Universidad San Francisco de Quito

Alejandra Ospina (00212243)

Sistemas Operativos – NRC: 2362

2023-02-21

Tarea 1

1. En esta primera parte, defina los siguientes términos (incluya la bibliografía utilizada): Sistema operativo, CPU, memoria RAM, dispositivo de almacenamiento, dispositivo de entrada y salida, computar, programas de aplicación, compilador, Ley de Moore, kernel, micro-kernel, kernel monolitico, middleware, interrupción, proceso, hilo, acceso directo a memoria, interrupt chaining, arquitectura de von-Neumann, registro, instrucción, memoria cache, discos ópticos, cintas magnéticas, L1 Cache, L2 Cache, NUMA, multiprogramación, multitarea.

- Sistema operativo: Es un software que tiene dos funciones principales. La primera es proporcionar una base y ambiente para la ejecución de programas de aplicación. La segunda es administrar los recursos de hardware de forma adecuada.
- CPU: Conocida como unidad central de procesamiento, es el cerebro de la computadora, ya que en este hardware se realiza todo el procesamiento de operaciones y control, ejecutando operaciones aritméticas y lógicas.
- Memoria RAM: Memoria de acceso aleatorio, siendo un tipo de memoria volátil que permite almacenar los datos y programas temporalmente que se utilizan por el CPU, dando acceso de forma rápida.
- Dispositivo de almacenamiento: Componentes de hardware cuya función principal es almacenar datos de forma permanente.
- Dispositivo de entrada y salida: Componentes de hardware, usualmente periféricos, que permiten al usuario comunicarse o interactuar con la computadora de manera fácil.
- Computar: Proceso realizado por una computadora que consiste en realizar cálculos y procesamiento de datos.
- Programas de aplicación: Son programas que están en contacto directo recibiendo instrucciones del usuario.
- Compilador: Un compilador es un traductor de código de lenguaje de alto nivel a lenguaje de máquina para que la máquina entienda lo que codificamos.
- Ley de Moore: Es una observación y predicción que aproximadamente cada dos años se duplica el número de transistores de los procesadores de las computadoras.
- Kernel: Es el núcleo del sistema operativo, parte esencial que tiene funciones de manejo de recursos, archivos y seguridad.
- Microkernel: Tipo de kernel que solo se ocupa de servicios importantes y el resto se ejecutan como procesos en el espacio de usuario. Es decir, solo posee llamadas al sistemas básicas.
- Kernel monolítico: Tipo de sistema operativo en el que todos los servicios y controladores de dispositivos se ejecutan en el espacio del kernel.
- Middleware: Capa de software cuya función es la comunicación entre los programas de aplicación y los servicios del sistema operativo. Da una interfaz al
- Interrupción: Una señal que usualmente proviene de eventos externos, como dispositivos de entrada y salida, para que la ejecución de sus instrucciones sea prioritaria.

- Proceso: Es una instancia de un programa en ejecución que incluye el código, los datos y el estado de ejecución.
- Hilo: Una unidad de procesamiento dentro de un proceso y puede compartir espacio de memoria con otros.
- Acceso directo a memoria: Es una técnica que permite al hardware externo acceder a la memoria del sistema sin la necesidad de que interfiera el CPU.
- Interrupt chaining: Es un método para manejar interrupciones de manera secuencial. Si se produce una interrupción, el sistema operativo puede mandar otra para manejarla.
- Arquitectura de von Neumann: Arquitectura de computadora que se caracteriza por tener un CPU, una sola memoria, una unidad aritmética lógica y sus dispositivos de entrada y salida. La memoria almacena los datos y los programas que necesita el CPU ejecutar.
- Registro: Es un almacenamiento que se encuentra en el CPU de alta rapidez de acceso, más que la memoria principal.
- Instrucción: Una instrucción es un comando de máquina que el CPU puede procesar. Las instrucciones incluyen operaciones aritméticas y lógicas, transferencias de datos y llamadas a funciones de
- Memoria caché: La memoria caché almacena temporalmente los datos que se usan con frecuencia para acelerar el acceso a los mismos, reduciendo así la carga en la memoria principal.
- Discos ópticos: Discos que para ser leídos por la maquina en la que se encuentran se utilizan láseres ya que estos tienen una cobertura o capa fotosensible.
- Cintas magnéticas: Dispositivo de almacenamiento que tiene un recubrimiento magnético que se utilizan para guardar grandes cantidades de datos. Utilizadas hace mucho tiempo en audio y video, en la actualidad ya casi no se las utiliza pos su durabilidad.
- L1 Cache: Tipo de memoria de alta velocidad que está integrada en el CPU para alojar datos e instrucciones de uso frecuente de este.
- L2 Cache: Tipo de memoria de alta velocidad que se encuentra afuera pero cerca del CPU más grande que el L1 y asimismo guarda datos e instrucción de uso frecuente.
- NUMA:Arquitectura de computadora en que se tienen varios procesadores y estos tienen acceso a solo una parte de la memoria pero no a toda, incrementando el rendimiento. Es decir limita el acceso a la memoria.
- Multiprogramación: Técnica que permite a un sistema operativo ejecutar varios programas al mismo tiempo. Funciona dividiendo el tiempo de procesamiento de la CPU entre varios programas y alternando entre ellos en intervalos de tiempo muy cortos, lo que da la impresión de que se están ejecutando simultáneamente.
- Multitarea: La multitarea es una técnica que permite a un sistema operativo ejecutar varias tareas simultáneamente. La multitarea es similar a la multiprogramación, pero en lugar de alternar entre programas a intervalos muy cortos, la CPU asigna diferentes tareas en momentos diferentes según su prioridad y necesidad.

