UNA
UNIVERSIDAD
NACIONAL
COSTA RICA

Contando Bytes

Proyecto # 1

Jorge Durán Campos
Luis Antonio Montes de Oca Ruiz
Diego Quirós Artiñano

EIF-212 Sistemas Operativos
Profesor Eddy Ramirez Jiménez

18 de abril de 2023

ÍNDICE

I. Resumen Ejecutivo

El presente proyecto trata sobre un problema de manejo de hilos, estos se van a dividir en dos categorías, productores y consumidores. Los productores van a leer un archivo byte por byte y los colocan en un buffer, los consumidores toman los datos del buffer, igual byte por byte, y cuenta las repeticiones de cada byte, colocando la cantidad de repeticiones en un arreglo, en una posición i, donde i representa el byte, es decir si el byte 254 se repite 3 veces, en la posición del 254 del arreglo va a estar el entero 3. Para poder solucionar el problema planteado se implementaron varios mecanismos como los MUTEX (mutual exclusion) [1], wait conditions, y threads ya que es el punto esencial de este proyecto.[2] [3] [4] [5]

Esta solución se implementa por medio de distintos métodos, uno para leer el archivo deseado byte por byte por cada hilo productor, otro para que los hilos consumidores tomen los bytes de los productores y se coloque la cantidad de repeticiones en el arreglo, como se mencionó anteriormente, y el uso del algoritmo de ordenamiento merge sort, ya que, al ser 256 posiciones por ordenar, se consideró que este era la mejor opción. Finalmente, cada uno de estos métodos se llaman dentro del main, junto con las verificaciones necesarias y la ejecución y unión de los hilos productores y consumidores, y se termina con la impresión en pantalla del resultado de las repeticiones por cada byte. Estos metodos se describen a continuacion

El primer metodo es el metodo auxiliar de merge sort, llamado merge() este implementa toda la logica de ordenamiento del merge sort, y luego está el metodo mergesort() que se encarga de unir el resultado en un arreglo auxiliar, utilizando como guia el arreglo de solucion que se genera en el metodo adding_to_array(). Despues se crea el metodo reading_file() el cual es la tarea de los hilos lectores, aqui se hace uso del archivo que se indica al ejecutar el programa y se hace la lectura byte por byte, existen tambien varias verificaciones como si se llegó a final de archivo, se hace uso de bloqueos al buffer de lectura por medio de MUTEX, de flags para saber si se llegó a final de archivo, y tambien de broadcasts para indicar que se terminó la lectura. Por último se encuentra el metodo adding_to_array() el cual es el que se les asigna a los hilos consumidores, este se va a encargar de tomar cada byte introducido en el buffer por los productores, identificar cuantas repeticiones hay de un mimso byte y colocarlo en un arreglo de solucion, en este metodo tambien se manejan verificaciones como que si la posicion de los consumidores es mayor o igual a la posicion de los productores en el archivo, se usa tambien bloqueos pero esta vez en el arreglo de solucion utilizando MUTEX, existen condiciones de espera como la mencionada anteriormente y por ultimo igual se hace un boradcast de que los hilos consumidores terminaron su tarea de procesar todo el buffer. Finalmente, dentro del main, como se mencionó anteriormente, existen verificaciones para la cantidad de hilos productores y consumidores, para la direccion del archivo y la correcta apertura del mismo, si se realizó correctamente la creacion de los hilos y si estos se unieron correctamente al hilo principal, se realiza el ordenamiento merge sort y se presenta en pantalla el resultado de todo el programa, ordenando cada byte de mayor a menor de acuerdo a su cantidad de repeticiones en el archivo indicado, al final se destruyen los MUTEX y condiciones declaradas al inicio del programa

Finalmente, se realizaron varias pruebas de lectura con 7 archivos de distintos tamaños y de distintos tipos, con cuatro distintos casos de prueba, misma cantidad de hilos productores y consumidores, 10 productores 1 consumidor, 1 productor 10 consumidores, y 5 productores 8 consumidores. De estas se obtuvo la conclusión que el mejor caso es cuanto existe una cantidad igual de ambos tipos de hilos, y la peor cuando existen más lectores que consumidores, ya que estos consumidores van a quedar esperando hasta que el único productor haga la lectura de un byte y esto genera un proceso de “back and forth” donde los consumidores deben esperar al productor continuamente.

