OS Assignment 1
Process management and system calls

Madhur Singhal
2015CS10235

February 2017

1 Introduction

Operating systems are designed to act as effecient managers of computing resources. They perform tasks like scheduling, file system management, device interaction, sandboxing and communication. Xv6 is a tiny operating system based on the Unix design methodology. In this assignment we study the xv6 source code to understand it's functioning and add some small bits of code to it to demonstrate our understanding. The main areas we are concerned with here is the system call pipeline and the process table.

2 System Call Tracing

The first task is to add a system call logging facility to the OS. This means that whenever a system call is executed, a line of output should be printed identifying the system call used and the number of times that particular system call has been utilized till now. We also add a new system call to tutn off/on this logging facility. For this task we add three data structures, an int array **numcalls** to store the number of times each system call has been invoked, a string array **strs** to store the name of the system call at a particular index and an int **trace_on** to indicate whether logging is on or off at the moment.
We add these structures to **syscall.c** and add three lines to print the system call whenever one is called in the same file. Further we add the new system call declarations in **usys.S**, **syscall.h** and **user.h**. We also add the toggle code to **sysproc.c** to modify the **trace_on** variable in **syscall.c**. Finally we add a user program called **user_toggle.c** whose only job is to call

the toggle system call onceThe Makefile is also modified in the appropiate manner so that the new user program and the assign files are linked properly.

3 sys_add System Call

This is very easy, we follow the same methodology as the above one to create a new **sys_add** system call. The main difference is inside **sysproc.c** where inside the code for the add system call we have to use the **argint** function twice to get the two numbers which were passed as parameters to the system call. argint calls fetchint to read the value at that address from user memory and write it to the given integer pointers from where they can be used by the kernel to perform the system call. The declaration in **user.h** is also modified a bit to expect two integers as inputs.

4 sys_ps System call

This is a bit tricky since the **sysproc.c** file does not have access to the process table. Thus to create this system call we modify the **procdump** method inside **proc.c**. We just go over the whole **ptable** and print all processes whose state is not **UNUSED**,**EMBRYO** or **ZOMBIE**. The procesure to add the system call is identical, with modifications to **syscall.c**, **syscall.h**,**usys.S**, **sysproc.c** and **user.h**. Here the **sys_ps** function inside **sysproc.c** just calls the **procdump** function in **proc.c**