Historial de Comandos y Códigos Fuente (14/08/19 al 04/09/19)

Miércoles 14 de Agosto

```
echo $MANPATH #mostrar el valor de la variable $MANPATH (contiene varias rutas, separadas po
ls -l /usr/man/man2/ #ver el contenido de una de las rutas includas en $MANPATH
man man #consultar el manual de usuario del comando que sirve para ver los manuales de usuar
        #  en este manual puede observarse que la sección 2 de los manuales corresponde a li
man 2 open #consultar el manual de la llamada al sistema para abrir archivos
man 2 write #consultar el manual de la llamada al sistema para escribir en archivos
which audacious #consultar cuál es el binario que se invoca cuando se ejecuta el comando "au
file /usr/bin/audacious #ver tipo del archivo
                        #  /usr/bin/audacious: ELF 64-bit LSB executable, x86-64, version 1
man 2 inb #consultar el manual de la llamada al sistema para leer (a bajo nivel) de disposit
```

Lunes 19 de Agosto

```
ls -lh PROKUF_DEV.log #listar información de un archivo compreso en modo largo (con detalles
xz PROKUF_DEV.log  #comprimir el archivo (nótese que al comprimirlo gana el sufijo .xz)
unxz PROKUF_DEV.log.xz #descomprimir el archivo (nótese que al descomprimirlo pierde el sufi
time xz PROKUF_DEV.log #comprimir el archivo pero tomando el tiempo que demora la operación
time cat PROKUF_DEV.log #mostrar el contenido del archivo

ls -l /lib64/ld-2.23.so #listar información del archivo que corresponde a la librería madre
date #mostrar la fecha/hora actual
strace date #ejecutar el comando date pero desde strace para que muestre todas las llamadas
strace date 2>&1 | less #lo mismo que el anterior, con algunos detalles:
                        #  dado que strace despliega las llamadas al sistema en la salida es
                        #  una vez hecho lo anterior, con | se pasa la salida estándar de st
which date #consultar cuál es el binario que se invoca cuando se ejecuta el comando date
ls -l /usr/bin/date #listar información del archivo; /usr/bin/date realmente es un archivo d
file /usr/bin/date #consultar el tipo del archivo, en este caso será un enlace simbólico
file /bin/date #consultar el tipo del archivo, en este caso será un ELF (Executable Linux Fo
/bin/date #ejecutar el binario directamente

tac #tac, al contrario de cat, muestra la entrada estándar al revés (primero las últimas lín
tac > ordenados #el redireccionador > enviará la salida estándar de tac hacia el archivo ord
cat ordenados #mostrar el contenido del archivo ordenados
ls -l ordenados desordenados #listar información de los archivos ordenados y desordenados, d
ls -l ordenados desordenados > salida #redireccionar la salida estándar al archivo salida (e
ls -l ordenados desordenados 2> error #redireccionar la salida estándar de error al archivo
cat salida #mostrar el contenido del archivo
cat error #mostrar el contenido del archivo
tac <ordenados > desordenados 2> error #redireccionar:
                                       #  con < la entrada estándar (será el archivo ordenad
```

```
                                        #  con > la salida estándar (será el archivo desorden
                                        #  con 2> la salida estándar de error (será el archi
cat desordenados
tac > ordenados
tac 1> ordenados 2> error #usar 1> es equivalente a >



#esto se ejecutó en otra consola mientras se ejecutaban los comandos tac mencionados antes
pgrep tac #buscar los procesos con nombre tac que existen
ls -l /proc/22853/ #una vez encontrado el PID del proceso de interés, consultar información
ls -l /proc/22853/fd #consultar información de sus archivos abiertos por el proceso
pgrep tac
ls -l /proc/23340/fd
```

Miércoles 21 de Agosto

```
#include "stdio.h"
#include "unistd.h"

int main (int argc, char *argv[]){

    printf("%s\n",argv[0]);
    printf("PID: %d\n",getpid());
    printf("usuario: %d\n",getuid());

    return 0; //25; //valor de retorno
}
```

```
pgrep xcalc #bucar el PID de los procesos llamados xcalc
date #mostrar la fecha/hora actual
date "+%Y-%m-%d" #agregar un parámetro al comando para mostrar la fecha en un formato disti
cal #mostrar el calendario del mes actual
cal -3 #agregar un parámetro para mostrar el calendario del mes actual, junto al mes anteri
cal -y #mostrar el año completo
cal 6 1997 #mostrar un mes en específico
firefox --ProfileManager #pasar parámetros a firefox
firefox www.opera.com
pgrep soffice #buscar proceso
xcalc #abrir una calculadora

cat /proc/sys/kernel/pid_max #consultar cuál es el máximo número de PID que asignará el ker
date #como este comando tendrá éxito, su valor de retorno, asignado a la variable $?, será
echo $? #consultar el valor de retorno del comando ejecutado inmediatamente antes
cal 1997 6 #como este comando fallará (lo correcto es pasarle parámetros de mes y año, no d
echo $?
