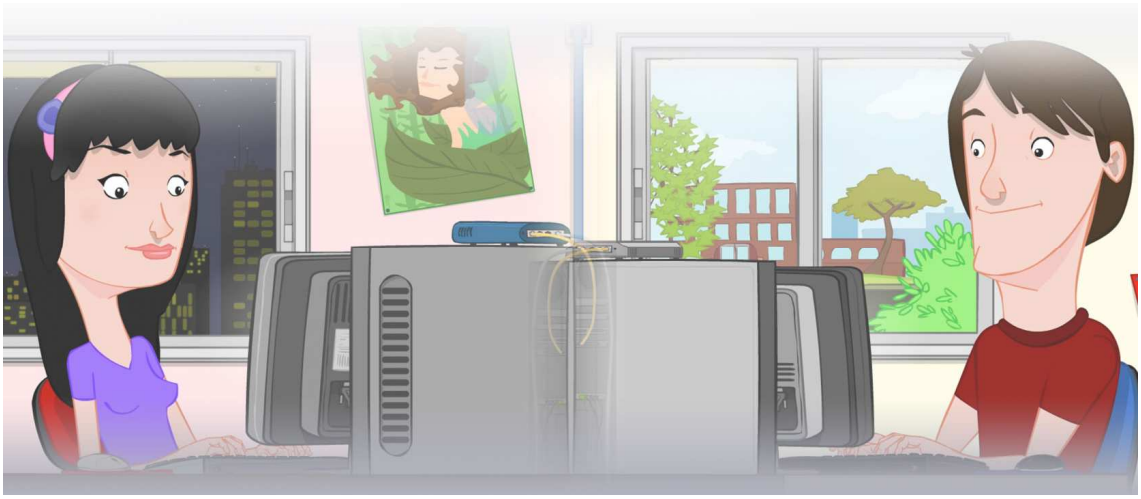
Práctica 1

Procesos y Señales



Curso 2014-2015

Grado en Ingeniería Técnica de Telecomunicación
(3º curso)

Área de Ingeniería Telemática

Práctica 1: Procesos y señales

1. Motivación

UNIX es un sistema operativo multiplataforma, multiproceso y multiusuario. Multiplataforma quiere decir que se puede encontrar este sistema operativo en muchas arquitecturas diferentes de computación, desde arquitecturas multiprocesador, a ordenadores personales o grandes ordenadores de empresa. Multiproceso es que el sistema permite ejecutar simultáneamente muchas aplicaciones o instancias de ejecución de programas. En este sentido existen sistemas UNIX que planifican procesos compartiendo el tiempo, otros que son capaces de asignar instancias de ejecución a diferentes procesadores de forma simultánea y otros diseñados para dar respuestas en el rango de tiempo real. Por último la característica multiusuario quiere decir que en la misma máquina se pueden atender a varios usuarios del sistema de forma simultánea. Esto es gracias a la característica de multiproceso del sistema que permite crear contextos de proceso específicos para atender las necesidades de cada usuario. Antiguamente, los usuarios se conectaban a los ordenadores a través de terminales "tontos" que no eran capaces de ejecutar procesos computacionales, sino que sólo servían para interactuar con el usuario vía teclado y pantalla. Las órdenes se comunicaban al ordenador central a través de la línea de comunicaciones y era ese sistema central (generalmente una máquina UNIX) la que daba la capacidad de cómputo a un entorno con múltiples usuarios, terminales y aplicaciones. Como podéis ver, las comunicaciones era algo intrínseco al diseño original de los sistemas UNIX. Eso ha hecho que casi todos los productos actuales se basen en sistemas embebidos con la filosofía UNIX. Aprender cómo funcionan estos sistemas ayudará a los alumnos a entender el mundo de las comunicaciones.

En esta práctica, se pretende que el alumno se familiarice con esta característica de multiproceso de los sistemas UNIX. Por ello se propone lanzar desde un programa diferentes instancias de ejecución que pueden ser tanto procesos como hilos de ejecución. Convendría aprender las diferencias entre programa-proceso-hilo. También se pretende introducir al alumno en los mecanismos básicos de comunicación entre procesos como son las señales. Las señales sirven para dar avisos entre procesos para manejar determinadas incidencias. Una señal en principio no transmite ninguna información más que ella misma ha ocurrido. En el mundo de las comunicaciones entre terminales de un sistema UNIX, las señales han jugado un papel fundamental. Muchas de ellas han sido diseñadas para identificar situaciones concretas de la línea de comunicaciones de un terminal con el ordenador central, y avisar a los programas de esas situaciones para poder reaccionar. Un ejemplo de ello es la caída de la línea de comunicaciones de un terminal remoto. Eso supone en un sistema UNIX que todos los procesos asociados a esa línea deben de finalizarse. Cuando pulsamos la combinación <CTRL>C de nuestro terminal de teclado estamos simulando esa circunstancia a nuestro terminal y eso se comunica a los procesos a través de una señal que en principio está programada para eliminar todos los procesos de esa terminal.

