PRÁCTICAS DE SISTEMAS OPERATIVOS I

SEGUNDA PRÁCTICA EVALUABLE (2025-26)

Los Ángeles de Charlie

---

1. **Enunciado.**

   El programa que hay que presentar constará de un único fichero fuente de nombre `charlie.c`. La correcta compilación de dicho programa, producirá un fichero ejecutable, cuyo nombre será obligatoriamente `charlie`. Respetad las mayúsculas/minúsculas de los nombres, si las hubiere.

   La ejecución del programa producirá unos hechos basados en la conocida serie de televisión de finales de los años 70 del siglo pasado. En la ejecución participan seis personajes, representados por procesos: Charlie, Bosley, los tres ángeles (Sabrina, Jill y Kelly) y el malo.

   Esto es lo que hace cada uno de los personajes:
   1. **Charlie**: es el proceso padre. Crea el proceso de Bosley. Bosley le avisa cuando haya creado los procesos de los ángeles y, a continuación, crea el proceso del malo. Avisa a Bosley de que el malo ha sido creado.
   2. **Bosley**: es el enlace entre Charlie y los ángeles. El proceso que lo representa es hijo del proceso de Charlie. Crea a los ángeles, avisa a Charlie de que han sido creadas y espera a que Charlie le avise de que ha creado al malo. Una vez el malo ha sido creado, avisa a los ángeles de que pueden comenzar a disparar sobre él.
   3. **Los ángeles**: son tres. Al menos en esta práctica, no en la serie que se sepa, son hijas de Bosley. Su vida consiste en disparar al malo, tratando de cargárselo. Comienzan a disparar cuando les avisa Bosley.
   4. **El malo**: es malo, malísimo. Sigue la tradición de su familia: una larga saga que se remonta a tiempos prehistóricos. Alguno de ellos incluso llegó a participar en "Operación Triunfo", pero mejor correr un tupido velo. El malo es curioso en el sentido de que se reencarna. Su primera encarnación es como hijo de Charlie. Pasado un tiempo, el proceso tiene un hijo y se muere. Así lo hará veinte veces. Si alguno de los ángeles es capaz de matar cualquiera de esas reencarnaciones, habrán ganado. Si el malo llega a su reencarnación vigésima, los ángeles habrán perdido.

   El modo en que los ángeles pueden disparar al malo es el que sigue. Enviarán una señal SIGTERM a dicho sujeto. Debido a que el malo está continuamente reencarnándose el PID no es fijo. Para que los ángeles puedan conocer el PID del malo, se va a usar un fichero proyectado en memoria, cuyo nombre será `pids.bin`. El fichero tendrá cabida para 20 números enteros almacenados en formato binario (cuatro bytes por cada entero). Al principio todos los enteros estarán a cero. Cuando se reencarna el malo, escribe su nuevo PID en una posición libre del fichero al azar. Cuando un ángel quiere disparar, elegirá una posición al azar del fichero. Si vale cero, significa que la pistola se ha encasquillado y no manda señal. Si vale distinto de cero, manda la señal al PID que ha leído. Si coincide que dicho PID es el de la reencarnación actual del malo, lo mata y acaba. Si no, continúa disparando.

   Los ángeles disponen de tres disparos para tratar de acabar con el malo. Devolverán a Bosley un 0 si han acabado con el malo y un 1, si no lo han podido hacer. Bosley recogerá los códigos de retorno de los ángeles y, en función de ellos devolverá un valor diferente a Charlie. Charlie escribirá en la pantalla un mensaje final como los que siguen, en función del resultado de la misión:
   - `CHARLIE: "El pAjaro volO. Ahora se pone tibio a daiquiris en el Caribe"`
   - `CHARLIE: "Bien hecho, Sabrina, siempre fuiste mi favorita"`
   - `CHARLIE: "Jill, donde pones el ojo, pones la bala"`
   - `CHARLIE: "Bravo por Kelly"`
   - `CHARLIE: "Kelly, mala suerte, tus compaNeras acertaron y tU, no"`
   - `CHARLIE: "Sabrina, otra vez serA, te apuntarE a una academia de tiro"`
   - `CHARLIE: "Jill, no te preocupes, las pistolas no suelen funcionar cuando mAs lo necesitas"`
   - `CHARLIE: "Pobre malo. Le habEis dejado como un colador... Sois unos Angeles letales"`

   Los tiempos que transcurren entre las acciones de los procesos se simularán mediante `sleep`. Así, los ángeles emplean, al azar, entre 6 y 12 segundos antes de poder disparar. El malo vive entre 1 y 3 segundos, al azar, en cada reencarnación.

