

INSTITUTO POLITÉCNICO NACIONAL

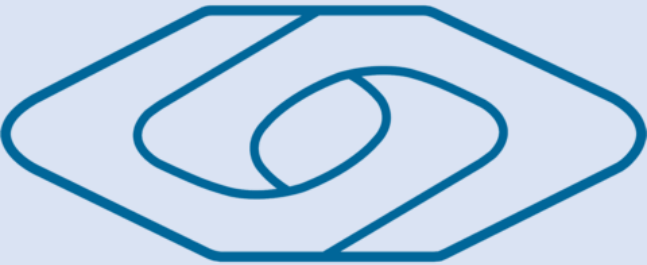
ESCUELA SUPERIOR DE COMPUTO



“Practica 4”

Sistemas Operativos

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

Equipo 7

- Héctor Chávez Rodríguez
 - María Fernanda Delgado Mendoza
 - Jonathan Said Gómez Marbán
 - Luis Antonio Ramírez Fárias
-
- **Grupo:** 4CM1

I. Introducción Teórica

¿Qué es un administrador de procesos?

Un administrador de procesos es un programa de computo que se utiliza para proporcionar información sobre los procesos y programas que se están activos en la computadora, esta aplicación se utiliza para detener o comunmente matar procesos.

¿Qué es un proceso?

Un proceso podría ser una instancia de un programa en ejecución. A los procesos frecuentemente se les refiere como tareas. El contexto de un programa que esta en ejecución es lo que se llama un proceso. Linux, es un sistema operativo multitarea y multiusuario. Esto quiere decir que múltiples procesos pueden operar simultáneamente sin interferirse unos con los otros. Cada proceso tiene la "ilusión" que es el único proceso en el sistema y que tiene acceso exclusivo a todos los servicios del sistema operativo.

Programas y procesos son entidades distintas, múltiples instancias de un programa pueden ejecutarse simultáneamente. Cada instancia es un proceso separado. Por ejemplo, si usuarios desde equipos diferentes, ejecutan el mismo programa al mismo tiempo, habría tantas instancias del mismo programa, es decir, procesos distintos.

Cada proceso que se inicia es referenciado con un número de identificación único conocido como Process ID PID, que es siempre un entero positivo. Prácticamente todo lo que se está ejecutando en el sistema en cualquier momento es un proceso, incluyendo el shell, el ambiente gráfico que puede tener múltiples procesos, etc. La excepción a lo anterior es el kernel en si, el cual es un conjunto de rutinas que residen en memoria y a los cuales los procesos a través de llamadas al sistema pueden tener acceso.

La administración de procesos es algo crucial a la hora de poder explotar todas las características del hardware y sistema operativo, entonces se hace imperioso el poder entender ciertas características de los mismos.

TASKLIST y TASKKILL son dos de los comandos que incluye el ejecutable cmd.exe que resultan muy útiles y nos auxilian cuando nos vemos en problemas en Windows.

Podemos usarlos directamente en la consola de CMD o Símbolo del sistema, en archivos batch o en scripts, para administrar completamente los procesos y tareas ejecutándose en nuestro equipo. Podemos con ellos obtener información y crear listas detalladas, detener aplicaciones, tareas y procesos aun cuando están bloqueados y no responden.

II. Desarrollo de la práctica

1.-

`pthread_create()`:

```
#include < pthread.h >
```

```
int pthread_create (pthread_t * hilo , const pthread_attr_t * attr ,  
void * (* start_routine ) (void *), void * arg );
```

La función `pthread_create ()` inicia un nuevo hilo en el proceso de llamada. El nuevo hilo comienza la ejecución invocando `start_routine ()`; `arg` se pasa como el único argumento de `start_routine ()`.

