2017

Proyecto 1

PRINCIPIO DE SISTEMAS OPERATIVOS

Estudiantes:
Karen Lépiz Chacón – 2013005341
André Solís Barrantes - 2013389035

Contenido

Introducción.

El proyecto consta en modificar un archivo base para que las diferentes tareas sean realizadas por medio de hilos y procesos. El objetivo principal es aprender sobre el comportamiento de un pool de procesos/hilos a la hora de ejecutar las asignaciones dadas y comparar cual tiene mejor rendimiento.

Tile_bars será el archivo base, su función es buscar en un archivo las palabras claves que el usuario de e imprimir una barra donde se muestre la cantidad de apariciones de dicha palabra por párrafo, además también muestra el tiempo que uso en ejecución.

Descripción del problema.

Se deben realizar dos pruebas independientes, en la primera se debe modificar el código del programa para dividirlo en múltiples sub procesos Unix. En este caso será necesario contar con un mecanismo de memoria distribuida para pasar datos entre los sub procesos. Se deben crear sub procesos independientes para leer diferentes archivos, imprimir la barra, e inclusive para procesar cada línea. La cantidad de sub procesos a crear debe ser limitada y se deben manejar como un “pool” asignando a los procesos que ya terminaron, una nueva tarea. La segunda prueba utilizará múltiples hilos de ejecución para realizar la misma tarea colaborativa. Sin embargo, en este caso no será necesario crear la memoria compartida pues todos los hilos comparten la misma memoria. Al igual que en el caso de múltiples procesos se debe manejar un “pool” de hilos de un tamaño definido y se deben asignar hilos a las diferentes tareas de leer archivos, imprimir la barra, y procesar cada línea. Se deben realizar múltiples pruebas para determinar el tamaño óptimo del “pool” tanto para los sub procesos como para los hilos. Se utilizará el tiempo de ejecución para determinar dicho tamaño óptimo. Note que el proceso principal debe esperar hasta que terminen todos los sub procesos/hilos para imprimir el tiempo total de ejecución.

Definición de estructura de datos.

Procesos

- Se utilizan listas para los estados de los procesos.
- Se utiliza shmctl() para realizar la memoria compartida y facilitar su uso mediante un stuct que viene predefinido:

```
struct shmid_ds {
    struct ipc_perm shm_perm;    /* Ownership and permissions */
    size_t          shm_segsz;   /* Size of segment (bytes) */
    time_t          shm_atime;   /* Last attach time */
    time_t          shm_dtime;   /* Last detach time */
    time_t          shm_ctime;   /* Last change time */
    pid_t           shm_cpid;    /* PID of creator */
    pid_t           shm_lpid;    /* PID of last shmat(2)/shmdt(2) */
    shmatt_t        shm_nattch;  /* No. of current attaches */
    ...
};
```

Imagen 1. Componentes del struct de shmctl, incluido en sys/shm.h.

Hilos.

- Se utilizan diferentes structs.
    - Para realizar el pool se utilizaron las siguientes:

```
// formas de apagado
typedef enum {
    apagado_inmediato = 1,
    apagado_descente  = 2
} threadpool_tipoApagado;

typedef enum {
    threadpool_graceful       = 1
} threadpool_destroy_flags_t;

// struct para pasar las funciones
typedef struct {
    void (*function)(void *); // nombre de la funcion
    void *argument; // argumentos que tiene la funcion (entradas)
} threadpool_tareas;
```

```c
struct threadpool {

  // datos necesarios que debe el hilo para manejarlo en el pool
  pthread_mutex_t lock;
  pthread_cond_t notify; // encargado de notificar a los hilos
  pthread_t *hilos; // array que contiene los hilos.
  threadpool_tareas *cola; // struct para indicar la tarea
  int cant_hilos; // numero de hilos que hay o se crearon
  int tam_cola; // tamano de la cola tarea
  int head; // indexa el primer elemenot
  int tail; // indexa el siguiente elemento
  int cant; // numero de tareas pendientes
  int estado; // pool esta apagada (flag)
  int started; // subprocesos iniciados

};
```

- Para pasar datos que necesitan las tareas para ser ejecutadas:

```c
//..................... datos que ocupan las funciones en un struct ..........................

struct datos_funciones {

    int num_keys;
    int num_line;
    int *counters;
    char* line;
    char* filename;
    char* token;
    threadpool *pool;
    int id;
};
```

- Se usaron listas para guardar todas las líneas del archivo que se va a leer y de las palabras claves que se van a usar. Se puede un máximo de 6 palabras claves.

Descripción detallada y explicación de los componentes principales del programa.

File_process.

- Objetivo: Es la función encargada de obtener las diferentes líneas del archivo que se mandaran al line_process, además se tendrá el número total de líneas que se leyeron.
- Entradas:
  - Procesos: el nombre del archivo, keywords y el número de palabras claves.
  - Hilos: un structs con los datos necesarios para su ejecución.
- No retorna nada.

Line_process:

- Objetivos: guarda en una lista la cantidad de veces que aparece la palabra clave en la línea que se está procesado, es decir, guarda en un campo específico el resultado que retorna keywords_process.
- Entradas:
  - Procesos: arreglo de líneas, el número de línea que se va a procesar, el arreglo en donde están las palabras claves, el número de palabras claves y un arreglo contador en donde se almacena la cantidad de veces que se repite la palabra clave en la línea que se esté procesando.
  - Hilos: un structs con los datos necesarios para su ejecución.
- No retorna nada.

