Cooperación entre la aplicación y el kernel para la planificación de flujos, en sistemas multiprocesadores, como soporte al paralelismo

Marisa Gil Gómez

Mayo de 1994

Departament d'Arquitectura de Computadors - Universitat Politècnica de Catalunya

Trabajo presentado para la obtención del título de Doctor en Informática por la Universitat Politècnica de Catalunya y dirigido por el Doctor José Ignacio Navarro Mas

"Cuando yo uso la palabra -insistió Humpty Dumpty en tono desdeñoso-
quiere decir lo que yo quiero que diga, ni más ni menos.
-La cuestión es -insistió Alicia- si se puede
hacer que las palabras signifiquen cosas diferentes."

Lewis Carroll: "A través del espejo"

Este trabajo ha sido subvencionado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), bajo los contratos TIC 89/392 "Arquitectura, Herramientas de Programación y Sistemas Operativos para Multiprocesadores" y TIC94-0439 "Cooperación entre el microkernel y las aplicaciones para explotar el paralelismo en sistemas multiprocesadores".

Agradecimientos

A través de este trabajo que presento como autora, está también plasmado el esfuerzo y la investigación de otras personas a las que quiero agradecer su colaboración. Lo mismo que a esas otras que han participado de otra forma, quizá menos científica pero igualmente personal y humana.

En primer lugar, a Jordi Domingo como tutor de mis cursos de doctorado y a José Ignacio Navarro, como director de este trabajo y de toda mi trayectoria en el campo de la investigación.

A José María Llabería, director del Departamento y a Mateo Valero, director del CEPBA, por la confianza que han mostrado en mi trabajo.

Al Departamento de Arquitectura de Computadores, en general, por los medios que ha puesto a nuestra disposición, sin los cuales no hubiera sido posible la realización práctica, y aún teórica, de este trabajo; en particular, a cada uno de sus miembros, por las preguntas que nos han respondido y las que nos han hecho. Y sobre todo por las caras amigas que han salido siempre por detrás de los terminales.

A Ana Moreno y Cristina Barrado, con las que he compartido, además de amistad, despacho, ánimos y desánimos en estos años de trabajo.

A Teo Jové, que tan bien supo hacer de comodín cuando las circunstancias lo requirieron.

Al grupo de investigación de sistemas operativos, con los que hemos pasado ratos de trabajo y ratos de tertulia. A Xavier Martorell por su inestimable colaboración en la programación y diseño, y a Fermín J. Reig. También a Ernest Artiaga, Cinta Duatis, Siscu, Montse Rubia y Laura Pena. Por su ilusión y sus ganas de colaborar en un proyecto que a veces se les hacía grande.

Y a mi familia, por todo.

A mis padres y abuelos

Contenido

Introducción y objetivos del trabajo

I.1. Introducción 5
I.2. Entornos paralelos de desarrollo 6
I.3. Nuestro planteamiento 7
I.4. Contribuciones 8
I.5. Organización del documento 9

Parte I

Estado del arte de los sistemas operativos para entornos multiprocesadores

Capítulo 1: Arquitecturas y Sistemas Operativos para Multiprocesadores con Memoria Compartida. 15

1.1. Introducción 16
1.2. Multiprocesadores con memoria compartida 17
1.2.1. ¿Qué es un multiprocesador? 17
1.3. Sistemas Operativos en multiprocesadores 19
1.3.1. Transporte de SO monoprocesador: UNIX 19
1.3.2. Diseño de sistemas operativos para multiprocesadores 20
1.3.3. Tecnología actual: Microkernels 20
1.4. Soporte de los sistemas operativos a la multiprogramación 21
1.4.1. Objetos planificables por el sistema 21
1.4.2. Estructuras para gestionar la planificación por parte del sistema 22
1.5. Políticas de planificación de flujos para sistemas multiprocesadores 23
1.5.1. Políticas de tiempo compartido 23
1.5.2. Planificación de grupos 24
1.5.3. Políticas de espacio compartido 25
1.6. Factores que influyen en la planificación de los multiprocesadores multiprogramados de propósito general 26
1.6.1. Mecanismos de optimización en la sincronización 27
1.6.2. Afinidad al procesador 28
1.7. Referencias y Bibliografía 29