```

```
vim prueba.c #editar un programa en C
gcc -o prueba prueba.c #compilar el programa, con -o se indica cómo se llamará el archivo ej
./prueba #ejecutar el programa recién compilado
echo $?
man 2 getpid #consultar el manual de la llamada al sistema
man getuid #consultar el manual de la llamada al sistema
echo $UID #mostrar el valor de la variable $UID que contiene el identificador del usuario qu

less /proc/cpuinfo #consultar información de los CPUs de la computadora
less /proc/meminfo #consultar información de la memorua de la computadora
cat /proc/partitions #consultar información de las particiones de disco disponibles
cat /proc/uptime #consultar desde hace cuánto tiempo (en segundos) se arrancó el sistema ope
uptime #mostrar cuándo tiempo lleva corriendo el sistema operativo
ls -l /proc/ #hacer una lectura larga del contenido del directorio /proc
ls /proc/ #hacer una lectura del contenido del directorio /proc
cat /proc/version #consultar la versión del kernel
cat /proc/filesystems #consultar información de los sistemas de archivos existentes
pgrep soffice #buscar proceso por su nombre
ls /proc/20517/ #consultar contenido del directorio del proceso
ls -l /proc/20517/exe #exe es un enlace simbólico hacia el binario que se invocó para crear
cat /proc/20517/cmdline #muestra la línea de comando con sus parámetros ejecutada para inic
pgrep xpenguins #buscar proceso
cat /proc/10080/cmdline
ls /proc/20517/
cat /proc/20517/environ #mostrar las variables de ambiente definidas para el proceso
less /proc/20517/limits #mostrar los límites de recursos establecidos para el proceso
ls -l /proc/20517/fd #consultar los archivos utilizados por el proceso (0: entrada estándar,
ls -l /proc/20517/fd/28 #consultar un archivo en específico utilizado por el proceso
cat /proc/20517/fdinfo/28 #consultar detalles de un archivo en específico utilizado por el p
ls -l /proc/20517/task/ #ver los hilos, subprocesos o procesos livianos que tiene el proceso
                        #  en este directorio habrá un directorio por cada hilo del proceso
ls /proc/20517/task/20569 #consultar información del hilo 20569 del proceso 20517
ls -l /proc/20517/cwd #ver el directorio de trabajo actual del proceso
cd /tmp/ #cambiarse de directorio
echo $$ #variable que contiene el PID del proceso actual
ls -l /proc/20523/cwd #ver el directorio de trabajo actual del proceso
ls -l /proc/20523/root #enlace simbólico al sistema de archivos raíz (/) desde donde se est
less /proc/20517/maps #información del uso de memoria del proceso
cd
man 2 fork #consultar el manual de la llamada al sistema fork
pstree #ver el árbol de procesos
ps axjf #otra forma de ver el árbol de procesos
ps axjf | sed -n '1p;/konsole/{N;N;N;N;p}' #filtrar un poco la salida del árbol de procesos
                                           #  primero la salida de ps axjf se pasa a través
                                           #  sed es un editor de flujo, puede hacer transf
```

```
#    -n hace que sed por defecto no imprima ningu
#    luego se le pasan los distintos comandos de
#      1p indica que imprima la línea 1 (la de la
#      /konsole/ busca la palabra que está entre
#        N hará que lea la siguiente línea de la
#        p hará que se imprima finalmente la líne
```

```
#lo siguiente se ejecutó como root
su #cambiarse al usuario root (switch user)
cat /proc/sys/kernel/pid_max #consultar cuál es el máximo número de PID que asignará el kern
echo 25000 > /proc/sys/kernel/pid_max #modificar el número máximo de PID que asignará el ker
