Process Pool Copy Program

Estudiantes:

Gerald Núñez Chavarría - 2021023226, Sebastián Arroniz Rojas - 2021108521

Profesor: Armando Arce Orozco

Instituto Tecnológico de Costa Rica

IC6600 Principios de Sistemas Operativos

Fecha: 18/04/2024

Introducción

El objetivo de este proyecto es comparar el rendimiento de realizar una tarea colaborativa utilizando múltiples procesos. Para ello se deberá desarrollar una versión multiprocesos del programa `copy` que permita copiar el contenido de un directorio completo.

Una vez desarrollado el programa se debe realizar un análisis de resultados para comparar el mismo programa utilizando una cantidad diferente de procesos para copiar un único directorio muy grande y así concluir cuál es la cantidad estándar adecuada de procesos que brinda una mejora en el rendimiento de esta tarea. Este proyecto se desarrolla en el lenguaje de programación C y el entorno del sistema operativo Unix.

Descripción del Problema

A continuación se presenta el enunciado del proyecto, que se convierte en la especificación base para su realización.

Enunciado

El objetivo de este proyecto es comparar el rendimiento de realizar una tarea colaborativa utilizando múltiples procesos. Para ello se deberá desarrollar una versión multiprocesos del programa copy que permita copiar el contenido de un directorio completo.

En este proyecto se deben realizar múltiples pruebas para determinar la cantidad óptima de procesos para ejecutar este tipo de tarea sobre un único directorio muy grande.

Uso del Programa

El programa `copy` recibe el nombre del directorio origen y el directorio destino:

```
copy directorio_origen directorio_destino
```

Por omisión el programa debe mostrar el nombre de cada archivo que se copia y su cantidad de bytes.

Lectura de Directorios

El proceso principal deberá leer el directorio indicado, crear un pool de procesos y asignar a cada proceso la copia de uno de los archivos en dicho directorio. La lectura de directorios y archivos se puede realizar utilizando como base los tutoriales respectivos (vistos en clase). Se debe procesar el directorio y todos los subdirectorios bajo él.

Manejo de Múltiples Procesos

El programa debe contar con múltiples procesos que se encarguen de copiar los archivo. Debido a que la copia se realiza sobre múltiples archivos no sería eficiente crear un único proceso. Es por ello que es necesario crear un conjunto compartido de procesos (pool de procesos) que incluya solo una cantidad limitada de procesos, y en donde cada proceso trabajará sobre un archivo específico. Como parte del proyecto se deben realizar diferentes pruebas (con diferentes directorios grandes) para determinar un tamaño "óptimo" del pool de procesos. Se utilizará el tiempo de ejecución para determinar dicho tamaño óptimo.

En este caso se utilizará un pool de procesos estático, es decir, el programa creará la totalidad de los procesos al inicio del programa y únicamente utilizará estos para realizar todo el trabajo. Tal como usted debe suponer, al proceso que termine de copiar un archivo deberá asignarse otro archivo para copiar hasta terminar con todo el directorio.

Comunicación Entre Procesos

El proceso principal (coordinador) debe comunicarse con los procesos copiadores (esclavos) mediante mensajes (NO sockets). El mensaje debe especificar el nombre del archivo a copiar, y el proceso copiador debe enviar de vuelta un mensaje indicando que terminó la copia. Note que se debe utilizar una única cola y utilizar un identificador para determinar el proceso que recibe el mensaje.
Al finalizar de copiar todos los archivos el proceso principal deberá enviar mensajes de terminación a cada uno de los procesos esclavos. Hasta que cada uno de estos procesos no haya terminado, no podrá terminar el proceso principal.

Análisis de Resultados

Se deberán realizar varias pruebas cambiando la cantidad de procesos en el pool. Una vez que se calcule el tiempo de ejecución final, se deben crear tablas y gráficas en donde se muestre cual cantidad de procesos brinda el mejor rendimiento para realizar la copia de archivos en paralelo. Cree un archivo de bitácora (logfile) desde el proceso principal que escriba una entrada por cada archivo que se copia, indicando el nombre del archivo, el subproceso que lo copió, y el tiempo

que se duró. Utilice preferiblemente un formato CSV para poder realizar el análisis de resultados desde Excel o R.

Definición de Estructuras de Datos

Para el desarrollo del programa se necesitó definir dos estructuras de datos. La primera permite controlar el pool de procesos, manejando un arreglo para tener el `pid` de cada proceso y un entero que indica el número de procesos que se encuentran disponibles para realizar tareas de copia de directorios o archivos. La segunda sirve para utilizarla como buffer en la cola de mensajes, esta permite almacenar el tipo del mensaje, el texto (nombre del archivo a copiar), la acción que se debe realizar, el proceso que lo envía y una variable para almacenar la duración que se obtuvo copiando ese archivo.