La función que nos permite crear un nuevo hilo de ejecución es `pthread_create()` que admite cuatro parámetros:

- `pthread_t *` es un puntero a un identificador de thread. La función nos devolverá este valor relleno, de forma que luego podamos referenciar al hilo para "hacerle cosas", como matarlo, esperar por él, etc.
- `pthread_attr_t *` son los atributos de creación del hilo. Hay varios atributos posibles, como por ejemplo la prioridad. Un hilo de mayor prioridad se ejecutará con preferencia (tendrá más rodajas de tiempo) que otros hilos de menor prioridad. Se puede pasar NULL, con lo que el hilo se creará con sus atributos por defecto. Si queremos un programa que cree y destruya hilos continuamente, no vale NULL, ya que con esta opción dejaremos memoria sin liberar cada vez que termine un hilo.
- `void *(*)(void *)` No es más que el tipo de una función que admite un puntero `void *` y que devuelve `void *`. Eso quiere decir que a este parámetro le podemos pasar el nombre de una función que cumpla lo que acabamos de decir. Esta función es la que se ejecutará como un hilo aparte. El hilo terminará cuando la función termine o cuando llame a la función `pthread_exit()`.
- `void *` es el parámetro que se le pasará a la función anterior cuando se ejecute en el hilo aparte. De esta manera nuestro programa principal puede pasar un único parámetro (que puede ser cualquier cosa, como una estructura compleja) a la función que se ejecutará en el hilo. La función del hilo sólo tendrá que hacer el "cast" adecuado.

`pthread_exit()`

```
void pthread_exit(void *value_ptr);
```

Se utiliza para la terminación del subproceso de llamada. Después de una llamada a esa función, se inicia un mecanismo de limpieza complicado. Cuando se completa el hilo se termina. La API `pthread_exit()` también se llama implícitamente cuando se produce una llamada a la rutina `return ()` en un hilo creado por `pthread_create ()`.

`pthread_join()`

`int pthread_join(pthread_t thread, void **value_ptr);`

La función `pthread_join` suspende la ejecución del hilo invocador hasta que el hilo identificado con `thread` termine, ya sea por que llamó a `pthread_exit` o por que fue cancelado. Esta llamada es similar a `waitpid` en el nivel de procesos. Si `value_ptr` no es `NULL`, entonces el valor retornado por `thread` es almacenado en la ubicación apuntada por `value_ptr`.

`pthread_self()`

`pthread_t pthread_self(void);`

La función `pthread_self` devuelve el identificador del hilo que esta invocando la función.

`scandir()`

La función `scandir()` rastrea el directorio `dir`, llamando `select()` en cada entrada de directorio. Las entradas para las que `select()` devuelve un valor distinto de cero se almacenan en cadenas (strings) reservadas vía `malloc()`, ordenadas usando `qsort()` con la función de comparación `compar()`, y puestas en la matriz `namelist` que está reservada vía `malloc()`. Si `select` es `NULL`, se seleccionan todas las entradas.

La función `scandir()` devuelve el número de entradas de directorio seleccionadas, o -1 si hubo algún error.

`stat()`

Se utiliza para mostrar por pantalla los atributos de un archivo.

El comando `stat` utiliza una estructura muy básica, ya que simplemente es necesario indicarle el nombre del archivo como argumento del comando.

`stat {option} FILE`

Programa 2

```
void *thread_routine(void *arg){  
  
    printf("\nHola mundo desde un hilo en UNIX\n");  
  
    return 0;  
}  
  
int main(int argc, char *argv[]){  
  
    pthread_t thread1;  
    int value = atoi(argv[1]);  
  
    if(0 != pthread_create(&thread1, NULL, thread_routine, &value))  
        return -1;  
  
    pthread_join(thread1, NULL);  
  
    return 0;  
}
```

Este programa es la visualización de crear hilos en c (Linux), con ayuda de la función pthread, aquí podemos observar una función puntero tipo void con un parámetro void puntero, la cual llevará un mensaje de que se ha creado el hilo.

En la función principal "main", crearemos una variable tipo "pthread" y una variable entera, la cual guardará un casteo del argumento ingresado a la hora de compilar.

