CSE2005 - Operating Systems

Module 4 –L2

Process Synchronization

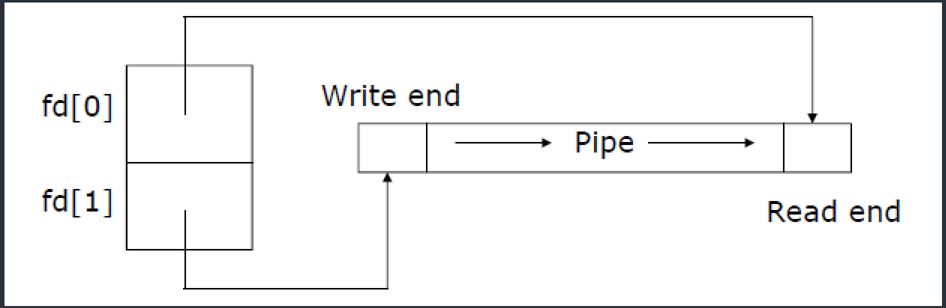
Dr. Rishikeshan C A
VIT Chennai

IPC in Unix

- The pipe is the IPC mechanism in UNIX
- It provides a reliable unidirectional byte stream between two processes
- Created by the system call pipe().
- Each end of a pipe has an associated file descriptor.
- Pipe is of Fixed size - it buffers the output of the writer and suspends the writer if the pipe gets too full
- Usually kept in memory by the normal block buffer cache
- In FreeBSD, pipes are implemented as a special case of the socket mechanism
- The socket mechanism can be used by unrelated processes

Pipes

- The following data structure would be created when the a pipe system call is executed:
  - int fd[2];
  - pipe(fd);



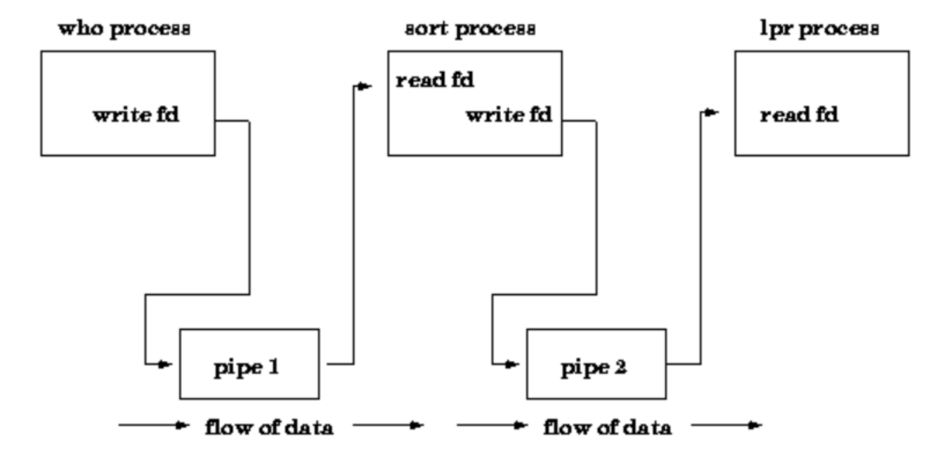
Source* - Abraham Silberschatz, Peter B. Galvin, Greg Gagne-Operating System Concepts, Wiley (2018

Unix Pipes

- There are two kinds of pipes
  - unnamed pipes
    - used for communication between a parent process (creating the pipe) and its child, with one process writing and the other process reading
  - named pipes
    - Any process can communicate with another using named pipes

Unnamed pipes

- The typical sequence of events for a communication is as follows:
  - The parent process creates an unnamed pipe using “pipe()”
  - The parent process forks
  - The processes communicate by using “write()” and “read()” calls
  - Each process closes its active pipe descriptor when it’s finished with it



