thank you for listening!

Are there any questions/feedback?