

MEMORIA PRÁCTICA 3

Ejercicio2:

En el ejercicio 2 se pide crear una zona de memoria compartida en la que, primero, se pide el número de hijos que va a tener el programa. Cada hijo pide un nombre por teclado, avisa al padre de que ha terminado y éste se encarga de mostrarlo por pantalla. Como podemos ver en la imagen, si esperamos un pequeño instante de tiempo, los hijos no se esperan los unos a los otros y piden la información todos de golpe sin controlar un área de memoria compartida. Además, el padre tampoco tiene control, él lee la memoria compartida cada vez que un hijo le avisa luego no hay ningún tipo de control sobre la memoria compartida y podríamos estar leyendo y escribiendo datos a la vez en esa zona sin control.

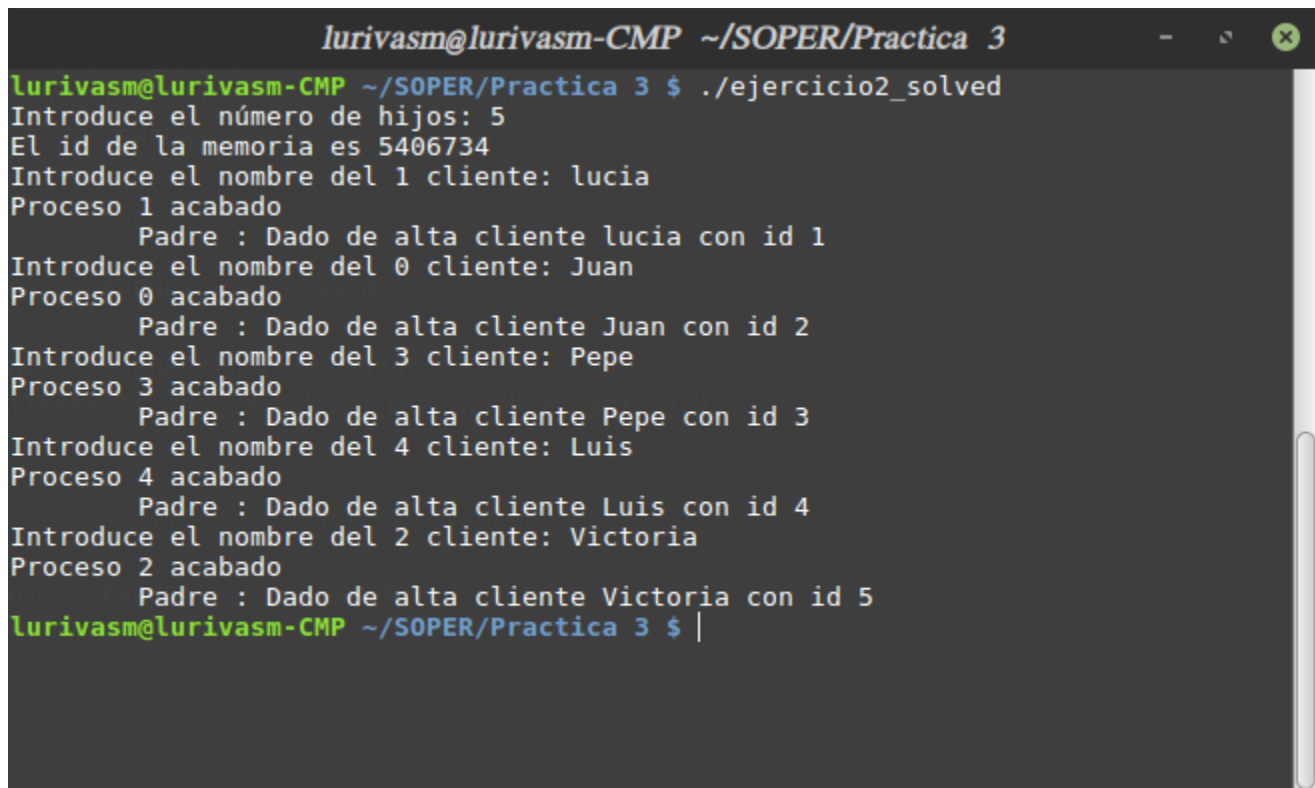
```
lurivasm@lurivasm-CMP ~/SOPER/Practica 3
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ ./ejercicio2
Introduce el número de hijos: 5
El id de la memoria es 1572876

Introduce el nombre del cliente:
Introduce el nombre del cliente:
Introduce el nombre del cliente:
Introduce el nombre del cliente:
Introduce el nombre del cliente: lucia
Proceso 0 acabado
    Padre : Dado de alta cliente lucia con id 1
luis
    Padre : Dado de alta cliente luis con id 2
Proceso 1 acabado
pepe garcia
    Padre : Dado de alta cliente pepe garcia con id 3
Proceso 4 acabado
Juan Romero
    Padre : Dado de alta cliente Juan Romero con id 4
Proceso 2 acabado
Jesus Rodriguez
    Padre : Dado de alta cliente Jesus Rodriguez con id 5
Proceso 3 acabado
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ |
```

