La implementación de estructuras de hilos de usuario en un sistema operativo didáctico

Hugo Ryckeboer, Nicanor Casas, Graciela De Luca, Martín Cortina, Gerardo Puyo, Waldo Valiente
{hugor,ncasas,gdeluca,mcortina,wvaliente}@unlam.edu.ar

Abstract. This is the first attempt of the team to the threads implementation in an operative system with didactic features. In this work we specify the limitations we experienced without going into the analysis of the different threads types. We are also showing here the way the user-level threads structure was plannified with different types of blocked and unblocked sentences, system calls and their problems, the structure of the threads table and the used commands and the reasons why the Kernel–level threads implemenation was not achieved.

Keywords: User-Level Thread, System Calls, Blocked Sentences, Unblocked Sentences, Kernel-Level Threads

Resumen. Este es el primer acercamiento, del equipo, a la implementación de hilos en un sistema operativo de características didácticas. Especificamos en este trabajo las limitaciones a las que tuvimos que someternos sin ingresar al análisis de los diferentes tipos de hilos. Presentamos aquí la forma en que se planificó la estructura de hilos de usuario con diferentes tipos de sentencias, bloqueantes y no bloqueantes, los envíos de señales y sus problemas, la estructura de la tabla de hilos y los comandos utilizados y el por que no se llegó a la implementación de hilos de kernel.

Palabras: Hilos de Usuario, Llamadas al Sistema, Sentencias Bloqueantes, Sentencias No Bloqueantes, Hilos de Kernel

1 Introduction

Existen diferentes argumentos por los cuales programar hilos y podemos enumerarlos de acuerdo a nuestro interés. Un argumento valedero es que representa un desafío, a llevar a cabo en la implementación de un sistema operativo de características didácticas, la implementación de hilos. Desde un principio el sistema operativo SODIUM se basó, como la generalidad de los sistemas operativos, en un solo hilo generando el correspondiente overhead [07].

Otro argumento es poder ver la habilidad de las entidades de descomponer una aplicación en varias entidades que se pueden ejecutar cuasi en paralelo [06].

Otro argumento es el de que los hilos son más rápidos en los cambios que los procesos comunes [05].

Hay diferentes definiciones sobre los tipos de hilos y por lo tanto del manejo de los mismos, generadas por los desarrolladores de sistemas operativos comerciales, lo que genera una marcada confusión, y por ende, algunos errores de interpretación, por lo que en este informe abocamos todos los esfuerzos en un solo tipo de hilo dejando en

claro que el resto de los diferentes tipos no se expondrán en este informe debido a los alcances del mismo.

Como en todo desarrollo es importante tener en cuenta las restricciones que aparecen, en lo referente a la arquitectura de que se dispone, por lo tanto se establece que todos los desarrollos se deben apuntar a máquinas con un procesador INTEL y palabra de 32 bits [01] [02] [03], que corresponde al parque de máquinas que se encuentran en el laboratorio de la facultad y que es donde se realizan las pruebas de funcionamiento.

Es muchas veces fácil explicar la composición de un hilo de usuario o un hilo de kernel separándolo de la aplicación específica [04] [05] [06] pero analizándolo desde el punto de vista de una aplicación, en este caso un sistema operativo, no es fácil determinar los diferentes tipos de recursos que se necesitan para su implementación.

2 Estableciendo el marco operativo. Discusiones

El primer paso fue establecer los lineamientos necesarios para llevar a cabo nuestra propuesta y seguir avanzando con el desarrollo del SODIUM; establecer el marco operativo en el que se debe basar el sistema operativo. Esto debería incluir los límites y alcances de programación incluyendo los de la arquitectura.

La primera discusión fue establecer qué tipo de definición de hilos, qué conceptos serían válidos para nosotros, dado que las mismas no abundan. Cuando se habla del concepto de hilo [06] lo que se da es un ejemplo y no una definición lo que hace difícil entender el concepto. Por tal motivo hemos tomado la definición de Ralf S. Engelshall diseñador de GNU Portable Threads [08]. Para esto tuvimos que tener en cuenta los componentes más importantes de un proceso, que son: la estructura de datos en la cual figura la asignación de memoria del proceso, el manejo de señales, el conjunto de descriptores de archivos y el contexto de máquina. El contexto de máquina incluye los registros de CPU, el contador de programa y el puntero al stack [02]. A este contexto de máquina es lo que podríamos denominar el hilo de ejecución o hilo [06], el cual usualmente comparte todos los atributos con el proceso pesado menos el contexto de máquina.