2. Enunciado de la práctica

Esta práctica tiene un tiempo de desarrollo de tres semanas, y está estructurada en torno a dos pruebas.

En la primera prueba el alumno deberá desarrollar una pequeña aplicación, en la que se utilizarán los recursos del sistema operativo presentados en la parte teórica. Basándose en los requerimientos de la aplicación se deberán resolver aspectos de diseño e implementación. Se incluyen en el enunciado algunas indicaciones y sugerencias para guiar al alumno respecto a los contenidos con los que se deberá familiarizar en esta práctica.

En el segundo apartado, se realizarán una serie de ejercicios guiados en los que el alumno realizará programas que se comunicarán con una aplicación de evaluación que se denomina Monitor. Estos ejercicios permiten al alumno auto-evaluar su dominio de las funciones y procedimientos básicos relativos a procesos y señales.

2.1 Descripción de la primera prueba

Crea un programa que al ejecutarse se desdoble el proceso padre en otros dos procesos hijos. El proceso padre se quedará dormido e igualmente ocurrirá con el primer hijo. El segundo hijo estará pendiente del teclado (proceso de teclado) esperando recoger una orden del operador que le indique si envía una señal al proceso padre, si lo hace al hijo o si finaliza la comunicación. Cada proceso mandará un mensaje a la pantalla al recibir la señal del proceso de teclado y continuará pendiente de recibir más señales.

Por otro lado, el proceso de teclado debe tener una opción para construir un mensaje en un fichero llamado “mensaje” que será traducido de minúsculas a mayúsculas por un thread del proceso hijo primero. La traducción se realizará cuando el proceso teclado envíe una señal al proceso hijo primero que le dé a entender que el mensaje está ya disponible para su traducción. El resultado de la traducción aparecerá en pantalla informando el thread de qué proceso lo ha realizado.

En el menú del proceso de teclado se contemplará una opción para cerrar el programa de una forma ordenada.

Una vez lanzado el programa conviene observar los siguientes aspectos:

- ¿Cuáles son los números de procesos asignados por el sistema?
- ¿Cómo puedes eliminar los programas en ejecución desde otra consola?
- ¿De qué información dispone cada proceso del resto de procesos del programa?
- ¿Cómo se debe finalizar la ejecución de los procesos?
- ¿Cómo se deben procesar las señales que recibe cada proceso?
- ¿qué son los procesos que aparecen en el sistema como <defunt> y cómo se puede evitar eso?

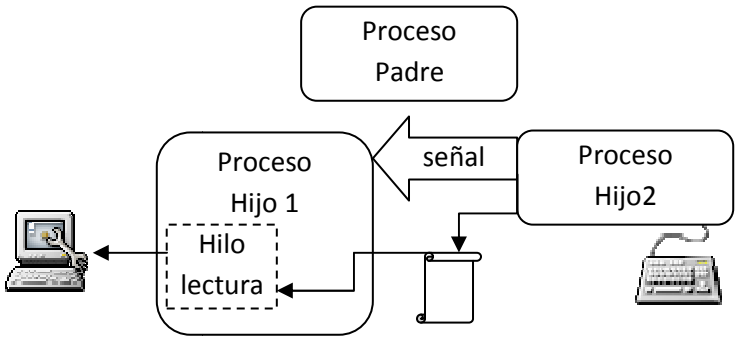
Competencias a desarrollar por el alumno

1. Manejo de las primitivas de creación de instancias de ejecución; fork(), exec(), pthread(), getpid(),…
2. Envío de señales entre procesos y recepción de las mismas: kill(), signal(), alarm(), sleep()…
3. Comandos básicos de supervisión de procesos en Linux: ps, kill, top…
4. Manejo del sistema de entrada/salida (I/O) estándar: open(), write(), read(), fgets(), printf(), perror(),…
5. Finalización de procesos: wait(), waitpid(), exit()…
6. Organización de una aplicación: Makefile, includes, librerías,…