Ejercicio2 solved:

Para solucionar el problema del ejercicio 2, hay que introducir semáforos para proteger la zona de memoria compartida. Hay dos semáforos, el semáforo 0 y el semáforo 1, ambos inicializados a 0. El padre, tras crear sus signals y la máscara se mete en un bucle para realizarlo una vez por cada hijo, en primer lugar hace un Up del semáforo 0 para dejar pasar a los hijos y luego se bloquea en sigsuspend hasta que un hijo le mande un SIGUSR1 o pulsemos CTRL+Z. Cuando le llega la señal hace un down del semáforo 1 e imprime por pantalla la información de la memoria compartida.

Por otro lado, el hijo, duerme un tiempo aleatorio, hace un down del semáforo 0, pide por teclado un nombre, avisa al padre de que ha terminado mandándole SIGUSR1 y hace un up del semáforo 1 para que el padre pueda leerlo. En la imagen podemos ver el resultado sincronizado.



```
lurivasm@lurivasm-CMP ~/SOPER/Practica 3
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ ./ejercicio2_solved
Introduce el número de hijos: 5
El id de la memoria es 5406734
Introduce el nombre del 1 cliente: lucia
Proceso 1 acabado
    Padre : Dado de alta cliente lucia con id 1
Introduce el nombre del 0 cliente: Juan
Proceso 0 acabado
    Padre : Dado de alta cliente Juan con id 2
Introduce el nombre del 3 cliente: Pepe
Proceso 3 acabado
    Padre : Dado de alta cliente Pepe con id 3
Introduce el nombre del 4 cliente: Luis
Proceso 4 acabado
    Padre : Dado de alta cliente Luis con id 4
Introduce el nombre del 2 cliente: Victoria
Proceso 2 acabado
    Padre : Dado de alta cliente Victoria con id 5
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ |
```

Tanto en este ejercicio como en el ejercicio2.c hemos agregado a la máscara la señal SIGINT por si queremos cancelar el programa. Además, hemos hecho que la función captura mande un ipcrm a la memoria compartida y a los semáforos para no tener que borrarlos a mano desde la terminal, mandando además un SIGKILL al padre para que acabe.

Ejercicio3:

En el ejercicio 3 se nos pide realizar un productor-consumidor. Para ello necesitamos dos semáforos, semáforo 0, inicializado a 1, y semáforo 1, inicializado a 0. En el ejercicio hemos llamado al hijo productor y al padre consumidor. El productor lo primero que hace es un Down del semáforo 0, genera un carácter del abecedario o un número del 0 al 9, dependiendo de la iteración en la que se encuentre, hace un Up del semáforo 1 y vuelve a comenzar en el Down del semáforo 0, en el que se queda bloqueado hasta que pueda pasar.

El consumidor, primero realiza un Down del semáforo 1, que como estaba inicializado a 0 se queda bloqueado hasta que el productor realiza el Up del semáforo 1, lee la zona de memoria compartida, la imprime por pantalla, realiza un Up del semáforo 0 para que el productor continúe con la siguiente iteración y vuelve a empezar quedándose bloqueado otra vez en el down del semáforo 1. Así sucesivamente hasta acabar con el abecedario y los números.

