Project Three Report
Introduction to Operating Systems
New Beginnings Spring 2018

Gavin Megson

5 May 2018

Description

For this assignment, I modernized process management in xv6 by creating lists for every process state and modifying system calls to move processes onto and off of these lists when changing the state. Not having to search all of `ptable` for a type of process is more efficient and organized. In addition, I learned more about console command implementation by implementing more facilities for printing processes.

Deliverables

The following features were added to xv6:

- The process state lists were added to the `ptable` struct, comprising of six lists:
  - Free list: containing `UNUSED` processes
  - Embryo list: containing `EMBRYO` processes
  - Running list: containing `RUNNING` processes
  - Ready list: containing `RUNNABLE` processes
  - Sleep list: containing `SLEEPING` processes
  - Zombie list: containing `ZOMBIE` processes
- New console control commands were added to assist in displaying information relevant to these lists:
  - Ctrl-f: displays the number of `UNUSED` processes in `ptable`
  - Ctrl-r: displays the PIDs of all processes on the Ready list
  - Ctrl-s: displays the PID of every `SLEEPING` process
  - Ctrl-z: displays the PID and PPID of all `ZOMBIE` processes

Implementation

State Lists

A new struct `StateLists` was added to `ptable` in `proc.c` (lines 13 – 36):

```
struct StateLists {
    struct proc* ready;
        struct proc* readyTail;
        struct proc* free;
        struct proc* freeTail;
        struct proc* sleep;
        struct proc* sleepTail;
        struct proc* zombie;
        struct proc* zombieTail;
        struct proc* running;
        struct proc* runningTail;
        struct proc* embryo;
        struct proc* embryoTail;
};

struct {
```

```
    struct spinlock lock;
    struct proc proc[NPROC];
#ifdef CS333_P3P4
    struct StateLists pLists;
#endif
} ptable;
```

These represent head and tail pointers to linked lists for each process state.

- Helper functions for adding and removing from the state lists were provided (`proc.c` lines 935 – 992), as well as asserting a process is in the correct state (lines
- To support linked list iteration, the `proc` struct was given a next field (`proc.h` line 82) which points to the next process in a list. This is updated by the aforementioned helper functions.
- When `userinit` first runs, it initializes the process lists by calling a function to set each head and tail pointer in `pLists` to 0, and a function to set all processes to `FREE` (`proc.c` lines 149 – 152, 994 – 1029). During this time, the `ptable` lock is held to prevent a race condition.
- The lock is also held whenever processes are moved onto different lists.
- The general procedure for changing the state of a process has been updated as follows:
  - acquire lock, if necessary (some functions will have been called with an implicit lock)
  - remove of appropriate process list by calling helper function
  - assert the process is in the assumed state
  - change the process STATE field to desired state
  - add process to tail of appropriate process list by calling helper function
  - check the return code from helper functions and panic if there was a failure
  - release lock at appropriate time
- The following functions were changed in `proc.c` to implement this procedure:
  - `allocproc` (lines 74 – 77, 89 – 93, 102 – 110) will change a process from free to embryo, and back again if unsuccessful
  - `fork` (lines 229, 242 – 246, 276 – 280) will change a process from embryo to ready if successful, or embryo to free if unsuccessful
  - `exit` (lines 334 – 405) will change a process from running to zombie
    - note that `exit` also has changed from searching every possible process location in `ptable` to searching the relevant process lists for a child to pass to `init`
  - `wait` (lines 452 – 535) will change a process from zombie to free, if appropriate
    - note that `wait` also has changed from searching every possible process location in `ptable` to searching the relevant process lists for a child or zomibe process
  - `scheduler` (lines 592 – 637) will change a process from ready to running
  - `yield` (lines 672 – 676) will change a process from running to ready
  - `sleep` (lines 728 – 733) will change a process from running to sleeping
  - `wakeup1` (lines 762 – 780) will change a process on the appropriate channel from sleeping to ready

  - `kill` (lines 816 – 869), in addition to setting the killed flag, will change a process from sleeping to ready, if appropriate
    * note that `kill` also has changed from searching every possible process location in `ptable` to searching the relevant process lists for a process to kill

- System calls `kill`, `wakeup1`, and `wait` perform actions on a specific process, and so will remove a process from any point in the appropriate state list. All other functions preserve round robin functionality by taking the next process of the correct state from the head of the appropriate state list. All processes are added to the tail of the appropriate state list.
- Other than the above changes, the logic of process state change is unaltered.