II. Introducción

Este documento tiene como objetivo servir como documentación del primer proyecto del curso de sistemas operativos. Este consta de un resumen ejecutivo en el cual se resume el proyecto junto a su solución y resultados, un marco teórico donde se describen aspectos relacionados al proyecto, como el leguaje y las bibliotecas, una descripción de la solución implementada al problema planteado en el enunciado del proyecto, los resultados de las pruebas de la solución, las conclusiones donde se analizan los resultados y el proyecto en general, y por último se presentan los aprendizajes obtenidos al realizar el proyecto. Así mismo, el proyecto se basa en realizar un programa lector de cualquier tipo de archivo, sea un video, un texto o demás, si este se sobrepasa de la cantidad de memoria disponible se realiza la lectura poco a poco, este archivo es leído de manera binaria usando hilos "productores" que van byte por byte, estos bytes se entregan a los hilos "consumidores", y van a colocar cada uno de los 256 posibles valores de bytes en un arreglo de 256 espacios, en el que cada índice representa la cantidad de repeticiones de un byte en particular.

III. Marco teórico

El lenguaje utilizado en este proyecto es C, el cual se creó a principios de los años 1970 por Dennis M. Ritchie en Bell Laboratories. Este fue diseñado como un lenguaje minimalista para la creación de sistemas operativos para minicomputadoras, las cuales eran computadoras más baratas y menos potentes que una supercomputadora, pero más caras y potentes que una computadora personal. El principal motivo fue el deseo de migrar el kernel del sistema pronto a ser terminado, UNIX, a un lenguaje de alto nivel, teniendo las mismas funciones, pero con menos líneas de código. C se basó en CPL, o Combined Programming Language, por sus siglas en inglés, el cual a su vez sirvió de base para el lenguaje B. De este, Ritchie reescribió varias funciones de CPL para crear C y después reescribió UNIX en este nuevo lenguaje.[6] [7]

Desde 1977 hasta 1979 ocurrieron distintos cambios en el lenguaje, y durante este tiempo se publicó un libro que sirve como manual para el lenguaje, titulado "The C Programming Language", publicado en 1978 por Ritchie y Brian W. Kernighan. Cinco años después, se estandarizó C en el American National Standards Institute, y desde ese momento, al lenguaje se le refiere como ANSI Standard C. De C salieron varios lenguajes derivados, tales como Objective C y C++. Además, también surgió Java, el cual se creó como un lenguaje que simplifica C.[8]

Continuando con los recursos utilizados en este proyecto, se utilizó la librería pthread. Esta es una librería de POSIX la cual es un estándar para el uso de hilos (threads), en C y C++, los cuales permiten un flujo de procesos de manera concurrente. El uso de estos hilos es más efectivo en procesadores con múltiples núcleos, ya que se pueden asignar los procesos a distintos núcleos, haciendo la ejecución más rápido.[9]

Además, se escogió CLion como el IDE preferido para este proyecto. Este IDE es de la empresa JetBrains y fue lanzado al mercado en 2014[10] con herramientas que ayudan a la creación de código, tales como refactoring, generación de código para sets/gets, terminación de código y arreglos rápidos del código escrito.[11]

Finalmente, ya que se mencionó tanto POSIX como UNIX se va a brindar información sobre estos. Portable Operating System Interface for UNIX, o POSIX, son estándares establecidos por la IEEE y publicados por la ANSI e ISO (International Organization for Standardization), estos estándares permiten el desarrollo de código universal, para que pueda correr en todos los sistemas operativos que implementen POSIX, tales como macOS o Ubuntu, la mayoría de los sistemas basados en UNIX cumplen con POSIX.[12] UNIX es un sistema operativo que fue creado para brindar a programadores funciones simples pero potentes, y que permitiera el uso de múltiples usuarios y de multi tarea, este se compone de tres partes, el kernel, los archivos de configuración de sistema y los programas.[13]

IV. Descripción de la solución

El primer paso de la solución fue crear las distintas variables y definiciones para el programa. Primero los MUTEX para poder sincronizar los hilos productores y consumidores, manejando uno para la lectura de los bytes y otro para el consumo

de estos.[14] [15] Seguidamente, se declaran condiciones de pthread para los productores y consumidores, estos funcionan como indicador de que terminaron su tarea correspondiente.[16] Existe un alias llamado BUFFERLEN el cual se va a utilizar como el tamaño del buffer de lectura para productores, este buffer es un array que se crea con un espacio de 1000 bytes y asimismo se utiliza otro array para la solución con 256 espacios, que representa cada byte posible. También, se crean dos indicadores de posición para los productores y consumidores, que indica la última posición de lectura. Se declara la variable que indica la cantidad deseada de hilos productores y consumidores, la dirección del archivo a probar, el tamaño de este archivo y la posición final de los lectores. Por último, se crea un flag que indica cuando se llega al final de la lectura del archivo.

