Lucía Rivas Molina
Daniel Santo-Tomás

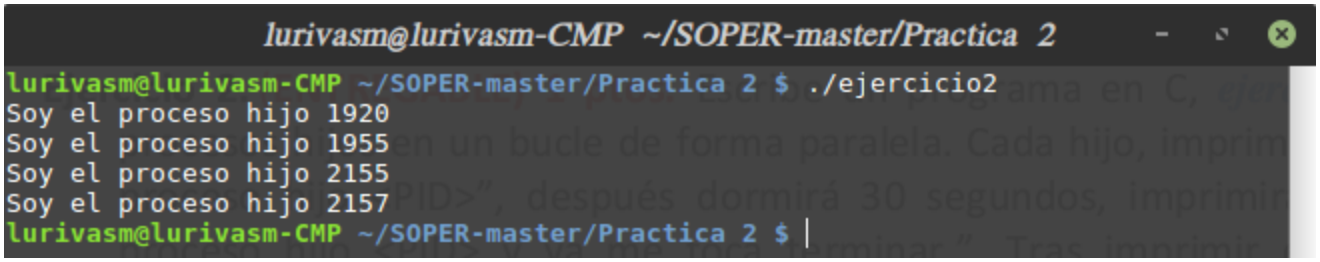
Grupo 2201

MEMORIA PRÁCTICA 2

Ejercicio2:

En este ejercicio se nos pedía realizar forks en los que el hijo imprimía “Soy el pid x”, a continuación dormía 30 segundos y justo antes de acabar imprimiría “Estoy acabando”. Sin embargo, el padre, tras crear el hijo duerme 5 segundos y le manda una señal SIGTERM al proceso que acaba de crear, para a continuación crear el siguiente. Al mandar esa señal, el hijo termina y como el padre duerme 25 segundos menos que el hijo, al hijo no le da tiempo a escribir por pantalla el mensaje de “Estoy acabando” por lo que, dando igual cuantos hijos creemos, el padre siempre los va a mandar acabar antes de que estos puedan. Si los hijos durmieran menos segundos que el padre entonces si podrían acabar.

Podemos observar en esta captura del ejercicio que efectivamente solamente imprimen el primer mensaje y no el segundo. En este caso hemos definido 4 hijos en la macro HIJOS:



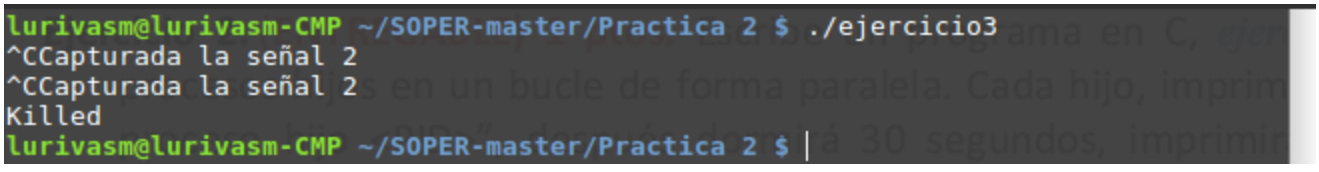
Ejercicio3:

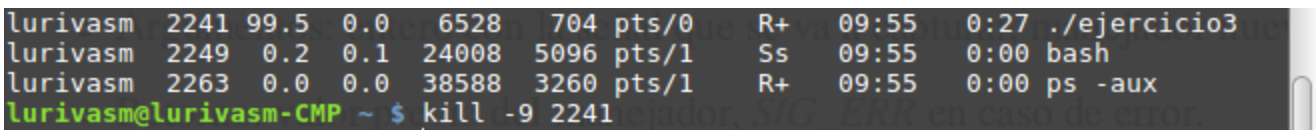
En el ejercicio 3 implementamos un pequeño programa capturador de señales, en este caso de CTRL+C igual que aparece en la foto. Como tiene un bucle while(1) para matarlo mandamos una señal SIGKILL a su proceso como se indica en la segunda foto.

a. Llamar a la función signal no implica que se ejecute la función captura, esta solo se ejecutará cuando el proceso capture la señal que le pasemos como argumento a signal, en este caso SIGINT.

b. El printf aparece cuando llamamos a la función captura cada vez que pulsamos CTRL+C.
c. Cuando una señal no tiene captura, por defecto el kernel llama a una función distinta para cada señal pues cada una tiene una acción y un propósito distinto explicados en el pdf de clase. Por ejemplo, SIGKILL elimina el proceso, SIGUSR1 y 2 son para programadores…

d. Nunca sale por pantalla el printf puesto que SIGKILL elimina al proceso luego no le da tiempo a capturar la función.





