TP II – Unidad I a IV - OBLIGATORIO

Objetivo: Implementar una serie de ejercicios de programación ANSI C a modo de sintetizar toda la ejercitación propuesta en clase y en los prácticos no obligatorios de las Unidades I a IV del programa de estudios de la asignatura.

1 Implementación del programa Shell (en su versión más completa, que soporta procesos foreground y background; que haga uso de la señal SIGCHLD para verificar la finalización de procesos en background) que evite la existencia de procesos zombies. El programa termina cuando el usuario ingresa el comando "salir". Si el usuario presiona CTRL+C el programa esperará a que terminen todos los procesos y luego finalizará sin dejar procesos zombies o huérfanos.

Links de ayuda para este ejercicio:
Minishell 01
https://www.youtube.com/watch?v=8eZUwHpPVjE

C Linux - Relación entre wait y exit - 05
https://www.youtube.com/watch?v=KKmqjPQu3Oo

C Linux Señales 01
https://www.youtube.com/watch?v=RqGh52B5B5A

C Linux Señales 02 Ejemplo
https://www.youtube.com/watch?v=3mkP_GAxKfs

Klingon Vs. La Federación Interestelar (Señales y Conjunto de Señales)
https://www.youtube.com/watch?v=ZzXqR1VVBOU

2 Implementar un proceso al cual se le indique por línea de comando la cantidad de procesos a crear, todos los procesos a crear serán hermanos; cada uno de ellos retornará un valor entero distinto al proceso padre. Los procesos hijos quedan en un loop eterno, en una espera no activa, cuando recibe la señal SIGUSR1 o SIGINT el proceso hijo termina retornando un valor entero distinto al de sus hermanos. El proceso padre reportará por pantalla la sumatoria de los retornos de los procesos hijos creados. No se permitirá que existan procesos huérfanos o zombies.

3 Implementación de una sincronización de los hilos A, B y C de forma tal, que la secuencia de ejecución y acceso a su sección crítica sea la siguiente: ABAC... detener el proceso luego de N iteraciones completas (el número N se ingresa por línea de comandos). Resolver la sincronización con variables Mutex (librería pthread).

4 Implementación de una sincronización con procesos independientes A, B y C de forma tal, que la secuencia de ejecución y acceso a su sección crítica sea la siguiente: ABAC... detener el proceso luego de N iteraciones completas (el número N se ingresa por línea de comandos). Resolver la sincronización con: semáforos SVR4.

Links de ayuda:

Sincronización BACA
https://www.youtube.com/watch?v=UZLaeemleRk

C Linux Semáforos System V 01
https://www.youtube.com/watch?v=ub2YTpCq3Aw

C Linux Semáforos System V Ejemplo 02
https://www.youtube.com/watch?v=vvI2-CZ1Bik

C Linux Semáforos System V Otros Dos Ejemplos 03
https://www.youtube.com/watch?v=Gy4baWme_ek

C Linux Semáforos System V ABoCDoE 07
https://www.youtube.com/watch?v=vLZt2yYjX6w

Semáforos POSIX Sin Nombre en Procesos Independientes, ejemplo en lenguaje C
https://www.youtube.com/watch?v=T6ZOOxhDSe8

5 Implementación de una sincronización con procesos emparentados PadreA, HijoB y HijoC de forma tal, que la secuencia de ejecución y acceso a su sección crítica sea la siguiente: PadreAHijoBPadreAHijoC... detener el proceso luego de N iteraciones completas (el número N se ingresa por línea de comandos). HijoB e HijoC son hermanos. Resolver la sincronización de la forma que a Ud. le parezca más apropiada (no usar señales).

6 Realizar un programa que reciba por línea de comandos un comando a ejecutar y sus argumentos. El programa va crear un proceso hijo con dicho comando y el proceso padre leerá la salida del comando. Para lograr esto, cree un pipe entre proceso padre e hijo y utilice la función dup() o dup2() para duplicar la salida en el pipe y luego leer la salida usando el pipe. No está permitido usar la función popen() que resume o simplifica la técnica propuesta. No generar procesos huérfanos ni zombies.

Links de ayuda para este ejercicio:
Pipes 1
https://www.youtube.com/watch?v=WigeRo_m6VE

Pipes 2
https://www.youtube.com/watch?v=XM8DbmgI5TE

Pipes 3
https://www.youtube.com/watch?v=tXJMeasmlI8

7 Implementar un servidor (basado en hilos) usando sockets tcp/ip. El cliente (use telnet) envía un comando Linux al servidor, el servidor lo ejecuta y devuelve la salida del comando al cliente. La interacción entre cliente y servidor termina cuando el cliente envía el comando “salir”. Reusar el código del ejercicio anterior.

