

National University of Computer and Emerging
Sciences Chiniot-Faisalabad Campus



Lab 05

CL2006 – Operating System - Lab

Course Instructor	Ms Juhinah Batool
Lab Instructor	Ms Juhinah Batool
Semester	Fall 2024

FAST School of Computing

Department of Artificial Intelligence & Data Science

Instructions

1. Make a PDF document with the convention “ROLLNO_ LAB#_ SECTION” and put all your source code and snapshots of its output in it.
2. Plagiarism is strictly prohibited, if you take a code snippet off the internet, mention its reference.
3. Do not discuss solutions with one another. Copying the solution from any source can lead to ZERO marks.

Lab Tasks

Task 1

Task: Implement Inter-Process Communication Among Parent and Multiple Child Processes

Objective: Write a program that demonstrates inter-process communication using a parent process and three child processes, each performing distinct operations.

Requirements:

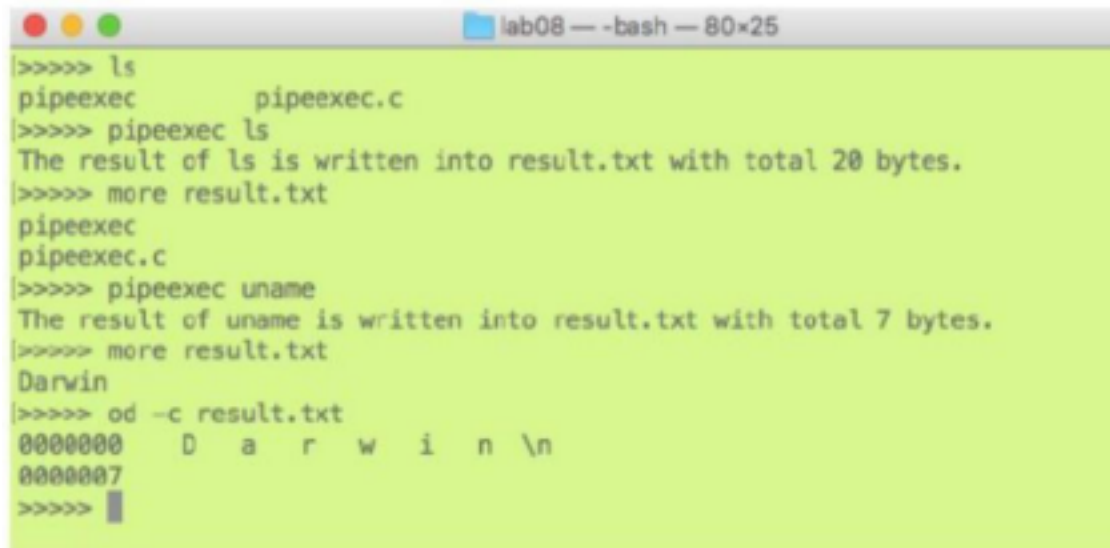
1. **Process Structure:**
 - Create a parent process and three child processes (C1, C2, and C3).
2. **User Input:**
 - The parent process will prompt the user for two numbers.
3. **Operation Assignment:**
 - The parent process will send the numbers to C1 for addition and subtraction.
 - After receiving the results from C1, the parent process will take another set of inputs from the user.
 - Send this second set of numbers to C2 for multiplication.
 - Finally, the parent will send the same numbers to C3 for division.
4. **Creating Additional Child:**
 - After completing the operations, C1 will create another child process, C4, to handle the display of results.
5. **Result Management:**
 - C1 will send the results of addition and subtraction to C4.
 - C2 will send the multiplication result to C4.
 - C3 will send the division result to C4.
6. **Output Handling:**
 - C4 will output all results collected from C1, C2, and C3.
7. **Process Termination:**
 - After the results are displayed, the parent process will terminate all child processes (C1, C2, C3, and C4).

Note: Please Use attached task1.c file for implementation. You are required to add you logic only in places where it is mentioned todo.