Se realiza un merge sort para poder ordenar los resultados de mayor a menor repeticiones por cada byte, este merge sort se maneja por un método auxiliar llamado merge() y otro llamado mergesort(), el auxiliar se encarga de toda la lógica de ordenamiento en el auxiliar y el mergesort() se encarga de unir la respuesta dentro de un array final.

Se crea el método reading_file() el cual es el que los hilos productores se encargan de ejecutar. Este va a abrir el archivo indicado en la dirección dada en la ejecución del programa.[17] [18] [19] Por mientras que no se haya llegado al final del archivo, se realiza un bloqueo del mutex para el buffer de lectura de los productores, para que ningún otro hilo productor toque este buffer, si el hilo que bloqueó este mutex se encuentra al final del documento, entonces se va a marcar el flag indicando que se llegó al final del archivo, se desbloquea para asegurarse que ningún otro hilo se quede pegado dentro de un wait condition, si no, se continua con la lectura del byte del archivo, si la posición siguiente del productor es igual que la posición del consumidor entonces se llama un wait condition, utilizando la condición de consumo y el mutex del buffer, esto para detener a los productores hasta que se consuma, si esto no ocurre se sigue normal, se aumenta el indicador de posición de productores, si se llega al fin del documento se desbloquea el mutex del buffer y se indica que se terminó con la señal de condición de read condition con un broadcast, se cierra la lectura del archivo, se marca el flag en 0 indicando que se llegó al final del archivo, y por último se destruyen los hilos productores.[20]

Luego, se crea el método adding_to_array(), el cual es el proceso de los consumidores. Este por mientras que la variable filelen sea 1 se ejecuta toda su lógica, si no se eliminan los threads consumidores. Si esta variable es true entonces se empieza el trabajo de los consumidores, primero se bloquea el mutex del array de la solución, si filelen es 0 entonces se desbloquea el array y se eliminan los consumidores, si no se continua. Además, existe un while donde si la posición de los consumidores es igual o mayor a la posición de los productores y el flag de final de archivo está en true entonces los consumidores esperan a que los productores generen más bytes para leer y estos indiquen cuando terminen, y también hay un if que verifica si el indicador filelen es falso entonces se desbloquea el mutex del array de solución y se eliminan los threads consumidores. Finalmente, si no se cumple ninguna de las condiciones anteriores entonces, como cada posición del array indica el byte correspondiente, se aumenta en 1 la posicion i para indicar la cantidad de repeticiones por cada byte i, esta posición se modifica leyendo cada posición del buffer, ya que este almacena el byte leido, tomando como índice el restante de la división de la posición de los consumidores entre el BUFFERLEN, se aumenta el índice de posición de los consumidores, se disminuye filelen y por último se desbloquea el mutex de la solución, se envía la señal que se terminó de consumir por medio de un broadcast y se eliminan los consumidores.

Para terminar, dentro del main primero creamos un array auxiliar para la solución, donde se ordenan los bytes de mayor a menor repetición del array original de solución. Luego se realizan verificaciones para revisar si se da el número de hilos productores o consumidores deseados o la dirección del archivo a leer. Luego se abre el documento para probar si se logra abrir correctamente, si no se termina la ejecución del programa, y luego se recorre todo el archivo para encontrar su tamaño, se devuelve el puntero que controla lectura y se cierra. Luego se hace la creación de productores con su asignación de la