Para comprobar la correcta sincronización entre los procesos, el productor también tiene un printf indicando la iteración por la que va. En la imagen podemos ver que van de uno en uno y no hay ni interbloqueo ni una mala sincronización.

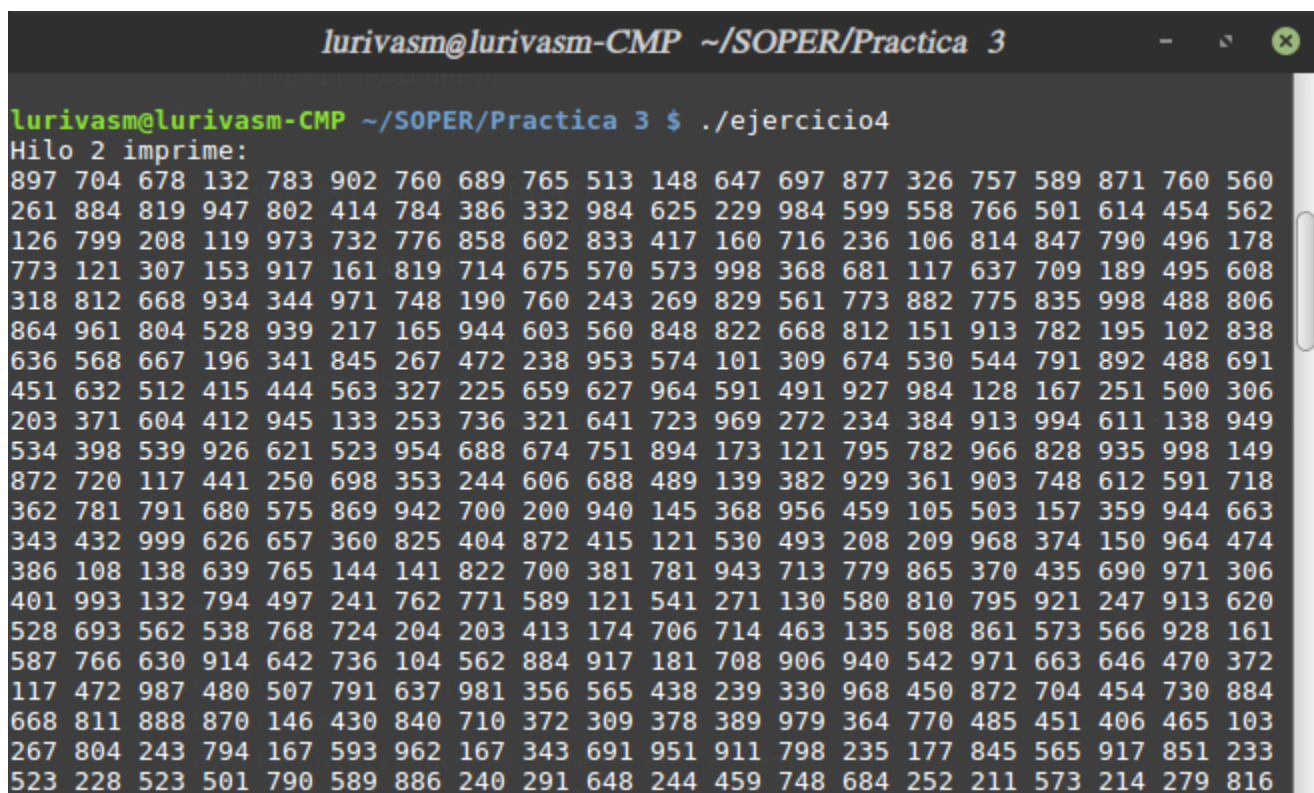
```
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ ./ejercicio3
El id de la memoria es 1703948
Productor A
Soy el consumidor e imprimo : A
Productor B
Soy el consumidor e imprimo : B
Productor C
Soy el consumidor e imprimo : C
Productor D
Soy el consumidor e imprimo : D
Productor E
Soy el consumidor e imprimo : E
Productor F
Soy el consumidor e imprimo : F
Productor G
Soy el consumidor e imprimo : G
Productor H
Soy el consumidor e imprimo : H
```

```
Soy el consumidor e imprimo : Z
Productor 0
Soy el consumidor e imprimo : 0
Productor 1
Soy el consumidor e imprimo : 1
Productor 2
Soy el consumidor e imprimo : 2
Productor 3
Soy el consumidor e imprimo : 3
Productor 4
Soy el consumidor e imprimo : 4
Productor 5
Soy el consumidor e imprimo : 5
Productor 6
Soy el consumidor e imprimo : 6
Productor 7
Soy el consumidor e imprimo : 7
Productor 8
Soy el consumidor e imprimo : 8
Productor 9
Soy el consumidor e imprimo : 9
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ |
```