La segunda discusión fue qué cantidad de hilos se generarían en una biblioteca usuario, debido a las diferencias, mínimas, entre la cantidad de hilos de usuario que LINUX permite y nuestra realidad y la de los procesadores IA-32.

En las primeras versiones de Linux se producía un desborde y el sistema operativo seguía creando hilos por un lado y eliminándolos por otro. En las actuales versiones se permite un máximo de 8192 hilos total, produciendo un marcado debilitamiento del hardware. En consecuencia se estipuló que en primera instancia la cantidad total de hilos que permitirá el sistema será de 4095 más uno para el manejador de hilos, lo que corresponde exactamente a la mitad de los que permite Linux.

La tercera discusión fue qué cantidad de hilos debería generar el sistema operativo para atender los hilos generados por una aplicación teniendo en cuenta la relación de SMP [06]. Para ello se tomó en cuenta los diferentes tipos de hilos que se pueden generar en un sistema de cómputos y se estableció que la relación será de varios a uno [06] [07] para facilitar la programación de los mismos, teniendo en cuenta que en un

soporte de hilos a nivel del kernel, los procesos no son entidades planificables, sino que son los hilos las entidades planificables y los procesos se convierten en contenedores lógicos para los hilos [10]. Como el sistema operativo SODIUM posee únicamente hasta el momento procesos pesados y diferentes tipos de planificadores configurables, no reconociendo la existencia de hilos y dadas las dificultades que representa para los alumnos la comprensión de los algoritmos de planificación de los hilos de usuario, se decidió como primera aproximación el diseño de hilos a nivel usuario. Para esto se necesita crear una biblioteca en el entorno del proceso usuario que permita las operaciones esenciales para el manejo de hilos, tales como creación, finalización, cambio de contexto de hilo, administración de la cola de planificación de hilos con su algoritmo correspondiente, siendo el cambio de contexto una de las operaciones críticas en la mejora de perfomance y la visualización del estado de las estructuras en el momento del context switch tanto de los hilos como la de los procesos pesados lo que permitirá a los alumnos reconocer el estado de los hilos y de los procesos, pudiendo diferenciarlos fácilmente.

El trabajo comenzó con un profundo análisis del funcionamiento de los hilos de usuario a través de la biblioteca y estructurarlo a espaldas del sistema operativo, es decir, sin que éste trabajara con su contexto. Crear la estructura de una biblioteca para la administración de hilos con el código suficiente como para crear, destruir, y fundamentalmente planificar la ejecución de los hilos; permitir que los hilos generen a su vez otros hilos y permitir que el hilo padre aporte su estructura de datos al nuevo.

La decisión es trabajar con estructuras del tipo system calls reducidas, almacenadas en una tabla diferente y estructuradas sin perjuicio de las verdaderas llamadas al sistema, por lo tanto los nombres que se utilizarán para las llamadas entre hilos no tendrán una estructura POSIX sino una estructura SODIUM.

La planificación de hilos abarcará algoritmos non preemptive y premeptive, en principio, pero a diferencia de los procesos sólo se incluirán dos algoritmos del primer planificador, que serán FIFO y PRIORIDADES.

Para el planificador preemtive se programará un algoritmo ROUND ROBIN, dejando la posibilidad de generar otros algoritmos planificadores para el futuro. Este último se incluye para poder analizar el comportamiento de la biblioteca y del sistema operativo con las especificaciones conflictivas de la terminación conjunta de cuantos (QUANTUM o TIME SLICE) y la resolución de prioridades entre ambos. Es necesario especificar que a prima facies, el equipo tiene el concepto de poca operatividad y complejidad de operación para esta planificación pero por otro lado una gran expectativa para el análisis del comportamiento teniendo en cuenta que el sistema permitirá variar la prioridad de uno con respecto al otro.

Los algoritmos enunciados anteriormente tienen una aplicación diferente si se incluyen en su ejecución sentencias que bloqueen o no al proceso que se está ejecutando. Estos puntos son los que trataremos a continuación.

3 Planificación de la estructura para el manejo de hilos

En términos generales los procesos requieren de sentencias de entrada/salida ya sea para interactuar con algún dispositivo o para imprimir en pantalla.