tarea del método reading file, si falla se detiene el programa, al igual con los consumidores con su método adding to array, si falla también se detiene. Por último, se hace join al hilo principal del programa a los productores y consumidores, si falla se detiene el programa.[21] Seguidamente, se ejecuta el mergesort para ordenar el solution array dentro del solution aux declarado al principio del main, y luego imprimiendo el resultado de las repeticiones, usando ambos arrays de solución, uno para indicar el byte y otro para indicar sus repeticiones, y finalmente se destruyen los mutex y las condiciones.
El primer paso de la solución fue crear las distintas variables y definiciones para el programa. Primero los MUTEX para poder sincronizar los hilos productores y consumidores, manejando uno para la lectura de los bytes y otro para el consumo de estos.[14] [15] Seguidamente, se declaran condiciones de pthread para los productores y consumidores, estos funcionan como indicador de que terminaron su tarea correspondiente.[16] Existe un alias llamado BUFFERLEN el cual se va a utilizar como el tamaño del buffer de lectura para productores, este buffer es un array que se crea con un espacio de 1000 bytes y asimismo se utiliza otro array para la solución con 256 espacios, que representa cada byte posible. También, se crean dos indicadores de posición para los productores y consumidores, que indica la última posición de lectura. Se declara la variable que indica la cantidad deseada de hilos productores y consumidores, la dirección del archivo a probar, el tamaño de este archivo y la posición final de los lectores. Por último, se crea un flag que indica cuando se llega al final de la lectura del archivo.

Se realiza un merge sort para poder ordenar los resultados de mayor a menor repeticiones por cada byte, este merge sort se maneja por un método auxiliar llamado merge() y otro llamado mergesort(), el auxiliar se encarga de toda la lógica de ordenamiento en el auxiliar y el mergesort() se encarga de unir la respuesta dentro de un array final.

Se crea el método reading_file() el cual es el que los hilos productores se encargan de ejecutar. Este va a abrir el archivo indicado en la dirección dada en la ejecución del programa.[17] [18] [19] Por mientras que no se haya llegado al final del archivo, se realiza un bloqueo del mutex para el buffer de lectura de los productores, para que ningún otro hilo productor toque este buffer, si el hilo que bloqueó este mutex se encuentra al final del documento, entonces se va a marcar el flag indicando que se llegó al final del archivo, se desbloquea para asegurarse que ningún otro hilo se quede pegado dentro de un wait condition, si no, se continua con la lectura del byte del archivo, si la posición siguiente del productor es igual que la posición del consumidor entonces se llama un wait condition, utilizando la condición de consumo y el mutex del buffer, esto para detener a los productores hasta que se consuma, si esto no ocurre se sigue normal, se aumenta el indicador de posición de productores, si se llega al fin del documento se desbloquea el mutex del buffer y se indica que se terminó con la señal de condición de read condition con un broadcast, se cierra la lectura del archivo, se marca el flag en 0 indicando que se llegó al final del archivo, y por último se destruyen los hilos productores.[20]

Luego, se crea el método adding_to_array(), el cual es el proceso de los consumidores. Este por mientras que la variable filelen sea 1 se ejecuta toda su lógica, si no se eliminan los threads consumidores. Si esta variable es true entonces se empieza el trabajo de los consumidores, primero se bloquea el mutex del array de la solución, si filelen es 0 entonces se desbloquea el array y se eliminan los consumidores, si no se continua. Además, existe un while donde si la posición de los consumidores es igual o mayor a la posición de los productores y el flag de final de archivo está en true entonces los consumidores esperan a que los productores generen más bytes para leer y estos indiquen cuando terminen, y también hay un if que verifica si el indicador filelen es falso entonces se desbloquea el mutex del array de solución y se eliminan los threads consumidores. Finalmente, si no se cumple ninguna de las condiciones anteriores entonces, como cada posición del array indica el byte correspondiente, se aumenta en 1 la posición i para indicar la cantidad de repeticiones por cada byte i, esta posición se modifica leyendo cada posición del buffer, ya que este almacena el byte leído, tomando como índice el restante de la división de la posición de los consumidores entre el BUFFERLEN, se aumenta el índice de posición de

los consumidores, se disminuye filelen y por último se desbloquea el mutex de la solución, se envía la señal que se terminó de consumir por medio de un broadcast y se eliminan los consumidores.