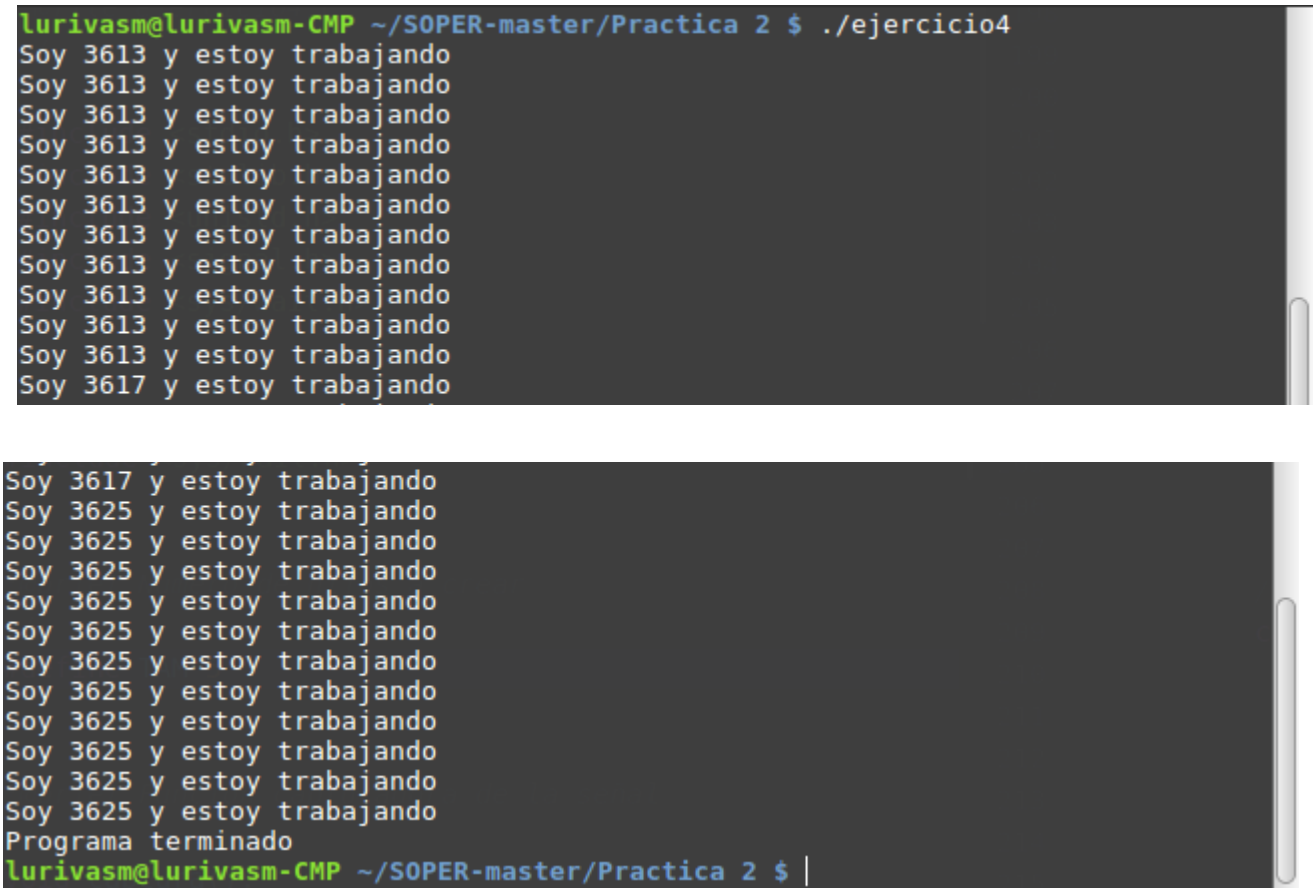
Lucía Rivas Molina
Daniel Santo-Tomás

Grupo 2201

Ejercicio4:

El ejercicio 4 consistía en crear tantos hijos como la macro TAM tenga, el padre crea el primer hijo y se queda en pause() esperando a recibir la señal SIGUSR1 capturada con la función captura, la cual significa que el hijo ha terminado y el padre deberá crear el siguiente hijo para que lo releve en el puesto. Una vez el segundo hijo se ha creado, éste, primero, le manda la señal SIGTERM al primer hijo para decirle que termine ya y luego continúa su trabajo imprimiento por pantalla. El padre, antes de quedarse en pause() a esperar al segundo hijo, realizará un wait para el primero y luego ya se quedará en pause(). Este proceso se repite tantas veces como hijos haya. Al finalizar, es el padre el que se encarga fuera del bucle de mandarle un SIGTERM al hijo y de realizar el wait pues no hay otro hijo que le pueda relevar.