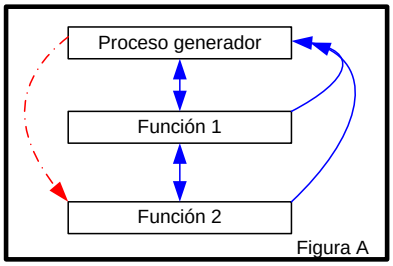
Estas sentencias tienen, por necesidad, conexión con llamadas al sistema; porque se producirá una espera del proceso y por consiguiente la no continuación del resto de los hilos que este proceso haya generado. Esta estructura es la que aparece en todos los libros correspondientes a la materia Sistemas Operativos [04] [05] [06].

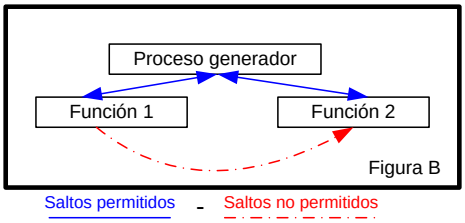
Estamos concientes que esta forma de estructurar la biblioteca de hilos es un avance importante en el desarrollo del sistema operativo, no sólo por el rendimiento de los procesos, sino porque sirve de una gran experiencia para generar los programas necesarios para su tratamiento.

Otra definición importante es que el tratamiento de segmentos de código de dos o más hilos se ejecutará por medio de saltos (JUMP) largos. Un proceso podrá direccionarse a cualquier hilo por él generado pero en principio se tienen las siguientes restricciones:

a) Un proceso no podrá realizar un salto largo a un hilo no generado directamente por él. Si el hilo fue generado por otro hilo se deberá seguir la línea de descendencia para el pase de datos pero si se permitirá que cualquier función pueda pasar información al proceso generador. (Figura A)
b) Realizar un salto largo de un hilo para pasarle el control a otro hilo que no sea de la misma familia, generado por el hilo, como se marca en la en la figura B.

Nótese que la línea roja (o línea cortada) muestra la restricción mencionada





Esta limitación, en primera instancia, se tomó para facilitar el trabajo de programación de los alumnos y en segunda instancia para poder seguir mejor el comportamiento de los hilos y su planificación

4 Análisis de System Calls.

El uso de System Calls en hilos de usuario puede generar el problema de bloqueo en los otros hilos [15], debido a que el kernel sólo reconoce al proceso si la system call realiza la relación con el file system o con el sistema de entrada/salida, el kernel podrá bloquear al proceso en ejecución o no.

El uso de sentencias bloqueantes en hilos dentro del plazo del cuanto de tiempo unificará a los tres algoritmos que se enunciaron anteriormente, independientemente

de ser apropiativos o no. El kernel bloquea y reanuda la ejecución del hilo a nivel de kernel, sin notificación al hilo a nivel de usuario.

4.1 Análisis aplicando sentencias bloqueantes.

Esta estructura es la de más fácil comprensión debido a que no existe la necesidad de controlar a los otros hilos debido a que cuando se llama a una system call todo el proceso se detiene. Cuando una aplicación requiere un servicio del kernel, esto es hecho mediante una llamada de sistema, éstas proporcionan la interfaz necesaria entre las aplicaciones a nivel de usuario y el kernel. Ellas se usan a menudo cuando la aplicación requiere los servicios del hardware del sistema subyacente, como el reloj interno o un dispositivo de entrada/salida. El requerimiento y devolución de un resultado de los servicios del sistema operativo son realizados según la definición de la llamada de sistema.

La llamada que realiza el hilo, necesariamente debe devolver un resultado para que el hilo pueda continuar, tal es el caso de un Read en un archivo, el proceso debe ser bloqueado hasta obtener el resultado requerido habilitado para su uso, por lo tanto el hilo que solicitó la system call y todos los otros hilos de este proceso serán bloqueados. En el caso de otras entradas-salidas, donde no se necesita bloquear el proceso, podría llegar a suceder que el dispositivo esté ocupado, y se deba bloquear el proceso hasta que se desocupe el dispositivo, obteniendo lo mismo que en el ejemplo anterior todos los hilos a nivel usuario del proceso serán bloqueados.

Otra situación que se analizó es el caso del uso de semáforos para generar regiones críticas donde proteger las modificaciones a variables globales, esto también requiere que los hilos generado por el usuario sean bloqueados por la biblioteca correspondiente.