Mediante una condición indicaremos que si la función pthread_create es diferente a 0 retornará -1, de lo contrario con la función "pthread_join" concluirá el proceso del hilo creado.

```
hector@hector-VirtualBox:~/Documentos/S0$ ./a.out 1
Hola mundo desde un hilo en UNIX
hector@hector-VirtualBox:~/Documentos/S0$
```

Programa 3

```
void *hilo(void *arg);
int main(void)
{
    pthread_t id_hilo;
    char* mensaje="Hola a todos desde el hilo";
    int devolucion_hilo;
    pthread_create(&id_hilo,NULL,hilo,(void*)mensaje);
    pthread_join(id_hilo,(void**)&devolucion_hilo);
    printf("valor = %i\n",devolucion_hilo);
    return 0;
}
void *hilo(void *arg)
{
    char* men;
    int resultado_hilo=0;
    men=(char*)arg;
    printf("%s\n",men);
    resultado_hilo=100;
    pthread_exit((void*)resultado_hilo);
}
```

Con base en el anterior programa, se realiza la misma estructura, la diferencia es que aquí agregaremos un valor al hilo creado, para enviar este valor del hilo, mediante la función pthread_join enviaremos la variable donde se almacenará el resultado el cual cuando se invoque la función hilo, actualizará el nuevo valor y lo imprimirá cuando se esté creando el hilo.

```

hector@hector-VirtualBox:~/Documentos/SO$ ./a.out 1
Hola a todos desde el hilo
valor = 100
hector@hector-VirtualBox:~/Documentos/SO$

```

Programa 4

```

DWORD WINAPI funcionHilo(LPVOID lpParam);
typedef struct Informacion info;
struct Informacion
{
    int val_1;
    int val_2;
};

int main(void)
{
    DWORD idHilo; /* Identificador del Hilo */
    HANDLE manHilo; /* Manejador del Hilo */
    info argumentos;
    argumentos.val_1=10;
    argumentos.val_2=100;
    // Creacion del hilo
    manHilo=CreateThread(NULL, 0, funcionHilo, &argumentos, 0, &idHilo);

    // Espera la finalización del hilo
    WaitForSingleObject(manHilo, INFINITE);
    printf("Valores al salir del Hilo: %i %i\n", argumentos.val_1, argumentos.val_2);

    // Cierre del manejador del hilo creado
    CloseHandle(manHilo);
    return 0;
}

DWORD WINAPI funcionHilo(LPVOID lpParam)
{
    info *datos=(info *)lpParam;
    printf("Valores al entrar al Hilo: %i %i\n", datos->val_1, datos->val_2);
    datos->val_1*=2;
    datos->val_2*=2;
    return 0;
}

```

Usando la lógica del programa anterior en Linux, windows es un poco similar, ya que se usa la misma función `CreateThread`, ya que esta creara el hilo a manejar, como en Linux, Windows debemos cerrar el manejador de este hilo, por ende la función `WaitForSingleObject` lo esperara el final del proceso de este y con la función `CloseHandle` cerraremos el hilo.

De igual podemos ver como pasamos informacion al hilo ejecutado, mediante variables de una estructura, enviaremos la información necesaria a mostrar.