cat /proc/sys/kernel/pid_max
pgrep xcalc
echo 22600 > /proc/sys/kernel/pid_max
echo 23000 > /proc/sys/kernel/pid_max
pgrep xcalc
echo 32768 > /proc/sys/kernel/pid_max #regresar a su valor original
```

Lunes 26 de Agosto

```
#include "stdio.h"
#include "unistd.h"

int main (int argc, char *argv[]){

    /*
    char *arg[]={
        "./20190821",
        NULL
    };
    char *amb[]={
        NULL
    };
    char *arg[]={
        "/bin/date",
        "+%Y",
        NULL
    };
    char *amb[]={
        "LANG=es_ES",
        NULL
    };
    */
    char *arg[]={
```

```
        "/usr/bin/glxgears",
        NULL
    };
    char *amb[]={
        "HOME=/home/gerardo",
        "DISPLAY=:1.0",
        NULL
    };

    printf("%s\n",argv[0]);
    printf("PID del proceso: %d\n",getpid());

    execve(arg[0],arg,amb);

    return 0;
}
```

```
ls -l /usr/src/linux #directorio con el código fuente del kernel
cd /usr/src/linux
du -shc #ver cuánto espacio en disco ocupa el directorio de trabajo actual (también puede po
ls -l
vim kernel/pid.c #ver en el fuente del kernel el valor mínimo al que regresa el número de PI
cat /proc/sys/kernel/pid_max #consultar el valor máximo posible para un PID

echo $PS1 #variable de ambiente que define el formato del prompt primario
echo $PS2 #variable de ambiente que define el formato del prompt secundario
cat \ #con la \ del final, Bash entiende que el comando está incompleto entonces muestra el
export PS1="$ " #cambiar el prompt por uno más corto (cambiando el valor de la variable cor
pwd #mostrar el directorio de trabajo actual

vim 20190826.c #editar
gcc 20190826.c #compilar, como no se especifica un archivo de salida (con -o) entonces se cr
ls -ltr #hacer una lectura del directorio en formato largo, ordenando por tiempo a la invers
./a.out #ejecutar programa recién compilado
vim 20190821.c
gcc -o 20190821 20190821.c
ls -ltr
file a.out 20190821 #consultar el tipo de los archivos
vim 20190821.c #editar
gcc -o 20190821 20190821.c #compilar
./20190821 #ejecutar
vim 20190826.c
gcc -o 20190826 20190826.c
./20190826
./20190821
```

```
which date #consultar dónde se encuentra el binario de date
ls -l /usr/bin/date #lectura larga
ls -l /bin/date
/bin/date #ejecutar directamente el binario de date
vim 20190826.c
gcc -o 20190826 20190826.c
./20190826

glxgears #ejecutar aplicación gráfica que muestra unos engranajes
lsmod | less #consultar los módulos del kernel cargados actualmente y pasarle la salida al
vim 20190826.c
which glxgears
ls -l /usr/bin/glxgears
vim 20190826.c
echo $HOME #mostrar variable de ambiente que contiene la ruta del directorio de trabajo por
echo $DISPLAY #mostrar variable de ambiente que contiene el identificador del ambiente gráf
glxgears
vim 20190826.c
gcc -o 20190826 20190826.c
./20190826

man 2 kill #consultar manual de la llamada al sistema kill
glxgears
kill -l #mostrar la lista de señales existentes
man 1 kill #consultar manual del comando kill
which kill
ls -l /bin/kill
glxgears
glxgears & #ejecutar glxgears en segundo plano (para que no deje atrapada la consola)
pgrep glxgears #buscar un proceso por su nombre
kill -s INT 20754 #enviar señal INT (de interrupción) al proceso con PID 20754
glxgears &> /dev/null & #unir la salida estándar con la salida estándar de error y ambas re
pgrep glxgears
kill -s INT 20947
glxgears &> /dev/null &
echo $! #consultar variable que contiene el PID del último proceso enviado a segundo plano
kill -s TERM $! #enviar señal TERM (terminación)
glxgears &> /dev/null &
kill -s KILL $!
