Operating Systems 0512.4402

Homework 4: Adding system calls in xv6

In this exercise we'll add new system calls to xv6, which return information about processes to user space, and write a user space program, a simple version of `ps` in Linux, to display the list of processes.

We'll implement 3 new system calls:

- A system call `getNumProc()` that returns the total number of active processes in the system (either in embryo, running, runnable, sleeping, or zombie states).
- A system call `getMaxPid()` that returns the maximum PID amongst the PIDs of all currently active processes in the system.
- A system call `getProcInfo(pid, &processInfo)`. This system call takes as arguments an integer PID and a pointer to a structure `processInfo`. This structure is used for passing information between user and kernel mode. It should return 0 on success, or negative number on error (when a process with that PID does not exist).

The `processInfo` structure is defined as follows:

```
struct processInfo
{
    int state;    // process state
    int ppid;     // parent PID
    int sz;       // size of process memory, in bytes
    int nfd;      // number of open file descriptors in the process
    int nrswitch; // number context switches in
};
```

Define this sturcture in a file `processInfo.h`. Include this structure in `user.h`, so that it is available to userspace programs. You may also want to include this header file in `proc.c` to fill in the fields suitably. The information about the process that must be returned includes the process state (embryo, running, runnable, sleeping, or zombie), the parent PID, the process size in bytes, the number of open file descriptors, and the number of times the process was context switched *in* by the scheduler. Note that while some of this information is already available as part of the `struct proc` of a process, you will have to add new fields to keep track of some other extra information. Note that the parent PID of the init process (PID=1) is set to 0 by convention.

With all of these system calls put together, it is possible to iterate over all active processes in the system, and print their information to screen, just like the `ps` command does in Linux. Write a user space program `ps.c` that compiles into a user program `ps`.

When running `ps`, it should display:

```
Total number of active processes: %d\n
Maximum PID: %d\n

PID         STATE         PPID         SZ         NFD         NRSWITCH
```

And next it should display a list of all the active processes in the system, sorted by increasing PID number. For each process there should be a line containing the following fields (seperated by TAB and/or spaces such that the table is aligned): pid, state (write the string name: embryo, running, runnable, sleeping, or zombie), ppid, sz, nfd, nrswitch, each one below the column names defined above.

Note: It is important to keep in mind that the process table structure `ptable` is protected by a lock. You must acquire the lock before accessing this structure for reading or writing, and must release the lock after you are done. Please ensure good locking discipline to avoid bugs in your code.

Submission guidelines

- The solution should be submitted in moodle in a gzipped tar file called hw4_id1_id2.tgz, where id1 and id2 are the "tehudat zehut" of the two students (or hw4_id.tgz if submitting alone).
- The tgz file should contain a subdirectory hw4_id1_id2, and in the subdirectory include all the files that you modified in xv6, and any new files that you added, so that when we copy those files over the original public xv6 directory, the kernel will compile using make, and when running it under QEMU with make qemu-nox, the new ps user space command will be available in the filesystem.
- Submit an external documentation pdf in a file called hw4_id1_id2.pdf, summarizing the changes you made to xv6 files in your solution.