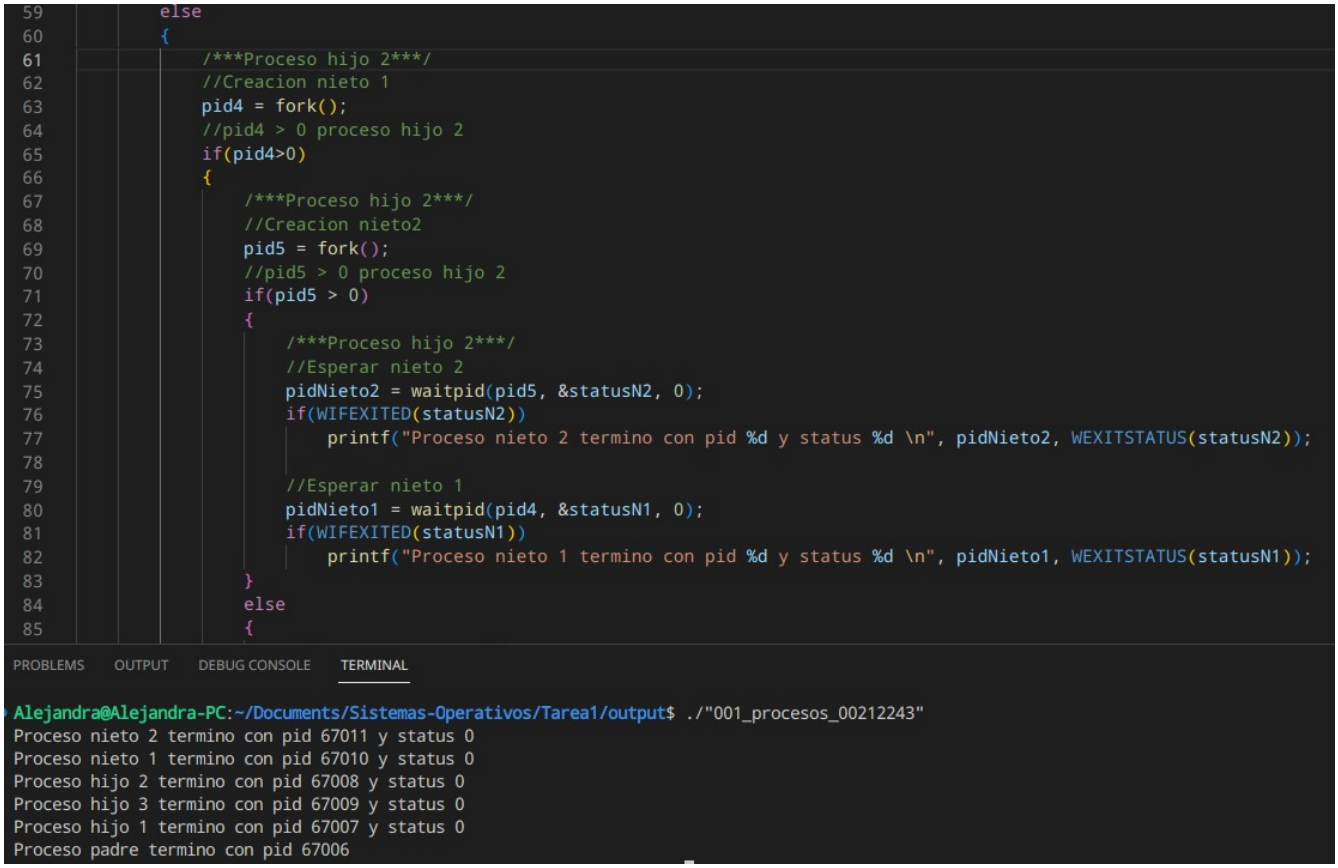
Referencias

Tanenbaum, A. & Woodhull, A. (2006). Operating systems : design and implementation. Upper Saddle River, N.J: Pearson/Prentice Hall.