Para terminar, dentro del main primero creamos un array auxiliar para la solución, donde se ordenan los bytes de mayor a menor repetición del array original de solución. Luego se realizan verificaciones para revisar si se da el número de hilos productores o consumidores deseados o la dirección del archivo a leer. Luego se abre el documento para probar si se logra abrir correctamente, si no se termina la ejecución del programa, y luego se recorre todo el archivo para encontrar su tamaño, se devuelve el puntero que controla lectura y se cierra. Luego se hace la creación de productores con su asignación de la tarea del método reading file, si falla se detiene el programa, al igual con los consumidores con su método adding to array, si falla también se detiene. Por último, se hace join al hilo principal del programa a los productores y consumidores, si falla se detiene el programa.[21] Seguidamente, se ejecuta el mergesort para ordenar el solution array dentro del solution aux declarado al principio del main, y luego imprimiendo el resultado de las repeticiones, usando ambos arrays de solución, uno para indicar el byte y otro para indicar sus repeticiones, y finalmente se destruyen los mutex y las condiciones.

V. Resultados de pruebas

Para las pruebas se utilizó un equipo con un procesador Intel Core i7-8550U, el cual corre a una velocidad base de 1,8 GHz y un turbo hasta 4,00 GHz, con 4 cores y 8 threads, y cuenta con 8 GB de RAM.

Estas pruebas se realizaron con 7 archivos de distintos tamaños y tipos, dos archivos txt, dos PDF, una foto de 24.5 megapíxeles y por último dos videos, uno 4K HDR y otro en 8K. Además, por cada archivo se realizaron cuatro casos de prueba, un hilo para productor y consumidor, 10 productores 1 consumidor, 10 consumidores 1 productor, y 5 productores 8 consumidores.

La primera prueba realizada fue la lectura del txt más pequeño de 33 bytes. Este se leyó con el programa en los siguientes tiempos 0.002s, 0.003s, 0.002s, y 0.003s para cada caso correspondiente, aquí se puede notar que al ser un archivo sumamente pequeño no hay mucha diferencia de tiempo entre cada caso. Seguidamente, se realiza la lectura de otro archivo txt de 7.4 kB, los tiempos respectivos de cada prueba fueron 0.011s, 0.018s, 0.129s, y 0.062s. Se logra percibir una diferencia en el caso donde existen 10 consumidores y 1 productor que tardo mas tiempo que el resto de las pruebas, seguido del caso de 5 productores 8 consumidores.

Luego se usó un PDF de 152.7 kB. Sus tiempos por cada caso fueron 0.193s, 0.276s, 2.371s, y 1.191s. Aquí ya se puede observar un patrón donde el caso de igual cantidad de productores y consumidores es el mejor y el peor es más consumidores que productores.

Se continua con la lectura de otro PDF de 4.2 MB. Sus tiempos respectivos fueron 5.253s, 16.084s, 2m 48.18s, y 1m 14.46s. Se confirma la observación realizada en las pruebas anteriores, siendo el mejor caso la cantidad igual de productores y consumidores que produjo el menor tiempo.

A continuación, se realizó la prueba con la fotografía con un tamaño de 7.6 MB, dando los siguientes tiempos 12.317s, 32.694s, 5m 4.31s, y 2m 18.13s. Confirmando una vez más que el mejor tiempo se realizó con la misma cantidad de productores y consumidores, siendo el peor cuando hay 10 consumidores y 1 productor.

Por último, ya que el equipo ha demostrado durar bastante tiempo en correr los archivos más pesados de las pruebas anteriores, y, además, se logró confirmar que el mejor caso es cuando existe la misma cantidad de productores y consumidores, ambos videos se van a correr usando este caso para dar una idea del tiempo que podría durar en los peores casos. El tamaño es 437.8 MB para el video 4K HDR y 936,6 MB para el video 8K. El video 4K HDR se leyó en un tiempo de 17m 01.58s y el 8K con un tiempo de 38m 19.41s, si el promedio de diferencia entre el mejor y peor caso es un aumento del

tiempo de 30 veces, entonces se podría aproximar que el peor caso para cada video duraría 8 horas para el video en 4K y 16 horas para el video en 8K.

VI. Conclusiones

Se puede concluir que, por medio de los resultados anteriores, lo mejor es manejar la misma cantidad de hilos productores y consumidores, ya que cualquier espera que pueda ocurrir por cualquiera de los hilos se reduce a casi cero, por que conforme los productores colocan bytes en el buffer, los consumidores los procesan, y, como se pudo observar en el peor caso, cuando hay más consumidores que productores, estos se quedan esperando por bytes para consumir ya que al ser mayor la cantidad de consumidores, el procesamiento de estos se hace mucho más rápido que la colocación de bytes en el buffer por parte de los productores y esto genera varias esperas de los consumidores para poder hacer su trabajo.

