Universidad de Mendoza - Computacion 2

Guía de ejercicios de laboratorio

Temario

NOTA:.....2
Ejercicio 1 - Getopt.....3
Ejercicio 2 - Getopt.....3
Ejercicio 3 - Fork.....3
Ejercicio 4 - Fork.....3
Ejercicio 5 - Fork.....3
Ejercicio 6 - Señales.....4
Ejercicio 7 - Señales.....4
Ejercicio 8 - Señales.....4
Ejercicio 9 – Señales.....4
Ejercicio 10 – Señales / Pipes.....4
Ejercicio 11 – Pipes.....5
Ejercicio 12 – FIFO / pipes.....5
Ejercicio 13 – FIFO.....5
Ejercicio 14 – Multiprocessing.....5
Ejercicio 15 – Multiprocessing / Pipe.....6
Ejercicio 16 – Multiprocessing / Pipe / FIFO.....6

NOTA:

Todos los ejercicios propuestos deben ser resueltos y publicados en el repositorio GitLab personal. Junto con los documentos de cátedra se encuentra una guía para crear y trabajar los pasos básicos de GIT con GitLab.

Ejercicio 1 - Getopt

Crear una calculadora, donde se pase como argumentos luego de la opción -o el operador que se va a ejecutar (+,-,*,/), luego de -n el primer número de la operación, y de -m el segundo número.

Ejemplo:

```
./calc -o + -n 5 -m 6
5 + 6 = 11
```

Considerar que el usuario puede ingresar los argumentos en cualquier orden. El programa deberá verificar que los argumentos sean válidos (no repetidos, números enteros, y operaciones válidas.

Ejercicio 2 - Getopt

Escribir un programa que reciba dosmbres de archivos por línea de órdenes utilizando los parámetros “-i” y “-o” procesados con getopt().

El programa debe verificar que el archivo pasado a “-i” exista en el disco. De ser así, lo abrirá en modo de solo lectura, leerá su contenido, y copiará dicho contenido en un archivo nuevo cuyombre será el pasado a “-o”. Si el archivo nuevo ya existe, deberá sobreescribirlo.

Ejercicio 3 - Fork

Escribir un programa que en ejecución genere dos procesos, uno padre y otro hijo. El hijo debe escribir "Soy el hijo" 5 veces. El padre debe escribir "Soy el padre" 2 veces. El padre debe esperar a que termine el hijo y mostrar el mensaje "Mi proceso hijo termino".

Ejercicio 4 - Fork

Escribir un programa en C que en ejecución genere 2 procesos (padre e hijo). Este programa recibirá como único argumento del intérprete de órdenes un entero. El padre incrementará ese valor en 4 y lo mostrará por pantalla. El hijo decrementará ese valor en 2 y lo mostrará en pantalla.

Ejercicio 5 - Fork

Realice otro programa que genere la siguiente configuración de procesos, y cada proceso al iniciar deberá mostrar “Soy el proceso N, mi padre es M” N será el PID de cada hijo, y M el PID del padre.

```
            padre
           /  |  \
      hijo1 hijo2 hijo3
```

Ejercicio 6 - Señales

Hacer un programa que capture la señal que se envía al presionar Ctrl+c y la ignore la primera vez, mostrando un mensaje por pantalla "Esta vez me saldré, inténtalo nuevamente". Establecer los valores por defecto para que al segundo intento se realice lo esperado.

Ejercicio 7 - Señales

Desde un proceso padre crear un hijo y enviarle una señal SIGUSR1 cada 5 segundos. El proceso hijo estará a la espera y escribirá un aviso en pantalla cada vez que llega la señal de su padre.

