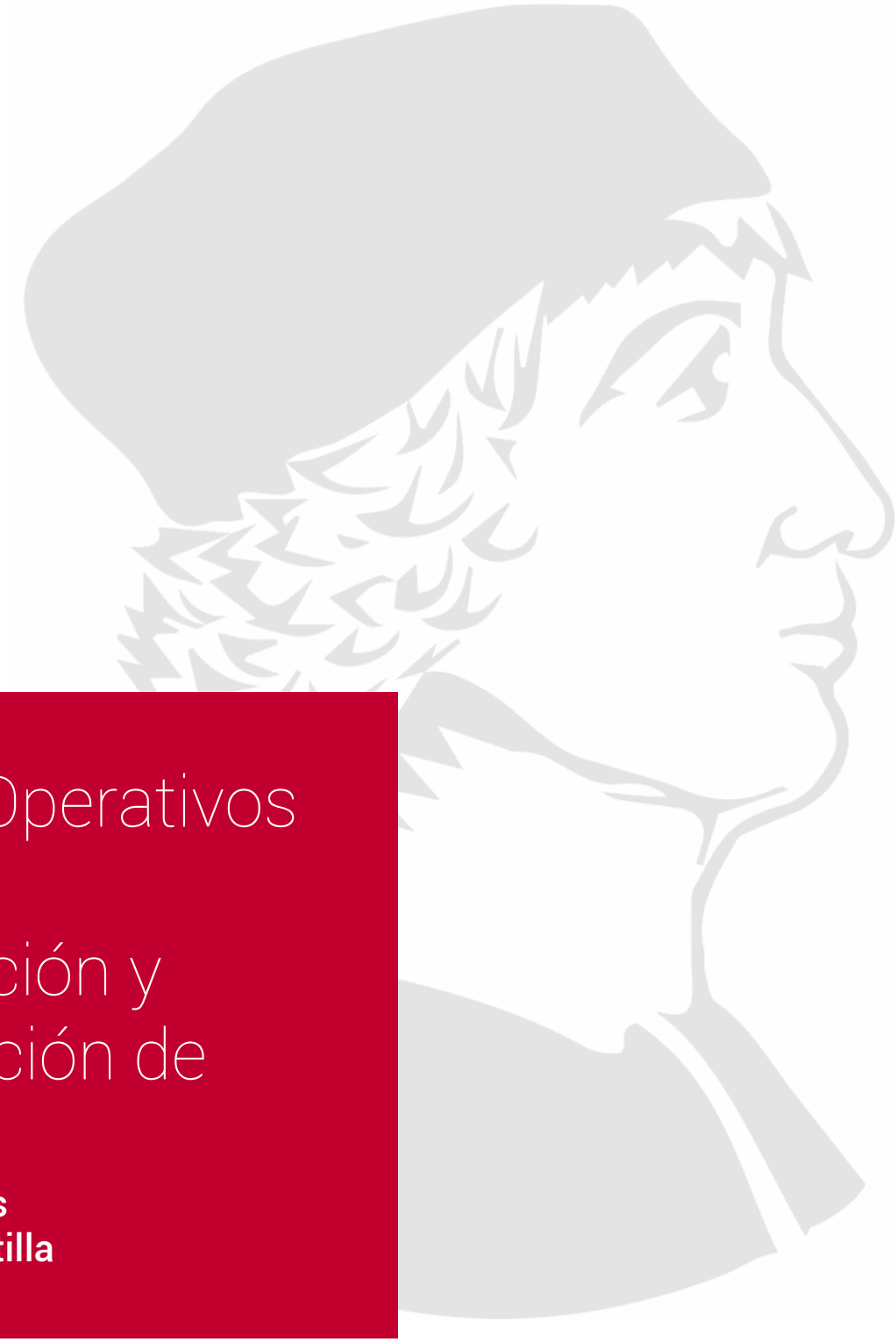


Sistemas Operativos Práctica 3: Comunicación y Sincronización de Procesos

Nicolás Sela Ramos
Arturo Requejo Portilla

UNIVERSIDAD NEBRIJA

Tabla de contenido

I. Enunciados

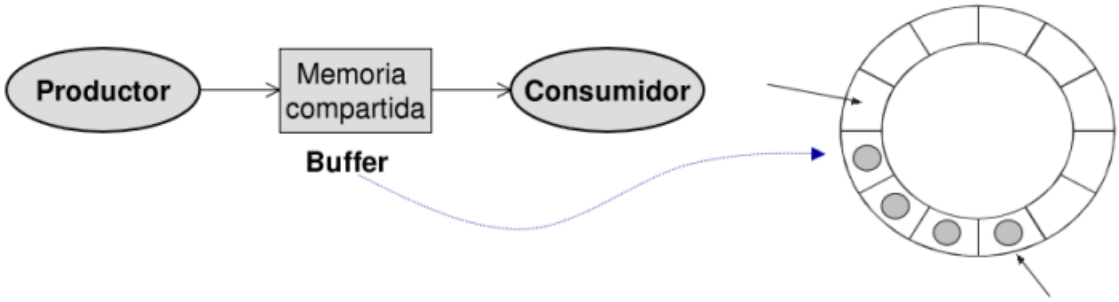
A. Ejercicio 1

Implementar un programa productor-consumidor mediante tuberías/pipes donde el proceso hijo genera un mensaje y el proceso padre los consume e imprime. **Justificar el proceso 3 PUNTOS**.

B. Ejercicio 2

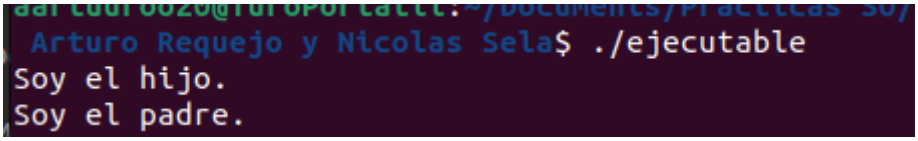
El problema del productor-consumidor con buffer limitado (circular): Planteamiento: **Justificar el proceso 7 PUNTOS**.

1. El proceso productor produce información y la almacena en un buffer.
2. El proceso consumidor accede al buffer y consume la información.
3. El productor y el consumidor comparten variables.
4. El productor no puede acceder al buffer si está lleno.
5. El consumidor no puede acceder al buffer si está vacío



II. Resultados

A. Ejercicio 1



En este ejercicio, primero creamos las dos "*pipes*", luego comprobamos mediante un condicional si estamos en el proceso padre, si esto se cumple, cerramos la posibilidad de lectura en el pipe al hijo, y la escritura en el pipe al padre, permitiendo la lectura del padre y copiamos en un buffer el dato a enviar, después abrimos la escritura en el pipe al hijo, medimos el tamaño del dato del buffer y se cierra la escritura en este *pipe*. Luego, haciendo uso de un bucle while leemos el dato del pipe al padre y lo almacenamos en *readbytes*, y por último cerramos la posibilidad de lectura. En caso de que el *pid* no sea el del padre, estamos en el hijo y realizamos el mismo proceso, cerramos la posibilidad de escritura en el pipe al hijo, así como la lectura del pipe al padre. Usando un bucle while, leemos el dato del buffer del pipe al hijo y se almacena igualmente en readbytes. Se cierra la posibilidad de lectura del *pipe* al padre una vez finalizada. Y, por último, almacenamos igualmente el dato a mandar al padre en otro buffer a través de la escritura del *pipe* al padre y la cerramos.

B. Ejercicio 2

Para hacer este ejercicio donde debemos crear un buffer circular donde debemos consumir y producir una posición. Para ello hemos creado dos funciones una que consuma y otra que produzca. El tamaño del buffer para todos los ejemplos es 10.
La función que produce va añadiendo en cada posición del buffer un dato, en concreto la posición + 1.
La función que consuma va quitando datos de la posición de la posición mayor hacia atrás.

Un ejemplo de que añade los datos correctamente es lo siguiente:

```
aartuuroo20@TuroPortatil:~/Documents/Practicas SO/Practica 3 Sistemas Operativos Arturo Requejo y Nicolas Sela$ ./ejecutable
Dato producido: 1
1 0 0 0 0 0 0 0 0 0
Dato producido: 2
1 2 0 0 0 0 0 0 0 0
Dato producido: 3
1 2 3 0 0 0 0 0 0 0
Dato producido: 4
1 2 3 4 0 0 0 0 0 0
Dato producido: 5
1 2 3 4 5 0 0 0 0 0
Dato producido: 6
1 2 3 4 5 6 0 0 0 0
Dato producido: 7
1 2 3 4 5 6 7 0 0 0
Dato producido: 8
1 2 3 4 5 6 7 8 0 0
Dato producido: 9
1 2 3 4 5 6 7 8 9 0
Dato producido: 10
1 2 3 4 5 6 7 8 9 10
```

En este ejemplo vemos como ha completado y lleno el buffer correctamente.

