



Objetivo

- Comunicar procesos/hilos
- Compartir datos
- Cooperar para lograr un objetivo común.



C++

Comunicación entre hilos

- Compartir una variable, TDA, etc. en el área de datos del proceso al cual están asociados ya que es un área de memoria compartida entre todos los hilos del proceso.

Python

Comunicación entre procesos

- Señales
- Tuberías (Pipes)
- Tuberías nombradas (FIFOs)
- Colas (módulo multiprocessing)
- Memoria Compartida
- Sockets

Comunicación entre hilos

- Variables, TDA, etc.
- Colas (módulo queue)
- Eventos (módulo threading)

Señales

- Un proceso emite una señal hacia otro proceso o grupo de procesos para **comunicar la ocurrencia de algún evento**.



Señales de LINUX

```
~/ForkWait$ kill -l
 1) SIGHUP    2) SIGINT    3) SIGQUIT    4) SIGILL    5) SIGTRAP
 6) SIGABRT   7) SIGBUS    8) SIGFPE     9) SIGKILL   10) SIGUSR1
11) SIGSEGV  12) SIGUSR2  13) SIGPIPE   14) SIGALRM   15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD  18) SIGCONT   19) SIGSTOP  20) SIGTSTP
21) SIGTTIN  22) SIGTTOU  23) SIGURG    24) SIGXCPU  25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF  28) SIGWINCH  29) SIGIO    30) SIGPWR
31) SIGSYS    34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

<https://www.softprayog.in/programming/signals-in-linux>

<https://www.softprayog.in/programming/posix-real-time-signals-in-linux>

Señales de LINUX

Signal	Standard	Action	Comment
SIGABRT	P1990	Core	Abort signal from <code>abort(3)</code>
SIGALRM	P1990	Term	Timer signal from <code>alarm(2)</code>
SIGBUS	P2001	Core	Bus error (bad memory access)
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGCLD	-	Ign	A synonym for SIGCHLD
SIGCONT	P1990	Cont	Continue if stopped
SIGEMT	-	Term	Emulator trap
SIGFPE	P1990	Core	Floating-point exception
SIGHUP	P1990	Term	Hangup detected on controlling terminal or death of controlling process
SIGILL	P1990	Core	Illegal Instruction
SIGINFO	-		A synonym for SIGPWR
SIGINT	P1990	Term	Interrupt from keyboard

<https://man7.org/linux/man-pages/man7/signal.7.html>

Señales en Python

■ `signal.signal(signalnum, handler)`

```
1  import os
2  import signal
3  import time
4  import sys
5
6  CHILD = 0
7
8  def manejador(sig, frame):
9      print("Recibido ", signal.Signals(sig).name)
10
11  def main():
12      signal.signal(signal.SIGCONT, manejador)
13
```

<Sobre el handler>

Debe aceptar dos argumentos (aunque no se utilicen); número de señal y frame object, ya que son enviados por defecto al generar la señal.

Señales en Python

```
14 v     for i in range(2):
15         process = os.fork()
16 v         if process < 0:
17             print("Error al crear el proceso ",i+1)
18             sys.exit(1)
19
20 v         if process == CHILD:
21             signal.raise_signal(signal.SIGSTOP)
22             os._exit(0)
23
24         time.sleep(2)
25         os.killpg(os.getpggrp(), signal.SIGCONT)
26
27 v     for i in range(2):
28         os.wait()
```

Tuberías (PIPEs)

- Puede comunicar un padre y un hijo
- El flujo de los datos es unidireccional
- Proporciona sincronización implícita.
- Los datos se leen en el orden en que fueron escritos.



Pipes en Python

- `multiprocessing.Pipe([duplex])`

Retorna un par de objetos (`conn1`, `conn2`) tipo `Connection` que representa los extremos del pipe.

```
1  from multiprocessing import Pipe
2  import os
3  import sys
4
5  CHILD = 0
6
7  def main():
8      recepcion, envio = Pipe(False)
9
10     process = os.fork()
11     if process < 0:
12         sys.exit("Error al crear el nuevo proceso")
13
```

Si `duplex` es `False`, `conn1` es lectura y `conn2` escritura

Pipes en Python

- `multiprocessing.connection.Connection`
 - `send(obj)` → Envía un objeto al otro extremo del pipe.
 - `recv()` → Recibe un objeto enviado desde el otro extremo del pipe.
 - `close()` → Cierra la conexión.

