KERNEL KREW PROJECT REPORT:

GITHUB LINK:

https://github.com/reineB5/KernelKrew_xv6

Project Goals: To enhance the xv6 operating system by introducing new, functional system calls and implementing additional OS features that extend its capabilities. The project also aims to ensure seamless integration of these changes with existing xv6 components while maintaining rigorous testing for reliability and correctness.

The project is organized into three distinct sprints, each targeting a specific set of objectives and milestones.

SPRINT ONE

Sprint one main objective was to familiarize ourselves with xv6 and its architecture.

In this sprint, we added two new system calls to the xv6 kernel:

- syscall_counts, which receives a syscall number and returns how many times that syscall has been called.
- getproccount, which returns the number of active processes currently running in the system

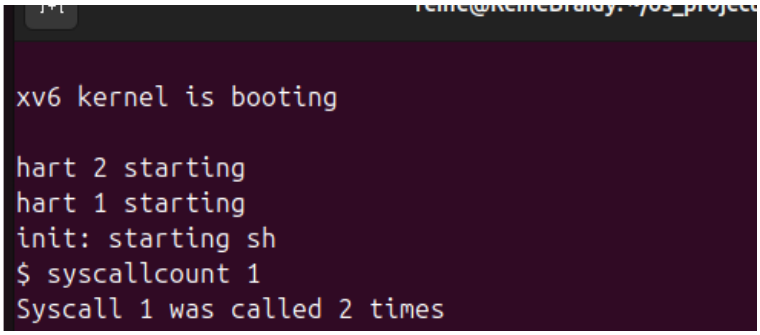
Modifications/additions:

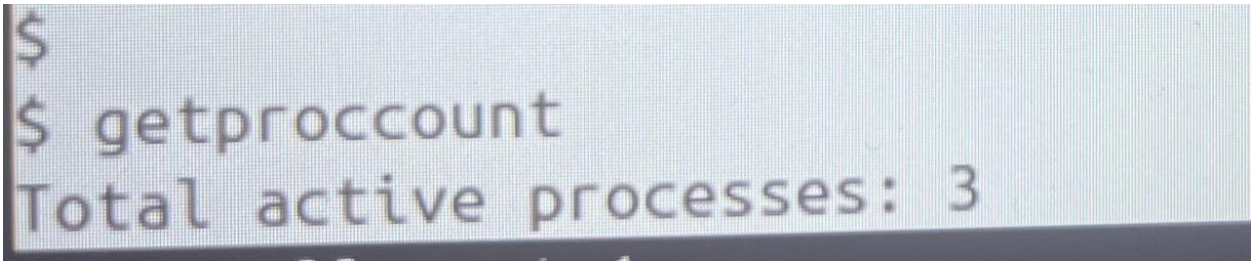
1. **syscall.c**
    - Initialized a global int array: int syscallcounter[30] = {0};
    - Incremented the counter for each syscall number inside the syscall(void) function: syscallcounter[num]++;
    - Added the function sys_syscall_count(void) that returns syscallcounter[x].
    - Added the extern uint64 sys_getproccount(void); declaration.
    - Registered both sys_syscall_count and sys_getproccount in the syscall dispatch table.
2. **syscall.h**
    - Defined new syscall numbers:
      #define SYS_syscall_count 23
      #define SYS_getproccount 24
3. **sysproc.c**

- Implemented syscall_count function that returns the count for a given syscall number.
- Implemented sys_getproccount function that counts active processes by iterating over the process table.
- Declared the process table as external: extern struct proc proc[NPROC];
- Used appropriate return types (int for syscall count, uint64 for process count).

4. **param.h**
    - Added #define numofcalls 30 to control the syscall counter array size and validate inputs.
5. **user.h**
    - Declared syscall interface prototypes:
      int syscall_count(int x);
      int getproccount(void);
6. **User Programs**
    - syscallcount.c: Accepts a syscall number from the command line, calls syscall_count, and prints the result.
    - getproccount.c: Calls getproccount syscall and prints the number of active processes.
    - Used correct kernel and user includes and proper usage of printf and exit.
7. **Makefile**
    - Added _syscallcount and _getproccount to the UPROGS list to include new user programs.
8. **usys.pl**
    - Added entries for both syscalls:
      entry("syscall_count");
      entry("getproccount");

Testing:





SPRINT 2:

In this sprint, we added two new functionalities to the xv6 kernel:

- touch command: Creates a new empty file or updates the timestamp of an existing one.
- search command that allows users to search for a keyword within the contents of a file.

Modifications/additions:

1. user/touch.c

- Takes a filename as an argument
- Opens the file using the O_CREATE | O_RDWR flags
- If the file exists, nothing happens. If it doesn't exist, it's created
- The file is closed and the program exits

2. user/search.c

- Takes a filename and keyword as command-line arguments.
- Opens the file and reads its contents.
- Uses a function to search for the keyword in the file.
- Prints whether the keyword was found.