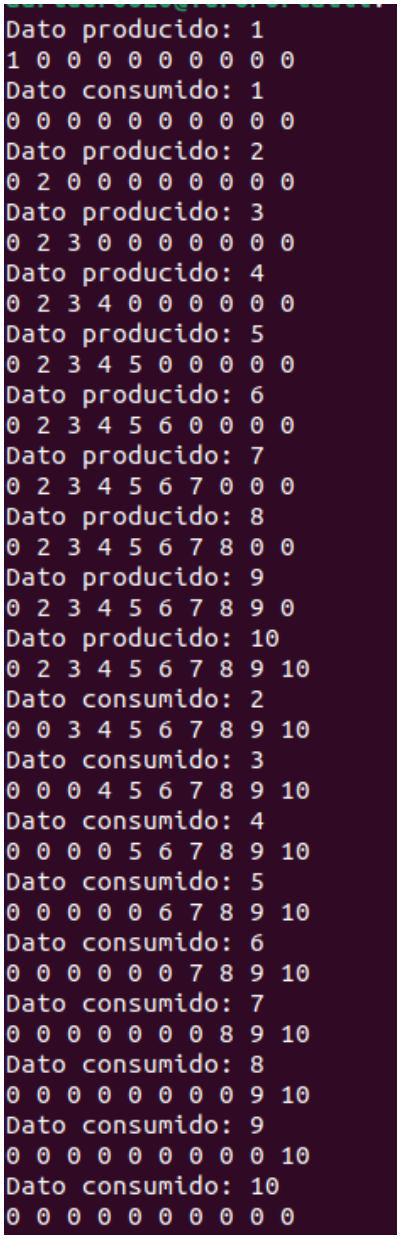
Un ejemplo de elimina todas las posiciones del array seria lo siguiente:

```
2 3 4 5 6 7 8 9 0 0
Dato consumido: 9
2 3 4 5 6 7 8 0 0 0
Dato consumido: 8
2 3 4 5 6 7 0 0 0 0
Dato consumido: 7
2 3 4 5 6 0 0 0 0 0
Dato consumido: 6
2 3 4 5 0 0 0 0 0 0
Dato consumido: 5
2 3 4 0 0 0 0 0 0 0
Dato consumido: 4
2 3 0 0 0 0 0 0 0 0
Dato consumido: 3
2 0 0 0 0 0 0 0 0 0
Dato consumido: 2
```

En este ejemplo vemos como los datos introducidos antes se están eliminando correctamente.
También en estos ejemplos hemos visto como si el buffer esta lleno no puede añadir mas datos mientras que si el buffer este vacío no puede consumir más datos.

Nos faltaría que al consumirse una posición el productor vuelva a entrar en escena y añade datos. Para ello, podemos introducir la implementación de un contador para controlar la producción y consumición, donde una vez introducidos los datos, los consuma.

Una forma de controlar los tiempos del programa y de comprobar también que las funciones de producir y consumir se mantienen ejecutándose de manera circular es cambiar el tiempo de los *sleeps* de producción y consumición, por ejemplo, si ponemos un tiempo de consumición alto y un tiempo de producción bajo, vemos cómo el buffer se llena y una vez está lleno empieza a consumirse, un ejemplo de ello:



Si ponemos un tiempo de consumición bajo y un tiempo de producción alto el dato. El consumidor se queda esperando a que se añada un dato, una vez introducido ese dato, se consume instantáneamente.

C. Conclusiones

De esta práctica hemos aprendido a implementar comunicación full dúplex entre dos procesos mediante el uso de los pipes en el ejercicio 1, así como el uso e implementación de las técnicas de planificación y sincronización de procesos, como los semáforos en el ejercicio 2.

D. Bibliografía

pipe() System call – GeeksforGeeks https://www.geeksforgeeks.org/pipe-system-call/