Capítulo 2: Concurrencia y Paralelismo en Aplicaciones 35

2.1. Introducción 36

2.2. Paradigmas de programación concurrente 37

2.2.1. Compiladores y arquitecturas especializadas 38

2.2.2. Estructuras del lenguaje 39

2.2.3. Librerías 40

2.3. Programación con procesos ligeros 40

2.3.1. Mecanismos de creación 41

2.3.2. Mecanismos de sincronización y exclusión mutua 42

2.4. Soporte del sistema operativo a la concurrencia y paralelismo para las aplicaciones ..43

2.4.1. Procesos ligeros y Procesos 44

2.5. Algunas distribuciones actuales de paquetes multiflujo 46

2.5.1. Simplificar el trabajo sobre *threads* de kernel: CThreads 46

2.5.2. En busca de un estándar: Pthreads 47

2.6. Primitivas de sincronización a nivel sistema 47

2.6.1. Primitivas de sincronización ofrecidas para el usuario 47

2.6.2. Sincronización dentro del propio sistema 48

2.7. Aligerar los cambios de contexto: continuaciones explícitas 49

2.7.1. Modelos de ejecución en el kernel 49

2.8. Aligerar el contexto de los flujos: cooperación usuario-sistema 50

2.8.1. Scheduler-activations 50

2.8.2. Otros mecanismos de comunicación kernel-usuario: Objetos de primera clase ..52

2.9. Planificación en Mach 3.0 53

2.9.1. Colas de preparados 54

2.9.2. Estados de planificación y ejecución 55

2.9.3. *Threads* de kernel en el kernel de Mach 55

2.10. Mecanismo de *handoff* en el kernel de Mach 57

2.11. Continuaciones explícitas en el kernel de Mach 58

2.12. Referencias y Bibliografía 59

Parte II

La aplicación como ente y gestora de su planificación

Capítulo 3: La aplicación como entidad 67

3.1. Introducción 68

3.2. Perfil de las aplicaciones a gestionar en un multiprocesador de propósito general. 69

3.3. La aplicación como entidad planificable 69

3.3.1. Adaptación de la aplicaciones a la política de planificación 71

3.4. Composición de la aplicación como objeto 72
3.4.1. La task 72
3.4.2. Visibilidad de los datos y semántica de la abstracción de flujo 72
3.5. Desarrollos actuales del concepto de aplicación 73
3.5.1. La aplicación como conclusión de diferentes modelos de programación: Psyche 74
3.5.2. La aplicación como descomposición de un problema 74
3.5.3. La aplicación como programación multiflujo: Mach 75
3.5.4. El futuro de las aplicaciones de alto rendimiento 76
3.6. Aportaciones de este trabajo 77
3.7. Referencias y Bibliografía 79

Capítulo 4: Mecanismos y políticas fuera del kernel. 83

4.1. Introducción 84
4.2. Algunas políticas ya existentes a nivel de usuario 84
4.3. Entorno de desarrollo: CThreads 85
4.3.1. Establecimiento del vínculo proceso ligero-procesador virtual 87
4.3.2. Tratamiento de bloqueos a nivel usuario 88
4.3.3. Robustez de la librería y opciones de compilación 88
4.3.4. Conclusiones acerca de la librería CThreads 88
4.4. Algunos desarrollos experimentales con CThreads 89
4.5. Nueva funcionalidad: prioridades a nivel de usuario 89
4.5.1. Prioridades en CThreads^{+} 90
4.5.2. Prioridades estáticas versus prioridades dinámicas 90
4.6. Selección directa de flujos en la planificación a nivel de usuario 91
4.7. Optimizaciones en las primitivas del propio paquete 91
4.8. Evaluación del rendimiento de la nueva librería de CThreads^{+} 92
4.9. Conclusiones 96
4.10. Referencias y Bibliografía 96

