Universidad Católica Andrés Bello
Escuela De Ingeniería Informática
Sistemas de Operación
Profesor: Richard Duarte

Documentación del Proyecto

Integrantes:

Ricardo Salvatorelli (C.I.: 26.967.602)
José Manuel Ramírez (C.I.: 26.902.002)
Leonardo Ruiz (C.I.: 25.209.081)
Sebastián González (C.I.: 27.234.142)

Caracas, 7 de enero de 2020.

Introducción

El presente informe contiene la documentación necesaria del proyecto de la materia Sistemas de Operación el cual está conformado por 5 programas, los cuales fueron desarrollados usando el lenguaje de programación C, y usando el sistema operativo Ubuntu (Linux).

Como Ejecutar el Proyecto

- Colocar todos los archivos del proyecto en una carpeta.
- Abrir la terminar de Linux en la carpeta.
- Escribir el comando “make” para compilar todos los programas del proyecto.
- Para ejecutar el Fibonacci Secuencial, escribir el comando “./FibSecuencial”.
- Para ejecutar el Fibonacci Multihilos, escribir el comando “./FibMulti”.
- Para ejecutar el Messenger de Procesos Hijos, escribir el comando “./procmsg”.
- Para ejecutar el ucp, se utiliza un archivo de prueba que tiene de tamaño 1.4 megabytes llamado “pruebaParaUCP.txt”. Para ejecutar la copia, se escribe el comando: “./ucp #buffer nombreArchivoFuente nombreArchivoDestino”.
- Para ejecutar el timeprog, se utiliza el comando “./timeprog ./programa parametro1 parametro2 …”, para probar el timeprog junto con ucp se coloca:
  “./timeprog ./ucp #buffer nombreArchivoFuente nombreArchivoDestino”.
- Si se desean eliminar los ejecutables rápidamente, se escribe en consola “make clean”.

Documentación de cada programa

MAKEFILE:

El makefile del proyecto básicamente consiste de 7 posibles comandos, los cuales son “All, timeprog, ucp, procmsg, FibSecuencial, FibMulti y clean”. Para utilizar el makefile, se debe colocar en consola “make” seguido del comando correspondiente.

Los comandos “timeprog, ucp, procmsg, FibSecuencial y FibMulti”, generan un ejecutable de sus respectivos archivos “.c” utilizando el compilador GCC.

El comando “All”, es el comando por defecto del makefile, al utilizar esta opción se ejecutan todos los comandos correspondientes para generar los ejecutables del proyecto. Para utilizar esta opción no es necesario colocar “make all” en la consola, sencillamente se coloca “make” en consola y se ejecuta esta opción.

El comando “clean” realiza un comando en consola “rm” que incluye a todos los ejecutables del proyecto, eliminando estos ejecutables.

MESSENGER:

Consiste en escribir un programa en donde un padre crea dos procesos hijos, C1 y C2 y crea un pipe para que C1 se comunique con C2. Programar C1 para que reciba una señal SIGALRM cada 2 segundos. Cada vez que C1 reciba una señal SIGALRM envía el mensaje recibido a C2, el cual lo imprime por pantalla al recibirlo. C1 deberá terminar una vez que haya recibido 10 señales SIGALRM y enviado 10 mensajes a C2. El proceso C2 deberá terminar una vez que haya recibido todos los mensajes y los haya impreso por pantalla. Terminar el proceso padre una vez que ambos hijos hayan terminado.

Para realizar este programa se creó una variable llamada “contador” la cual cuenta las veces que se ha recibido el SIGALRM en el proceso hijo C1, también se creó una variable llamada “flag” para controlar el SIGALRM y se creó

una función llamada “recibirAlarma” la cual cuando se ejecuta señala que se ha recibido un SIGALRM en C1 colocando el flag en 1, al recibir la alarma se incrementa el contador de control de C1 y si no ha recibido ya 10 alarmas, se configura otra alarma dentro de 2 segundos. Posteriormente se crearon dos variables las cuales contendrán a los procesos C1 y C2 , seguidamente se declara el pipe para la comunicación entre los procesos y se muestra en pantalla el proceso padre con su PID , después se creó el pipe utilizando la variable declarada previamente, a continuación se crea el proceso hijo C1 , si hay un error al crear el proceso hijo se termina el programa, si el proceso es el hijo C1 se muestra en pantalla su PID y el de su padre, si el proceso es el padre , dicho padre crea otro proceso llamado C2 , igualmente si hay un error al crear el proceso hijo la ejecución del programa finaliza, si el proceso es el hijo C2 se muestra en pantalla su pid y el del padre.

Ulteriormente si el proceso es el hijo C1 y no el hijo C2 se configura una alarma en dos segundos que generará un SIGALRM el cual al ser recibido por C1 ejecutará la función “recibirAlarma”, seguidamente se crea un ciclo while el cual revisa cada segundo si se ha recibido una alarm , lo cual es notificado por el flag que cambia en la función “recibirAlarma” , si se recibió una alarma se envía el mensaje a través del pipe y se reinicia el flag , cuando C1 haya enviado 10 mensajes se termina el ciclo y se cierra el proceso C1.

Si el proceso es el hijo C2 y no el hijo C1, se inicializa un contador para contar cuantas veces el proceso C2 ha mostrado el mensaje, mientras C2 no haya mostrado el mensaje 10 veces, se ejecuta un ciclo, dentro de dicho ciclo se leerá el pipe de los procesos y cada vez que se reciba el mensaje de C1, C2, lo notificará por pantalla e incrementará su contador de control una unidad.