Indicaciones para el diseño y la implementación de la aplicación

Estas indicaciones están enfocadas en proponer una metodología sencilla de trabajo en laboratorio que pueda servir para formar suficientemente al alumno para conseguir sus objetivos. Algunos aspectos que se deberían practicar en el tiempo de laboratorio son los siguientes:

1. Familiarizarse con el entorno de desarrollo. Es aconsejable utilizar una consola pues es la interfaz estandarizada de cualquier sistema UNIX y el uso de aplicaciones con salida textual es importante para formar a los alumnos en la administración de este tipo de sistemas. En este sentido se aconseja: (a) utilizar el editor vi como editor de programas, (b) compilar con gcc, (c) organizar el proceso de compilación con make, etc.
2. De cara a esta práctica y a otras posteriores, conviene que el alumno se familiarice con el mecanismo de recogida de parámetros de la línea de comandos desde la función main(). Al menos el manejo de los parámetros argc y argv de esa función main es importante. Para alumnos más avanzados se propone que aprendan a manejar el estándar de tratamiento de parámetros de UNIX consultando el manual de getopt().
3. Aunque en el enunciado de la práctica adicional se propone una jerarquía clara de procesos, reflexionar sobre qué procesos se van a ejecutar y qué función va a realizar cada proceso en la aplicación. Hacer un programa sencillo que despliegue los procesos y, posteriormente, monitorizarlos desde el sistema con ps, top, etc. Pensar bien en qué información maneja cada proceso para decidir quién debe realizar qué funciones dentro de la jerarquía.



4. Monitorizar los procesos y threads creados con los comandos del sistema ps, top, etc. Se pueden mandar señales desde comando kill de la Shell para poder realizar pruebas.
5. En la aplicación se debe interactuar con el usuario mediante un menú. Centrar los esfuerzos en recoger del teclado la opción de un usuario de selección de un menú. Se desaconseja utilizar scanf() pues esta llamada sirve para entradas formateadas y los usuarios difícilmente se ajustan a un formato. Se aconseja estudiar como leer de la entrada estándar con fgets() y si se quiere identificar un formato determinado utilizar funciones de transformación de strings como atoi() e incluso sscanf(). Con estas ideas hacer una función de selección de opciones en menú.
6. Hay que saber programar la recepción de una señal (signal()) y saber enviar las señales adecuadas (kill()). En sistemas UNIX las señales están ya definidas y muchas de ellas están asociadas a determinadas funciones. Conviene aprender el uso estándar de algunas de ellas. Consultar el manual con el comando man de algunas de esas funciones (Ej: man –a signal). La sección SEE ALSO permite ver comandos y funciones asociadas. Aprender cómo se organiza el man de UNIX en secciones es de interés. La sección de funciones es la 2, así que si queremos saber la información de una función se debe usar p.e. man 2 signal.
7. En la aplicación se debe manejar un fichero para introducir un mensaje que debe ser traducido por otro proceso que debe leerlo y traducirlo en pantalla a mayúsculas. Las operaciones sobre ficheros de texto se pueden realizar con open(), close(), read(), write() que son llamadas estándares o con fopen(), fclose(), fread() y fwrite() que son similares pero de un mayor grado de abstracción para el sistema. En principio se aconseja usar estás últimas pues permiten controlar el tamaño de los búferes intermedios. Estudiarlas con man 2.
8. Hacer una función sencilla de traducción de cadenas de minúsculas a mayúsculas. Esta función se puede llamar toupper_str() y se puede apoyar en las macros toupper() para traducir cada carácter de la cadena a mayúsculas. Analizar qué diferencia hay entre macro y función.

2.2 Descripción de la segunda prueba

El alumno debe realizar un programa denominado <client.c> que se ejecutará como <./client> en el directorio de trabajo donde reside el programa <./monitor-1>. El programa monitor propone la realización de 6 ejercicios que deben realizarse en un único programa de forma secuencial. Con el fin de facilitar la sincronización con el programa monitor, es conveniente que el programa cliente espere la intervención del usuario para avanzar en la ejecución de los diferentes ejercicios (mediante la pulsación de una tecla, por ejemplo).

Ejercicio 1: Enviar señales

El programa monitor esperará durante un minuto a recibir la señal SIGUSR1 desde el proceso <client>. Cuando esto se produzca desvelará el secreto <1>.