Programa 5

```
//funcion que imprime el identificador de el hilo
void* idHilo(void* p) {
    printf("El identificador del hilo es= %ld\n", pthread_self());
    pthread_exit(NULL);
    return NULL;
}
```

```
//imprime el mensaje de Practica 2
void* imprimirMensaje(void* mensaje){
    char* cadena = (char*) mensaje;
    printf("%s \n", cadena);
    return NULL;
}
```

```
int main(){
    pthread_t hilo[15];

    int identificador = fork();
    // creacion de un proceso por copia exacta
    if(identificador == 0){
        //imprimir el id del proceso
        printf("El identificador del proceso hijo es= %d\n", identificador);
        //creacion de los hilos
        for(int i=0; i<15; i++){
            pthread_create(&hilo[i], NULL, idHilo, NULL);

            for(int j=0; j<10; j++){
                pthread_create(&hilo[j], NULL, idHilo, NULL);

                //creacion de los hilos terminales
                for(int k=0; k<5; k++){
                    pthread_create(&hilo[k], NULL, imprimirMensaje, "Practica 2");
                }
            }
        }
        // esperar la terminaci3n de los hilos.
        for(int i=0; i<15; i++){
            pthread_join(hilo[i], NULL);
            for(int j=0; j<10; j++){
                pthread_join(hilo[j], NULL);
                for(int k=0; k<5; k++){
                    pthread_join(hilo[k], NULL);
                }
            }
        }

        exit(0);
    }
    return 0;
}
```

```

//funcion que imprime el identificador de el hilo
DWORD WINAPI idHilo(LPVOID p) {
    printf("El identificador del hilo es= %d \n", GetCurrentThreadId());
}

//imprime el mensaje de Practica 2
DWORD WINAPI imprimirMensaje(LPVOID mensaje){
    char* cadena = (char*) mensaje;
    printf("%s \n", cadena);
}

int main(){
    HANDLE hilos[15];
    HANDLE proceso;
    STARTUPINFO sinfo;
    PROCESS_INFORMATION pinfo;
    memset(&sinfo, 0, sizeof(STARTUPINFO));
    memset(&pinfo, 0, sizeof(PROCESS_INFORMATION));

    sinfo.dwFlags = STARTF_USESHOWWINDOW;
    sinfo.wShowWindow = SW_SHOWMAXIMIZED;

    BOOL bSuccess = CreateProcess("C:\\Users\\jona-\\Documents\\Jona\\ESCOM\\semestre 4\\Sistemas Operativos\\Practica y tarea 4\\notepad.exe",
    NULL, NULL, NULL, FALSE, CREATE_DEFAULT_ERROR_MODE, NULL, NULL, &sinfo, &pinfo);
    printf("El id del proceso hijo: %d\n", _getpid() );
}

```

```

for(int i=0; i<15; i++){
    hilos[i] = CreateThread(NULL, 0, idHilo, NULL, 0, NULL);

    for(int j=0; j<10; j++){
        hilos[j] = CreateThread(NULL, 0, idHilo, NULL, 0, NULL);

        //creacion de los hilos terminales
        for(int k=0; k<5; k++){
            hilos[k] = CreateThread(NULL, 0, imprimirMensaje, "Practica 2", 0, NULL);
        }
    }
}

for(int i=0; i<15; i++){
    WaitForSingleObject(hilos[i], INFINITE);

    for(int j=0; j<10; j++){
        WaitForSingleObject(hilos[j], INFINITE);

        for(int k=0; k<5; k++){
            WaitForSingleObject(hilos[k], INFINITE);
        }
    }
}

for(int i=0; i<15; i++){
    CloseHandle(hilos[i]);

    for(int j=0; j<10; j++){
        CloseHandle(hilos[j]);

        for(int k=0; k<5; k++){
            CloseHandle(hilos[k]);
        }
    }
}

return 0;

```


Programa 6

```
jona@ubuntu:~$ gcc -o Hilos hilos.c -lpthread
jona@ubuntu:~$ ./Hilos
8      8      8      8      8      8      8
8      8      8      8      8      8      8
8      8      8      8      8      8      8
8      8      8      8      8      8      8
8      8      8      8      8      8      8
8      8      8      8      8      8      8
8      8      8      8      8      8      8


196    168    140    112    84     56     28
196    168    140    112    84     56     28
196    168    140    112    84     56     28
196    168    140    112    84     56     28
196    168    140    112    84     56     28
196    168    140    112    84     56     28
196    168    140    112    84     56     28


-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6
-6     -4     -2     0      2      4      6


1      1      1      1      1      1      1
2      2      2      2      2      2      2
3      3      3      3      3      3      3
4      4      4      4      4      4      4
5      5      5      5      5      5      5
6      6      6      6      6      6      6
7      7      7      7      7      7      7


jona@ubuntu:~$
```




multDeMat
rices.txt


restaDeMa
trices.txt


sumaDeMa
trices.txt


transDeMa
trices.txt

transDeMatrices.txt ~/Documents							
Open	▼	+					
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7

sumaDeMatrices.txt ~/Documents							
Open	▼	+					
1	8	8	8	8	8	8	8
2	8	8	8	8	8	8	8
3	8	8	8	8	8	8	8
4	8	8	8	8	8	8	8
5	8	8	8	8	8	8	8
6	8	8	8	8	8	8	8
7	8	8	8	8	8	8	8

