Testing Characteristics of the Linux Page Replacement Policy

Callen Rain & Justin Cosentino

April 8, 2014

Abstract

A Brief summary of problem examining and results found

1 Data

Phase	Case	Page Faults	Error
1	1	8608.00	± 86.23
	2	7775.00	± 41.76
2	1	128.00	± 3.87
	2	978.00	± 32.60

Table 1: MRU versus LRU test data *Description*

Phase	Case	Page Faults	Error
1	1	7708.00	± 158.27
	2	6832.00	± 128.12
1	1	976.00	± 20.27
	2	993.00	± 20.12
2	1	976.00	± 8.89
	2	979.00	± 33.21
1	1	0.00	± 0.00
	2	983.00	± 15.59

Table 2: FIFO versus LRU test data *Description*

2 Introduction

Here are some font examples: *this is an example of how to get italics fonts* here is **bold** here is a code font `foo(int x);`.

If you want a new paragraph, just put a blank line between the current paragraph

and the next paragraph.

noindent will remove the auto indenting of paragraphs within a section.

Phase	Case	Page Faults	Error
1	1	3614.00	± 131.24
	2	5142.00	± 320.20
1	1	1.00	± 1.00
	2	1.00	± 0.00
2	1	3990.00	± 24.58
	2	2163.00	± 13.96

Table 3: Working Set versus LRU test data *Description*

If you want a larger break between paragraphs end the line with slash slash and add a blank line between it and the next paragraph

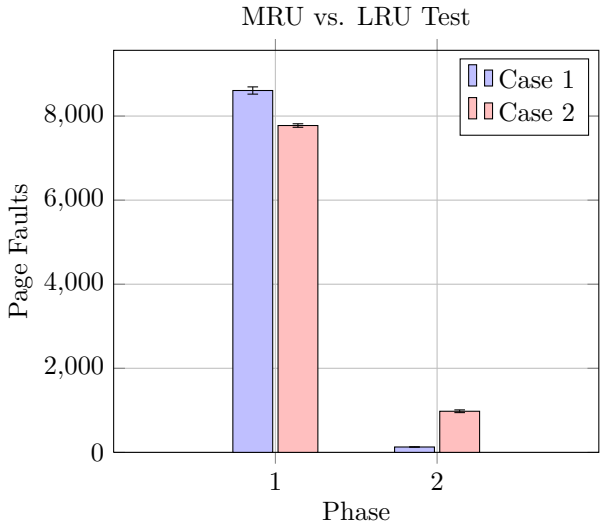
To refer to a figure tha appears anywhere in the document use `backslash ref{labelname}` and note where the labelname is defined in the figure (the tilda glues it to the preceding word of text). For example: (see Figure **??**).

Here is an example of how to incorporate a figure in a document...you need to use one of the psfig or epsfig style files to include a postscript or an encapsulated postscript file. These style files also include macros for positioning your figure around the text, for changing the size of your figure for adding captions and labels...

I can force a figure to the top or bottom of a page by using the [t] or [b]. I can also try to force it to come after the prose it follows in the .tex file by using [!htb]:

The .cls and .sty files typically contain comments on how to use them in your latex document...if you look at psfig.sty, the file contains comments that summarize commands implemented by this style file and how to use them

Here is an example of how to print somthing using a courier font function `invokeMethod` while interpreting AP method `foo`. The application program developer wants performance data that measures the



3 Here is how to create an enumerated list:

1. **CPUtime, </APCode/foo.class/foo>**: note the dollar signs around the above are interpreted as math mode. In this mode some characters have special meaning that result in special formatting. Latex is particularly good for representing mathematical functions via math mode. You can also put a large block of text in math mode using begin and end.
2. The next thing
3. **methodCallTime, </APCode/foo.class/foo>**: I used to have a mathematical definition for this.

3.1 Tables in Latex

Tables are not the easiest interface to use, but you can have latex generate tables for you. The first part `e.g. |c|l|r|rr|` specifies the number of columns implicitly and the formating for each column (centered, left, right). If you put a vertical bar between entries, then a vertical bar will be drawn between the columns otherwise not. The first example below has 3 columns, the first uses centered formatting that next two right. Each row of values is listed on a separate line. Ampersands are used between each row's column values and backslash-backslah ends a row. hline can be used to draw horizontal lines. multicolumn can be used to span columns and to change the default column formatting.

Here is a larger example that also demostrates multicolumn directive:

4 Some random prose to make this longer

This course is a introduction to the theory, design, and implementation of operating systems. An operating system is the software layer between users and the computer hardware. It implements abstractions that are easier to program than the underlying hardware (e.g. processes, virtual memory, file systems), and it manages the machine's resources (e.g. memory, cpus, disk, network interfaces, and other devices). We will cover the following topics: processes (including synchronization, communication, and scheduling), memory (main memory allocation strategies, virtual memory, and page replacement policies), file systems (including naming and implementation issues), I/O (including devices, drivers, SSDs and disks, and disk scheduling), protection and security, distributed systems and other advanced topics. Prerequisites: CPSC 35 and CPSC 31 required.

This course is a broad introduction to computer science that focuses on how a computer works and how programs run on computers. We examine the hardware and software components required to go from a program expressed in a high-level programming language like C or Python to the computer actually running the program. This course takes a bottom-up approach to discovering how a computer works, and introduces parallel and distributed computing with a

specific focus on parallelism for multicore and other shared memory systems. Topics include theoretical models of computation, data representation, machine organization, assembly and machine code, memory, I/O, the stack, the operating system, compilers and interpreters, processes and threads, and synchronization. In addition to parallel programming, we will discuss parallel computers and system-level support for parallel computing. Prerequisite: Completion of CS21 or its equivalent.

