Trabajo práctico 0: Infraestructura básica

Daddario Rubén, *Padrón Nro. 83122*
ruben.daddario@gmail.com

Amura Federico, *Padrón Nro.*
federicoamura@gmail.com

Grupo Nro. X - 2do. Cuatrimestre de 2014
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Jueves 10 de septiembre de 2014

Abstract

El presente informe se corresponde al trabajo práctico número 0 de la asignatura 66.20 Organización de Computadoras. El trabajo consiste en desarrollar un programa portable con el objetivo de familiarizarse con el entorno de desarrollo a utilizar en la materia y las distintas herramientas disponibles.

Contents

1 Introducción

MIPS (Microprocessor without Interlocked Pipeline Stages) es un procesador con arquitectura RISC (reduced instruction set computer) desarrollado por MIPS Technologies. Este tipo de procesadores se encuentra en muchos sistemas embebidos y dispositivos de la actualidad como consolas de videojuegos, routers, etc. En particular, debido a que los diseñadores crearon un conjunto de instrucciones tan claro, los cursos sobre arquitectura de computadoras en unviersidades y escuelas técnicas a menudo se basan en la arquitectura MIPS.

Existen múltiples versiones de este procesador, siendo los más recientes MIPS32 y MIPS64, implementadas para 32 y 64 bits, respectivamente.

2 Desarrollo

El programa a elaborar es una implementación en ANSI C del comando nl de unix con un conjunto reducido de parámetros. Permite a grande rasgos enumerar las líneas de un documento a partir de un archivo de entrada.

3 Diseño y detalles de implementación

3.1 Recursos utilizados

Para resolver el problema planteado utilizamos la herramienta GXemul, para emular un entorno MIPS y el sistema operativo NetBSD. También utilizamos code::blocks para la codificación de la aplicación y un repositorio SVN para compartir el proyecto.

Para implementar el programa utilizamos el lenguaje de programación C y para la realización del informe utilizamos LaTeX.

3.2 Diseño

El programa se compone de múltiples archivos de cabecera .h con sus correspondientes archivos .c donde se realiza la implementación de los métodos utilizados.

```
tp0 [OPTION]...[FILE]...
```

Las opciones pueden ser:

- -h, –help Print this message and quit

- -s, –number-separator (mandatory argument)
- -v, –starting-line-number (mandatory argument)
- -i, –line-increment (mandatory argument)
- -l, –join-blank-lines (mandatory argument)
- -t, –non-blank

3.3 Implementación

3.3.1 Portabilidad

Para proveer portabilidad se implementaron las funciones en lenguaje C para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

3.3.2 Código y algoritmos

El programa se compone de un archivo main con el punto de entrada al ejecutable y desde aquí se invócan al resto de los archivos que definen las estructuras de datos necesarias para la ejecución.

4 Mediciones. Corridas de prueba y resultados obtenidos

A continuación presentamos una corrida de prueba de la implementación, compilación y ejecución desde NetBSD.

Figure 1: Compilación y construcción del ejecutable. Salida del parámetro -h o –help

Figure 2: Diversas resultados mostrados por standard output

Utilizando las herramientas time y gprof logramos hacer un análisis de la ejecucíon del programa sometido a diferentes tamaños de carga.

Vamos a tomar como ejemplo para un seguimiento de mediciones al siguente comando:

time ./tp0 -s="–¿" -v 1 –line-increment=1 inputlong.txt ¿ output.txt

Descripcion del comando Este caso en particular tiene varias ventajas. En primer lugar vemos de manera notoria que le estamos haciendo procesar el archivo "inputlong.txt" unas 32 veces. Esto lo hacemos para que nos de las mediciones de procesar muchos archivos y cada uno con mucha carga, de esta manera logramos agrandar las mediciones de tiempo de las funciones que consumirian mas tiempo. Ademas, dado que le damos como linea inicial y como salto de lineas los valores mas bajos posibles se va a procesar la totalidad de lo que le enviemos. Tambien le decimos que la salida, en vez de hacerla por consola la haga hacia un archivo dado que no es el foco de nuestro analisis la salida del programa si no las mediciones de tiempo y recorrido que hacemos sobre el, de esta manera evitamos que la salida por consola nos tape los tiempos de proceso con los que corresponderian al mostrado en pantalla.

