

PONTIFICIA UNIVERSIDAD CATÓLICA MADRE Y MAESTRA

FACULTAD DE CIENCIAS E INGENIERÍA



Sistemas Operativos II

ISC ISC365-101

L3 – Usando mandatos Linux.

Presentado por:

Junior Hernandez 2018-0999 10135069

Entregado a:

Juan Ramón Felipe Núñez Pérez

Fecha de realización: 30 de octubre del 2022

Fecha de entrega: 19 de noviembre del 2022

Santiago de los caballeros; República Dominicana.

Introducción

Algunos podrían argumentar que C no es el mejor lenguaje para principiantes, en lo que trabajar con Linux se trata. C y Unix, y más tarde Linux, están estrechamente vinculados, por lo que parecía natural comenzar nuestra serie de desarrollo con C. Desde el kernel, una parte sustancial del cual está escrito en C, hasta muchas aplicaciones de usuarios cotidianos, C se usa masivamente en su sistema Linux. Por ejemplo, GTK se basa en C, por lo que si usa aplicaciones Gnome o XFCE, está usando aplicaciones basadas en C. C es un lenguaje de programación antiguo y bien establecido, una herramienta vital en muchas partes del mundo de TI, desde sistemas integrados hasta mainframes. Por lo tanto, es justo suponer que las habilidades C no solo enriquecerán su CV, sino que también lo ayudarán a resolver muchos problemas en su sistema Linux, eso solo si se toma esto en serio y practica mucho leyendo y escribiendo código C.

Marco Teórico

Linux es una familia completa de sistemas operativos Unix de código abierto, que se basan en el kernel de Linux. Esto incluye todos los sistemas basados en Linux más populares como Ubuntu, Fedora, Mint, Debian y otros. Más exactamente, se llaman distribuciones o versiones.

Desde que Linux se lanzó por primera vez en 1991, ha seguido ganando popularidad debido a su naturaleza de código abierto. Las personas pueden modificarlo y redistribuirlo libremente bajo su propio nombre.

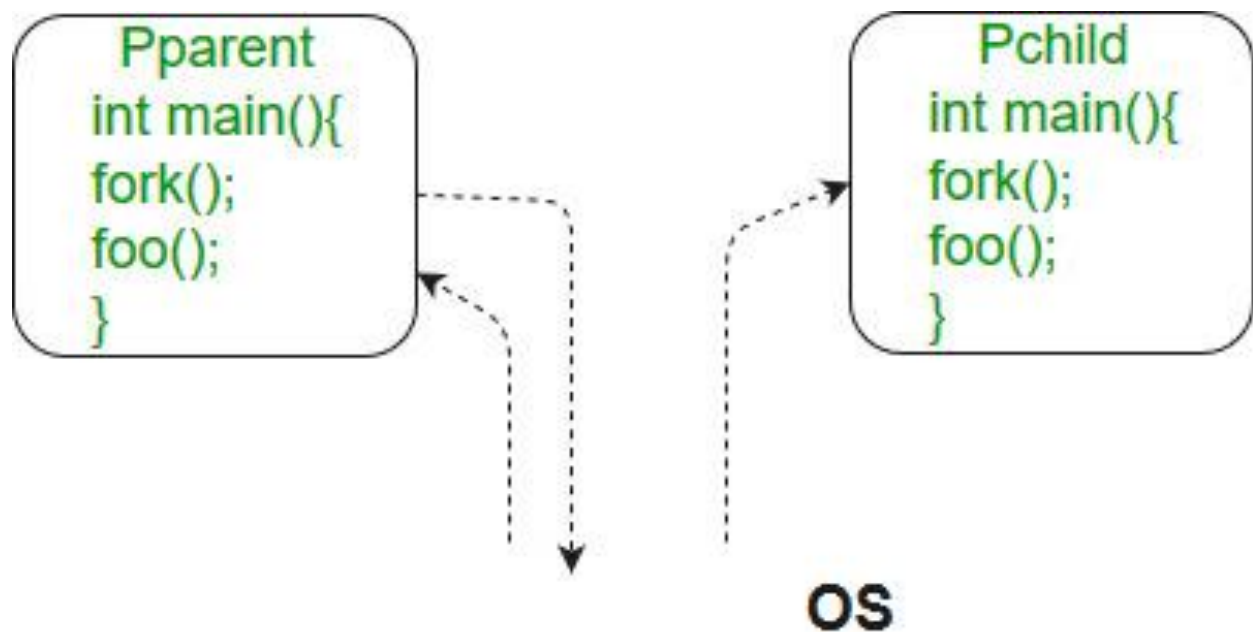
Al operar un sistema operativo Linux, debes usar un shell, una interfaz que te da acceso a los servicios del sistema operativo. La mayoría de las distribuciones de Linux utilizan una interfaz gráfica de usuario (GUI) como shell, principalmente para proporcionar facilidad de uso a sus usuarios. Es una práctica común utilizarlos cuando se gestiona un VPS.