Keywords_process:

- Objetivo: Es en donde se cuenta cuantas veces aparece la palabra clave en la línea dada.
- Entradas: la línea que se está procesando y la palabra clave que se quiere buscar.
- Retorna el número de veces que aparece la palabra clave en la línea dada (counters)

Print_bars:

- Objetivo: Imprime una barra en escala de grises donde muestra que tan seguido aparece la o las palabras claves en la línea.
- Entradas:
  - Procesos: un arreglo con las palabras claves, el número de palabras claves que hay, el número de línea y el arreglo counters
  - Hilos: un structs con los datos necesarios para su ejecución.
- Imprime una barra en escala de grises.

Threadpool_añadirTarea:

- Objetivo: es la función que se encarga de agregar en cola una tarea.
- Entradas: cantidad de hilos que tendrá el pool, el tamaño de la cola y flag.

Threadpool_generar:

- Objetivo: función que genera el pool de hilos de acuerdo a una cantidad de hilos especifico y tamaño de la cola.
- Entradas: Se le asigna el pool con el que va a trabajar, la tarea, argumentos que ocupa la función y flag.

Mecanismo de creación y comunicación de procesos.

Procesos: Los procesos se comunican por medio de una memoria compartida que se crea de la siguiente manera:

```
/* we create the shared memory */
id_zone = shmget (key, sizeof(int)*500*num_keys, 0777 | IPC_CREAT);
int num_zone = shmget (keyNum, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
if (num_zone == -1 || id_zone == -1) {
   fprintf (stderr, "Error with id_zone \n");
}
```

Y para crear el pool de procesos:

```
// creacion del pool de procesos
for (int i=0; i<POOL; ++i){
    if ((pid = fork()) <=0 ) {
        break;
    }
}
```

Hilos: En los hilos las tareas se van ejecutando de acuerdo al orden en que se hayan agregado a la cola de tareas.

```
//..... ingresar la tarea en la cola .....

pool->cola[pool->tail].function = function;
pool->cola[pool->tail].argument = argument;
pool->tail = next;
pool->cant += 1;
```

El pool se crea de la siguiente forma:

```
assert((pool = threadpool_generar(THREAD, QUEUE, 0)) != NULL);
```

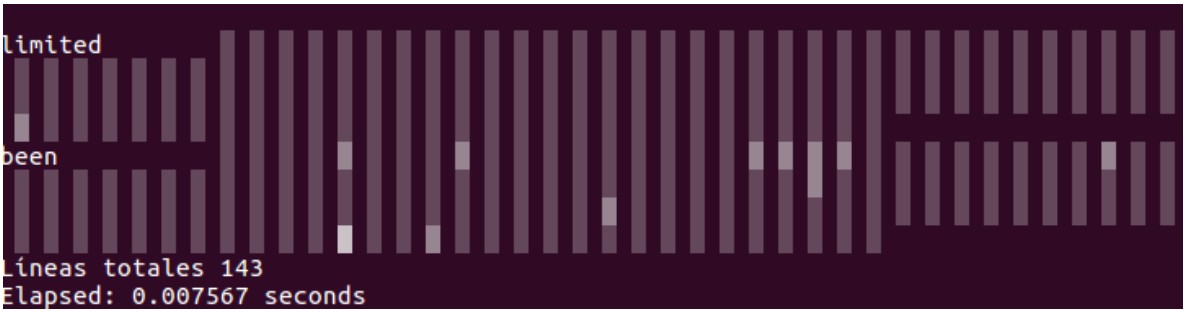
Los procesos obtienen los datos necesarios para su ejecución por medio de un struct , que se mencionó anteriormente.

Pruebas de rendimiento.

Prueba 1: archivo1.txt, keywords: limited, been

Hilos.

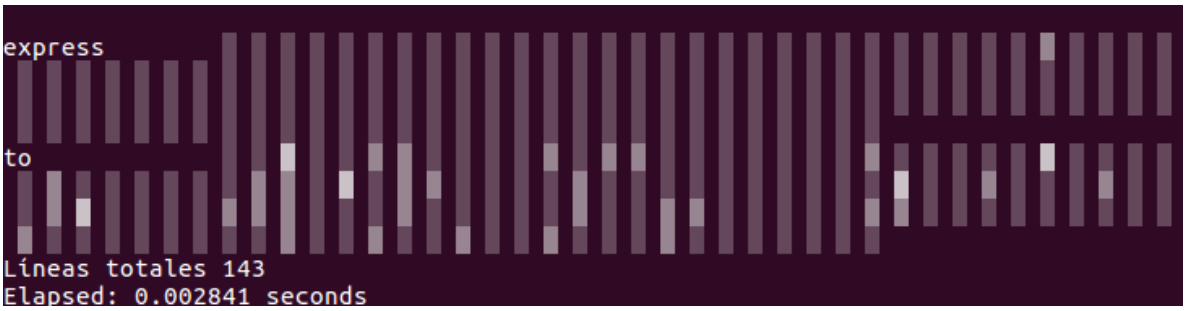
Procesos



Prueba 2: archivo3.txt, keywords: express, to.

Hilos

Procesos



Conclusiones.

Según las pruebas de rendimiento que se realizaron, indican que ejecutar los procesos por hilos terminan su ejecución más rápida que hacerlas por procesos. Sin embargo, los resultados podrían variar al cambiar el tamaño del pool, ya sea colocando un tamaño más grande del necesario o el más óptimo. También se concluye que el procedimiento para realizar una serie de tareas es diferente entre hilos y procesos, debido a que por hilos se van realizando por medio de una cola por lo que se debe tener en cuanta en el orden en que se tiene que ejecutar las tareas para obtener los resultados esperados. Además, el hecho de que los procesos se ejecutan en memoria retrasa el tiempo que dura en ejecución pues tiene que estar consultado en memoria cada vez que quiera realizar una tarea.