restaDeMatrices.txt ~/Documents							
Open	▼	+					
1	-6	-4	-2	0	2	4	6
2	-6	-4	-2	0	2	4	6
3	-6	-4	-2	0	2	4	6
4	-6	-4	-2	0	2	4	6
5	-6	-4	-2	0	2	4	6
6	-6	-4	-2	0	2	4	6
7	-6	-4	-2	0	2	4	6

multDeMatrices.txt ~/Documents							
Open	▼	+					
1	196	168	140	112	84	56	28
2	196	168	140	112	84	56	28
3	196	168	140	112	84	56	28
4	196	168	140	112	84	56	28
5	196	168	140	112	84	56	28
6	196	168	140	112	84	56	28
7	196	168	140	112	84	56	28

```
/*inicializarMatriz1 no regresa nada, rellena la matriz del 1 al 7 en cada renglon*/
void inicializarMatriz1(int mat1[FILAS][COLUMNAS]){
    int i, j;
    int numeros[7] = {1,2,3,4,5,6,7};
    //Se llena la matriz 1 con valores del 1 al 7
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat1[i][j] = numeros[j];
        }
    }
}

/*inicializarMatriz1 no regresa nada, rellena la matriz del 7 al 1 en cada renglon*/
void inicializarMatriz2(int mat2[FILAS][COLUMNAS]){
    int i, j;
    int numeros[7] = {1,2,3,4,5,6,7};
    //Se llena la matriz 2 con valores del 7 al 1
    for(i = 0; i < 7; i++){
        int k=6;
        for(j = 0; j < 7; j++){
            mat2[i][j] = numeros[k];
            k--;
        }
    }
}
```

```
/*sumarMatrices no regresa nada, suma la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void sumarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int i, j;
    //Se suman los elementos de las matrices
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat3[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}

/*restarMatrices no regresa nada, resta la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void restarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int i, j;
    //Se restan los elementos de las matrices
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat3[i][j] = mat1[i][j] - mat2[i][j];
        }
    }
}
```

```

/*multiplicarMatrices no regresa nada, multiplica la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void multiplicarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int c,d,k, sum=0;
    //Se multiplican los elementos de las matrices
    for (c = 0; c < 7; c++){
        for (d = 0; d < 7; d++){
            for (k = 0; k < 7; k++){
                sum = sum + mat1[c][k]*mat2[k][d];
            }
            mat3[c][d] = sum;
            sum = 0;
        }
    }
}

/*imprimirMatriz no regresa nada, pero imprime todos los renglones y columnas*/
void imprimirMatriz(int mat[7][7]){
    int i, j;
    printf("\n\nMatriz: ");
    //Se imprimen los elementos de la matriz
    for(i = 0; i < 7; i++){
        printf("\n");
        for(j = 0; j < 7; j++){
            printf("%d ", mat[i][j]);
        }
    }
}

```

```

/*procesoSumarMatriz, permite pasarla como parametro en pthread_create
Realiza la operacion de suma entre matrices ademas de escribir el resultado de la
suma en un archivo*/
void* procesoSumarMatriz(void* args){
    //Se hace un casting para usar la estructura
    struct argumentos* arg = (struct argumentos*) args;
    sumarMatrices(arg->mat1,arg->mat2,arg->matR);
    //Se guarda la direccion del archivo en una variable de la estructura
    arg->direccion = "Documents/sumaDeMatrices.txt";
    crearArchivo("Documents/sumaDeMatrices.txt");
    escribirMatriz(arg->matR, "Documents/sumaDeMatrices.txt");
    //leerImprimirArchivo("Documents/sumaDeMatrices.txt");
}

/*procesoRestarMatriz, permite pasarla como parametro en pthread_create
Realiza la operacion de resta entre matrices ademas de escribir el resultado de la
resta en un archivo*/
void* procesoRestarMatriz(void* args){
    //Se hace un casting para usar la estructura
    struct argumentos* arg = (struct argumentos*) args;
    restarMatrices(arg->mat1,arg->mat2,arg->matR);
    //Se guarda la direccion del archivo en una variable de la estructura
    arg->direccion = "Documents/restaDeMatrices.txt";
    crearArchivo("Documents/restaDeMatrices.txt");
    escribirMatriz(arg->matR, "Documents/restaDeMatrices.txt" );
    //leerImprimirArchivo("Documents/restaDeMatrices.txt" );
}