Ejercicio 8 - Señales

Realice un programa que genere dos procesos. El proceso hijo1 enviará la señal SIGUSR1 al proceso padre y mostrará la cadena "ping" cada 5 segundos. El proceso padre, al recibir la señal SIGUSR1 enviará esta misma señal al proceso hijo2. El proceso hijo2, al recibir dicha señal mostrará la cadena "pong" por pantalla.

```
Soy proceso hijo1 con PID=1545: "ping"
Soy proceso hijo2 con PID=1547: "pong"

[... 5 segundos mas tarde ...]
Soy proceso hijo1 con PID=1545: "ping"
Soy proceso hijo2 con PID=1547: "pong"
[... y así sucesivamente ...]
```

Ejercicio 9 – Señales

Escribir un programa que genere 3 procesos hijos. El proceso padre enviará una señal a cada uno de sus hijos (SIGUSR1). Cada hijo mostrará por pantalla el mensaje: “Soy el PID xxxx, recibí la señal yyy de mi padre PID zzzz”, donde xxxx es el pid del hijo en cuestión, yyy el número de señal, y zzzz el pid del proceso padre inicial.

Ejercicio 10 – Señales / Pipes

Escribir un programa que genere un proceso hijo (fork), y a su vez, éste genere un nuevo proceso hijo (fork) (ver esquema abajo). Todos los procesos estarán comunicados mediante un pipe.

```
    A → B → C
```

El proceso B escribirá en el pipe el mensaje “Mensaje 1 (PID=BBBB)\n” en el momento en que el proceso A le envíe la señal USR1.

Cuando esto ocurra, el proceso B le enviará la señal USR1 al proceso C, y C escribirá

en el pipe el mensaje “Mensaje 2 (PID=CCCC)\n”.

Cuando el proceso C escriba este mensaje, le enviará al proceso A la señal USR2, y el proceso A leerá el contenido del pipe y lo mostrará por pantalla, en el formato:

```
A (PID=AAAA) leyendo:
Mensaje 1 (PID=BBBB)
Mensaje 2 (PID=CCCC)
```

Notas:

- AAAA, BBBB, y CCCC son los respectivos **PIDs** de los procesos A, B y C.
- Inicialmente el proceso B envía un mensaje al pipe cuando A le hace llegar la señal USR1, por lo que B deberá quedar esperando al principio una señal.
- El proceso C también deberá quedar esperando inicialmente una señal que recibirá desde el proceso B para poder realizar su tarea.
- El proceso A inicialmente envía la señal USR1 al proceso B, y automáticamente debe quedar esperando una señal desde el proceso C para proceder a leer desde el pipe.
- Los saltos de línea en los mensajes (“\n”) son necesarios para que el pipe acumule varias líneas, y no concatene los mensajes en la misma línea.
- El pipe es uno solo en el que los procesos B y C deberán escribir, y el proceso A deberá leer.
- ***Pista***: la utilización de señales es para sincronizar los procesos, no necesariamente los handlers deben gestionar el pipe.

Ejercicio 11 – Pipes

Escribir un programa que lance dos procesos hijos (fork).

Uno de los procesos hijos se encargará de leer desde la entrada estándar líneas de texto, y en la medida en que el usuario escriba, el proceso las irá enviando por un pipe que compartirá con el otro proceso hijo.

El segundo proceso se encargará de leer desde el pipe las líneas que el primer proceso escriba, y las irá mostrando por pantalla en el formato “Leyendo (pid: 1234): mensaje”, donde 1234 es el pid de este segundo proceso, y “mensaje” es el contenido leído desde el pipe.

Ejercicio 12 – FIFO / pipes

Crear un sistema Productor-Consumidor (Escritor-Lector) donde un proceso productor almacene un mensaje de saludo en una tubería FIFO. Ese mensaje será enviado mediante línea de comandos como argumento del programa. Ejemplo

```
./saludofifo HOLA
```

Otro programa (consumidor) deberá leer el mensaje desde la tubería FIFO, generar un proceso hijo (fork) y enviarle por PIPE el mensaje. El proceso hijo mostrará por pantalla el mensaje recibido.

```
Proc1_fifo_escritor → FIFO → Proc2_fifo_lector → pipe → Proc2hijo
```

Ejercicio 13 – Multiprocessing

Escribir un programa que genere tres procesos hijos (fork) cuyos nombres serán “Hijo1”, “Hijo2” e “Hijo3”.