Estructura para controlar el pool de procesos:

```
struct process_pool_control

{

  int pids[PROCESS_POOL_SIZE][2];

  int available_processes;

};
```

Estructura para el buffer de la cola de mensajes:

```
struct msgbuf

{

  long mtype;

  char mtext[MAX_MSG_LEN];

  int action;

  int sender_pid;

  double copy_duration;

};
```

Componentes Principales del Programa

A continuación se explican los cinco componentes principales que tiene el programa, cada uno se encuentra dividido en un archivo para separar la lógica.

Componente de Archivos

Se encuentra en el archivo `archives.c`. Este componente contiene funciones para manipular archivos y directorios. Algunas de las operaciones que realiza incluyen:

1. Crear un directorio si no existe (`create_dir`).

2. Copiar un archivo de origen a un archivo de destino (`copy_file`).
3. Copiar el contenido de un directorio de origen a un directorio de destino, incluyendo subdirectorios y archivos (`copy_directory`). Esta función utiliza mensajes de cola para coordinar la copia de archivos entre múltiples procesos.

Componente de Procesos

Se encuentra en el archivo `processes.c`. Este componente se encarga de gestionar un "pool" de procesos en un entorno. Algunas de las operaciones que realiza son:

1. Crear el "pool" de procesos (`create_process_pool`). Esta función crea varios procesos hijos utilizando la llamada al sistema `fork()`. Retorna 0 si el proceso es un hijo y 1 si es el proceso padre.
2. Eliminar el "pool" de procesos (`delete_process_pool`). Esta función envía la señal `SIGKILL` a cada proceso hijo para terminarlos.
3. Obtener el PID de un proceso libre (`get_free_process_pid`). Retorna el PID de un proceso que está marcado como libre en el "pool".
4. Establecer el estado de un proceso (`set_process_state`). Esta función actualiza el estado de un proceso en el "pool" (1 para libre, 0 para ocupado).

Estas funciones son útiles para gestionar y controlar la ejecución de múltiples procesos concurrentes.

Componente de Registros

Se encuentra en el archivo `logs.c`. Este componente proporciona funciones relacionadas con la creación y escritura de registros en un archivo de registro. Las funciones incluidas son:

1. Crear un archivo de registro (`create_log_file`). Esta función crea un archivo de registro nuevo o sobrescribe uno existente. El archivo se abre en modo de escritura (`"w"`), lo que significa que si el archivo existe, su contenido se eliminará.
2. Agregar una entrada al archivo de registro (`add_entry_log_file`). Esta función agrega una nueva entrada al archivo de registro existente. La entrada contiene el nombre del archivo copiado, el ID del proceso que realizó la copia y la duración de la operación de copia en segundos. La información se escribe en formato CSV.

Estas funciones son útiles para mantener un registro de las operaciones de copia de archivos realizadas por un programa, lo que es útil para poder analizar el rendimiento según el número de procesos utilizados.

Componente de la Cola de Mensajes

Se encuentra en el archivo `msg-queue.c`. Este componente proporciona funciones para la creación, gestión y uso de colas de mensajes. Algunas de las operaciones que realiza son:

1. Crear una cola de mensajes (`create_msg_queue`). Esta función crea una nueva cola de mensajes o reemplaza una existente si ya hay una con la misma clave. Retorna el identificador de la cola de mensajes (`msqid`).
2. Comprobar si una cola de mensajes está vacía (`is_msg_queue_empty`). Esta función verifica si la cola de mensajes especificada está vacía o no.
3. Eliminar una cola de mensajes (`delete_msg_queue`). Esta función elimina la cola de mensajes especificada.
4. Enviar un mensaje a la cola de mensajes (`send_msg`). Esta función envía un mensaje a la cola de mensajes especificada. El mensaje contiene un tipo, un texto, una acción, el PID del

remitente y una duración de copia. La bandera `flag` determina si el envío es bloqueante o no.
5. Recibir un mensaje de la cola de mensajes (`receive_msg`). Esta función recibe un mensaje de la cola de mensajes especificada. La función puede ser bloqueante o no, dependiendo de la bandera `flag`.
6. Obtener la longitud actual de la cola de mensajes (`len_msg_queue`). Esta función devuelve el número actual de mensajes en la cola de mensajes.
7. Obtener el PID del último proceso que envió un mensaje a la cola (`get_last_sender`). Esta función proporciona el PID del último proceso que envió un mensaje a la cola de mensajes especificada.

Estas funciones son esenciales para la comunicación entre procesos. Permiten la transferencia eficiente de datos entre procesos de manera asíncrona, facilitando la implementación de la copia de archivos mediante múltiples procesos.

Mecanismo de Creación y Comunicación de Procesos

Los procesos del programa se crean mediante el uso de la función llamada `create_process_pool`. Que es una función que utiliza un ciclo en el que llama la función `fork()` para crear hijos y se detiene cuándo se alcanza el tamaño definido mediante la macro `#define PROCESS_POOL_SIZE 2`. De esta manera ya se tiene un pool de procesos para utilizarlo durante todo el programa.