New Control Commands

The following control commands were added:

- Ctrl-F prints the total number of free processes on one line to console
- Ctrl-R prints the PID of each ready process on the ready state list, in order from head to tail
- Ctrl-S prints the PID of each sleeping process on the sleep state list, in order from head to tail
- Ctrl-Z prints the PID and PPID of each zombie process on the zombie state list, in order from head to tail

The following files were modified to add the new control commands:

- `console.c`. Logic for recognizing the keyboard characters f, r, s, and z as control inputs was added (lines 207 – 222), and logic for calling the appropriate system call was added (lines 255 – 272).
- `proc.c`. Each control command has its own associated system call which acquires a lock on `ptable`:
  - `freedump` (lines 1087 – 1101) traverses the entire free list, incrementing a counter before printing the number of free processes
  - `readydump` (lines 1066 – 1085) traverses the entire ready list, printing each process sequentially
  - `sleepdump` (lines 1103 – 1122) traverses the entire sleep list, printing each process sequentially
  - `zombiedump` (lines 1124 – 1143) traverses the entire zombie list, printing each process sequentially

Testing

Free List Initialization

With `NPROC` set to 64, after `init` and `sh` begin running, I am going to display the free list size with `Ctrl-F` and the active processes with `Ctrl-P`.

`NPROC` set to 64, minus the two running processes, yields 62 free processes.

This test `PASSES`.