4.1 Goals for the Course:

- To understand how a sequential or parallel program goes from being expressed in a high-level programming language to being run on the underlying system.
- To understand and analyze the systems costs associated with application performance.
- To understand the role of the operating system and some of the abstractions it implements to support efficiently running programs.
- To understand shared memory parallel computing and to be able to "think in parallel" algorithmically.
- To become proficient in using gdb and valgrind to debug and examine program execution state.
- To design and implement solutions to large problems using the C programming language.
- To design and implement parallel solutions to programming problems that require synchronization using pthreads.

5 More Random Prose

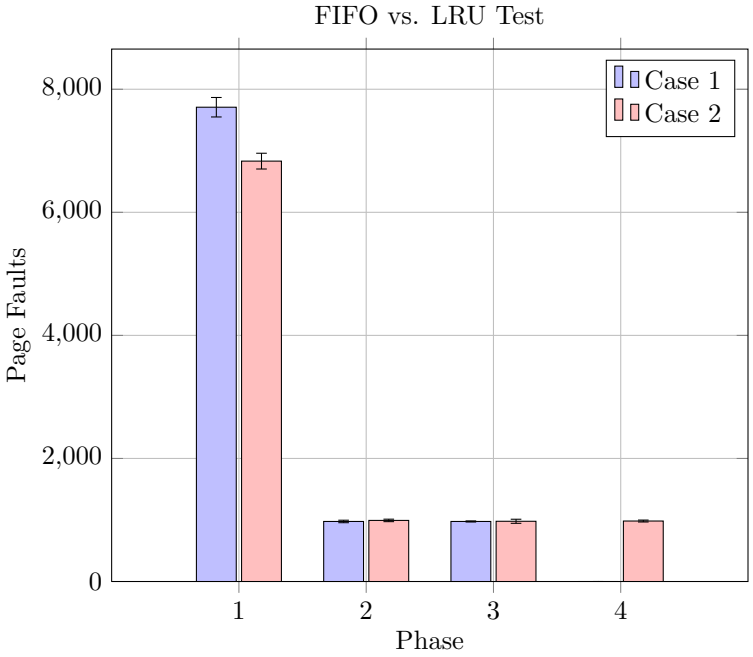
This course covers a broad range of topics related to parallel and distributed computing, including parallel and distributed architectures and systems, parallel and distributed programming paradigms, parallel algorithms, and scientific and other applications of parallel and distributed computing. In lecture/discussion sections, students examine both classic results as well as recent research in the field. The lab portion of the course includes programming projects using different programming paradigms, and students will have the opportunity to examine one course topic in depth through an open-ended project of their own choosing. Course topics may include: multi-core, SMP, MMP, client-server, clusters, clouds, grids, peer-to-peer systems, GPU computing, scheduling, scalability, resource discovery and allocation, fault tolerance, security, parallel I/0, sockets, threads, message passing, MPI, RPC, distributed shared memory, data parallel languages, MapReduce, parallel debugging, and applications of parallel and distributed computing.

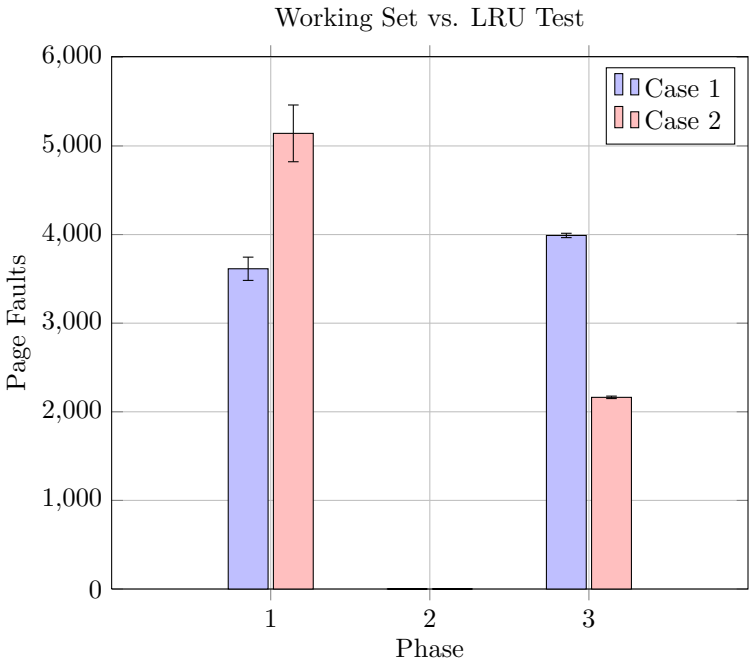
Class will be run as a combination of lecture and seminar-style discussion. During the discussion based classes, students will read research papers prior to the class meeting that we will discuss in class. During the first part of the course, we will examine different parallel and distributed programming paradigms. During the second part of the course, students will propose and carry out a semester-long research project related to parallel and/or distributed computing. The department's gigabit cluster, two eight processor workstations, as well as the CS lab machines, are available for course projects.

Previous course work in Operating Systems, Networking, Databases, or Architecture is helpful, but not necessary for taking this course.

6 Conclusion

To figure out how to do something in latex, look at some examples and some on-line documentation, or google for it. Often times it is very difficult to get latex to format something differently from what its default formatting is.


FIFO vs. LRU Test
Case 1
Case 2
Page Faults
8,000
6,000
4,000
2,000
0
1
2
3
4
Phase


Working Set vs. LRU Test
Case 1
Case 2
Page Faults
6,000
5,000
4,000
3,000
2,000
1,000
0
1
2
3
Phase