

System Design Document  
for  
CMSC 421 Project 2  
Phase: Complete  
Spencer R. Shimko  
2004-10-19

# Table Of Contents:

1	Introduction.....	3
1.1	Purpose.....	3
1.2	Scope.....	3
1.3	References.....	3
1.4	Overview.....	3
1.5	Constraints.....	3-4
2	System Overview.....	4
3	System Architecture.....	4
3.1	Architectural Design.....	4
3.2	Architectural Alternatives.....	4
3.3	Design Rationale.....	4
4	Data Design.....	5
4.1	Global Data Structures.....	5
4.2	Functional.....	5
5	Human Interface Design.....	5
6	Testing.....	5
7	Timeline.....	5

# 1 Introduction

## 1.1 Purpose

This document describes the requirements fulfilled by adding an additional system call to the Linux kernel. The reader is expected to have a fundamental understanding of programming and data structures.

## 1.2 Scope

For this project two separate software components are produced. The first is the Linux kernel implementation of an additional system call. The second is the user space driver to exercise this system call. These components combined will enable a user to easily obtain information about the maximum number of threads supported and the current number of threads running. Each of these will be enumerated in the sections following.

## 1.3 References

### Project Description:

- <http://www.csee.umbc.edu/courses/undergraduate/421/Fall104/project2.htm>

### System Call:

- [http://www.csee.umbc.edu/courses/undergraduate/CMSC421/fall102/burt/projects/howto\\_add\\_systemcall.html](http://www.csee.umbc.edu/courses/undergraduate/CMSC421/fall102/burt/projects/howto_add_systemcall.html)  
- [http://www.csee.umbc.edu/courses/undergraduate/CMSC421/fall102/burt/projects/LinuxJournal\\_SystemCalls.html](http://www.csee.umbc.edu/courses/undergraduate/CMSC421/fall102/burt/projects/LinuxJournal_SystemCalls.html)  
- <http://appsrv.cse.cuhk.edu.hk/%7Ecsc3150/tutnote/notel/kernel.html#syscall>

### Kernel Source:

- <http://lxr.linux.no/source/?v=2.2.19>

## 1.4 Overview

The rest of this document will be dedicated to clarify the statements made in section 1.2 above (Scope). Specifically a more descriptive overview will be presented followed by a discussion of the architecture and data design. Finally this document will briefly discuss the human interface, testing, and time line.

## 1.5 Constraints

Aside from the constraints placed upon this project from the description (the function prototypes, etc) there are a few other restrictions to consider. Data must be copied safely from kernel space to user space. This will

be done using pointers passed from the user space which must be verified for security reasons.

## **2 System Overview**

This system involves modifying the standard RedHat Enterprise Linux kernel version 2.4.21.EL-20. A system call will be added to this kernel that will facilitate querying the kernel for thread information. Specifically a user space driver will be written to execute a new system call which, upon success, will modify two pointers to integers passed to the system call as arguments. The intent of this project is to introduce the implementers to details of system calls in the Linux kernel and see why and how they are used to execute privileged instructions in a modern operating system.

## **3 System Architecture**

### **3.1 Architectural Design**

This section intentionally left blank for this design document as diagrams are not required for this system implementation.

### **3.2 Architectural Alternatives**

This system can be implemented any hardware platform supported by the Linux kernel such as Sparc or PPC. There is a specific method to add system calls to any of these platforms and while the method of adding them might differ, the concept remains the same. The alterations made to "sys.h" and "unistd.h" are cross-platform as is the system call function and header files. However, the *system call table*, which is hardware specific, must be modified to handle the additional system call. Read the section below entitled "Design Rationale" for an example of added a call to the i386 system call table.

### **3.3 Design Rationale**

This system is implemented on Intel 32-bit 80386 class hardware. To add a system call must be added to "entry.S", which is an i386 Intel assembly file. This platform was chosen because it is familiar to the implementers.

## **4 Data Design**

### **4.1 Global Data Structures**

The data structures used in this system implementation are limited to four main variables and a few variables used as error flags. The four main variables are the kernel space `max_threads` and `nr_threads`, and the user space pointers to their respective counterparts `threads_max` and `threads_running`.

### **4.2 Functional**

The main system call prototype which will be used is:  
`int getThreadInfo(int *threads_max, int *threads_running)`  
This function returns an integer representative of it's exit status. It returns 1 for success and 0 for failure. It takes two pointers to integers as arguments. The integer they point to will be altered to reflect the current state of threads in the kernel.

## **5 Human Interface Design**

The limited human interface is provided by a driver program that executes the system call and displays the current thread information via the standard out. Any errors are reported to standard error. Additional notification of system call entry and exit is printed to `syslogd` through `printk()`.

## **6 Testing**

This system was tested by passing the system invalid and out of range arguments. This testing revealed the need to use additional protection such as the kernel function `access_ok()`, found in "uaccess.h".

## **7 Time line**

Released: Sep 30th, 2004

Design complete: Oct 7th, 2004

Implementation complete: Oct 15th, 2004

Debug and fine tuning: Oct 17th, 2004

System Design Document complete: Oct 20th, 2004

Completion: Oct 20th, 2004