Universidad Nacional Autónoma de México
Facultad de Ciencias

Sistemas Operativos - Practica 3

Septiembre de 2018

Descripción de programas

1. mmap()
   - Leer un archivo a través de la llamada al sistema read(), imprimir su contenido y el valor de su descriptor de archivo.
   - Describir como se mapea este archivo en memoria
   - Abrir un archivo mediante la llamada al sistema mmap() e imprimir su contenido.
   - Describir como se mapea este archivo en memoria.
   - Realizar una comparación entre ambos métodos y las ventajas de uno sobre el otro
   - Para ambos casos realizar un diagrama de la estructura del proceso en memoria
2. Child's Play
   - Crear un programa que cree un proceso hijo a través de la llamada al sistema fork(), el proceso original deberá imprimir su Process ID, y su Parent Process ID.
   - Incluir una condición para la creación del proceso hijo, es decir, ejecutar el fork() unicamente cuando ocurra un evento en el sistema operativo que lo desencadene.
   - El proceso hijo debe imprimir de igual manera su PID y su PPID, y deberá remarcarse la asociación entre ambos procesos.
   - Generar código diferente para cada proceso teniendo en cuenta lo siguiente

```
p = fork();
if(p>0)
    Codigo para el XXXXX
else
    Codigo para el XXXX
```

3. Tuberias

   - Escribir un programa el cual genere dos procesos, P1 y P2. P1 recibirá una cadena de caracteres y se la enviará a P2. P2 concatenará la cadena recibida con otra cadena definida en P2 sin emplear funciones de manejo de cadenas contenidas en string.h. La cadena generada será devuelta a P1 para ser impresa en la salida estandar.
   - En este problema deberan utilizarse las llamadas al sistema read(), write(), close(), fork() y pipe() teniendo en cuenta lo siguiente

```
int fd[2]; //Arreglo de descriptores de archivos
pipe(fd); //Creacion del pipe, pasando como argumento el
arreglo de descriptores
fd[0]; // Descriptor de archivo del pipe para lectura
fd[1]; // Descriptor de archivo del pipe para escritura
```

   - Considerar que se deben crear dos pipes, y por lo tanto dos arreglos de descriptores de archivos, y que el código debe ser diferente para el proceso padre y el hijo.

4. Signal

   - A través de la implementación de "signal.h" comunicar dos procesos (un proceso padre y un hijo), permitiendo la interacción entre ambos, y finalizar el programa "matando" al proceso hijo.
   - Cada senal a implementar deberá desencadenar una acción, por ejemplo la impresión de un mensaje.
   - Tener en cuenta que debe implementarse una función que será llamada con la ejecución de cada senal.
   - La llamada al sistema kill() recibe como parametro el PID y la senal a enviar

```
if (pid == 0) { //Hijo
signal(SIGHUP, sighup);
signal(SIGINT, sigint);
signal(SIGQUIT, sigquit);
for (;;);
}

else // Padre
{
printf("\nPadre: enviando SIGHUP\n\n");
kill(pid, SIGHUP);
...

void sighup()
```

```
{
    signal(SIGHUP, sighup);
    printf("HIJO: Recibi un SIGHUP, ahora debo...\n");
}
```

5. Overflow (EXTRA)

   - Sobreescribe la variable val, desbordando el tamano de almacenamiento permitido para la variable buff empleando el siguiente código.

```
    #include <stdio.h>
#include <stdlib.h>

int main(){
long val=0x41414141;
char buf[20];

printf("Cambia el valor de val de 0x41414141 a 0x42414241!\n");
printf("Introduzca los datos: ");
scanf("%24s",&buf);

printf("buf: %s\n",buf);
printf("val: 0x%08x\n",val);

if(val==0x42414241)
printf("Bien!!!\n");
else {
printf("NO!!!!!!!!!\n");
exit(1);
}

return 0;
}
```

   - Recordar little endian y big endian
   - Describir la vulnerabilidad y ejemplificar con el mapa del proceso en memoria

NOTAS

- Cada programa deberá estar debidamente documentado con imagenes y pruebas de ejecución en un archivo README.md
- En el directorio raíz deberán estar especificados los nombres de los integrantes del equipo que participan en la practica.