En la foto TAM está definido para 3 hijos, los procesos 3313, 3317, 3325 pero en el ejercicio lo dejamos con 5.

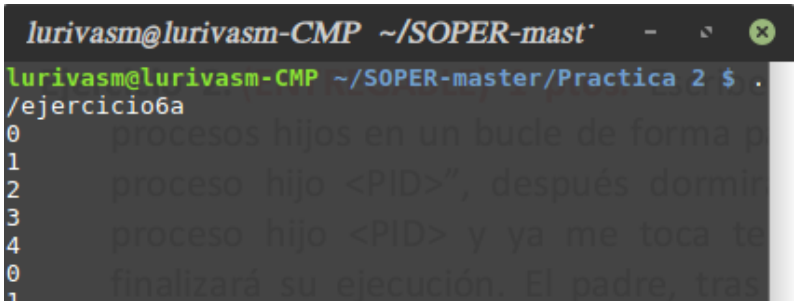


Ejercicio 6A:

En el ejercicio 6 se nos pedía modificar un código dado: En primer lugar realizamos un emptyset para vaciar nuestra máscara, y a continuación agregamos las señales SIGUSR1, SIGUS2 y SIGALRM, ya que las máscaras son heredables por los hijos. Una vez hecha la máscara, pasamos a crear los hijos (en este caso son 5, el número que guarda la macro NUM_PROC) añadimos un alarm(SEC) a los hijos nada más ser creados, donde sec es una macro que contiene un 40, de modo que salte el SIGALRM en 40 segundos. Después de cada bucle los hijos sacan de la máscara para desbloquear SIGUSR1 y SIGALRM y cuando comienza el bucle de impresión de números las vuelven a bloquear añadiéndolas a la máscara. A los 40 segundos, tal y como vemos en la foto, el proceso acaba saliendo del while(1) pues al acabar de contar, como desbloqueamos SIGALRM, salta la alarma.

