

Synchronization

Blocking or non blocking →

↓
Sender waits for the proper delivery of the message

Synchronous mode of comm.

Sender continues to do whatever it wants to do
asynchronous

Receiver will not check, it just tries to receive a message from mailbox/shared mem/buffer etc.

Synchronous: Waits till a message is available

(NO risk of worrying "what if there is nothing to send/receive")

asynchronous has that worry

Buffering

if sender is yet to send/receiver is not ready

| | Zero Capacity | Bounded | Unbounded |
|--------------------------------------|-------------------------------------|---|-----------|
| | blocking | N messages | ∞ queue |
| Sender blocks till receiver receives | (no link can have waiting messages) | ($(n+1)^{th}$ message can't be in link) ↳ sender blocks | |

Pipe

Pipe
defn

- Conduit to allow two processes to communicate
- flow from source to a destination via a Channel
- One is source (producer) one is dest (Consumer)
Writes to pipe Reads from pipe

Concerns

Unidirectional

Ordinary
Pipe

↳ write end
read end

Requires

Parent-child } Child
relationship } inherits pipe

(first the pipe creation
is done)

OR Bidirectional

Named Pipe

Not destroyed when creator
is terminated

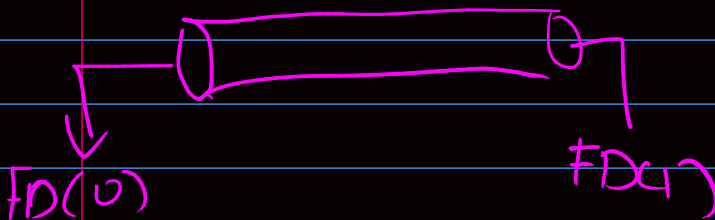
mkfifo()

Bidirectional

pipe
is
a special
file

System

pipe(int[fd]);

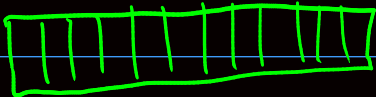


Can't be accessed
from an outer process

child to write,
close the other
side

(ideally)
EOF is inserted

Message Queue

Process A →  → Process B

asynchronous communication

Step 1 (connect/create)
msgget() msgrcv() (retrieve message)
msgsnd() msgrctl() (control
write messages operations/
delete)

Context Switching

timesharing - Quickly switch between active processes

multi-programming → assign CPU ^{to another process} between waiting times of one process

Save the context of one process, load the other

It is an overhead (it must be minimized)
Also h/w dependent (varies from system-to-system)