```
if process == CHILD:
    envio.close()
    print("Recibido: ", recepcion.recv())
    recepcion.close()
    os._exit(0)
else:
    recepcion.close()
    envio.send((1, 'Yerba "La Tranquera"'))
    envio.close()
    os.wait()
```

Tubería con nombre (FIFO)

- Puede utilizarse entre procesos **NO emparentados**
- La comunicación es unidireccional
- Tiene un nombre en el sistema de archivos
- Proporciona sincronización implícita
- Los datos se leen en el orden en que fueron escritos (First In, First Out)



Fifos en Python

■ Programa A

```
1  import sys
2  import os
3
4  try:
5      os.mkfifo("./fifo", 0o600) # Crear el FIFO si no existe
6  except OSError as err:
7      print(f"Error al crear fifo: {err}")
8      sys.exit(1)
9
10 try:
11     fifo = os.open("./fifo", os.O_WRONLY) # Abrir el FIFO para escribir
12     os.write(fifo, "Café La Morenita".encode()) # Escribir en el FIFO
13     os.close(fifo) # Cerrar el FIFO
14     os.unlink("./fifo") # Eliminar el FIFO
15 except OSError as err:
16     print(f"Error al trabajar con el FIFO: {err}")
17     sys.exit(1)
```

Fifos en Python

■ Programa B

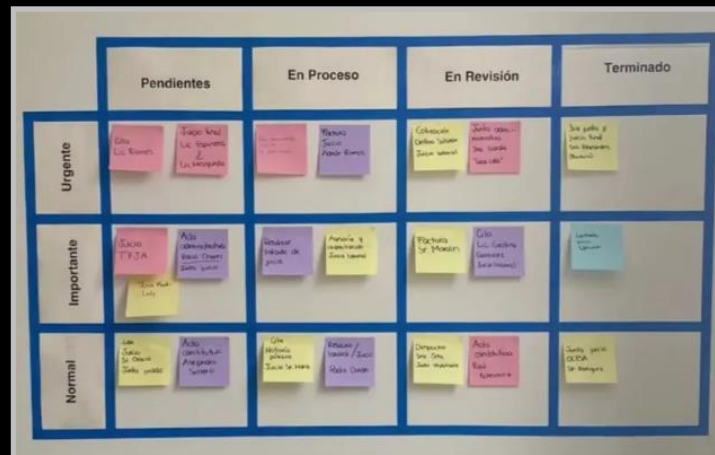
```
1  import os
2
3  BYTES = 1024
4
5  # Intentar abrir el FIFO para lectura
6  try:
7      fifo = os.open("./fifo", os.O_RDONLY)
8
9      # Leer datos del FIFO (hasta BYTES bytes)
10     data = os.read(fifo, BYTES)
11
12     # Imprimir los datos leídos después de decodificar
13     print(data.decode())
14
15     # Cerrar el FIFO después de leer
16     os.close(fifo)
17
18 except OSError as e:
19     print(f"Error al abrir o leer el FIFO: {e}")
```

Fifos en Python

- **Bytestring**: Secuencia de bytes que se utiliza para representar datos binarios como audio, imágenes o videos.
- **os.write(fd, bytestring)**
 - Escribe el bytestring en el descriptor fd.
 - Retorna la cantidad de bytes escritos.
- **os.read(fd, n)**
 - Lee n bytes del descriptor fd.
 - Retorna un bytestring.

Memoria compartida

- Varios procesos acceden a un mismo segmento de memoria común
- Permite comunicación bidireccional
- Es el mecanismo IPC mas rápido
- Pueden producirse problemas de concurrencia debido al acceso simultáneo
- Suele necesitar mecanismos de sincronización



Memoria Compartida en Python

- Módulo multiprocessing
 - `shared_memory.SharedMemory`
 - `managers.SharedMemoryManager`
 - `shared_memory.ShareableList`

https://docs.python.org/3.8/library/multiprocessing.shared_memory.html

Memoria Compartida en Python