El proceso padre debe mostrar por pantalla el mensaje “Lanzando hijo NN” en la medida que ejecuta cada uno de los procesos.

Cada hijo que se lance mostrará por pantalla su pid, el pid de su padre, esperará un segundo, y luego terminará.

El padre debe esperar a la terminación de sus hijos.

Ejercicio 14 – Multiprocessing / Pipe (REVIEW)

Replicar el comportamiento del ejercicio 11 utilizando el módulo Multiprocessing.

Ejercicio 15 – Multiprocessing / Pipe / FIFO

Replicar el comportamiento del ejercicio 12 utilizando el módulo Multiprocessing para la creación del proceso hijo del programa consumidor.

Ejercicio 16 – Mini Chat

Escribir dos programas que interactúen entre si como si fuesen clientes peer-to-peer. El funcionamiento será el siguiente:

- Llamemos AA al primer programa en ejecutarse, y BB al segundo.
- AA quedará escuchando un texto desde el otro programa.
- BB permitirá al usuario escribir por entrada estándar un mensaje. Al dar *enter* se enviará dicho mensaje a AA, y automáticamente BB quedará esperando un mensaje remoto.
- AA al recibir el mensaje lo mostrará por pantalla, y permitirá al usuario ingresar desde entrada estándar un texto.
- Se repite el comportamiento anterior, pero en sentido inverso.
- La idea es que se produzca una sucesión de mensajes de ida y vuelta entre AA y BB, donde BB enviará el primer mensaje, y todos los mensajes serán leídos desde la entrada estándar.

- El mecanismo de comunicación entre procesos a utilizar queda a decisión del diseñador y programador :)

Ejercicio 17 – Fork / MQ

Escriba un programa que cree diez procesos hijos (fork).

Cada proceso al iniciar debe escribir por pantalla "Proceso xx, PID: xxxx"

A su vez, cada proceso debe esperar i segundos, donde i es el número de proceso hijo, y luego escribir en un formato "xxxxx\t" el pid en una cola de mensajes.

El padre debe esperar a que todos los hijos terminen, y luego leer el contenido de la cola de mensajes y mostrarlo por pantalla.

Ejercicio 18 – Multiprocessing / MQ

Reescriba el programa anterior pero esta vez haciendo uso del módulo multiprocessing.

Ejercicio 19 – Multiprocessing / MQ

Reescriba el programa anterior pero esta vez haciendo uso de pools de procesos.

Ejercicio 21 – Pool of workers

Escriba un programa que haga uso de un pool de procesos para calcular las raíces cuadradas de los números impares comprendidos entre los valores N y M pasados por argumentos al programa mediante los modificadores -n y -m (validar las entradas con getopt).

Ejemplo de ejecución:

```
python3 calculador.py -n 12 -m 20
Raíz cuadrada de 13: 3.605
Raíz cuadrada de 15: 3.872
Raíz cuadrada de 17: 4.123
Raíz cuadrada de 19: 4.358
```

Escriba dos versiones del programa:

- Una haciendo uso de la función **map**
- Una haciendo uso de la función **apply**

Conversor de subtítulos

Ejercicio 10 FIFO / SHM

Escriba un programa que lea desde el stdin contenido, y lo escriba en un segmento de memoria compartida.

El programa debe recibir por parámetros elmbre de un archivo (inexistente).

Luego debe escribir en un fifo elmbre del archivo leído por parámetro.

Otro proceso debe leer desde el fifo elmbre del archivo, y cree el archivo en el disco, y le escriba lo que hay en el shm.

Llamada al programa 1:

```
./prog1 /tmp/archivo.txt <enter>
hola mundo
este es el contenido
<ctrl+d>
Llamada al programa 2:
./prog2
```

Ejercicio 14 – Sockets Unix DGRAM

Escriba un programa cliente-servidor con sockets Unix/DGRAM que tenga el siguiente comportamiento.