Silberschatz, A., Galvin, P. & Gagne, G. (2012). Operating system concepts. Hoboken, NJ: J. Wiley & Sons. ISBN: 9781118112731

2. Procesos: Implemente un programa en C (usando la mayor cantidad de llamadas al sistema de Linux) que permita crear la siguiente familia de procesos (ver Figura 1). Su trabajo es sincronizar los procesos para que terminen su ejecución en orden, primero niegos, luego hijos y finalmente el padre.

Varios if y else para manejar la creación de los hijos y nietos con llamada fork(). Lo que se hizo es con la función waitpid() primero hacer esperar al padre a que termine el hijo 2 y este a su vez que espere a los nietos, luego de que el padre espera por el hijo 2, espera por el resto de hijos. WIFEXITED() Permite saber si los procesos hijos terminaron con exit y WEXITSTATUS permite ver el status con el que terminaron. Guardo el pid de cada proceso antes de que terminen.



```
59          else
60          {
61              /***Proceso hijo 2***/
62              //Creacion nieto 1
63              pid4 = fork();
64              //pid4 > 0 proceso hijo 2
65              if(pid4>0)
66              {
67                  /***Proceso hijo 2***/
68                  //Creacion nieto2
69                  pid5 = fork();
70                  //pid5 > 0 proceso hijo 2
71                  if(pid5 > 0)
72                  {
73                      /***Proceso hijo 2***/
74                      //Esperar nieto 2
75                      pidNieto2 = waitpid(pid5, &statusN2, 0);
76                      if(WIFEXITED(statusN2))
77                          printf("Proceso nieto 2 termino con pid %d y status %d \n", pidNieto2, WEXITSTATUS(statusN2));
78
79                      //Esperar nieto 1
80                      pidNieto1 = waitpid(pid4, &statusN1, 0);
81                      if(WIFEXITED(statusN1))
82                          printf("Proceso nieto 1 termino con pid %d y status %d \n", pidNieto1, WEXITSTATUS(statusN1));
83                  }
84                  else
85                  {
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./"001_procesos_00212243"
Proceso nieto 2 termino con pid 67011 y status 0
Proceso nieto 1 termino con pid 67010 y status 0
Proceso hijo 2 termino con pid 67008 y status 0
Proceso hijo 3 termino con pid 67009 y status 0
Proceso hijo 1 termino con pid 67007 y status 0
Proceso padre termino con pid 67006
```

3. Copia de Archivos: Implemente un programa en C (usando la mayor cantidad de llamadas al sistema de Linux, en particular read and write) para copiar un archivo en otro. (Básicamente reproducir el comando cp de Linux).

Se pasa al argv al correr el programa el comando y los nombres de los archivos, chequea si el comando no esta bien escrito, si faltan argumentos, si el archivo para copia no existe, si no existe el archivo en el que se va a copiar se crea. Se utiliza la función strcmp para copiar los datos del arreglo argv en variables. Se utilizan las llamadas open(), write(), read(), close(). En el caso del archivo en donde se copia, si no existe se crea pasando como argumentos en la función open O_CREAT, O_TRUNC y 0700 para crear sino existe y si existe y es un archivo regular y permite escribir va a ser truncado a 0

```
32
33      //Si es el primer parametro no es igual a cp no se hace la copia
34      if(strcmp(commandInput, command)!=0)
35      {
36          printf("Comando incorrecto\n");
37          return 1;
38      }
39
40      //Si no se puede abrir el archivo de lectura
41      if ((readFile = open(nameReadFile, O_RDONLY)) == -1)
42      {
43          printf("No se pudo abrir archivo %s para leer\n", nameReadFile);
44          return 2;
45      }
46
47      //Si no se puede abrir el archivo de escritura
48      if ((writeFile = open(nameWriteFile, O_WRONLY|O_CREAT|O_TRUNC, 0700)) == -1)
49      {
50          printf("No se pudo abrir archivo %s para escritura\n", nameWriteFile);
51          close(writeFile);
52          return 3;
53      }
54
55      //Bucle para que ha medida que lee de un archivo lo escriba en el otro
56      while ((num = read(readFile, &buffer, sizeof(char))) > 0)
57      {
58          write(writeFile, &buffer, num);
59      }
```

```
PROBLEMS 5   OUTPUT   DEBUG CONSOLE   TERMINAL

Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./002_copiadearchivos_00212243 cp a.txt
Número de argumentos no válido
Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./002_copiadearchivos_00212243 cp file1.txt
Número de argumentos no válido
Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./002_copiadearchivos_00212243 cp file1.txt file2.txt
Archivo 'file1.txt' se copio correctamente al archivo 'file2.txt'
Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./002_copiadearchivos_00212243 cp a.txt file2.txt
No se pudo abrir archivo a.txt para leer
Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./002_copiadearchivos_00212243 c a.txt file2.txt
Comando incorrecto
```

