# TASK 5

# Detail about Task 3

In Task 3, I took a deep dive into the Linux commands that serve as the backbone for file manipulation. I analyzed the cp, rm, and cat commands, unveiling their operational mechanics and pinpointing avenues for refinement. The cp command, in particular, became the center of my innovation. I retrofitted it with a real-time progress bar, meticulously integrating it to relay detailed feedback during file transfer operations. This enhancement meticulously counts the bytes, offering users a transparent view of the copy speed and the total elapsed time, thus morphing a mere utility into an interactive experience. The culmination of this task was not only a technical triumph but also a stride towards user-centric command-line interfaces, marrying the robustness of Linux with the intuitiveness demanded by modern users.

# Detail about Task 4

In Task 4, We identified a susceptible gets function and developed a payload to manipulate the program's execution flow. With a careful blend of theory and practice, we navigated through memory analysis using GDB to locate the precise injection point for our exploit. We demonstrated a buffer overflow attack by exploiting the gets function in a controlled environment. We pinpointed this vulnerability through code analysis gets notoriously lacks input size validation, leading directly to a buffer overflow if the input exceeds buffer capacity. We constructed a tailored payload with ample 'A' characters to fill the buffer, followed by a crafted return address and NOP sled leading to our shellcode. The payload was injected during the gets call, overwriting the return address and diverting execution flow to our shellcode, which successfully initiated a shell. This practical exercise served as a stark reminder of the criticality of secure coding practices and the potential perils of neglecting them. This process underlined the importance of understanding buffer overflow vulnerabilities and led us to devise robust countermeasures, such as bounds checking and stack canaries, to fortify our program against such attacks.