xeyes & #abrir aplicación en segundo plano
pgrep xeyes
echo $!
kill -s STOP $! #enviar señal STOP (detener, congelar el proceso)
kill -s CONT $! #enviar señal CONT (para que un proceso detenido/congelado continúe)

xeyes
```

```
pgrep xeyes
kill -s CONT 21608
kill -9 21608 #enviar una señal por su valor numérico (9 = KILL)
kill -l
xeyes &
kill -15 21881 #enviar señal TERM usando su valor numérico
xeyes &
killall -s SIGKILL xeyes #enviar señal KILL a todos los procesos con nombre xeyes
xeyes &
pgrep xeyes
pkill xeyes #enviar señak a proceso por su nombre
xeyes &
pgrep xeyes
pkill -n xeyes #enviar señal al proceso xeyes más nuevo
man pkill
pkill -o xeyes #enviar señal al proceso xeyes más viejo
pkill xeyes

xkill #abrir aplicación gráfica para matar un proceso del ambiente gráfico (al ejecutarla,
xkill
pgrep xkill
pkill xkill

ping 127.0.0.1 #enviar paquetes de red de prueba
#en otra consola
  pgrep ping #buscar el proceso
  kill -s QUIT 22687 #enviarle señal QUIT para que imprima estadísticas (ping cacha la señal
  pgrep ping
  kill -s QUIT 22958

man man
man 7 signal #ver manual de las señales, entre otras cosas, acá también está la lista de se
xeyes
pkill xeyes
pkill -s CONT xeyes #congelar proceso
kill -s CONT xeyes #continuar proceso
pgrep xeyes
kill -s CONT 23260
man pkill

bc #ejecutar calculadora
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.68.tar.xz #descargar un archiv
```

Miércoles 28 de Agosto

```c
#include "stdio.h"

int main (int argc, char *argv[]){

    while(1){
        //sensar/consultar algun evento
        //if(...) //ya son las ocho y media?
            return 0;
    }

    while(1){
        //if(...) //ya son las ocho y media?
            return 0;
        sleep(60);
    }

    while(1){
        //if(...) //ya son las ocho y media?