file1.txt M × | *file2.txt M*

Tarea1 > output > file1.txt

```
1  Si estas leyendo este mensaje, espero que estes bien y tu dia haya sido excelente
```

file1.txt M | *file2.txt M* ×

Tarea1 > output > file2.txt

```
1  Si estas leyendo este mensaje, espero que estes bien y tu dia haya sido excelente
```

3. Sistema de Archivos: Implemente un programa en C (usando la mayor cantidad de llamadas al sistema de Linux) que permita listar el contenido de un directorio. (En otras palabras, un programa que reproduzca el comportamiento de ls de Linux).

Asimismo se colocan los argumentos se chequea que se utiliza el comando correcto, el número de argumentos. Si solo son dos, se utiliza el path en que se encuentra ahora para esto se utiliza la llamada getcwd(). Si son tres se utiliza lo que ingrese el usuario. Se utilizan las llamadas o funciones opendir(), readdir() y closedir() específicamente para directorios. De cada archivo se obtienen información (el tamaño en bytes, el nombre y el modo que se puede visualizar en hexadecimal) con la función stat(). Para visualizar los permisos, se utiliza una función que si el valor de st_mode el valor de ciertas constantes dan true o 1 es que si tiene permiso de escritura, lectura o ejecución.

```
38
39          if(argc == 2)
40              getcwd(path, sizeof(path));
41          if(argc == 3)
42              strcpy(path, argv[2]);
43
44          //Si es el primer parametro no es igual a ls
45          if(strcmp(commandInput, command)!=0)
46          {
47              printf("Comando incorrecto\n");
48              return 1;
49          }
50
51          //Se abre el directorio
52          directory = opendir(path);
53
54          //Si el directorio actual o el proporcionado no se puede abrir
55          if(directory == NULL)
56          {
57              printf("No se pudo abrir el directorio %s\n", path);
58              return -1;
59          }
60
61          //Leer archivos del directorio
62          printf("\nName\t\t\t\t\tSize(bytes)\tMode\tPermissions\n");
63          while((dent = readdir(directory)) != NULL)
64          {
65              stat(dent->d_name, &statistics);
66              printf("%-32s\t%-12d\t%-6X", dent->d_name, statistics.st_size, statistics.st_mode);
67              printPermission(statistics.st_mode);
68          }
```

```
PROBLEMS 7   OUTPUT   DEBUG CONSOLE   TERMINAL

Alejandra@Alejandra-PC:~/Documents/Sistemas-Operativos/Tarea1/output$ ./003_sistemadearchivos_00212243 ls

Name                                Size(bytes)     Mode    Permissions
001_procesos_00212243               69720           81ED    rwxr-xr-x
002_copiadearchivos_00212243        54128           81ED    rwxr-xr-x
003_sistemadearchivos_00212243      55768           81ED    rwxr-xr-x
file2.txt                           81              81C0    rwx------
                                    4096            41ED    rwxr-xr-x
```

```c
//Funcion para obtener los permisos de los archivos a partir del mode_t
void printPermission(unsigned int m)
{
    putchar('\t');
    putchar( m & S_IRUSR ? 'r' : '-' );
    putchar( m & S_IWUSR ? 'w' : '-' );
    putchar( m & S_IXUSR ? 'x' : '-' );
    putchar( m & S_IRGRP ? 'r' : '-' );
    putchar( m & S_IWGRP ? 'w' : '-' );
    putchar( m & S_IXGRP ? 'x' : '-' );
    putchar( m & S_IROTH ? 'r' : '-' );
    putchar( m & S_IWOTH ? 'w' : '-' );
    putchar( m & S_IXOTH ? 'x' : '-' );
    putchar('\n');

}
```