El programa <client> debe, inicialmente, recoger como parámetro de la línea de comandos un dato que le informe del número de proceso del programa monitor. Después de enviar la señal SIGUSR1 al proceso monitor, el cliente deberá dormir durante 5 segundos.

Ejercicio 2: Recibir señales

Para avisar al monitor del comienzo de la prueba, el cliente debe inicialmente enviar la señal SIGUSR1 al proceso monitor. Éste esperará la señal durante unos segundos después de seleccionada la opción adecuada.

Cuando se reciba la señal de arranque, el proceso monitor enviará al proceso cliente la señal SIGUSR2 y esperará recibir como respuesta, de forma inmediata, un “eco” consistente en la señal SIGUSR2 enviada desde el proceso <client>.

Si todo se realiza de forma correcta el proceso monitor desvelará el secreto <2>

Ejercicio 3: Crear procesos

Para avisar al monitor del comienzo de la prueba, el cliente debe inicialmente enviar la señal SIGUSR1 al proceso monitor. Éste esperará la señal durante unos segundos después de seleccionada la opción adecuada.

El proceso <client> debe enviar la señal SIGUSR1 de arranque de la prueba y después de **un segundo** proceder a la creación de dos procesos hijos denominados <hijo 1> e <hijo 2>. Para avisar de la finalización del ejercicio el cliente enviará al monitor de nuevo la señal SIGUSR1. Entre las dos señales SIGUSR1 de arranque y finalización no debe transcurrir más de 2 segundos. Si todo se realiza de forma correcta el proceso monitor desvelará el secreto <3>

Ejercicio 4: Crear hilos

El proceso <hijo 1> del programa <client> debe permanecer a la espera de la señal SIGUSR2 para lanzar un hilo de ejecución que presente en la pantalla datos de su actividad cada cinco segundos durante un tiempo de 20 segundos aproximadamente.

El proceso monitor, al activarse la opción, mandará un número aleatorio de señales SIGUSR2 al proceso <hijo 1> para la activación de hilos, durante un periodo máximo de 20 segundos. El alumno debe conocer cuántos hilos se han activado. Si todo se realiza de forma correcta el proceso monitor desvelará el secreto <4>

Ejercicio 5: Contar señales

Para avisar al monitor del comienzo de la prueba, el cliente debe inicialmente enviar la señal SIGUSR1 al proceso monitor. Éste esperará la señal durante unos segundos después de seleccionada la opción adecuada.

Posteriormente el proceso monitor enviará al proceso <hijo 2> del programa <client> una señal SIGUSR1 para avisar del inicio de un periodo de prueba que durará unos 60 segundos. Durante ese periodo el proceso monitor enviará una serie de señales SIGUSR2 aleatorias que el cliente debe contar y responder a ellas en tiempo con otra señal SIGUSR2 de eco. El periodo de prueba finalizará cuando el cliente reciba otra señal SIGUSR1.

Se trata de que el proceso <hijo 2> del programa <client> cuente el número de señales SIGUSR2 que el proceso monitor le ha enviado en un periodo de tiempo señalizado por dos señales SIGUSR1. Si todo se realiza de forma correcta el proceso monitor desvelará el secreto <5>

Ejercicio 6: Cerrar aplicación

Para avisar al monitor del comienzo de la prueba, el cliente debe inicialmente enviar la señal SIGUSR1 al proceso monitor. Éste esperará la señal durante unos segundos después de seleccionada la opción adecuada.

El cliente debe enviar la señal SIGUSR1 de arranque de la prueba y después de esperar un segundo proceder al cierre de la aplicación cliente de forma ordenada. No deben quedar procesos descolgados, ni en estado <zombie>. El monitor supervisará que se ha realizado todo de forma correcta en un tiempo máximo de dos segundos y en ese caso revelará el secreto <6>

FICHA DE PRUEBA DE LABORATORIO DE ARQUITECTURA DE SISTEMAS DE LA INFORMACIÓN

Nombre:…… Grupo: ……

DNI:…….. Fecha:……..

Ejercicio 1: Enviar señales	Secreto:	
Ejercicio 2: Recibir señales	Secreto:	
Ejercicio 3: Crear procesos	Secreto:	Padre: Hiio 1: Hiio 2:
Ejercicio 4: Crear hilos	Secreto:	Pid Hiio 1: Nº hilos creados:
Ejercicio 5: Contar señales	Secreto:	Pid Hiio 2: Nº Señales recibidas:
Ejercicio 6: Cerrar aplicación	Secreto:	

Firma