Comandos básicos de Linux

1. comando pwd	11. comando locate	21. comando chmod
2. comando cd	12. comando find	22. comando chown
3. comando ls	13. comando grep	23. comando jobs
4. comando cat	14. comando sudo	24. comando kill
5. comando cp	15. comando df	25. comando ping
6. comando mv	16. comando du	26. comando wget
7. comando mkdir	17. comando head	27. comando uname
8. comando rmdir	18. comando tail	28. comando top
9. comando rm	19. comando diff	29. comando history
10. comando touch	20. comando tar	30. comando man

(A., 2022)

La llamada al sistema de bifurcación se usa para crear un nuevo proceso, que se llama proceso secundario, que se ejecuta simultáneamente con el proceso que realiza la llamada a la bifurcación () (proceso principal). Después de que se crea un nuevo proceso secundario, ambos procesos ejecutarán la siguiente instrucción después de la llamada al sistema fork(). Un proceso hijo usa la misma computadora (contador de programa), los mismos registros de CPU, los mismos archivos abiertos que usa el proceso padre.



(Geeksforgeeks, n.d.)

El `const char *arg` y los puntos suspensivos subsiguientes en las funciones `execl()`, `execlp()` y `execle()` se pueden considerar como `arg0`, `arg1`, ..., `argn`. Juntos describen una lista de uno o más punteros a cadenas terminadas en nulo que representan la lista de argumentos disponibles para el programa ejecutado. El primer argumento, por convención, debe apuntar al nombre de archivo asociado con el archivo que se está ejecutando. La lista de argumentos debe terminar con un puntero NULL y, dado que se trata de funciones variables, este puntero debe convertirse `(char *) NULL`.

Las funciones `execv()`, `execvp()` y `execvpe()` proporcionan una serie de punteros a cadenas terminadas en nulo que representan la lista de argumentos disponibles para el nuevo programa. El primer argumento, por convención, debe apuntar al nombre de archivo asociado con el archivo que se está ejecutando. La matriz de punteros debe terminar con un puntero NULL. (Linux.die, n.d.)

Desarrollo

Codigo 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
/*
 * Se crea un proceso que ejecutara - con execvp - "ls -l /"
 */

int spawn (char* program, char** arg_list)
{
    pid_t child_pid;

    child_pid = fork ();
    if (child_pid != 0)
        return child_pid;
    else {
        execvp (program, arg_list);
        fprintf (stderr, "un error ocurrio con execvp\n");
        abort ();
    }
}

int main ()
{
    char* arg_list[] = { "ls", "-l", "/", NULL };

    spawn ("ls", arg_list);
    printf ("Finalizando con main \n");

    return 0;
}
```

Salida:

