FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ

AYMAN SAEID

2221221366

Distributed Calculator Using Pipes and Fork

Content

1- Project Topic

Distributed Calculator Using Pipes and Fork

This project involved the creation of a distributed calculator system implemented in C.
The system performs basic arithmetic operations (addition, subtraction, multiplication, and division) using separate processes for each operation. The main program communicates with these subprocesses through pipes to send data and retrieve results. Additionally, a separate saver process stores the results in a text file.

2- Tasks Completed During the Project

a. Main Program (Calculator)

- Designed a menu-driven interface for the user to select arithmetic operations.
- Used the fork() system call to create child processes for distributed computation.
- Managed inter-process communication using unidirectional pipes:

- **Parent-to-Child Pipe:** Sent user inputs (two numbers) to the subprocess responsible for the selected operation.
- **Child-to-Parent Pipe:** Sent the computed result back to the main program.

- Ensured proper synchronization and clean-up of resources such as closing unused pipe ends.

b. Operation Subprocesses

➢ Created four separate programs:

- sum2numbers.c: Handles addition.
- subtract2numbers.c: Handles subtraction.
- multiply2numbers.c: Handles multiplication.
- divide2numbers.c: Handles division and error-checking for division by zero.

➢ Each subprocess received input via a pipe, performed the calculation, and returned the result to the parent process.

➢ Integrated the execl function for executing operation-specific logic seamlessly.

c. Saver Process

➢ Developed a dedicated saver.c program that:

- Accepted the result as a command-line argument from subprocesses.
- Appended the result to a results.txt file for record-keeping.

➢ Ensured this subprocess was invoked by each arithmetic subprocess immediately after computing the result.

d. Error Handling

➢ Implemented error-checking for all critical operations:

- Ensured pipes were successfully created.
- Verified successful forking of subprocesses.
- Checked for invalid inputs and division by zero.
- Added fallbacks for failed execl calls.

e. Testing and Debugging

➢ Conducted rigorous testing to verify the accuracy of computations and proper functionality of inter-process communication.

➢ Debugged issues related to pipe communication, ensuring data integrity between processes.

➢ Verified the persistence of results in results.txt.

3- Additional Notes

Key Challenges:

- ✓ Synchronizing parent and child processes to avoid data corruption.
- ✓ Designing modular subprocesses to handle specific operations efficiently.

Key Learnings:

- ✓ Deepened understanding of Unix system programming, especially with fork(), pipe(), and execl().
- ✓ Gained experience in designing distributed systems with inter-process communication.

Future Enhancements:

- ✓ Extend functionality to include more operations like modulus or exponentiation.
- ✓ Implement advanced error logging for debugging complex scenarios.
- ✓ Add a user-friendly graphical interface for better usability.

4- References

Class lecture notes on system programming.

GeeksforGeeks: Inter-Process Communication (IPC).