Offline 2 on xv6 System Call

In this offline, you will add new system calls to xv6, which will help you understand how they work and will expose you to some of the internals of the xv6 kernel.

Relevant files

- The user-space codes that route system calls into the kernel are in `user/usys.S`, which is generated by `user/usys.pl` when you run `make`. Declarations are in `user/user.h`.
- The kernel-space code that routes a system call to the kernel function that implements it are in `kernel/syscall.c` and `kernel/syscall.h`.
- Process-related code are in `kernel/proc.h` and `kernel/proc.c`

Task-1

Implement a new system call `trace` that will control your program tracing. It should take one argument, an integer `sys_call_number` which denotes the system call number to trace for an user program. For example, to trace the `fork` system call, a user program calls `trace(SYS_fork)`, where SYS_fork is the syscall number of `fork` from `kernel/syscall.h`. You have to modify the xv6 kernel to print out a line when each system call is about to return for a process, if the current system call's number is the same as the argument pass in the trace system call. The line should contain the process id, the name of the system call and the return value of that system call. **The trace system call should enable tracing for the process that calls it but should not affect other processes.**

You will be given a `trace.c` user program that runs another program with tracing enabled. For example, when you run `trace 5 echo hello`, the trace user program will first enable tracing for sys_call_num 5 (i.e read) and run the `echo hello` program (look at the `trace.c` file). Make necessary changes so that you can use the given `trace.c` user program in the shell.

You might expect output like this:

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ trace 5 grep hello README
pid: 3, syscall: read, return value: 1023
pid: 3, syscall: read, return value: 961
pid: 3, syscall: read, return value: 321
pid: 3, syscall: read, return value: 0
$ grep hello README
$
```

The first command `trace 5 grep hello README` means you want to trace system call number 5 (i.e: read) for the program `grep hello README`. You can see 4 lines of output, so it called the `read` system call 4 times.
But the second command has no tracing output, because it did not set the tracing though the

`trace` user program.

Hint:

1. You might need to add an extra field in the `proc` structure (see `kernel/proc.h`) to remember which system call the process wants to trace.
2. The `syscall()` function in `kernel/syscall.c` is the point where the kernel decides which system call handler to invoke for a specific system call number, calls that handler and returns the return value. So it is a perfect place to modify to print the trace output. To print the `syscall` name, you might need to add an array of syscall names in that file to index into.

Task-2

Implement a system call named `sysinfo` that prints information about the current running system. When called from a user program, the syscall should print two information:

1. the number of free memory available in the system in bytes
2. the number of existing processes in the current system

An example format of the `sysinfo` syscall is given below (You don't have to match the formatting):

```
sysinfo system call prints:
free-memory: 133373952 bytes
n_proc  : 3
```

You will be given a user program `sysinfotest`. Make necessary changes so that you can use the given `sysinfotest.c` user program.
When you are done the output should be something like below: (The initial value might differ, but the change should be the same)

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ sysinfotest
sysinfotest: start

            Testing memory

Initial State

sysinfo system call prints:
free-memory: 133378048 bytes
n_proc  : 3

Using up one more page (4094 bytes).
Freemem should reduce by that much!

sysinfo system call prints:
free-memory: 133373952 bytes
n_proc  : 3
```

```
Giving back that one more page to the pool (4094 bytes).
Freemem should go back to the initial value!

sysinfo system call prints:
free-memory: 133378048 bytes
n_proc  : 3


            Testing nproc

Initial State

sysinfo system call prints:
free-memory: 133378048 bytes
n_proc  : 3

Created one new process. So nproc should increase by 1.
sysinfo system call prints:
free-memory: 133337088 bytes
n_proc  : 4

Created process ended. So nproc should go back to initial value.
sysinfo system call prints:
free-memory: 133378048 bytes
n_proc  : 3

sysinfotest: done
```

Hints:

1. The kernel maintains a linked-list (named `freelist`) of free memory pages in `kernel/kalloc.c` in the `kmem` struct.
2. xv6 has a fixed and hard-coded (see `NPROC` in `kernel/param.h`) maximum number of processes available. Each process has an associated state (see `procstate` enum in `kernel/proc.h`). The processes that are **not** in `UNUSED` state are the ones currently exists in the system.

General Guideline:

- Don't forget to acquire and release locks when needed, look out for the `proc` struct in `kernel/proc.h` and `kmem` struct in `kernel/kalloc.c`. You should look how other existing functions use the fields of those structs to get an idea. Remember xv6 is multi-core.

Attached Files

- trace.c
- sysinfotest.c

Submission Guideline:

Start with a fresh copy of xv6 from the original repository. Make necessary changes for this offline. In this offline, you will submit just the changes done (i.e: a patch file), not the entire repository.

Don't commit. Modify and create files that you need to. Then create a patch using the following command:

```
git diff HEAD > studentID.patch
```

Where studentID = your own six-digit student ID (e.g., 1805121).

Just submit the patch file, do not zip it. In the lab, during evaluation, we will start with a fresh copy of xv6 and apply your patch using the command:

```
git apply <studentID>.patch
```

Make sure to test your patch file after submission the same way we will run during the evaluation.

Please DO NOT COPY solutions from anywhere (your friends, seniors, internet, etc.). Any form of plagiarism (irrespective of source or destination), will result in getting -100% marks in this assignment. You have to protect your code.