Lucía Rivas Molina
Daniel Santo-Tomás

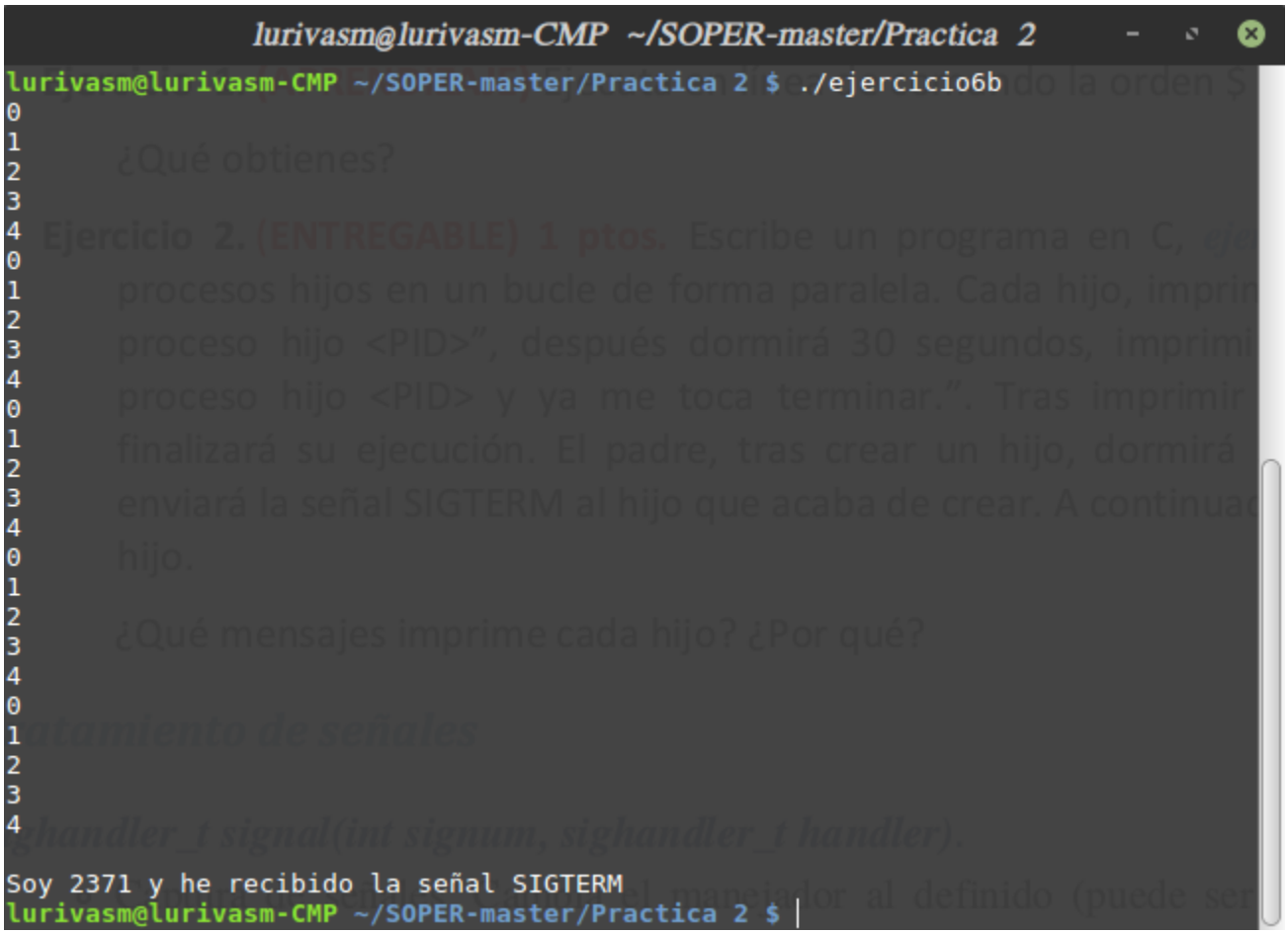
Grupo 2201



He acortado las dos fotos pues el programa, como duraba 40 segundos, quedaba muy largo.

Ejercicio 6B:

La mecánica de este ejercicio es la misma que el anterior solo que ahora es el padre el que se encarga de decirle al hijo que termine mandando un SIGTERM capturado con la función captura una vez han transcurrido 40 segundos. El hijo atmbién sale del bucle while como indica la foto.



Lucía Rivas Molina
Daniel Santo-Tomás

Grupo 2201

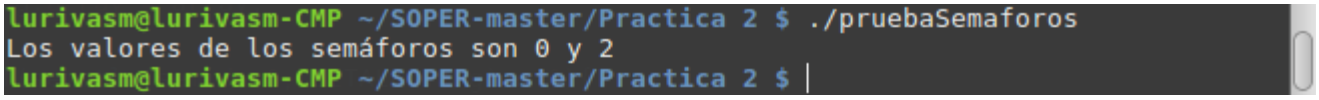
Ejercicio8:

El ejercicio 8 consistía en crear la librería de semáforos siguiendo los pasos del pdf.

- InicializarSemaforo: inicializa el semáforo pasado con los valores almacenados en el array con semctl.
- CrearSemaforo: crea size semáforos con la clave key.
- BorrarSemaforo: borra el semaforo pasado como argumento con semctl.

Los ups y downs funcionan de similar manera: suben o bajan el valor del semáforo pasado como argumento con semop pero además realiza un control de errores con el errno. Cuando un proceso realiza un down que ha realizado otro proceso antes, se queda bloqueado hasta que puede hacerlo, sin embargo, si le mandamos una señal se desbloquea y puede haber problemas. Para ello, en el do while comprobamos que eso no pasa, asignamos siempre al return un semop y a errno un 0 mientras estos devuelvan -1 y EINTR respectivamente. Al comprobar que ambos valen -1 y EINTR respectivamente realizamos de nuevo el bucle, es decir, mientras el proceso sea avisado por otra señal.

Para comprobar que funcionaban las librerías, a parte del ejercicio 9 hicimos un pruebaSemaforos.c que es básicamente el ejercicio 7 del pdf adaptado a las librerías. Lo hemos incluido en el archivo.