```

```

/*procesoMultiplicarMatriz, permite pasarla como parametro en pthread_create
Realiza la operacion de multiplicar matrices ademas de escribir el resultado de la
multiplicacion en un archivo*/
void* procesoMultiplicarMatriz(void* args){
    //Se hace un casting para usar la estructura
    struct argumentos* arg = (struct argumentos*) args;
    multiplicarMatrices(arg->mat1,arg->mat2,arg->matR);
    //Se guarda la direccion del archivo en una variable de la estructura
    arg->direccion = "Documents/multDeMatrices.txt";
    crearArchivo("Documents/multDeMatrices.txt");
    escribirMatriz(arg->matR, "Documents/multDeMatrices.txt" );
    //leerImprimirArchivo("Documents/multDeMatrices.txt" );
}

/*procesoTranspuestaMatriz, permite pasarla como parametro en pthread_create
Realiza la operacion de transpuesta de una matriz, ademas de escribir el resultado de la
la transpuesta en un archivo*/
void* procesoTranspuestaMatriz(void* args){
    //Se hace un casting para usar la estructura
    struct argumentos* arg = (struct argumentos*) args;
    transpuesta(arg->mat1,arg->matR);
    //Se guarda la direccion del archivo en una variable de la estructura
    arg->direccion = "Documents/transDeMatrices.txt";
    crearArchivo("Documents/transDeMatrices.txt");
    escribirMatriz(arg->matR, "Documents/transDeMatrices.txt" );
    //leerImprimirArchivo("Documents/transDeMatrices.txt");
}

```

```

int main(){
    struct argumentos args[5];
    pthread_t proceso[5];

    //Asignandole valores a diferentes matrices
    for(int i=0; i<5; i++){
        inicializarMatriz1(args[i].mat1);
        //imprimirMatriz(args[i].mat1);

        inicializarMatriz2(args[i].mat2);
        //imprimirMatriz(args[i].mat2);
    }

    //Creacion de hilos
    pthread_create(&proceso[0], NULL, procesoSumarMatriz,&args[0]);
    pthread_create(&proceso[1], NULL, procesoRestarMatriz,&args[1]);
    pthread_create(&proceso[2], NULL, procesoMultiplicarMatriz,&args[2]);
    pthread_create(&proceso[3], NULL, procesoTranspuestaMatriz,&args[3]);

    //Union de los hilos
    pthread_join(proceso[0], NULL);
    pthread_join(proceso[1], NULL);
    pthread_join(proceso[2], NULL);
    pthread_join(proceso[3], NULL);

    //Con un solo hilo pero con un for, se lee y se imprime el contenido de los archivos creados
    for(int i=0; i<4; i++){
        pthread_create(&proceso[4], NULL, leerImprimirArchivo,&args[i]);
    }
    pthread_join(proceso[4], NULL);

    return 0;
}

```

Conclusiones

Podemos concluir que un hilo es una unidad básica de utilización de CPU, la cual contiene un id de hilo, su propio counter, un conjunto de registros, y una pila; que se representa a nivel del sistema operativo con una estructura llamada TCB (thread control block).

Así mismo los hilos comparten con otros hilos que pertenecen al mismo proceso la sección de código, la sección de datos, entre otras cosas. Si un proceso tiene múltiples hilos, puede realizar más de una tarea a la vez.

También cabe aclarar que:

- Respuesta: el tiempo de respuesta mejora, ya que el programa puede continuar ejecutándose, aunque parte de él esté bloqueado.
- Compartir recursos: los hilos comparten la memoria y los recursos del proceso al que pertenecen, por lo que se puede tener varios hilos de ejecución dentro del mismo espacio de direcciones.
- Economía: Es más fácil la creación, cambio de contexto y gestión de hilos que de procesos.
- Utilización múltiples CPUs: permite que hilos de un mismo proceso ejecuten en diferentes CPUs a la vez. En un proceso mono-hilo, un proceso ejecuta en una única CPU, independientemente de cuantas tenga disponibles.