Named Pipes

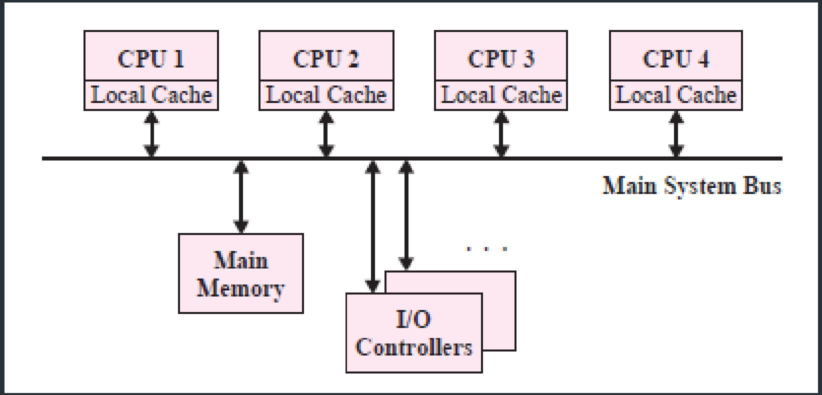
- Named pipes- often referred to as FIFOs(first in, first out)
- They have a name that exists in the file system
- They have a larger buffer capacity, typically about 40K.
- unidirectional
- Named pipes exist as special files in the file system and may be created in one of two ways:
  - by using the UNIX mknod utility
  - by using the “mknod()” system call

Close and remove a pipe

- When a process has finished using a named pipe, it should close it using “close()”
- when a named pipe is no longer needed, it should be removed from the file system using “unlink()”.
- Writer processes should open a named pipe for writing only, and reader processes should open a pipe for reading only.

Multiprocessing

- Multiprocessing systems are those that run multiple CPUs in a single system
- Multiprocessing is a less expensive option because a single system can share many expensive hardware components such as power supplies, primary and secondary storage, and the main system bus.



A simplified multiprocessor system architecture

Source* - Abraham Silberschatz, Peter B. Galvin, Greg Gagne-Operating System Concepts, Wiley (2018

Types of Multiprocessing

- Asymmetric multiprocessing
  - The OS runs on only one designated CPU.
  - The other CPUs run only applications.
  - not commonly used because of performance bottlenecks due to running the OS only on one processor.

- Symmetric multiprocessing (SMP)
  - OS can be running on any CPU
  - A running program obviously will be modifying its state (data
  - Multiple instances of the OS running on different CPUs must be prevented from changing the same data structure at the same time.

Locking

Why locks?

- In multiprocessing, different tasks run concurrently on different CPUs but concurrency can cause inconsistent results if multiple tasks try to use the same variables at the same time
- In order to make it consistent, serializability is required
- To ensure serializability, locks are used
- A lock is an object with two operations: acquire(L) and release(L)
- The lock has state: it is either locked or not locked
- A call to acquire(L) waits until L is not locked, then changes its state to locked and returns
- A call to release(L) marks L as not locked

Drawbacks:

- Locks can ruin performance

Lock free coordination

why do we want to avoid locks?

- Locks limit scalability
- Degrades performance
- complexity
- Possibility of deadlock
- priority inversion

- A lock-free data structure can be used to improve performance.
- A lock-free data structure increases the amount of time spent in parallel execution rather than serial execution, improving performance on a multi-core processor, because access to the shared data structure does not need to be serialized to stay coherent.

Lock free coordination

- A lock free algorithm protects a shared data structure through a non-blocking algorithm.
- A **non-blocking algorithm** ensures that threads competing for a shared resource do not have their execution indefinitely postponed by mutual exclusion.
- A non-blocking algorithm is **lock-free** if there is guaranteed system-wide progress and **wait-free** if there is also guaranteed per-thread progress

References

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne-Operating System Concepts, Wiley (2018).
2. Ramez Elmasri, A.Gil Carrick, David Levine, Operating Systems, A Spiral Approach - McGrawHill Higher Education (2010).
3. https://pdos.csail.mit.edu/6.828/2012/lec/l-lockfree.txt
4. https://en.wikipedia.org/wiki/Non-blocking_algorithm

Thank you