3. Makefile

Added the following entries under UPROGS:

$U/_touch\

$U/_search\

TESTING:

We tested the touch command inside xv6: $ touch test.txt
then $ ls to verify the file was created

```
$ touch test.txt
ls
$ .              1 1 1024
..               1 1 1024
README           2 2 2292
cat              2 3 34328
echo             2 4 33248
forktest         2 5 16248
grep             2 6 37576
init             2 7 33712
kill             2 8 33168
ln               2 9 32984
ls               2 10 36352
mkdir            2 11 33224
rm               2 12 33216
sh               2 13 54784
stressfs         2 14 34112
usertests        2 15 179416
grind            2 16 49456
wc               2 17 35280
zombie           2 18 32584
touch            2 19 33184
syscallcount     2 20 33256
console          3 21 0
test.txt         2 22 0
$
```

We tested the search command inside xv6 using the following steps:
Created a file with text using: echo hello world > file.txt
Ran: search file.txt world → Output: Found: world

```
$ echo hello world > file.txt
$ search file.txt world
Found: world
$
```

SPRINT3:

New Feature:

• vtop syscall: A new system call that translates a given virtual address to its corresponding physical address in xv6

Modifications/additions:

kernel/syscall.c:

-Declared the system call handler: extern uint64 sys_vtop(void);

-Mapped the system call number to the handler: [SYS_vtop] sys_vtop,

user/user.h:

-Declared the user-level interface for the new syscall: int vtop(void* vaddr);

user/vtopstub.c (New file):

-Implemented the user-side syscall wrapper

kernel/sysproc.c:

- Added the function uint64 sys_vtop(void) to fetch the user-supplied virtual address and call walkaddr() to obtain the physical address.

kernel/syscall.c

- Defined a syscall number SYS_vtop in the enum.

- Added an entry for SYS_vtop in the syscalls[] function pointer array.

user/usys.S:

- Declared the syscall stub:

.**global vtop**

vtop:

li a7, SYS_vtop

ecall

ret

user/vmtest.c:

- Created a user program vmtest that accepts a virtual address as input, invokes vtop, and prints the result.

user/vmtest.h:

To avoid build errors caused by the default user.h and types.h in xv6, we created a minimal custom header file (user/vmtest.h) for the vmtest program. This header defines the necessary types such as uint64 and declares only the essential function prototypes (vtop, printf, and exit). This approach simplified compilation and prevented implicit declaration errors by explicitly specifying the interfaces used in the user test program

Makefile:

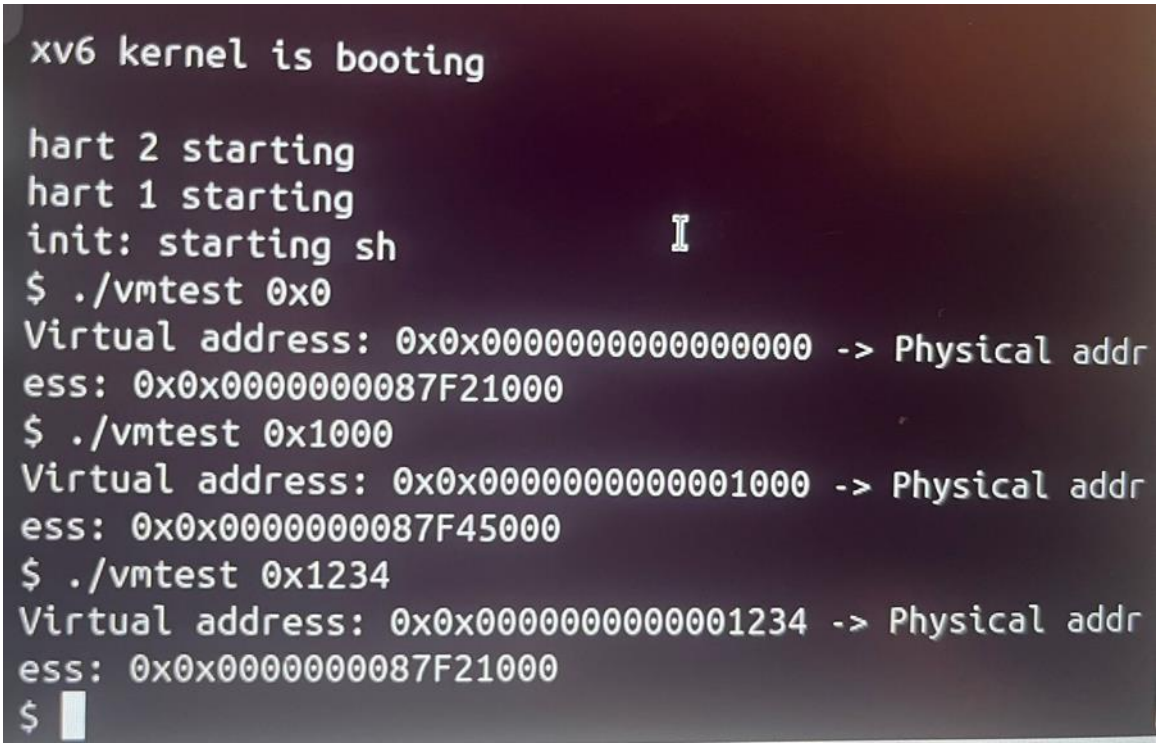
- Added the following entry under UPROGS: $U/_vmtest\

kernel/syscall.h

- Defined:

#define SYS_vtop <number>

TESTING:


xv6 kernel is booting
hart 2 starting
hart 1 starting
init: starting sh
$ ./vmtest 0x0
Virtual address: 0x0x0000000000000000 -> Physical addr
ess: 0x0x0000000087F21000
$ ./vmtest 0x1000
Virtual address: 0x0x0000000000001000 -> Physical addr
ess: 0x0x0000000087F45000
$ ./vmtest 0x1234
Virtual address: 0x0x0000000000001234 -> Physical addr
ess: 0x0x0000000087F21000
$