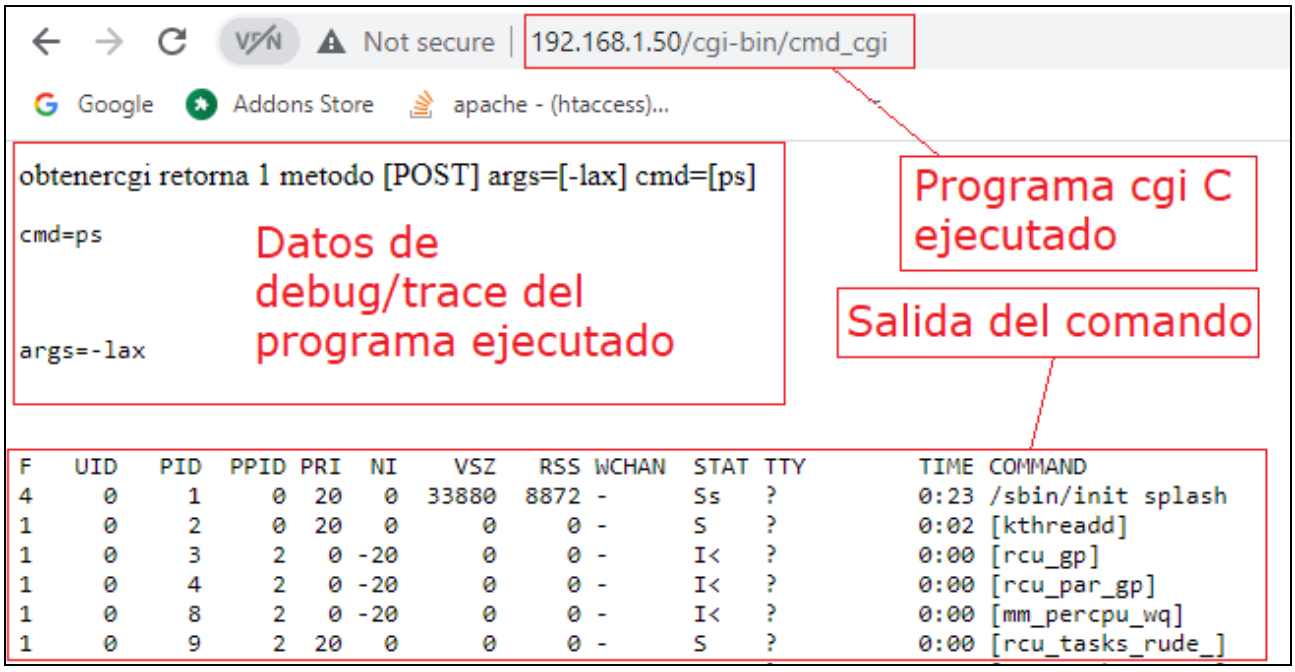
Ejercicios Opcionales:

8 Implementar un Sistema de Monitoreo Distribuido utilizando un servidor web (server1), colocando en la página index.html los siguientes links asociados a los siguientes comandos:

Link	Comando
Procesos	ps -lax
Memoria	free -h
Disco	df -h
IPC	ipcs

por ejemplo, supongamos que el computador a monitorear tiene la IP 192.168.1.50:

Entonces, cuando el operador hace click en el link “Procesos”, se ejecutará el programa CGI “genérico” que sirva para cualquier comando Linux (recibe –ya sea por método POST o GET- dos parámetros: “cmd” –que indica el comando a ejecutar- y “args” –que indica los argumentos del comando a ejecutar-). Aquí mostramos una salida posible, indicando algunos datos adicionales que usamos en la prueba del programa CGI:



Implemente el programa CGI genérico, instálelo en la carpeta cgi-bin del servidor web, implemente la página html index.html, no puede utilizar la función popen() .

Antes de instalar servidor web grchere, debe tener instalado:
Berkeley DB 5.3:

```
$ sudo apt-get install libdb5.3 libdb5.3-dev
```

Instalar gcc, make etc:

```
$ apt-get install build-essential manpages-dev gcc make
```

Instalar cliente git:

```
$ apt-get install git
```

Instalar servidor web grchere (https://gitlab.com/grchere/webserver):

```
$ git clone https://gitlab.com/grchere/webserver.git
$ cd webserver
$ ./configure
$ make
$ make apps
```

Los comandos anteriores compilan ./wserver/server1 y ./wserver/server2 ; server1 es un servidor web básico basado en procesos pesados que puede utilizar para resolver el ejercicio 8, server2 es un servidor web básico basado en procesos livianos y además implementa la posibilidad de hacer aplicaciones web usando librerías .so (share objects). Ver carpeta ./doc para más info.

Verifique la configuración de server1: editar archivo ./wserver/server1.conf
Verifique la configuración de server2: editar archivo ./wserver/server2.conf

Para ejecutar server1: (no usar usuario root, verifique que el puerto tcp/ip que va a usar este libre)

```
$ cd ./wserver
$ ./server1
```

Observe los mensajes en la consola, los servidores no tienen archivos .log, si todo ok, abra su navegador e introduzca: http://localhost:8200 si configuro usar el puerto tcp/ip 8200, si va a usar el puerto 80, simplemente introduzca http://localhost esto provocará que el servidor devuelva la página web index.html que se encuentra en ./www
Para resolver el ejercicio 8 deberá modificar ./www/index.html y deberá implementar un proceso CGI que deberá compilar y copiar en ./cgi-bin . No puede usar la función popen(), deberá hacer fork(), exec() y conectar pipes de la forma adecuada para leer la salida del comando.