```
junior@junior-VirtualBox:~/Descargas$ ./a.out
Finalizando con main
junior@junior-VirtualBox:~/Descargas$ total 1505360
lrwxrwxrwx    1 root root          7 oct 24 00:56 bin -> usr/bin
drwxr-xr-x    4 root root       4096 oct 24 01:32 boot
drwxrwxr-x    2 root root       4096 oct 24 01:02 cdrom
drwxr-xr-x   19 root root       4160 oct 24 11:00 dev
drwxr-xr-x  129 root root     12288 oct 24 10:45 etc
drwxr-xr-x    3 root root       4096 oct 24 01:17 home
lrwxrwxrwx    1 root root          7 oct 24 00:56 lib -> usr/lib
lrwxrwxrwx    1 root root          9 oct 24 00:56 lib32 -> usr/lib32
lrwxrwxrwx    1 root root          9 oct 24 00:56 lib64 -> usr/lib64
lrwxrwxrwx    1 root root         10 oct 24 00:56 libx32 -> usr/libx32
drwx-----   2 root root     16384 oct 24 00:56 lost+found
drwxr-xr-x    2 root root       4096 ago  9 07:48 media
drwxr-xr-x    2 root root       4096 ago  9 07:48 mnt
drwxr-xr-x    2 root root       4096 ago  9 07:48 opt
dr-xr-xr-x  318 root root          0 oct 24 10:25 proc
drwx-----   4 root root       4096 oct 24 01:44 root
drwxr-xr-x   33 root root        900 oct 30 11:10 run
lrwxrwxrwx    1 root root          8 oct 24 00:56 sbin -> usr/sbin
drwxr-xr-x   11 root root       4096 ago  9 07:55 snap
drwxr-xr-x    2 root root       4096 ago  9 07:48 srv
-rw-----   1 root root 1541406720 oct 24 00:56 swapfile
dr-xr-xr-x   13 root root          0 oct 24 10:25 sys
drwxrwxrwt   20 root root       4096 oct 30 11:20 tmp
drwxr-xr-x   14 root root       4096 ago  9 07:48 usr
drwxr-xr-x   14 root root       4096 ago  9 07:54 var
```

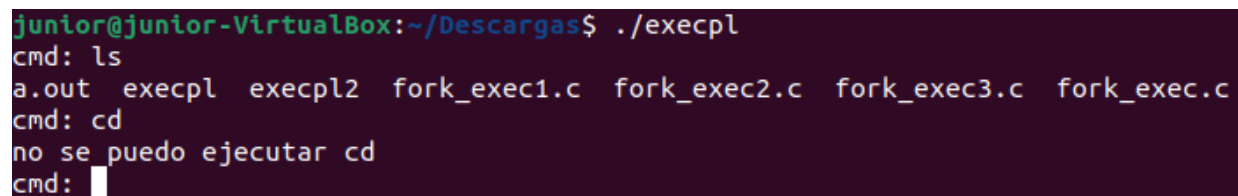
Como era lo que se esperaba el programa solo ejecuta el ls-l, ene l directorio raíz además utiliza execvp para agregar una lista de argumento, gracias a esto posibilita el argumento -l al comando ls como se ve.

Codigo 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <wait.h>
#include <sys/types.h>
#define EXIT_FAILURE 1
#define MAXI 512

int main (void) {
    char line [BUFSIZ];
    int process;
    for ( ; ; ) {
        (void) fprintf (stderr, "cmd: ");
        // if (fgets(line,MAXI,stdin) == (char *) NULL)
        if (fgets(line, MAXI,stdin) == (char *) NULL)
            exit (EXIT_FAILURE);
        if ((process = fork ()) > 0)
            (void) wait ((int *) NULL);
        else if (process == 0) {
            (void) execlp (line, line, NULL);
            (void) fprintf (stderr, "no se puedo ejecutar %s\n", line);
            exit (errno);
        } else if (process == -1) {
            (void) fprintf (stderr, "Problemas con fork\n");
            exit (errno);
        }
    }
}
```

Salida



```
junior@junior-VirtualBox:~/Descargas$ ./execpl
cmd: ls
a.out  execpl  execpl2  fork_exec1.c  fork_exec2.c  fork_exec3.c  fork_exec.c
cmd: cd
no se puedo ejecutar cd
cmd: 
```

Aquí se observa claramente un pequeño programa que funciona para ejecutar comandos básicos sin argumentos del cmd de Linux con la ayuda de fork, trabajando desde el directoria actual, con excepciones como cd ya que se necesita de chdir() para ello, las ejecuciones se realizan básicamente ejecutando execlp(), entonces no se pueden agregar argumentos a los comandos escritos.

Codigo 3:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <wait.h>
#include <sys/types.h>
#include <string.h>
#include <malloc.h>

#define EXIT_FAILURE 1
#define BUFSIZE 8192
#define MAXI 512
/*
 * Se crea un proceso se recibe un string
 * que es dividido en tokens para construir argumentos
 * de proceso a ejecutar
 * Y se ejecuta como un proceso hijo : solo se trabaja con
 * mandatos simples
 */
char * first_argu(char * str )
{
    char * stok;
    char buff[BUFSIZE];

    strcpy(buff, str);
    stok = strtok(buff, " ");
    return stok;
}
```

```

int argum(char ** argv, char * str)
{
    char * stok;
    char buff[BUFSIZE];
    strcpy(buff, str);
    int pos = 0;
    stok = strtok(buff, " ");

    while(stok != NULL)
    {
        strcpy(argv[pos], stok);
        stok = strtok(NULL, " ");
        pos++;
    }
    argv[pos] = (char*)0;
    return pos;
}

```