   La invocación de la práctica se hará con un argumento opcional:

   ```
   charlie [velocidad]
   ```

   El argumento opcional podrá valer `normal` o `veloz`. Si no se especifica este argumento, se entiende que su valor es `normal`. La diferencia estriba en que, a velocidad veloz, no se debe ejecutar ninguna pausa por parte de los procesos, aunque se indiquen en el enunciado. Esto es, a esa velocidad, no se invoca `sleep` nunca.

   Todos los procesos informarán por la salida estándar acerca de qué están haciendo. El mensaje final de Charlie será uno de los indicados más arriba. Los otros posibles mensajes serán:
   - `CHARLIE: "Bosley, hijo de mis entretelas, tu PID es x. Espero a que me avises..."`
   - `CHARLIE: "Veo que los Angeles ya han nacido. Creo al malo..."`
   - `CHARLIE: "El malo ha nacido y su PID es x. Aviso a Bosley"`
   - `BOSLEY: "Hola, papA, dOnde estA mamA? Mi PID es x y voy a crear a los Angeles..."`
   - `BOSLEY: "Los tres Angeles han acabado su misiOn. Informo del resultado a Charlie y muero"`
   - `SABRINA: "Hola, he nacido y mi PID es x"` (lo mismo para las otras dos)
   - `SABRINA: "Pardiez! La pistola se ha encasquillado"` (lo mismo para las otras)
   - `SABRINA: "Voy a disparar al PID x"` (lo mismo para las otras)
   - `SABRINA: "BINGO! He hecho diana! Un malo menos"` (lo mismo para las otras)
   - `SABRINA: "He fallado. Vuelvo a intentarlo"` (lo mismo para las otras)
   - `SABRINA: "He fallado ya tres veces y no me quedan mAs balas. Muero"` (lo mismo para las otras)
   - `MALO: "JA, JA, JA, me acabo de reencarnar y mi nuevo PID es: x. QuE malo que soy..."`
   - `MALO: "AY, me han dado... pulvis sumus, collige, virgo, rosas"`
   - `MALO: "He sobrevivido a mi vigEsima reencarnaciOn. Hago mutis por el foro"`

   Como veis, el malo, aunque muy malo, es un malo culto, que hasta sabe latín. No uséis otros mensajes diferentes a los anteriores y respetadlos íntegramente hasta la última coma, por si se usa, para la corrección, una herramienta automática.

   Los procesos, al hacer un `ps` desde la línea de órdenes, deben mostrar su nombre, es decir: charlie, bosley, sabrina, kelly, jill y malo. Para lograr esto, haced que los procesos recién nacidos que lo necesiten hagan un `exec` al mismo ejecutable, pero variando `argv[0]` de modo acorde. Si necesitárais pasar información extra, usad más argumentos. Nada más iniciar la función `main` una comprobación del valor de `argv[0]` os puede servir para guiar al proceso a ejecutar el código que le corresponde.

   Para que el buffer intermedio usado por `printf` no interfiera con la salida de los procesos, es importante usar `write` para la salida por pantalla en su lugar.

2. **Números aleatorios**

   Esta no es una práctica de programación, sino que es necesario programarla y, por consiguiente, saber programar. Es por ello que a continuación se incluyen unas pistas para generar números aleatorios (al azar) en vuestros procesos y no se deja para que lo investiguéis por vuestra cuenta.

   Los ordenadores son las máquinas más previsibles que os podéis encontrar. Para ellas, es muy difícil hacer algo al azar. Por eso, cuando quieren generar un número al azar, lo que hacen es usar una fórmula matemática que, partiendo de un número (la semilla) genera otro tratando de que el nuevo se parezca poco al anterior. El número nuevo pasa a ser la nueva semilla para generar el siguiente.

   Para obtener una ristra aleatoria se debe, pues, hacer dos pasos: primero, establecer la semilla inicial (**esto solamente se hace una vez**) y segundo, ir sacando números.

