01-preface

Programas básicos de manejo de *system calls* y *library calls*

- **01-hello.c:**
- – Muestra el más simple programa en C. Se hizo hincapié en la compilación y generación del ejecutable a través de make.

- **02-args.c:**
- – Se muestra qué recibe un programa desde el *shell* como argumentos de *main* y que retorna al *shell* como retorno de *main*;.
- – Este estado de terminación se puede ver en el *shell* imprimiendo la variable **?**

- **03-fork.c;**
- – Un ejemplo muy sencillo del uso de *fork* para mostrar como el hijo es una réplica del padre pero que constituye un proceso completamente diferente.
- – Se muestran en el programa los *identificadores de proceso* respectivos.

- **04-fork-wait:**
- – Se ha agregado en el código del padre, esperar la terminación del hijo, antes de terminar él mismo
- – El hijo, a su vez, una vez creado, hace una llamada a la función de biblioteca *sleep* esperando por 5 segundos, si el proceso se lanza sin argumentos, y sino por una cantidad de segundos igual a lo que indica un argumento opcional.

- – Al permitir, de esta última manera, que se pueda colocar una cantidad apreciable (por ejemplo 30 segundos) y lanzar el proceso en *background* mediante el símbolo **&**, da la posibilidad de ganar el control de la línea de comando mientras se sigue ejecutando el proceso y poder, mediante el comando **ps -l** ver el estado de bloqueo en que se encuentran ambos procesos.

- **05-zombie.c:**

- – En este programa, se crea un proceso hijo que termina inmediatamente, mientras que el padre, después de mostrar los datos de hijo, entra en el la función de biblioteca *sleep* por un período de tiempo que, como en el programa anterior, depende de si hay argumento en la línea de comando. Una vez que termina dicho período, espera la terminación del hijo (que para este entonces ya había terminado) y lo muestra;

- – Como en el caso anterior, es interesante poner un valor de argumento suficientemente alto como para poder ver el estado de los procesos (antes que termine el período en el cual el padre permanece bloqueado en *sleep*), mostrando que el hijo está en la tabla de procesos pero ya no compite por el procesador y está marcado como *<defunct>*

- **06-exec.c:**

- – En este programa, se pretende que el hijo tome la imagen de otro programa y que no se limite a tener el mismo código del padre, como ha sido hasta ahora en los ejemplos anteriores.

- – Ello se logra haciendo que en el código del hijo se llame a cambiar la imagen por aquella que se defina en el *system call* correspondiente a la familia denominada genéricamente *exec*, en este caso particular, al system call *execvp*.

- – El programa *06-exec* debe recibir como argumentos el nombre del programa a ejecutar por el hijo así como sus argumentos, por ejemplo: **$./06-exec ls -l**; estos argumentos los recibe *main* y, luego de verificar que hay más de uno, llama a la función *do_process* a quien pasa como único argumento el puntero al segundo argumento recibido.

- – Esta función procede a crear el hijo quien inmediatamente llama a *execvp* colocando como primer argumento el contenido del puntero recibido, que no es otra cosa que el *string* del nombre del programa a buscar y cargar, y como segundo argumento, la lista de argumentos que debe recibir el programa a la usanza normal.
- – Una nota importante del código del hijo: obsérvese que, si la llamada a *execvp* es exitosa, nunca más retornará a este código, por lo cual el código que está escrito después de *execvp* solo se ejecutará si esta llamada falla.

- **07-simple-shell.c**:
- – Mediante lo realizado en el programa anterior, se está muy cerca de realizar un *shell* muy simple.
- – Obsérvese que la función *do_process* se ha mantenido tal cual el ejemplo anterior y que en *main* el loop que existe ahora se repite para siempre y solamente se sale de él por una condición de retorno de la nueva función *get_command_string*.
- – Si la función *get_command_string* tiene éxito, lo que retorna dicha función, que se coloca en la variable *cmd_arg* se le pasa a la conocida función *do_process* con lo cual reemplaza lo que en el programa anterior era el puntero al comando que se desea que ejecute el hijo.
- – Sólo queda comprender lo que hace la función *get_command_string*: primero imprime en *stdout* lo que se denomina el *prompt* para invitar al usuario que ingrese el comando e inmediatamente se llama a *fgets* que permite entrar una línea terminada con *new line* desde la entrada *standard*, colocándole en el arreglo denominado *line*
- – En el particular caso que *fgets* retorne el puntero nulo, lo cual significa que en la entrada existió un *end of file* (lo cual si se está ingresando de teclado es equivalente a colocar *control-D*) o, si la longitud de la línea recibida en el arreglo *line* es cero, se retorna NULL, terminando de esta manera la ejecución del *shell*.
- – En el caso que lo ingresado sea válido, a nivel de lo que considera *fgets*, entonces en la línea siguiente, se anula la existencia del *new line* en el arreglo, si este existiese
- – Como ocurre en el *shell*, debe haber una forma elegante para salir del mismo, que generalmente es con la palabra *exit* lo cual se hace verificando mediante la función de biblioteca *strcmp* comparando lo ingresado con el string **exit**

- – Recién ahora, entonces, se está en condiciones de separar cada uno de los *tokens* o palabras existentes en la línea de entrada y colocándoles en el arreglo de punteros a *char* denominado *argv*; de acuerdo a lo requerido por *execvp* dicho arreglo debe tener un último valor con el puntero nulo, de manera que quien lea el arreglo sepa donde termina.
- – Esta separación se hace con la ayuda de la función de biblioteca *strtok* (por favor, use *man* o busque en *Google* para comprender exactamente la función y comprender porque está usada en dos instancias en este caso)

- **99-fbomb.c**:
- – Esta es la famosa bomba *fork*: **CUIDADO**: no hay problema en compilarla pero **no la ejecute**. piense como funciona y que le pasa al sistema operativo con éste programa en ejecución; averigüe en Google sobre ella y vea que también existe como script a nivel de *shell*.