```

char ** creat_arr(int n, int m)
{

```

```

    char ** args;
    int i ;
    args = malloc(n* sizeof(char*));
    for(i = 0; i < n; i++) args[i] = malloc(m*sizeof(char));
return args;
}

void clean_arr(char ** args, int n)
{
    int i, j;
    for (i = 0 ; i < n; i++) free(args[i]);
    free(args);
}

int main (void)
{
    char line [BUFSIZE];
    char * spath;
    char ** args;
    int process,i, nr;

    for ( ; ; ) {
        (void) fprintf (stderr, "cmd: ");
        if (fgets(line,MAXI,stdin) == (char *) NULL)
            exit (EXIT_FAILURE);
        if ((process = fork ()) > 0)
            (void) wait ((int *) NULL);
        else if (process == 0) {
            args = creat_arr(32,64);

            spath = first_argu(line);
            nr = argum(args,line);

            for(i = 0; i < nr -1; i++)printf("%s\n",args[i]);

            (void) execvp (spath, args);
            clean_arr(args,32); /* No creo que se ejecute */
            (void) fprintf (stderr, "no se puedo ejecutar %s\n", line);
            exit (errno);
        } else if (process == -1) {
            (void) fprintf (stderr, "Problemas con fork\n");
            exit (errno);
        }
    }
}

```

Salida:

```
junior@junior-VirtualBox:~/Descargas$ ./a.out
cmd: ls -l
total 72
-rwxrwxr-x 1 junior junior 16664 oct 30 13:37 a.out
-rwxrwxr-x 1 junior junior 16304 oct 30 13:35 execpl
-rwxrwxr-x 1 junior junior 16744 oct 30 13:30 execpl2
-rw-rw-r-- 1 junior junior 874 oct 30 13:35 fork_exec1.c
-rw-rw-r-- 1 junior junior 2047 oct 30 13:37 fork_exec2.c
-rw-rw-r-- 1 junior junior 306 oct 24 10:58 fork_exec3.c
-rw-rw-r-- 1 junior junior 579 oct 24 10:55 fork_exec.c
cmd: ls -la
total 80
drwxr-xr-x 2 junior junior 4096 oct 30 13:37 .
drwxr-x--- 16 junior junior 4096 oct 30 11:11 ..
-rwxrwxr-x 1 junior junior 16664 oct 30 13:37 a.out
-rwxrwxr-x 1 junior junior 16304 oct 30 13:35 execpl
-rwxrwxr-x 1 junior junior 16744 oct 30 13:30 execpl2
-rw-rw-r-- 1 junior junior 874 oct 30 13:35 fork_exec1.c
-rw-rw-r-- 1 junior junior 2047 oct 30 13:37 fork_exec2.c
-rw-rw-r-- 1 junior junior 306 oct 24 10:58 fork_exec3.c
-rw-rw-r-- 1 junior junior 579 oct 24 10:55 fork_exec.c
cmd: pwd date
pwd: los argumentos que no son opciones no serán tenidos en cuenta
/home/junior/Descargas
cmd: 
```

Bueno esto seria una mejora para el anterior, ya que con el anterior programa solo se podía ejecutar el comando sin ningún argumento en este lo hacemos con ayuda de `execvp()` para capturarlos y `strtok()` para separarlos, usando punteros y cadenas para poder ejecutar estos comandos con algún argumento.

Codigo 4:

```
#include <stdio.h>

#include <string.h>

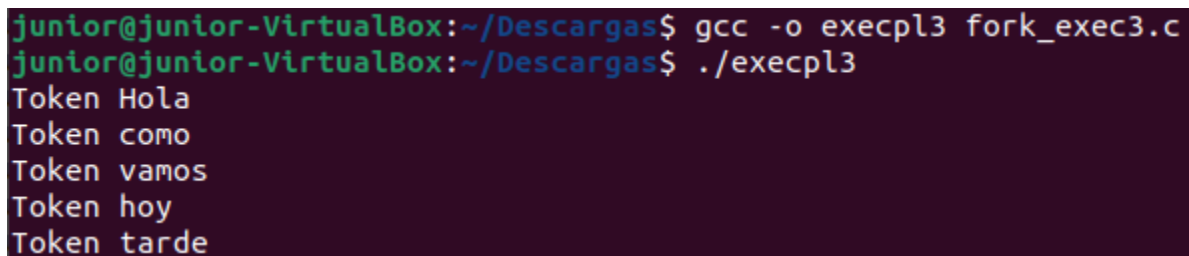
int main()