Lucía Rivas Molina Grupo 2201
Daniel Santo-Tomás

Ejercicio 9:

En primer lugar, hemos creado varias funciones:

- aleat_num(int inf, int sup) la cual devuelve un numero aleatorio entre inf y sup
- leer(char *archivo) la cual lee el int guardado en el archivo pasado y devuelve lo que ha leído.
- escribir(char *archivo, int cantidad) escribe en archivo la cantidad pasada como argumento.
- void captura(int sennal) la cual captura la señal enviada.
- vaciar(char *archivo) la cual vacía el archivo que le pasas como argumento abriéndola con w.
- y la función caja que realiza la función de los hijos explicada más adelante.

Por otro lado hemos creado una serie de macros:

- CAJAS: define el número de cajas a crear.
- SEMAFOROS: define el número de semáforos que necesitamos, siempre uno más que el número de cajas. Cada semáforo es para los archivos cajaTotali.txt, uno para cada caja i, luego hay un semáforo por caja y además uno extra para el archivo info.txt en el que cada vez que la caja quiere mandarle una señal al padre, ésta escribe en info.txt la señal que manda y qué caja es para que el padre pueda saber que acción realizar. La apertura de este archivo es en modo "a" de modo que se vayan almacenando todas las señales de los hijos que quieren mandar una. El padre las lee todas, realiza los cambios que tenga que hacer y vacía info.txt para que no las repita el padre varias veces, liberando el archivo para que los hijos lo sigan usando.

- OPERAC: número de veces que cobran las cajas, es decir, número de veces que lee el fichero clientesTotal.txt. Por defecto hemos puesto 50 como el enunciado indica.
- ALEAT: es un número aleatorio entre 0 y 300 para la escritura de los ficheros clientesTotal.txt.
- SECS: es un número aleatorio entre el 1 y el 5 para el sleep que realizan las cajas al terminar cada iteración.
- SEMKEY: key para los semáforos.

El programa comienza con un time(NULL) para los números aleatorios y vacía el archivo info.txt por si acaso no estuviese vacío, por defecto además, también se crea. Hemos modificado el Makefile de modo que al hacer make clean se eliminen los .txt de la carpeta. A continuación realiza la creación de los ficheros cajaTotal.txt y clientesTotal.txt para cada hijo, creándolos y escribiendo OPERAC veces un ALEAT. Después hace un signal para las señales que vamos a usar: SIGUSR1 y SIGUSR2 explicadas más adelante; y crea los semáforos.

Comienza el bucle, en caso del padre primero crea todos los hijos con un continue y después realiza un pause() hasta ser avisado; en caso del hijo llamamos a la función caja(…) explicada con detenimiento en el doxygen. Si todo va bien la función devuelve OK, el hijo libera el sem y realiza un exit(EXIT_SUCCESS).

La función caja(…) lo primero que hace es abrir el clientesTotal.txt y lo mantiene abierto hasta que se acaban las operaciones, va leyendo el dinero que paga cada cliente y guardándolo en la variable dinero. Realiza un down del fichero cajaTotal.txt, lee el valor (el inicial es 0 porque lo hemos escrito al principio del programa en la creación de ficheros) y lo guarda en la variable total, suma el nuevo dinero al total y lo vuelve a escribir en cajaTotal.txt, además realiza un printf de la caja que es y del total que tiene para controlar que todo va bien desde la terminal pues tarda en ejecutar todo. Por último, hace el up de cajaTotal.txt. Si el valor total es mayor o igual que 1000 avisamos al padre de la siguiente manera:

Hacemos un down del fichero info.txt, escribimos con "a" para no perder antiguos cambios la señal que mandamos, en este caso SIGUSR1, y el hijo que somos, le manda la señal al padre y realiza un up de info.txt, quedándose dormido con un sleep(SECS) y volviendo a comenzar. Este proceso lo repiten todos los hijos hasta

que clientesTotal.txt se acaba, en cuyo caso cerramos ese fichero y le mandamos un SIGUSR2 al padre de la misma manera que le mandamos SIGUSR1.

En caso de ser el padre, tras recibir una señal cualquiera, éste sale del pause y abre el fichero info.txt con los semáforos correspondientes para saber qué hijo le manda una señal y qué señal. Si le mandan SIGUSR1 le quitará 900 euros al dinero de cajaTotal.txt y volviendo a escribir esa cantidad, sumándole al total de todas las cajas 900 euros. En caso de ser SIGUSR2 significa que la caja ha terminado y tiene que quitarle todo el dinero, además realiza un cont++ para saber cuántos procesos han muerto y una vez mueran todos terminar el programa haciendo los waits y frees correspondientes.

En la siguiente foto podemos ver el ejercicio realizado con 5 cajas y 50 operaciones, que es como lo hemos dejado.