   Para establecer la semilla inicial, la biblioteca de C usa la función `void srand(unsigned int semilla)`. Pero, ¿qué semilla ponemos? Si ponemos un número que se nos ocurra, los números al azar que obtendremos serán buenos, pero siempre los mismos cada vez que ejecutemos el programa. Para obtener números diferentes, se le suele pasar a `srand` algo que dependa del momento en que se lanza el programa. Algo muy típico es:

   ```
   srand(time(NULL));
   ```

   Cuando tenemos varios procesos, la cosa se complica. Primero, los procesos van a heredar la misma semilla del padre, por lo que todos los hijos sacarán los mismos números de su chistera. Si el proceso, hace un `exec` es aún peor, pues todo ocurre como si nunca se hubiera llamado a `srand`. Hay, por consiguiente, que hacer un `srand` en cada nuevo proceso una sola vez y después de haber hecho sus `exec`s, si es que los hace. Ahí no acaban nuestros problemas, pues si ponemos la fórmula de arriba, que tiene una resolución de segundos, dará la misma semilla a todos los procesos, pues se crean muy rápidos. Lo mejor es que la modifiquéis a vuestro gusto para que también incluya el PID del proceso que hace la llamada.

   Una vez hemos iniciado el generador de números aleatorios, hay que obtener los propios números. La función que hace esto es `int rand(void)`. La dificultad estriba en que esta función nos devuelve un número al azar entre 0 y una macro llamada `RAND_MAX`, lo que es muy poco útil. Lo más normal es que queráis obtener un número entre a y b, ambos incluidos (por ejemplo, entre 1 y 6, para la tirada de un dado). Esta expresión logra vuestro objetivo:

   ```
   a+(int)(rand()/(1.0+RAND_MAX)*(b-a+1))
   ```

3. **Finalización ordenada**

   La práctica deberá acabar ordenadamente cuando el usuario pulse CTRL-C. Los procesos deben morir y el padre, una vez hayan muerto todos imprimirá la frase: "Programa interrumpido".

4. **Restricciones**

   - Se deberán usar llamadas al sistema siempre que sea posible, a no ser que se especifique lo contrario.
   - No está permitido usar la función de biblioteca `system`, salvo indicación explícita en el enunciado de la práctica.
   - No se puede suponer que los PIDs de los procesos de una ristra van a aparecer consecutivos. Puestos en plan exquisito, ni siquiera podemos suponer que estarán ordenados de menor a mayor (puede ocurrir que se agoten los PIDs y se retome la cuenta partiendo de cero).
   - No está permitido el uso de ficheros, tuberías u otro mecanismo externo para transmitir información entre los procesos, salvo que se indique en el enunciado.

5. **Plazo de presentación.**

   Consultad la entrada de la página web de la asignatura.

6. **Normas de presentación.**

   Acá están.

7. **LPEs.**

   I. Las tareas que tiene que realizar el programa son variadas. Os recomendamos que vayáis programándolas y comprobándolas una a una. No es muy productivo hacer todo el programa de seguido y corregir los errores al final. El esquema que os recomendamos seguir para facilitaros la labor se os muestra a continuación:
      1. Haced un pequeño programa al que se le pase los argumentos que se especifican en el enunciado. Imprimid los argumentos para depurar y considerad las opciones de error al meterlos. Una vez controlados, comentad la depuración.
      2. Crearemos primero el proceso del malo, pues tiene la vida más fácil de realizar. Hacemos el `fork`, cada proceso imprime quién es y los dejamos en `pause`. Al hacer un `ps` desde otro terminal, debemos observar a los dos procesos, uno hijo del otro.
      3. Los dos procesos se llaman `charlie` en el punto anterior. Debemos lograr que el malo cambie su nombre a `malo`. Para ello, usaremos un truco. Haremos una llamada a una función de la familia `exec`, por ejemplo, `execl`, que ejecute el ejecutable `charlie`, pero con `argv[0]` igual a `malo`. Daos cuenta que, al hacer el `exec`, el proceso *olvida* todo y recomienza su ejecución. Lo tenemos que redirigir a una función para que haga las tareas que le son propias. Pero para eso, nos podemos valer del contenido de `argv[0]`. Moved el mensaje donde el malo dice que es el malo a dicha función y comprobad que todo va como debe ir.
      4. Completad el funcionamiento del malo. Primero, duerme. Luego tiene un hijo y se muere. Así, veinte veces. Charlie debe esperar por la muerte del primer malo, para que no se quede zombie.
      5. Hecho ya el malo. Ahora, que Charlie cree a Bosley. Este código tiene que venir antes de la creación de los malos, como dice en el enunciado. Lo hacemos ahora aunque venga antes. Bosley se queda esperando a que Charlie le avise de que ha creado el malo. Charlie le avisa cuando eso ocurre.
      6. Toca el turno de crear a los ángeles. Bosley los crea y se ponen los mensajes y se avisa según se marca en el enunciado.
      7. Charlie crea el fichero de PIDs y lo pone a cero. Para ver si se ha creado el fichero de un modo correcto, podéis usar la orden de la línea de órdenes `od`, que sirve para ver el contenido binario de un fichero. El fichero debe medir 80 bytes justos. Si no lo mide, muy probablemente lo habéis creado y escrito en modo de texto, no en binario.
      8. El siguiente paso consiste en que el malo proyecte el fichero y vaya escribiendo su PID en un sitio vacío cada vez que se reencarna. Al final de la ejecución el fichero debe estar lleno con los PIDs que ha ido teniendo el malo.
      9. Haced que los ángeles disparen a la señal de Bosley. Imprimid los mensajes adecuados según el resultado de los disparos.
      10. Programad ahora que Bosley se quede esperando y que los ángeles le devuelvan el resultado de su misión. Una vez tenga todos, le devuelve un valor a Charlie, indicando el resultado global.
      11. Acabad de pulid los detalles que faltan: que acabe bien cuando se pulsa CTRL-C, etc.
      12. Probad el modo veloz. No os extrañe si en este modo siempre pasa lo mismo. Va a depender mucho del reparto de CPU. Lo que sí tiene que ocurrir es que los mensajes que aparecen tengan sentido.