La comunicación entre procesos se logra mediante el uso de una cola de mensajes que se crea mediante la función `create_msg_queue`. En el proceso padre, se envían mensajes a la cola de mensajes para coordinar la copia de archivos y directorios. Los procesos hijos reciben estos mensajes, ejecutan las tareas correspondientes y envían mensajes de vuelta al proceso padre para informar sobre el progreso y los resultados de la tarea.

Instrucciones de Uso

Compilación

Estando en la raíz del proyecto, ejecute el comando `sh compile.sh`.

Ejecución

Llame al ejecutable `main.out` que se encuentra en la carpeta `bin`. Este ejecutable recibe 3 argumentos aparte del nombre del programa:

1. Directorio fuente.
2. Directorio destino. Es necesario que este directorio exista.
3. Nombre del archivo de logs.

Ejemplo: `./main.out test copy-test log-file-01.csv`

Pruebas de Rendimiento

Se llevarán a cabo pruebas de rendimiento del programa, variando el tamaño del pool de procesos, con el objetivo de determinar la configuración más eficiente. Para ello, se realizará la copia de un directorio llamado "test", que contiene varios directorios y archivos, al directorio de destino denominado "copy-test".

La métrica de rendimiento seleccionada consiste en el tiempo total requerido para copiar todos los archivos. Esta métrica se obtiene mediante la suma de todos los tiempos registrados en el archivo de logs, el cual incluye el nombre del archivo copiado, el PID del proceso que lo copió y su duración en milisegundos. Las pruebas se realizarán utilizando 2, 4, 9, 10, 16 y 20 procesos hijos.

Para modificar la cantidad de procesos en el pool, simplemente se debe ajustar la macro `#define PROCESS_POOL_SIZE 2`, donde el número al final representa la cantidad de procesos en el pool.

Prueba 1

Se utilizan un total de 2 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-01.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 1 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 151.205 milisegundos de duración.

Prueba 2

Se utilizan un total de 4 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-02.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 2 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 86.511 milisegundos de duración.

Prueba 3

Se utilizan un total de 9 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-03.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 3 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 95.939 milisegundos de duración.

Prueba 4

Se utilizan un total de 10 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-04.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 4 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 75.966 milisegundos de duración.

Prueba 5

Se utilizan un total de 16 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-05.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 5 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 83.686 milisegundos de duración.

Prueba 6

Se utilizan un total de 20 procesos. El comando utilizado es: `./bin/main.out test copy-test logfile-06.csv`. Los registros en el archivo de logs proporcionan:

[Insertar Imagen 6 aquí]

Al sumar los valores de "duration", que representan la duración en milisegundos de cada proceso para copiar el archivo, se obtiene un total de 81.065 milisegundos de duración.

Análisis Gráfico de las Pruebas

Con el fin de visualizar mejor las pruebas, se realizó el siguiente gráfico que representa la relación entre la duración total de realizar la copia de archivos con el número de procesos que se utilizó:

Imagen Gráfica

Conclusión de las Pruebas

El mejor resultado se obtiene en la prueba número 4 dónde se utilizaron 10 procesos y la duración fue de 75.966 milisegundos. El peor resultado se obtiene en la prueba número 1 dónde se utilizaron 2 procesos y la duración fue de 151.205 milisegundos.

Al utilizar únicamente dos procesos, la copia de archivos es más lenta. No obstante, después de cuatro procesos, aumentar el número no significa disminuir la duración. Se concluye que después de cuatro procesos hijos, se va a mantener una duración promedio de 60 a 100 milisegundos después de cuatro procesos al realizar el proceso de copia.

Conclusiones

1. Se implementó un programa en C que utiliza múltiples procesos para copiar archivos de un directorio a otro. Se creó un pool de procesos estático al inicio del programa para realizar las tareas de copia de archivos de manera concurrente.
2. El programa consta de cinco componentes principales: archivos, procesos, registros, cola de mensajes y un mecanismo de creación y comunicación de procesos. Cada componente desempeña un papel crucial en la creación, gestión y ejecución eficiente de las tareas de copia de archivos.
3. Se llevaron a cabo pruebas variando el tamaño del pool de procesos (2, 4, 9, 10, 16 y 20 procesos) y se registró el tiempo total requerido para copiar todos los archivos. Los resultados mostraron que el mejor rendimiento se logró con 10 procesos, con una duración de 75.966 milisegundos, mientras que el peor rendimiento se obtuvo con solo 2 procesos, con una duración de 151.205 milisegundos.
4. Se realizó un análisis gráfico para visualizar la relación entre la duración total de la copia de archivos y el número de procesos utilizados. Se observó que después de cuatro procesos, aumentar el número no significaba una disminución significativa en la duración.
5. Se concluyó que después de cuatro procesos hijos, la duración promedio de la copia de archivos se mantuvo en el rango de 60 a 100 milisegundos. A partir de esta cantidad de procesos, agregar más no proporcionaba una mejora sustancial en el rendimiento.