Aquí nos encontramos con las variables locales que se transforman en variables globales. Cuando una variable es generada por el proceso padre, puede convertirse en una variable global sobre la que se deban utilizar semáforos para salvaguardar resultados debido a que los hilos producen una modificación continua de la misma. Esto en primera instancia no se había tomado en cuenta y recién se analizó cuando se presentó el problema.

4.2 Funciones aplicando sentencias no bloqueantes.

Esto realmente es un desafío, porque implica realizar un mini sistema operativo en base a los diferentes cambios de contexto entre hilos. En estos casos las llamadas al sistema no ayudan para una planificación estándar. Una de system call no bloqueante o asincrónica retorna sin esperar que ésta se complete, por ejemplo una system call solicitando una entrada/salida, tal es el caso del teclado y del mouse. Para poder mejorar el rendimiento de los hilos de usuario, debemos realizar la mayor cantidad de system calls posibles en forma asincrónica. Esto hará que la perfomance del proceso mejore, ya que cuando un hilo no puede ejecutar otro continúa.

4.3 Implementación de System Call bloqueantes en forma no bloqueante

Para poder implementar esto en el Sistema Operativo **SODIUM**, proponemos diseñar una biblioteca para el manejo de hilos donde el kernel envía una interrupción a la biblioteca de hilos para avisarle del comienzo y la finalización de la entrada y poder replanificar al hilo que solicitó la system call; de ese modo no necesitaríamos bloquear el proceso, ya que la biblioteca de hilos bloquearía al hilo solicitante de la system call y podría pasar otro a ejecución. Cuando recibe la interrupción avisando de la finalización de la system call, pondrá al hilo como listo para ejecutar en la cola de planificación de la biblioteca. Todos estos cambios de estado se producirán sin bloquear el proceso. Hay varias soluciones más a implementar y analizar cuando se realice la interface entre los hilos de usuario y los de Kernel. Una de las propuestas de los diseñadores del sistema Psyche es realizar una estructura de datos a la que denomina ***Procesador Virtual*** encargada de la relación entre el hilo a nivel de Kernel y los Hilos a nivel de usuario implementada en el espacio de direccionamiento del usuario junto con memoria compartida entre el kernel y la biblioteca de usuario. Esto le permite el manejo de interrupciones, señales, timers y planificación entre otros [09]. Otra propuesta es la de Dean [10] que utiliza un objeto denominado ***Continuation*** que describe totalmente una ejecución a futuro, Este objeto puede ser creado, invocado, y pasado como argumento a otros objetos similares. El primer tipo de Continuation describe el estado del hilo de usuario mientras ejecuta una rutina de hilos en C y se construye cuando el hilo cruza el límite de protección desde el espacio de usuario (nivel 3 en nuestro caso) al modo kernel (nivel 0). El otro Continuation se usa internamente para describir la ejecución que debería ocurrir cuando un hilo se reanuda después de un bloqueo.

5 Análisis del envío de señales.

Sabemos que la implementación de señales en el sistema operativo Linux tiene una serie de problemas y nuestra mayor preocupación fue que los mismos no se trasladaran a nuestro desarrollo ya que toda la base de programación se realiza sobre ese sistema operativo.

Lo primero que buscamos fue no recargar al hardware y es por ese motivo que se redujo la cantidad de hilos que puede controlar una biblioteca, de esa manera diminuiría la posibilidad de stress.

En nuestro caso al llegar a los 4096 hilos de usuario el proceso es abortado siguiendo la característica del comando específico.

Otro problema en el que se pensó es en el envío de señales internas que no se interpusieran con el envío de señales manejadas por el Kernel del sistema operativo. Esto hizo que se generara un nuevo paquete de señales que resulte eficaz para la eliminación de hilos que se encuentren en actividad o en estado de espera o bloqueo.

6 Estructura de la TCB (thread control block)

La estructura de la TCB es de una extensión menor que la correspondiente a la PCB pero no es la estructura estándar que se menciona generalmente en los libros de la materia.

Los campos que la forman son los que a continuación se mencionan.

TID (Thread Identification). Identificador del hilo o nombre del mismo. Es un número secuencial. El número comienza con el número de día que se está generando el hilo.