            return 0;
        //esperar un suceso externo: cierto tiempo, una señal, que se lea cierta informació
        //pause(...);
        //waitpid(...);
    }
}
```

```
yes #ejecutar comando que imprime repetidamente la letra "y" en pantalla
yes Hoy es miércoles y había tráfico miércoles #imprimir un texto particular repetidamente
yes Hoy es miércoles y había tráfico miércoles | sed -n '1~1000=' #pasar la salida de yes a
                                                                  #  -n: no muestra nada en
                                                                  #  1~1000: a partir de la
                                                                  #  =: mostrar el número d
yes Hoy es miércoles y había tráfico miércoles | sed -n '1~1000000=' #ídem pero para cada m
yes Hoy es miércoles y había tráfico miércoles > /dev/null & #abrir en segundo plano
top #ver procesos existentes, yes aparecerá en estado R (running) y usando el CPU
echo $! #PID del último proceso abierto en segundo plano
cat /proc/27342/stat #información del estado del proceso
cat /proc/27342/status #información del estado del proceso en un formato más amigable
grep ctxt /proc/27342/status #consultar la cantidad de cambios de contexto (cuando toma el
pkill yes #

bc #calculadora
#en otra consola
  pgrep bc
  grep ctxt /proc/27859/status
```

```
  less /proc/27859/status
  grep ctxt /proc/27859/status
  grep status /proc/27859/status
  less /proc/27859/status
  grep State /proc/27859/status

cd /usr/src/linux #directorio del código fuente del kernel
vim include/linux/sched.h #ver definición de estados en el fuente del kernel

#gcc -o x -lpthreads arch.c #ejemplo de cómo compilar con threads (hilos)
man pthread_create #manuales de funciones para manejo de hilos
man pthread_exit
man pthread_detach
man pthread_join
man pthread_cancel

vim 20190828.c
sleep 5 #esperar 5 segundos
man sleep
top
date
watch date #ejecutar el comando date repetidamente (por defecto, cada 2 seg.)
watch -n 10 date #ampliar el intervalo a cada 10 seg.
watch -n 0.1 date #intervalo de 1 décima de segundo
watch -n 0.01 date
watch -n 0.1 grep State /proc/27859/status #ejecutar repetidamente el comando grep
grep ctxt /proc/27859/status

ps -eLf | less #consultar los procesos existentes detallando informaciín de sus hilos
ps -eLf | sed -n '1p;/bc/p' #filtrar la salida de ps con sed:
                            #  -n: no imprimir
                            #  1p: imprimir la lónea 1 (encabezados)
                            #  /bc/p: buscar la palabra bc e imprimirla
echo light weight process
ls -l /proc/32767/task/ #consultar directorio que contiene los hilos de un proceso
ls -l /proc/32767/task/32767/ #consutar información de un hilo de un proceso
ls -l /proc/32767/task/
pgrep soffice
ls -l /proc/26939/task/
```

Miércoles 4 de Septiembre

```
#include "stdio.h"
#include "unistd.h"
#include "pthread.h"
```

```c
#define N 5

int sigId=32;
int espera=10;

pthread_t hilos[N];
pthread_mutex_t m;

void* siguienteId (void* p);
void* esperar (void* p);

int main (int argc, char *argv[]){

    /*
    siguienteId(NULL);
    siguienteId(NULL);
    siguienteId(NULL);
    siguienteId(NULL);
    */

    pthread_mutex_init(&m,NULL);

    for(int i=0;i<N;i++){
        //pthread_create(&hilos[i],NULL,&siguienteId,NULL);
        pthread_create(&hilos[i],NULL,&esperar,NULL);
    }
    //sleep(1);
    void **retval;
    for(int i=0;i<N;i++){
        pthread_join(hilos[i],retval);
    }

    pthread_mutex_destroy(&m);

    return 0;
}

void* siguienteId (void* p){

    pthread_mutex_lock(&m);

    printf("%d\n",sigId);
    sigId++;

    pthread_mutex_unlock(&m);
```

```
    return 0;
}

void* esperar (void* p){

    if(pthread_mutex_trylock(&m)==0){
        while(espera>0){
            printf("%d\n",espera);
            sleep(1);
            espera--;
        }

        pthread_mutex_unlock(&m);
    }
    else{
        printf("El mutex ya estaba bloqueado :(\n");
    }

    return 0;
}
```

```
vim 20190904_mutex.c #editar
gcc 20190904_mutex.c #compilar
./a.out #ejecutar

vim 20190904_mutex.c
gcc -lpthread 20190904_mutex.c #compilar incluyendo librería de hilos
./a.out

shuf rgmc/trabajo/sistemasOperativos/20192/alumnos | head -n5 #ordenar aleatoriamente el lis
```