{

    char str[4096];
```

```
char * pst;  
  
strcpy(str, "Hola como vamos hoy tarde");  
  
pst = strtok (str, " ");  
  
while (pst != NULL)  
{  
  
    printf ("Token %s\n",pst);  
  
    pst = strtok (NULL, " ");  
  
}  
  
return 0;  
  
}
```

Salida:



```
junior@junior-VirtualBox:~/Descargas$ gcc -o execpl3 fork_exec3.c  
junior@junior-VirtualBox:~/Descargas$ ./execpl3  
Token Hola  
Token como  
Token vamos  
Token hoy  
Token tarde
```

Este programa es simple, corto y directo al punto, toma la cadena y la divide en 5 tokens según el separador indicado “ ” todo realizado con Strtok.

Captu.sh:

```
#!/bin/bash
# Capturando una señal
nrcic=10
trap 'echo "capture señal 2 en ciclo `expr $nrcic - $i + 1`; continue' 2
for i in `seq $nrcic -1 1`
do
    echo " Faltan $i minutos para finalizar"
    sleep 60
done
echo " Hemos finalizado"
```

Salida:

```
junior@junior-VirtualBox:~/Descargas$ ./captu.sh
Faltan 10 minutos para finalizar
Faltan 9 minutos para finalizar
Faltan 8 minutos para finalizar
Faltan 7 minutos para finalizar
Faltan 6 minutos para finalizar
Faltan 5 minutos para finalizar
Faltan 4 minutos para finalizar
Faltan 3 minutos para finalizar
Faltan 2 minutos para finalizar
Faltan 1 minutos para finalizar
Hemos finalizado
```

El script bash, captura las señales que les son enviadas y espera entonces 1 minuto durante 10 ciclos para poder recibir e imprimir cual es la señal que esta recibiendo.

Freq.sh

```
#!/bin/bash
# calcular frecuencia de palabras
nr=10
if [ $# -ge "2" ]
then
    nr=$1
fi
tr      '[A-Z]' '[a-z]' |
tr -cs  '[a-z]'  '[\012*]' |
sort    |
uniq -c  |
sort -nr |
head - $nr
```

Salida:

```
junior@junior-VirtualBox:~/Descargas$ sudo find /usr/ -name '*txt' -print | xargs cat -v | ./freq.sh
cat: Module: No existe el archivo o el directorio
cat: 4.txt: No existe el archivo o el directorio
 38862 m
 34471 [
 33173 ]
 18270 a
 13558 f
 13109 the
 11958 d
 11679 c
 11502 letter
 10643 b
```

Como se muestra en la salida, esas palabras son las mas usadas en los archivos marcados en el comando. Teniendo a su izquierda la cantidad de veces que se repiten.

Conclusión

Como se observa durante el desarrollo de la practica fue posible correr y probar los códigos propuestos para esta asignación, se observa como fue posible comprender los resultados de cada uno de los códigos en C, en los cuales detallan el uso del fork y de otras funciones para simular una terminal de comandos cmd, en el directorio actual en que se encontraban susodichos programas, viendo como fork() y los demás funcionaban como llamadas al sistema para ejecutar una palabra ingresada al programa como un comando, trabajando desde un punto más específico con el primer código usando ls -l, hasta uno en el tercer código en el cual trabajamos el input de un comando mas un argumento y como dividir cadenas en tokens, además se vio la interpretación y ejecución archivos de comandos para la terminal de linux.

References

A., D. (2022, Septiembre 14). *Hostinger.es*. Retrieved from Hostinger.es:
<https://www.hostinger.es/tutoriales/linux-comandos>

Geeksforgeeks. (n.d.). *Geeksforgeeks.org*. Retrieved from Geeksforgeeks.org:
<https://www.geeksforgeeks.org/fork-system-call/>

Linux.die. (n.d.). *Linux.die.net*. Retrieved from Linux.die.net:
<https://linux.die.net/man/3/execlp>