Si el proceso es el padre, el padre espera hasta que sus dos hijos hayan terminado su ejecución y luego el proceso padre termina su ejecución

TIMEPROG:

Consiste en Escribir un programa denominado timeprog. Este programa recibirá como parámetros un programa (prog) y sus argumentos, para determinar cuánto tiempo demora la ejecución de dicho programa (prog). La sintaxis para timeprog es la siguiente: **timeprog <[arg1, arg2, ..., arg9]>** Los parámetros son opcionales y dependerán del programa prog a ser ejecutado por timeprog. La salida de timeprog deberá ser el número de segundos requeridos durante la ejecución del programa

Para realizar este programa se utilizaron las variables que permiten capturar los parámetros suministrados y el número de estos, para mediante de esos programas ejecutar el programa cuyo tiempo de ejecución va a ser medido y pasarle los argumentos en caso de que los necesite.

Se crearon las variables necesarias para guardar el tiempo en el que inició el programa a ser ejecutado y el tiempo en que terminó el programa a ser

ejecutado. Se declaró también una variable tipo string para guardar en dicha variable el programa a ser ejecutado junto a sus posibles argumentos. Se usó un ciclo “for” el cual recorre el array “argc” el cual contiene el programa a ser ejecutado y los posibles argumentos, este ciclo va uniendo el programa a ser ejecutado y sus argumentos en una sola variable tipo string, y va agregando espacios cuando es necesario. Luego se capturó el tiempo inicial utilizando la función “gettimeofday y se inicia el programa, seguidamente se captura el tiempo de finalización del programa utilizando la función “gettimeofday”, en una variable llamada tiempoTotal se guarda el tiempo que tardó el programa en ser ejecutado y luego se imprime dicho tiempo.

UCP:

Consiste en escribir una versión propia del programa para copiar archivos de Unix, cp.

La sintaxis para UCP es la siguiente: **ucp bufsize file1 file2**

Para realizar este programa en principio se crearon dos funciones, una llamada “error” para capturar los errores generados en el programa y otra llamada “copyFiles” que es la encargada de copiar los archivos a la ruta destino, también se utilizaron el número de argumentos que se pasan en consola y los argumentos, todo esto fue almacenado en sus variables respectivas. Si no hay ningún error en los argumentos suministrados el programa procede a copiar, pero si hay problemas en los argumentos suministrados el programa informa del error y termina su ejecución.

Utilizar timeprog para probar ucp copiando un archivo grande (640K o mayor). Los valores de prueba para bufsize deberán ser: 1, 32, 8192 y 16384. ¿Cuáles son los resultados?

Resultados: Para un archivo .txt de tamaño 1.4 megabytes.

Bufsize =1 → 4.1s

Bufsize=32 → 0.35s

Bufsize=8192 → 0.17s

Bufsize=16384 → 0.08s

FIBONACCI SECUENCIAL:

Consiste en deberá sumar un millón de números de FIBONACCI y generar de forma aleatoria entre 0 y 19 y a ese número le calculamos su FIBONACCI,

generar ese millón de números hacer la suma e imprimirlo por pantalla (usar las siguientes librerías: stdio.h, math.h, stdlib.h y time.h).

Se define randnum(), la cual generara un numero aleatorio en el rango establecido por los parámetros min máx, se creó una función llamada “serieSum” la cual es una función utilizada para calcular la secuencia de Fibonacci, en esta función se generará un número aleatorio de 0 a 19 que se utilizará para calcular la sucesión de Fibonacci utilizando el método del ciclo que irá sumando los números anteriores para calcular la sucesión, a continuación en el método main se crea una semilla para el randomizador utilizando el tiempo actual del reloj local , se inicia el contador del tiempo del procesamiento del programa, luego se ejecuta un millón de veces la sucesión de Fibonacci utilizando un ciclo y dentro de ese ciclo se llama a la función “serieSum” después se finaliza el contador de tiempo de procesamiento del programa y se muestran por pantalla los datos finales del programa.

FIBONACCI MULTIHILOS:

Consiste en usar en paralelo “HILOS” e imprimirlo por pantalla, el objetivo de este ejercicio es observar cual versión correr más rápido (usar función “pthread”).

Se define randnum(), la cual generara un numero aleatorio en el rango establecido por los parámetros min máx, se creó una función llamada “serieSum” la cual es una función utilizada para calcular la secuencia de Fibonacci, en esta función se generará un número aleatorio de 0 a 19 que se utilizará para calcular la sucesión de Fibonacci utilizando el método del ciclo que irá sumando los números anteriores para calcular la sucesión, después se creó una función llamada “Roll” que generará números aleatorios de 0 a 19 y llamará a la función serieSum para que calcule la sucesión de Fibonacci con el número aleatorio generado (esto lo hará 50000 veces), los resultados los irá almacenando en la variable suma, y al finalizar se pasará la variable parámetro de la función "arg", la suma total de los resultados, a continuación en el método main se crea una semilla para el randomizador utilizando el tiempo actual del reloj local, después se inicia el contador del tiempo del procesamiento del programa, se creó la variable “x” donde se almacenará la suma de las sucesiones calculadas, y se crea un array donde estarán las variables para cada hilo y así evitar la condición de carrera, posteriormente se inicializan las variables que utilizaran los hilos en 0 y después se crean 20 hilos que trabajarán en paralelo y calcularán 50000 sucesiones cada uno de ellos utilizando la función roll(void *arg). Luego se espera a que terminen los 20 hilos y se van sumando los resultados obtenidos en la variable de resultados "x", finalmente se finaliza el contador del programa y se imprimen los resultados en pantalla.