Capítulo 5: Cooperación entre la aplicación y el kernel en la planificación de flujos 101

5.1. Introducción 102
5.2. Colaboración del kernel con la aplicación 102
5.2.1. Relación entre procesadores físicos y procesadores virtuales: mapeo 1-1 103
5.2.2. Mecanismos para la notificación de eventos 106
5.2.3. Eventos que se notifican a la aplicación 107
5.3. Transferencia de la gestión del kernel a la aplicación 108
5.3.1. Entorno de trabajo 109
5.3.2. Inicialización de los eXc 110
5.3.3. Añadir un procesador a la aplicación 112
5.3.4. Aviso por temporizador 113

5.3.5. Bloqueo y desbloqueo de un flujo 114
5.3.6. Quitar un procesador a la aplicación 120
5.4. Un ejemplo de gestión de flujos en un entorno de eXc: bloqueo y desbloqueo por una petición de servicio 121
5.4.1. Momento de bloqueo: llamada write() al kernel 121
5.4.2. Momento de desbloqueo: finalización del write y vuelta a usuario 122
5.5. Gestión llevada íntegramente por el kernel: eventos “transparentes” 122
5.6. Nuevas funcionalidades en la gestión de flujos a nivel usuario 123
5.6.1. Información compartida por la aplicación y el kernel: objetos de primera clase 123
5.6.2. Planificación de flujos a nivel usuario: continuaciones explícitas 124
5.6.3. Tipos de flujos ejecutables en la aplicación 125
5.6.4. Los procesos nulos de la librería de CThreads: los *waiters* *126*
5.7. Referencias y Bibliografía 127

Capítulo 6: Realización de los contextos de ejecución 131

6.1. Introducción 132
6.2. Estructuras de datos utilizadas. 133
6.2.1. Información que mantiene el sistema 133
6.2.2. Información que mantiene el usuario 134
6.2.3. Cola de flujos bloqueados en servicios del sistema 135
6.3. Mecanismos de exclusión mutua entre la aplicación y el kernel 136
6.3.1. Atomicidad en el tratamiento de upcalls 136
6.3.2. Atomicidad en la cola de preparados 137
6.3.3. Abrazo mortal por desbanque de un flujo 140
6.4. Mecanismos de subida de notificaciones desde el kernel 143
6.4.1. Activar una upcall en el propio procesador 143
6.4.2. Desbancar un procesador para subir una notificación a la aplicación 144
6.5. Solapamiento de upcalls 146
6.5.1. Atomicidad en el tratamiento de upcalls: retardar la subida de nuevos avisos ..146
6.6. Repetición de s-a en el tratamiento de un mismo servicio 146
6.7. Gestión transparente al usuario: excepciones 147
6.8. Fases de un eXc ejecutando código en modo kernel 148
6.9. Ordenación en el tratamiento de los nuevos ASTs 148
6.10. Algunas optimizaciones en la gestión de upcalls 150
6.10.1. *Bypass* de notificaciones 150
6.10.2. Reciclaje de procesadores virtuales 151
6.11. Referencias y Bibliografía 151