```
1  from multiprocessing import shared_memory
2  import time
3  import numpy
4
5  ENTEROS = 5
6  TAM_ENT = numpy.dtype(numpy.int32).itemsize # Calcular bytes por entero
7
8  try:
9      mc = shared_memory.SharedMemory(name="mi_mem", create=True, size=TAM_ENT*ENTEROS)
10     print(f"Memoria compartida creada con nombre: {mc.name}")
11 except FileExistsError:
12     print("La memoria compartida ya existe.")
13
14 for i in range(ENTEROS):
15     mc.buf[i] = i+1
16
17 time.sleep(5)
18
19 mc.close()
20 mc.unlink()
21 print("Finalizando SHMA.py")
```

Memoria Compartida en Python

```
1  from multiprocessing import shared_memory
2  from multiprocessing.resource_tracker import unregister
3
4  ENTEROS = 5
5
6  try:
7      mc = shared_memory.SharedMemory(name="mi_mem")
8      print(f"Memoria compartida accedida: {mc.name}")
9  except FileNotFoundError:
10     print("La memoria compartida no existe.")
11     sys.exit(1)
12
13  for i in range(ENTEROS):
14     print( mc.buf[i] )
15
16  mc.close()
17  #Para evitar el comportamiento reportado en:
18  #https://bugs.python.org/issue39959#msg368770
19  unregister(mc._name, "shared_memory")
20  print("Finalizando SHMB.py")
```

Sockets

- Permiten comunicar procesos que corren en distintas computadoras
- La comunicación se realiza a través de una red.
- Se identifica el destino mediante una dirección IP y un número de puerto.
- Utilizan un protocolo de comunicación, como TCP o UDP.
- Permiten comunicación bidireccional

JAVA

Comunicación entre procesos

- Sockets

Comunicación entre hilos

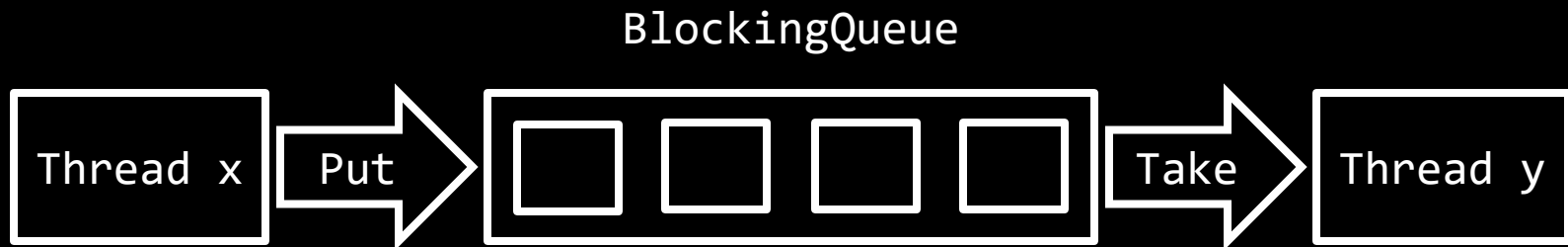
- Variables, TDA, etc.
- Colas (Queues)
- Pipes

Señales en Java

- Señalización entre hilos
- La clase `java.lang.Object` define tres métodos
 - `wait()` – el hilo llamador se bloquea a la espera de que otro hilo invoque `notify()` o `notifyAll()`
 - `notify()` – despierta a un thread que haya ejecutado `wait()`
 - `notifyAll()` – despierta a todos los threads que hayan ejecutado `wait()`

Comunicación en Java

- Interface BlockingQueue
- Uso típico para modelo productor – consumidor
- Garantiza que no haya condiciones de carrera entre los hilos (thread safe)



Comunicación en Java

■ Implementaciones

- ArrayBlockingQueue
- DelayQueue
- LinkedBlockingQueue
- PriorityBlockingQueue
- SynchronousQueue

Links de referencia

- <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html>
- <https://docs.python.org/3/library/socket.html#module-socket>