1. El usuario ejecutará el programa servidor pasándole dos argumentos:
    1. Elmbre del archivo de socket Unix.
    2. Elmbre de un archivo de texto inexistente
2. El servidor creará el socket y el archivo de texto en el path especificado.
3. El cliente recibirá un argumento por línea de comandos: elmbre del socket que utiliza el servidor.
4. El cliente comenzará a leer desde STDIN texto hasta que el usuario presione

Ctrl+d.

5. El cliente enviará todo el contenido por el socket al servidor.
6. El servidor leerá todo el contenido desde el socket hasta que encuentre un EOF.
7. El servidor almacenará todo el contenido en el archivo de texto creado.

Ejercicio 15 – Sockets Inet DGRAM

Ídem al ejercicio anterior, pero haciendo uso de sockets INET/DGRAM.

Ejercicio 16 – Sockets Inet STREAM

Ídem al ejercicio anterior, pero haciendo uso de sockets INET/STREAM.

Ejercicio 17 – Sock Inet Str / fork

Escribir un programa servidor que levante un socket INET STREAM en todas las direcciones locales, y en un puerto específico que será pasado por parámetros de línea de comandos.

El servidor debe escuchar en el puerto especificado, y debe permitir, mediante subprocesos atender a varios clientes simultáneamente.

Ejecutar el servidor, por ejemplo, así:

```
./servidor 1234
```

Escribir un programa cliente que utilice sockets INET STREAM que reciba por parámetros dirección ip del servidor y puerto de esta forma:

```
./cliente 127.0.0.1 1234
```

El cliente con esos datos debe conectarse al servidor al puerto especificado, y luego debe leer desde la entrada estándar (STDIN_FILENO) y todo lo que el usuario escriba debe ir enviandoselo al servidor. El servidor los recibirá e irá mostrándolos por pantalla.

Cuando el usuario escriba "exit" el cliente debe terminar la conexión y cerrar. El servidor debe cerrar la conexión con ese cliente.

Ejercicio 18 – Sock Inet Str / thread

Ídem al ejercicio anterior, pero utilizando multithreading en vez de multiprocesos del lado del servidor.

Ejercicio 19 – SHM / Semáforos

Escriba un programa que lance 15 procesos hijos, cada uno de los cuales deberá escribir una letra del alfabeto en un segmento SHM (por ejemplo, 'A' para el primer proceso, 'B' para el segundo, etc) cinco veces, con un delay de 1 segundo entre impresiones, en un archivo de texto que será pasado por parámetros en la línea de comandos. Ejemplo:

```
./programa -f /tmp/test.txt
```

Haga uso de getopt() para leer los parámetros de línea de órdenes.

Como condición, las cinco letras de cada proceso podrán estar intercaladas. Haga uso de semáforos POSIX para controlar la ejecución.

El resultado en el archivo, será, por ejemplo:

```
AAAAADDDDDEEEEEBBBBB ...
```

Ejercicio 20 – Sock Inet Str / thread

Escriba un servidor tcp que atienda en el puerto 1234, y reciba los siguientes comandos: ABRIR, CERRAR, AGREGAR, y LEER, por parte de los clientes.

Si el cliente envia un comando ABRIR, el servidor deberá solicitarle unmbre de archivo.

Si el cliente envía un comando "AGREGAR" el servidor deberá solicitarle una cadena de texto para agregar al final del archivo.

Si el cliente envía un comando "LEER" el servidor le deberá enviar al cliente el contenido del archivo.

Si el cliente envía un comando "CERRAR" el servidor deberá cerrar el archivo y cerrar la comunicación con el cliente.

El servidor deberá poder mantener conexiones en simultáneo con varios clientes mediante un sistema multihilo.

Deberá también programar a los clientes para que le soliciten comandos al usuario mediante un simple menú de opciones, e interactúe con el servidor para que pueda realizar sus tareas.