Parte III

Evaluación, conclusiones y líneas abiertas

Capítulo 7: Evaluación del rendimiento en la cooperación kernel-aplicación 157

7.1. Introducción 158

7.2. Caracterización de la carga, benchmark 158

7.3. Herramientas de monitorización software y hardware 159

7.3.1. Mecanismos hardware 160

7.3.2. Mecanismos software 160

7.4. Entorno de trabajo 160

7.5. Hardware de medición: High Resolution Clock 161

7.6. Software de medición 162

7.6.1. La herramienta de medición JEWEL 162

7.6.2. Traceado y extracción de contadores: nuevas llamadas al sistema 164

7.7. Rendimiento del traspaso de funcionalidades del kernel a la aplicación 165

7.7.1. Cambios de contexto en modo usuario y recurriendo al kernel 165

7.7.2. Creación de objetos planificables en usuario y en kernel 168

7.8. Tratamiento de las notificaciones mediante upcalls 169

7.8.1. Tratamiento y planificación de flujos desde la upcall de temporización 169

7.8.2. Tratamiento y planificación de flujos desde la upcall de asignar procesador 169

7.9. Planificación de la entrada / salida gestionada por la librería 170

7.10. Evaluación del entorno realizado en cuanto a su transportabilidad 172

7.10.1. Código modificado en el kernel 173

7.10.2. Rutinas añadidas en el kernel 173

7.11. Referencias y Bibliografía 174

Capítulo 8: Conclusiones y líneas abiertas 177

8.1. Introducción 178

8.2. Aportaciones realizadas en este trabajo 178

8.2.1. Concepto de aplicación 179

8.2.2. Realizaciones a nivel usuario 180

8.2.3. Realizaciones a nivel kernel 182

8.2.4. Comunicación entre la aplicación y el kernel 183

8.2.5. Evaluación de los acontecimientos del sistema y del mecanismo de upcalls 183

8.3. Estado actual de los microkernels 183

8.4. Líneas abiertas 184

8.4.1. Mecanismos de desbanque “conscious” 184

8.4.2. Gestión de memoria a nivel usuario 184

8.4.3. Estado de las aplicaciones de cálculo: registros de coma flotante 185
8.4.4. Potencia de planificación a nivel usuario 185
8.5. Conclusiones del trabajo realizado 185

Parte IV

Apéndices

Apéndice A: Diseño y realización de un servidor de procesadores ...190
A.1. Introducción 191
A.2. Descripción del interfaz 192
A.2.1. Cpu_server_request 194
A.2.2. Cpu_server_release 194
A.2.3. Cpu_server_info 194
A.3. Detalles del diseño sobre Mach 195
A.3.1. Identificación de las peticiones 195
A.3.2. Servidor multiflujo 195
A.3.3. Gestión de excepciones 195

Apéndice B: Nuevo interfaz de la librería CThreads+ 198
B.1. Introducción 199
B.2. Planificación por prioridades 199
B.2.1. Modificar la prioridad: cthread_set_prio 199
B.2.2. Consultar la prioridad: cthread_get_prio 200
B.2.3. Inicializar el rango de prioridades: cthread_init_prio 200
B.3. Cesión voluntaria del procesador virtual 201
B.3.1. Cthread_handoff 201
B.3.2. Cesión del procesador virtual a otro flujo más prioritario: cthread_hint 202

Apéndice C: Funciones e interfaz del mecanismo de upcalls 204
C.1. Introducción 205
C.2. Inicialización del interfaz del mecanismo de upcalls 205
C.2.1. Inicialización del entorno de planificación de flujos sobre eXc 205
C.2.2. Registro de las rutinas de upcall 205
C.2.3. Creación del espacio para el pool de pilas 206
C.2.4. Solicitud de la activación del mecanismo de upcalls 207
C.2.5. Programación de un temporizador 207
C.3. Interface de las upcalls (rutinas que atienden a cada upcall) 208
C.4. Interface interno a las rutinas de tratamiento de las upcalls 209
C.4.1. Salvar el contexto de usuario de un eXc 209

C.4.2. Encolar en la aplicación un eXc bloqueado en el kernel 210
C.4.3. Desencolar en la aplicación un eXc desbloqueado en el kernel 210
C.4.4. Inserción de un cthread en la tabla de preparados 210
C.4.5. Cesión del eXc a un flujo planificador 210
C.4.6. Planificación de un Cthread desde una upcall 211
C.4.7. Continuación explícita de un Cthread desbancado 211
C.5. Flujos nulos y planificadores a nivel de aplicación 211
C.5.1. *Handoff* de un cthread al flujo nulo 212
C.5.2. Actualización de la cola de preparados 212
C.5.3. Intento de bloqueo de un flujo nulo y cesión de su eXc 212
C.5.4. Nueva primitiva de *spin lock*: marcar quién está en posesión del *lock* *212*
C.5.5. Inhibición temporal del mecanismo de upcalls 213