# Operating System Lab

Lab 5

Inter Process Communication(IPC)

(PIPE Concept)

# What is IPC ?

- When two or more processes comunicate with each other, this event is called IPC.

# Types of IPC

- Among related processes on same system (PIPE).
- Among unrealted processes on same system (FIFO).
- Among unrealted processes on physically different systems (SOCKET).

# PIPE

- PIPE Pipes permit sequential communication from one process to a related process.
- A PIPE permits unidirectional communication.
- PIPE is a temporary file, which has different file-descriptors for reading and writing.
- Data written to the “write end” of the pipe is read back from the “read end” .
- Typically, a pipe is used to communicate between two threads in a single process or between parent and child processes.

# PIPE As a Command

- ls -l | wc -l.
- Shell produces two child processes, one for ls and one for wc.
- The shell also creates a pipe connecting the standard output of the ls process with the standard input of the wc process.
- The filenames listed by ls are sent to wc in exactly the same order as if they were sent directly to the terminal.

# Process Synchronization with PIPE

- A pipe’s data capacity is limited.
- If the pipe cannot store more data, the writer process blocks until more capacity becomes available.
- If the reader tries to read but no data is available, it blocks until data becomes available.
- Thus, the pipe automatically synchronizes the two processes.

# PIPE As a System Call

- Create int array of two elements: *int pipe_fds[2]*.
- System Call: *pipe(pipe_fds)*.
  - Creates temporary file with two file-descriptors.
  - pipe_fds[0] ->Read file descriptor
  - pipe_fds[1] ->Write file descriptor

# Using PIPE For IPC

- Create a PIPE.
- Create a new child.
- Both ends of file-descriprors will be copied into child.
- Close read end from writer process.
- Close write end from reader process.
- Write something on PIPE from writer process.
- Read that in reader process.
- You can use dup2 to connect read end of PIPE with stdin and write end with stdout in corresponding processes.

# FIFO (Named PIPE)

- Using FIFO IPC between two unrelated process are possible.
- Create a PIPE with name.
- We can access this file like an ordinary file.

# FIFO As a Command

- mkfifo /tmp/fifo.
- See properties ls -l /tmp/fifo
- From terminal 1 redirect cat output:
  - cat > /tmp/fifo //any command instead of cat.
- From terminal 2 redirect cat input:
  - cat < /tmp/fifo
- Type on terminal 1.
- Get same thing on terminal 2.
- Remove /tmp/fifo
  - rm /tmp/fifo

# FIFO As a System Call

- mkfifo(“/tmp/fifo”, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH).
- Including reader and writer, any one can create this FIFO.
- Open like normal file in writer process:
  - f = open(“/tmp/fifo”, O_WRONLY);
- Open like normal file in  process:
  - f = open(“/tmp/fifo”, O_RDONLY)

# Task 1

Write a program in which parent is  a writer and child is a reader using PIPE.

# Task 2

Repeat Task 2 using dup2. Duplicate read descriptor of PIPE with stdin in reader process, and write descriptor of PIPE with stdout in writer process

# Task 3

Write a program in which two children of same parent are communicating using PIPE.

# Task 4

Write a program to show IPC using FIFO.

Calculate the factorial of a number using FIFO.

# Task 5

Write a program to show producer-consumer problem in shared memory, so that at most BUFFER_SIZE - 1 buffer can be used.

(see book 3.4.1)

# Task 6

Solve above problem, and use full buffer.

(see book 6.1)

# Task 7 (Shared Memory)

Four external processes will communicate temperatures to a central process, which in turn will reply with its own temperature and will indicate whether the entire system has stabilized. Each process will receive its initial temperature upon creation and will recalculate a new temperature according to two formulas:

***new external temp = (myTemp * 3 + 2 * centralTemp) I 5;***

***new central temp = (2 * centralTemp +***

***four temps received from external processes) I 6;***

Initially, each external process will send its temperature to the mailbox for the central process. If all four temperatures are exactly the same as those sent by the four processes during the last iteration, the system has stabilized. In this case, the central process will notify each external process that it is now finished (along with the central process itself), and each process will output the final stabilized temperature. If the system has not yet become stable, the central process will send its new temperature to the mailbox for each of the outer processes and await their replies. The processes will continue to run until the temperature has stabilized.

# Task 8

Repeat above program using Named Pipe (FIFO).