Ejercicio4:

En el ejercicio 4 se pide que, a través de dos hilos, se utilice el mapeo de ficheros. En primer lugar el hilo 1 abre el archivo “ejercicio4.txt” o lo crea si no existe, e introduce una cantidad aleatoria de números aleatorios separados por comas, lo cierra y termina. En segundo lugar, el hilo 2 debe abrir ese mismo fichero, vuelca los datos en un buffer con un mapeo, lo recorre entero cambiando las comas por espacios e imprimiéndolo por pantalla, cierra el mapeo, cierra el fichero y sale.

En las imágenes podemos observar que tras la operación, el hilo 2 va cambiando las comas por espacios y que tras haber hecho el munmap el fichero “ejercicio4.txt” también ha cambiado.



```
lurivasm@lurivasm-CMP ~/SOPER/Practica 3
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ ./ejercicio4
Hilo 2 imprime:
897 704 678 132 783 902 760 689 765 513 148 647 697 877 326 757 589 871 760 560
261 884 819 947 802 414 784 386 332 984 625 229 984 599 558 766 501 614 454 562
126 799 208 119 973 732 776 858 602 833 417 160 716 236 106 814 847 790 496 178
773 121 307 153 917 161 819 714 675 570 573 998 368 681 117 637 709 189 495 608
318 812 668 934 344 971 748 190 760 243 269 829 561 773 882 775 835 998 488 806
864 961 804 528 939 217 165 944 603 560 848 822 668 812 151 913 782 195 102 838
636 568 667 196 341 845 267 472 238 953 574 101 309 674 530 544 791 892 488 691
451 632 512 415 444 563 327 225 659 627 964 591 491 927 984 128 167 251 500 306
203 371 604 412 945 133 253 736 321 641 723 969 272 234 384 913 994 611 138 949
534 398 539 926 621 523 954 688 674 751 894 173 121 795 782 966 828 935 998 149
872 720 117 441 250 698 353 244 606 688 489 139 382 929 361 903 748 612 591 718
362 781 791 680 575 869 942 700 200 940 145 368 956 459 105 503 157 359 944 663
343 432 999 626 657 360 825 404 872 415 121 530 493 208 209 968 374 150 964 474
386 108 138 639 765 144 141 822 700 381 781 943 713 779 865 370 435 690 971 306
401 993 132 794 497 241 762 771 589 121 541 271 130 580 810 795 921 247 913 620
528 693 562 538 768 724 204 203 413 174 706 714 463 135 508 861 573 566 928 161
587 766 630 914 642 736 104 562 884 917 181 708 906 940 542 971 663 646 470 372
117 472 987 480 507 791 637 981 356 565 438 239 330 968 450 872 704 454 730 884
668 811 888 870 146 430 840 710 372 309 378 389 979 364 770 485 451 406 465 103
267 804 243 794 167 593 962 167 343 691 951 911 798 235 177 845 565 917 851 233
523 228 523 501 790 589 886 240 291 648 244 459 748 684 252 211 573 214 279 816
```

Ejercicio5: Cadena de Montaje

En el último ejercicio se pedía el uso de mensajes entre procesos. Hemos hecho que el padre sea el proceso C, con dos hijos, el proceso A y el proceso B. Se deben introducir dos nombres de archivos como parámetros de entrada, el fichero a leer y el fichero de salida. Hemos dejado el fichero que hemos utilizado como entrada "entrada.txt". El enunciado pedía mandar mensajes de 16 kilobytes pero ya que era una cantidad muy grande, lo hemos puesto como 16 bytes para apreciar que de verdad se mandan varios mensajes.