TID_PARENT (Parent Thread Identification). Identificador del padre siempre y cuando el que lo haya generado sea el otro hilo. Debe permanecer en blanco si el hilo fue generado por el proceso. La información en este campo permitirá saber si quien quiere eliminarlo tiene el permiso suficiente para tal función

PROC_PARENT (Process Parent). Nombre del proceso padre. Es un campo obligatorio y siempre debe estar informado. Esto identifica la biblioteca en la cual se está trabajando.

NEXT_TCB (Puntero a la próxima TCB). Permite el ordenamiento de los hilos para su ejecución de acuerdo a la planificación elegida.

STATUS (Estado del hilo). Los estados permitidos para un hilo son:

A: activo

E: en ejecución

W: en espera o demora

B: bloqueado por propia voluntad

REGISTERS (Registros del hilo). Estado de los registros del procesador en el momento de ejecutarse un cambio de contexto entre hilos.

TIMESTAMP (Día y hora). Establece el día y la hora de creación del hilo.

PRIORITY (Prioridad). Establece la prioridad inicial del hilo. Esta prioridad puede ser modificada por el proceso o por otro hilo siempre que él mismo haya sido el generador del hilo que se va a modificar. No hay que confundir a esta prioridad como una prioridad interna que se puede modificar, como lo hace la sentencia pnice de Unix, a través del sistema para cumplimentar el progreso en el protocolo de sincronización.

7 Comandos habilitados

Queda establecido que la biblioteca de hilos a nivel de usuario creará y mantendrá el estado de los hilos y que la mayor parte de la planificación ocurrirá en el espacio de usuario. Asumimos que el Kernel se encarga del resto de la alocación y protección de recursos.

La mayor tarea de la biblioteca es crear y despachar el contexto de máquina, en la práctica la tarea que le sigue en importancia, es asegurar que ningún hilo se bloquee por accidente. Cuando un proceso desea bloquearse, la biblioteca deberá poder suspender la ejecución del hilo actual y despachar uno de los restantes hilos.

Se han definido los comandos para el manejo de la biblioteca de hilos. Esta estructura básica puede ser ampliada en el futuro a la luz de las necesidades existentes

en el desarrollo del sistema operativo. Es importante recordar que el sistema operativo es desarrollado por alumnos de la carrera de Informática en la materia Sistemas Operativos y que esto sólo es la base de lo solicitado teniendo ellos la posibilidad de agregar o quitar comandos de acuerdo a sus necesidades de programación.

Por otro lado desde el punto de vista del aprendizaje cada biblioteca de hilos deberá ser única en el sistema para ese proceso.

Para ello el nombre de la biblioteca de hilos deberá ser de la siguiente manera: Número de proceso + "THLIB", donde el número de proceso corresponde a la ID del mismo y THLIB es la combinación de THRAD y LIBRARY.

Esto significa que si la biblioteca no fue vaciada en su totalidad la misma quedará bajo la tutoría del INIT de la misma manera que los procesos, pero sólo al efecto de poder obtener resultados estadísticos.

Los comandos básicos solicitados son los siguientes

Crear_hilo: Permite al proceso generar un hilo usuario de tal manera que el mismo se encuentre limitado al espacio de la biblioteca de hilos.

La creación de hilos no puede ser especificada también a través de una sentencia EXEC__, la cual ocasionaría la eliminación del padre y debería ser ejecutada en el modo Kernel. En este caso específico se deberá cancelar al proceso que quiera ejecutar esta sentencia. Para el sistema operativo SODIUM pasa a ser una sentencia inválida.

También será inválida la creación de hilos a través de la sentencia CLONE.

Crear_hilo_hilo: Permite a un hilo crear otro hilo. Siempre determina una estructura de árbol.

Eliminar_hilo: Permite al proceso eliminar un hilo creado por el mismo o por cualquiera de su descendencia. En el caso de hilos creados por otros hilos sólo será permitida la eliminación del mismo, en el caso de hilos, por su creador.

Eliminar_todo: Permite al proceso eliminar a todos los hilos creados por él o por otros hilos. Es una estructura conveniente para vaciar la biblioteca de hilos antes del cierre del proceso y verificar si la generación de hilos no fue realizada en exceso. Permite realizar un listado de los procesos en actividad que no han sido eliminados por el creador o no han terminado por sí mismos al concluir la tarea encomendada.