Task 2

Write a C program with the parent process and a child process communicating with a pipe. The child process will execute the shell command provided by the user via command line arguments. The result of executing this shell command is passed to the parent process using a pipe. The parent process will write the result into a file called result.txt and acknowledge the user on the screen with the shell command and the total number of bytes in the result. For simplicity, the shell command

contains only the command name, no argument. You can only use read, write, close for pipe operation. Sample run:

A terminal window titled 'lab08 -- bash -- 80x25' with a light green background. It shows the execution of a program called 'pipeexec'. The user enters '>>>> ls' and the program responds with 'pipeexec pipeexec.c'. Then the user enters '>>>> pipeexec ls' and the program responds with 'The result of ls is written into result.txt with total 20 bytes.' followed by '>>>> more result.txt' and 'pipeexec pipeexec.c'. Next, the user enters '>>>> pipeexec uname' and the program responds with 'The result of uname is written into result.txt with total 7 bytes.' followed by '>>>> more result.txt' and 'Darwin'. Finally, the user enters '>>>> od -c result.txt' and the program outputs '00000000 D a r w i n \n' and '00000007'. The prompt '>>>>' is shown again at the bottom.

```
>>>> ls
pipeexec      pipeexec.c
>>>> pipeexec ls
The result of ls is written into result.txt with total 20 bytes.
>>>> more result.txt
pipeexec
pipeexec.c
>>>> pipeexec uname
The result of uname is written into result.txt with total 7 bytes.
>>>> more result.txt
Darwin
>>>> od -c result.txt
00000000  D  a  r  w  i  n  \n
00000007
>>>>
```

Task 3

Task: Implementing Named Pipe Communication with Two Programs

You will write two simple programs: `my_msg_reader.c` and `my_msg_writer.c` that use a named pipe to communicate.

Program Details

1. **`my_msg_reader.c`:**
 - This program will create a named pipe using `mkfifo()`.
 - It will open the named pipe for reading.
 - The program will read messages from the pipe until it receives the string "quit".
2. **`my_msg_writer.c`:**
 - This program will open the named pipe for writing.
 - It will read messages from the user and write them to the named pipe.
 - When the user enters "quit", the program will write this string to the pipe and then exit.

Execution Example

The execution should look something like this (note that you must start the reader first):

Reader:

```
$ ./my_msg_reader
Creating named pipe: /tmp/my_msg_pipe
Waiting for input... Got it: "Hello, World!"
Waiting for input... Got it: "How are you?"
Waiting for input... Got it: "quit"
Exiting
```

Writer:

```
$ ./my_msg_writer
Opening named pipe: /tmp/my_msg_pipe
Enter Input: Hello, World!
Writing buffer to pipe... done
Enter Input: How are you?
Writing buffer to pipe... done
Enter Input: quit
Writing buffer to pipe... done
Exiting
```

Task 4:

Task: Implementing Shared Memory Communication Between Processes

Write a program that creates a shared memory segment and waits for two other separate processes to write into that shared memory. After both processes have written to the shared memory, the main process will print the contents.

Program Details

1. Shared Memory Segment:

- Create a shared memory segment that can hold at least two strings (e.g., one for a greeting and another for a number).

2. **Process 1 (Main Process):**
 - Write the string “Welcome” to the first position of the shared memory.
 - Wait for the other processes to write their data.
3. **Process 2:**
 - Write the number 42 to the first position of the shared memory.
 - Then write the string “to the class” in the second position of the shared memory.
4. **Process 3:**
 - Write the number 1001 to the first position of the shared memory.
 - Then write the string “student ID” in the second position of the shared memory.
5. **Final Output:**
 - After both processes have written their data, Process 1 will print all the contents of the shared memory.

Example Execution

Assuming that you manage the process execution correctly, the final output of Process 1 should look something like this:

```
Process 1: Welcome
Process 2: 42 - to the class
Process 3: 1001 - student ID
```

Notes:

- Ensure proper synchronization between the processes to avoid race conditions.
- Use appropriate system calls to create and manage shared memory.