Devolucion del comando Nos arroja los siguientes mediciones de tiempo:

real 0m48.285s

user 0m15.144s

sys 0m8.793s

Con lo cual vemos que el programa demoro unos 48 segundos en correrse. Esto es muy util para calcular el SpeedUp frente a otras versiones o implementaciones del mismo programa.

Luego, dado que el programa fue compilado para funcionar juntando estadisticas de tiempo podemos ejecutar gprof con el siguiente comando:

gprof -z tp0

Lo cual nos trae todos los datos que recolecto nuestro programa mientras fue ejecutado. En particular nos interesa la primera tabla que indica la cantidad de tiempo que estuvo en cada funcion y cuantas veces fue llamada cada una.

79.92	0.51	0.51	32	15.98	19.43	readFile
7.05	0.56	0.05	240068128	0.00	0.00	isLineEmpty
5.84	0.59	0.04	24068064	0.00	0.00	shouldFormat
4.70	0.62	0.03	24068064	0.00	0.00	formatLine
3.13	0.64	0.02	•	•	•	readFiles
0.00	0.64	0.00	•	•	•	main

Solo mostramos las primeras 5 funciones ya que despues de esta empiezan a aparecer funciones mas internas al sistema o que presentan mediciones con valores practicamente cero, exactamente igual a la fila de main pero con el nombre correspondiente.

Lo mas importante para determinar donde seria optimo realizar mejoras esta en la primera y cuarta columna de esta tabla.

Viendo primero la cuarta columna vemos la cantidad de veces que se llamo cada funcion. Como podemos ver la funcion readFile se llamo 32 veces, una por cada vez que le indicamos que abriera el archivo. Luego las tres siguientes se llamaron 24068128 veces, por lo que una modificacion en esa parte es crucial para el tiempo de ejecucion dado que se trasladara en hacer una operacion esa cantidad de veces mas o menos. La primera columna nos indica el porcentaje de tiempo que el programa estuvo en esa funcion. Como podemos ver estuvo casi un 80 por ciento en la funcion readfile, que es la encargada de abrir el archivo desde el disco. Luego estan las funciones isLineEmpty, shouldFormat, formatLines y readFiles generando casi todo el tiempo que tarda en ejecutarse el programa, lo cual es logico dado la cantidad de veces que se llamaron. En definitiva la tabla se nos muestra ordenada con respecto a la tercer columna, self seconds, que indica cuandos

segundos totales estuvo esa funcion en ejecucion, un criterio simple seria tratar de optimizar segun este orden, pero dado que una mejora en readFiles se multiplicaria por 32, mientras que una en las otras funciones por 24068128, esto no es tan correcto.

Ya podemos concluir en las mejoras que debemos realizar, por un lado, considerando que el mayor tiempo lo gasta en readFiles, es nuestro principal punto de ataque, el primer objetivo seria acercar los archivos a una memoria mas cercana para trabajarlos desde ahi, mas eficientemente. Luego, lo ideal seria optimizar las funciones que se repiten miles de veces, incluso antes de optimizar esos aspectos de readFiles, dado que una mejora que nos ahorre una instruccion de procesador en estas funciones sera mucho mas recompensado que una en otro lugar del codigo. Luego se haria lo mismo con readFiles.

Luego de realizar las mejoras volvemos a correr el mismo comando de analisis, realizamos las mismas mediciones y veremos donde seguir enfocando las optimizaciones, ademas de poder calcular el SpeedUp que logramos con las mejoras.

5 Conclusiones

A partir de este trabajo adquirimos experiencia en el uso de GXEmul para MIPS32, y el uso de LaTeX. Ademas el uso de las herramientas time y gprof para determinar donde se esta produciendo la demora del programa y la efectividad de las mejoras.

References

[1] J. L. Hennessy and D. A. Patterson, Computer Architecture. A Quantitative Approach, 3ra Edicion, Morgan Kaufmann Publishers, 2000.

[2] Grupo de la materia http://groups.yahoo.com/group/orga6620/

[3] Wikipedia en.wikipedia.org