Demorar: Permite a un hilo bloquearse hasta que sea liberado por otro hilo, por el mismo proceso o por el reloj del sistema. Esta estructura genera variables globales de familia de tal manera que permita una comunicación entre generador y la descendencia generada para permitir al primero sacar del estado de hibernación a un hilo. Tiene como restricción que las variables de familia deberán ser declaradas en el proceso y no en los hijos. Esta función es muy importante porque permite el bloqueo de hilos sin intervención del sistema operativo dado que no se utilizan las system calls del mismo.

Demorar_tiempo: Permite a un hilo dormirse por un espacio de tiempo que el mismo establece. El reloj disparado por esta funcionalidad despertará al proceso en su momento. Un hilo que se encuentre en esta situación igualmente podrá ser eliminado siguiendo las especificaciones y restricciones establecidas en el punto correspondiente.

Cambiar_prioridad: Permite modificar la prioridad original del hilo. Esta instrucción es válida sólo si es invocada por el proceso o por el hilo generador.

8 Conclusiones

En principio el sistema operativo SODIUM consta de un único hilo de ejecución, o sea, un proceso un hilo. Con la implementación de la biblioteca de hilos a nivel de usuario estaríamos contando con el mismo tipo de modelo. Esto nos proporciona una mayor seguridad porque no tenemos que tocar el Kenel ya desarrollado.

En esta primera etapa los hilos nos permitirán poder mejorar las condiciones para la concurrencia, poder entender como funciona la planificación de hilos de usuario, se podrá visualizar el estado del PCB y de las TCB de los hilos en cada cambio de contexto mostrando el contenido correspondiente, también la cola de listos de hilos y las diferentes colas de los procesos.

Lo anterior permitirá posteriormente hacer comparaciones estadísticas sobre la performance de las tareas realizadas con procesos y las realizadas con hilos, gracias a la posibilidad de visualización durante los eventos del SODIUM.

Nos falta realizar la investigación profunda de los hilos de kernel y los LWP para, posteriormente desarrollar los hilos a nivel de Kernel, entonces SODIUM dejará de planificar procesos y pasará a planificar hilos de Kernel, con lo cual pretendemos poder implementar el modelo M:N con M>N.

References

[01] Angulo, J. M., Funke, E. M.; *Microprocesadores Avanzados 386 y 486 Introducción al PENTIUM y PENTIUM PRO,* Editorial Paraninfo, Cuarta Edición, 1998

[02] Brey B., *Los Procesadores Intel,* Prentice Hall, Quinta edición, 2000

[03] IA-32 *Intel Architecture Software Developer's Manual*, Volume 3: System Programming Guide, 2003.

[04] Silverschatz, A., Galvin, P., Gagne, G.; *Operating System Concepts,* Addison-Wesley Longman, Seventh Edition, 2005

[05] Stallings, W.; *Operating Systems Internals and design principles,* Prentice Hall International, Fifth Edition, 2005

[06] Tanenbaum A.S., *Modern Operatig System,* Prentice Hall, Third Edition, 2008

[07] Dijkstra E. W.,Cooperating *Sequential Processes*. 'Programming Ralph S. Languages', Genuys, F. (ed.), Academic Press, 1965.

[08] Engleschall, R. S.; *GNU Portable threads*, 1999; http://gnu.org

[09] Brian D. Marsh, Michael L Scott, Tomas J LeBlanc, Evangelos P. Marlatos. *First-Class User-Level Threads.* Computer Science Department Computer University of Rochester. 1991

[10] Randall D. W.; *Using continuation to build a User-Level Threads Library*. Cool of computers Science. Carnegiue Mllion University. Pittsburgh. Pubilshed in the proceedings of Usenix Match III Symposium. Santa Fe, New Mexico. 1993

[11] Pinkert J. & L.Wear Operating Systems: Concepts, Polices and Mechanisms, Englewood Cliffts NJ, Prentice Hall, First Edition, 1989

[12] Standard Microsystems Corporation *Application note 6.12,* 2000

[13] Turley J. *Advanced 386 Programming Techniques,* Editorial Osborne McGraw-Hills, Primera Edición, 2004
[14] Hans-J Boehm. *Threads Cannot Be Implemented As a Library.* HP Laboratories Palo Alto. 2005
[15] Jamie Cameron, Managing Linux Systems with Webmin: System Administration and Module Development
[16] Jasmin Blanchette, Mark Summerfield, C++ GUI Programming with Qt 3
[17] Rafeeq Ur Rehman, Christopher Paul, The Linux Development Platform: Configuring, Using, and Maintaining a Complete Programming Environment