   II. **No se puede usar `sleep()` o similares para sincronizar los procesos. Hay que usar otros mecanismos.**

   III. Sabéis que si usáis *espera ocupada* en lugares donde explícitamente no se haya dicho que se puede usar, la práctica está suspensa. No obstante, existe una variedad de espera ocupada que podríamos denominar *espera semiocupada*. Consiste en introducir una espera de algún segundo en cada iteración del bucle de espera ocupada. Con esto el proceso no consume tanta CPU como en la espera ocupada, pero persisten los demás problemas de la técnica del sondeo, en particular el relativo a la elección del periodo de espera. Aunque la práctica no estará suspensa si hacéis espera semiocupada, se penalizará en la nota bastante si la usáis. En conveniente que la evitéis.

   IV. Evitad, en lo posible, el uso de variables globales. Tenéis la posibilidad de declarar *variables estáticas*.

   V. Tened cuidado con el uso de `pause()`. Salvo en bucles infinitos de `pause`s, su uso puede estar mal. Mirad la solución a la práctica propuesta en la sesión décima acerca de él o el siguiente LPE.

   VI. El programa que he hecho se para a veces. O en mi casa se para, pero en clase, no. O en clase sí, pero en casa, no.
   Solución.

   VII. ¿Qué hago cuando mi programa se desboca para no perjudicar el funcionamiento de la máquina?
   Solución.

   VIII. Debéis tener cuidado con un efecto que se produce por el hecho de que los descriptores de fichero heredados comparten un único puntero de fichero. Si cualquiera de los procesos lo mueve con `lseek` el resto lo ve movido. Este efecto secundario hace que el siguiente código sea erróneo para tratar de bloquear el fichero:

   ```
   lseek(fd,0,SEEK_SET);
   lockf(fd,F_LOCK,0);
   ```

   La razón es que entre las dos instrucciones se puede perder la CPU y otro proceso nos puede mover el puntero que nosotros pensamos que está al principio. La solución pasa por no usar los descriptores heredados sino que cada hijo, al nacer haga algo similar a:

   ```
   case 0: /* COdigo del hijo */
     close(fd);   // Cerramos el descriptor heredado
     fd=open(...  // Lo volvemos a abrir
   ```

   IX. En el LPE anterior es evidente que el `open` que hace el hijo no puede llevar `O_TRUNC`. Si lo lleva, cuandoquiera que nazca un hijo, borrará el fichero pudiendo pillar justo antes de una lectura, que no leería nada. Lo debe abrir solamente con el flag `O_RDWR`

   X. Los procesos malos, al nacer uno detrás de otro y ser adoptados por init, son de difícil control. Sería interesante poder matarlos al final o cuando se usa CTRL-C. Una solución consiste en enviar una señal al PID 0. Si se hace así, la señal la reciben todos los procesos de la misma familia. Es decir, que si la manda Charlie la recibirán él mismo y todos los demás, incluidos los malos. Gracias a BraveSparks por probarlo antes de poder incluir este LPE