Operating Systems- COL331
Assignment 1

Ankesh Gupta
2015CS10435

Part 1

Pointers on System call Trace:

1. We had to trace system calls made by xv6 while its operating.
2. System calls in xv6 are uniquely identified by an identifier, an integer. So, to print name of the call, a mapping was maintained between unique system call id and its name.
3. The mapping was stored in ***array data structure***, which mapped system call name indexed by *unique call id.*
4. Also, as asked, another *(integer)array data structure* was maintained which kept track of system call count, i.e., how many times a particular system call has been invoked since the OS(emulator) began.
5. As soon as any system call is invoked, array's corresponding index was *incremented*.
6. The reason for choosing array data structure was that the unique id's assigned to system calls were *contiguous*. Other data structures can also easily handle this task, but array is quite a simple and intuitive choice for the problem.
7. The data structures are marked *static* as they have no role outside the module.
8. Mapping array was named *syscall_names*. Counter array was named *call_count*.

Pointers on Toggling System calls

1. As a continuation, we had to create a system call which can *toggle* tracing system calls.
2. A $flag(variable)$ was introduced which maintained the current toggling. The variable was externed so that they are visible from external modules.
3. Requisite changes were made in file *usys.S* so that the system call can be recognized and correct system call id is put in *eax register*.
4. Header *syscall.h* was also changed to allocate an unique id to new system call.
5. Header *user.h* was changed so that the system call is visible to user program.
6. Actual implementation of system call was done in *sysproc.c* to keep with convention. Here the state of toggle flag is reversed.
7. Necessary modifications were made in *syscall.c* so that the new system call is visible to kernel, and when called, kernel realizes what instructions to execute.
8. Flag was named $toggle\_flag$.

Part 2

Pointers on add System Call

1. In this, a system call was added which returns sum of 2 numbers.
2. This implementation helped understand how to pass arguments to system call, and how to use them in while implementing system call *handler*.
3. All implementation details are similar as mentioned in above case of adding sys_toggle system call.
4. Only difference is the way we get parameters. A call to *argint* method was made which returns the i^{th} system call argument(i is sent while invocation).
5. 2 calls were made and the sum was returned, which was placed in *eax register*.
6. No extra variables or data structures were introduced. Only few look-up's were updated.

Part 3

Pointers on printing currently running processes

1. We were required to list process id and process name for processes with running, runnable or sleeping state.
2. Implementation details are again similar to above implementations.
3. We required access to *ptable*, a structure which tracked states of all the processes.
4. Since it was available in *proc.c* module, a method was declared which used it to list the required processes.
5. The method was also externed in *sysproc.c* so that its visible in the file and can be accessed.
6. While printing, process's state was monitored and it was listed if:

$$process->state \in \{RUNNABLE, RUNNING, SLEEPING\}$$

7. Again, no new variables and data structures were introduced. Just few look-up's were updated.