Además, también se pudo observar que el manejo de hilos en un proyecto ayuda a que las tareas sean ejecutadas con mayor rapidez que si se manejara un único hilo principal, ya que a estos hilos se les asignan tareas para que sean ejecutadas en paralelo o de manera concurrente.[4]

Finalmente, los resultados anteriores se podrían mejorar haciendo uso de hardware más potente y moderno, principalmente utilizando un procesador con mayor cantidad de cores y threads, ya que este recurso sería mucho más rápido en el manejo de hilos paralelos y de ejecución de sus tareas.

VII. Aprendizajes

En esta sección se muestran los aprendizajes de cada integrante del grupo de trabajo.

Jorge Durán Campos: Aprendí que usando toda la capacidad de procesamiento de un multiprocesador actual se pueden realizar procesos de manera más rápida haciendo uso de hilos, debido a que estos amortiguan la carga de un proceso entre los diferentes núcleos, y de esta forma conseguir una disminución de tiempo de ejecución proporcional a la cantidad de núcleos utilizados para este fin. También me ayudó a complementar, junto con la teoría, la importancia de la sincronización entre hilos.

Luis Antonio Montes de Oca Ruiz: De este proyecto aprendí sobre el manejo de hilos y como estos ayudan a mejorar la ejecución de un programa, dividiendo las tareas en cada hilo y así mejorando el tiempo de ejecución y el uso de recursos compartidos. También el manejo de exclusiones MUTEX que permiten un manejo correcto de la región crítica y recurso compartido entre los distintos hilos, siendo esto una parte sumamente importante de este tipo de ejecución, ya que ocurren errores importantes si no se maneja de manera correcta, como la perdida de datos o ciclos de espera de un hilo por otro.

Diego Quirós Artiñano: Aprendí como implementar la teoría que vimos en clase. El proyecto me ayudó a entender en mayor detalle cómo es que los hilos corren durante su ejecución. Por ejemplo, a pesar de que son paralelos en teoría, en la práctica por la exclusión mutua entre los hilos el programa ejecuta de manera secuencial. Comprendí la importancia de los semáforos y la exclusión mutua. Además, entendí el uso de la función de cond_wait(), y la diferencia entre el cond_signal() y cond_broadcast().

REFERENCIAS

[1] WhileTrueThenDream, "Mutex. sincronización de hilos. programar en c, linux," 04 2020.
[2] S. Khalid, "What are pthreads in C/C++?." [Online; accessed 7. Apr. 2023].
[3] CodeVault, "Short introduction to threads (pthreads)," 12 2020.
[4] P. Courses, "Introduction to threads (pthreads) — c programming tutorial," 02 2022.
[5] WhileTrueThenDream, "Programar en c con hilos pthreads," 03 2018.
[6] E. Britannica, "C — computer programming language," 10 2022.
[7] D. Munoz, "After all these years, the world is still powered by c programming."
[8] D. M. Ritchie, "The development of the c language," 04 1993.
[9] G. Ippolito, "Linux tutorial: Posix threads," 2020.
[10] A. Avram, "Jetbrains clion, a c/c++ ide, and resharper for c++," 09 2014.
[11] JetBrains, "Intelligent coding assistance code analysis - features — clion."
[12] U. I. T. Services, "About posix," 12 2021.
[13] I. S. University, "What is unix?," 02 1997.
[14] cppdev, "c - pthreads mutex vs semaphore," 01 2010.
[15] ManRow, "c - static pthreads mutex initialization," 02 2011.
[16] T. O. Group, "pthread$_{c}ond_{w}ait$,$''$ 1997.
[17] ChaniLastnamé, "C - read file byte by byte using fread," 02 2015.
[18] cppreference.com, "fread - cppreference.com," 11 2021.
[19] M. Kerrisk, *fseek(3) - Linux manual page*. The Linux Programming Interface, No Starch Press, 10 2010.
[20] M. Kerrisk, *pthread*$_{e}xit(3).TheLinuxProgrammingInterface, NoStarchPress$, 102010.
[21] M. Kerrisk, *pthread*$_{j}oin(3).TheLinuxProgrammingInterface, NoStarchPress$, 102010.