En primer lugar, hemos creado una nueva estructura para los mensajes, la cual contiene un long que indica el tipo de mensaje, un int que indica si el proceso ha terminado y el array de char con la información que mandamos de tamaño 16 ya que sólo podemos mandar mensajes de 16 bytes. El tipo 1 de mensajes corresponde a los mensajes que manda A a B y el tipo 2 son los que manda B a C. El entero acabado se inicializa a 0 y una vez el proceso ha acabado lo cambia a 1 para indicarle a su receptor que ha terminado.

El proceso A comienza leyendo el fichero pasado como parámetro de entrada hasta que encuentre un \n. Va leyendo de 16 en 16 bytes y mandándole los mensajes de tipo 1 al proceso B, una vez A acaba cambia el entero acabado a 1 y sale. El proceso B espera en el buzón de mensajes de tipo 1, los lee, cambia las letras por su siguiente en el abecedario y manda el mensaje cambiado de tipo 2 al proceso C, una vez que acaba cambia el entero acabado a 1 para indicarle a C que ya ha terminado y no recibirá más mensajes. El proceso C espera recibir mensajes de tipo 2 del proceso B y los escribe en el fichero pasado como segundo argumento. Cuando recibe que B ha acabado, sale y termina todo bien.

Para cambiar las letras por su siguiente hemos creado la función cambiar(...) la cual solamente cambia las letras mayúsculas o minúsculas, manteniendo las comas, espacios... y otros signos para evitar alteraciones y que sea más simple a la vista.

En la imagen podemos ver que el programa acaba con éxito, y que entrada.txt se ha convertido en salida.txt.

```
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ cat entrada.txt
La estrategia de prevencion del interbloqueo consiste, de forma simplificada, en
diseñar un sistema de manera que se excluya la posibilidad del interbloqueo. Se
pueden clasificar los metodos de prevencion del interbloqueo en dos categorias.
Un metodo indirecto de prevencion del interbloqueo es impedir la aparicion de u
na de las tres condiciones necesarias listadas previamente (las tres primeras).
Un metodo directo de prevencion del interbloqueo impide que se produzca una espe
ra circular (cuarta condicion). A continuacion, se examinan las tecnicas relacio
nadas con las cuatro condiciones.
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ ./cadena_montaje entrada.txt salida.t
xt
Proceso A acabado
Proceso C acabado
Proceso B acabado
Trabajo acabado con éxito!
lurivasm@lurivasm-CMP ~/SOPER/Practica 3 $ cat salida.txt
Mb ftusbufhjb ef qsfwfodjpo efm jousfscmprvfp dpotjtuf, ef gpsnb tjnqmjgjdbeb, fo
ejtfnbs vo tjtufnb ef nbofsb rvf tf fydmvzb mb qptjcmjebe efm jousfscmprvfp. Tf
qvfefo dmbtjgjdbb mpt nfupept ef qsfwfodjpo efm jousfscmprvfp fo ept dbufhpsjbt.
Vo nfupep joejsfdup ef qsfwfodjpo efm jousfscmprvfp ft jnqfejs mb bqbsjdjpo ef v
ob ef mbt usft dpoejdjpoft ofdftbsjbt mjtubebt qsfwjbnfouf (mbt usft qsjnfsbt).
Vo nfupep ejsfdup ef qsfwfodjpo efm jousfscmprvfp jnqjef rvf tf qspevadb vob ftqf
sb djsdvmbb (dvbsub dpoejdjpo). B dpoujovbdjpo, tf fybnjobo mbt ufdojdbt sfmbdjpo
obebt dpo mbt dvbusp dpoejdjpoft.
```