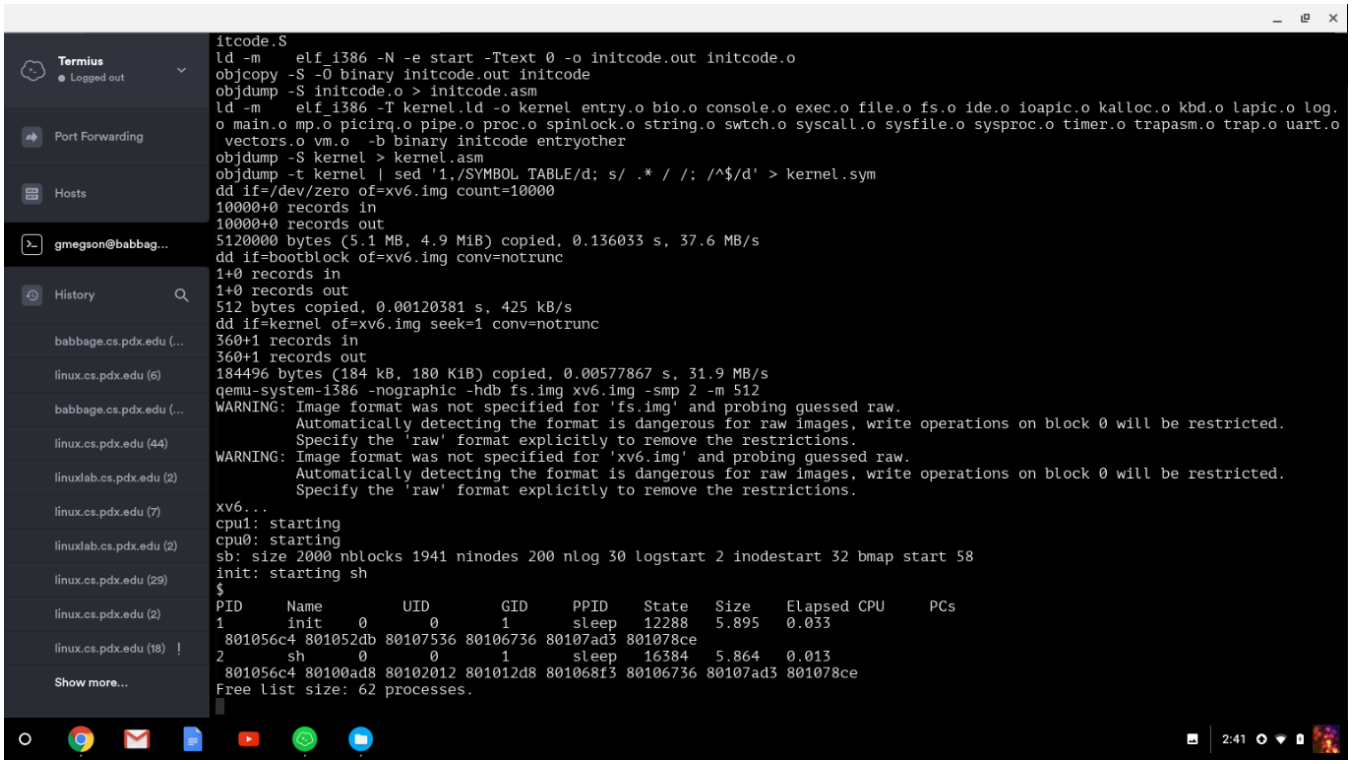

Figure 1: Free List Initialization

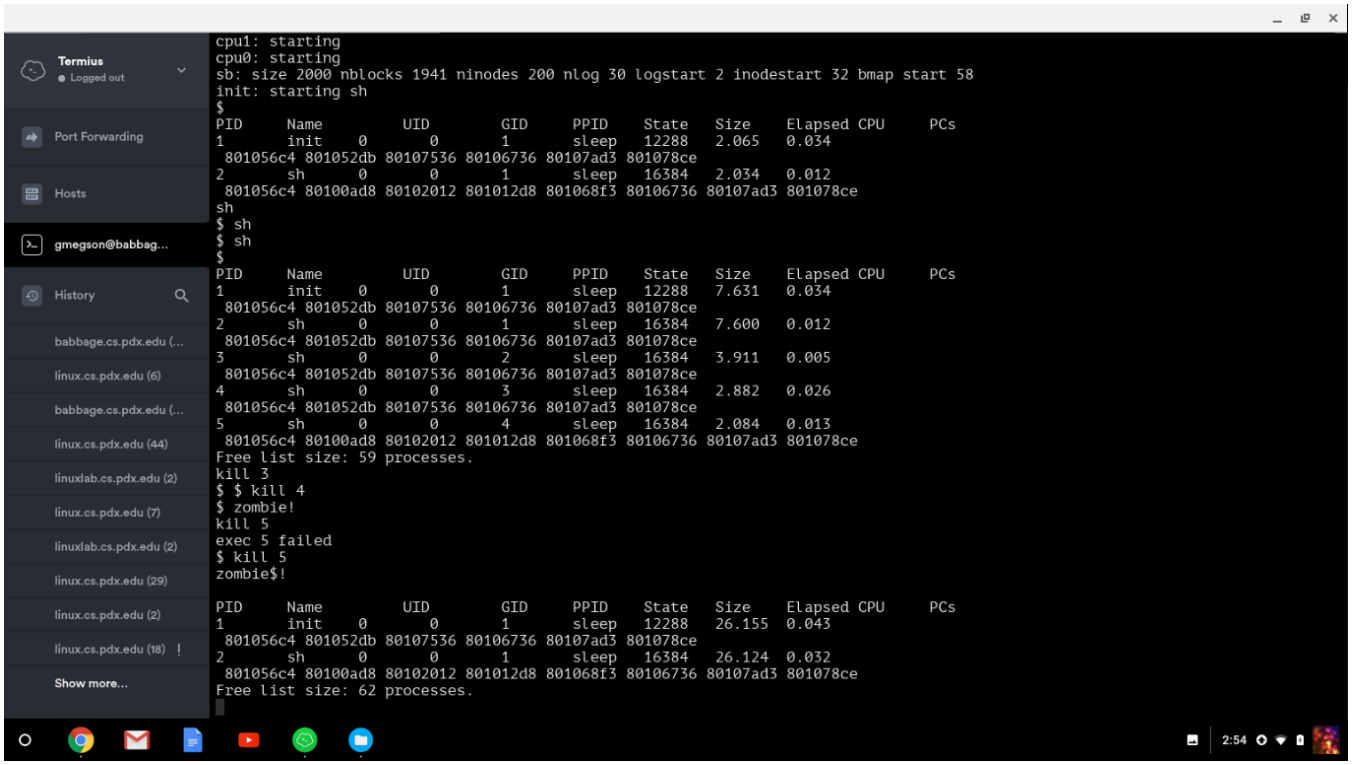

Figure 2: Free Allocation and Deallocation

Free List Allocation and Deallocation

I am going to create and then kill multiple shells and check the size of the free list at each step with `Ctrl-F` and `Ctrl-P`.

After creating 3 new shells, the free list correctly reduces to 59. After killing them off, the free list size is back to normal.

This test `PASSES`.

Sleep List

I am going to create and then kill multiple shells and check the printed sleep list at each step with `Ctrl-S` and `Ctrl-P`.

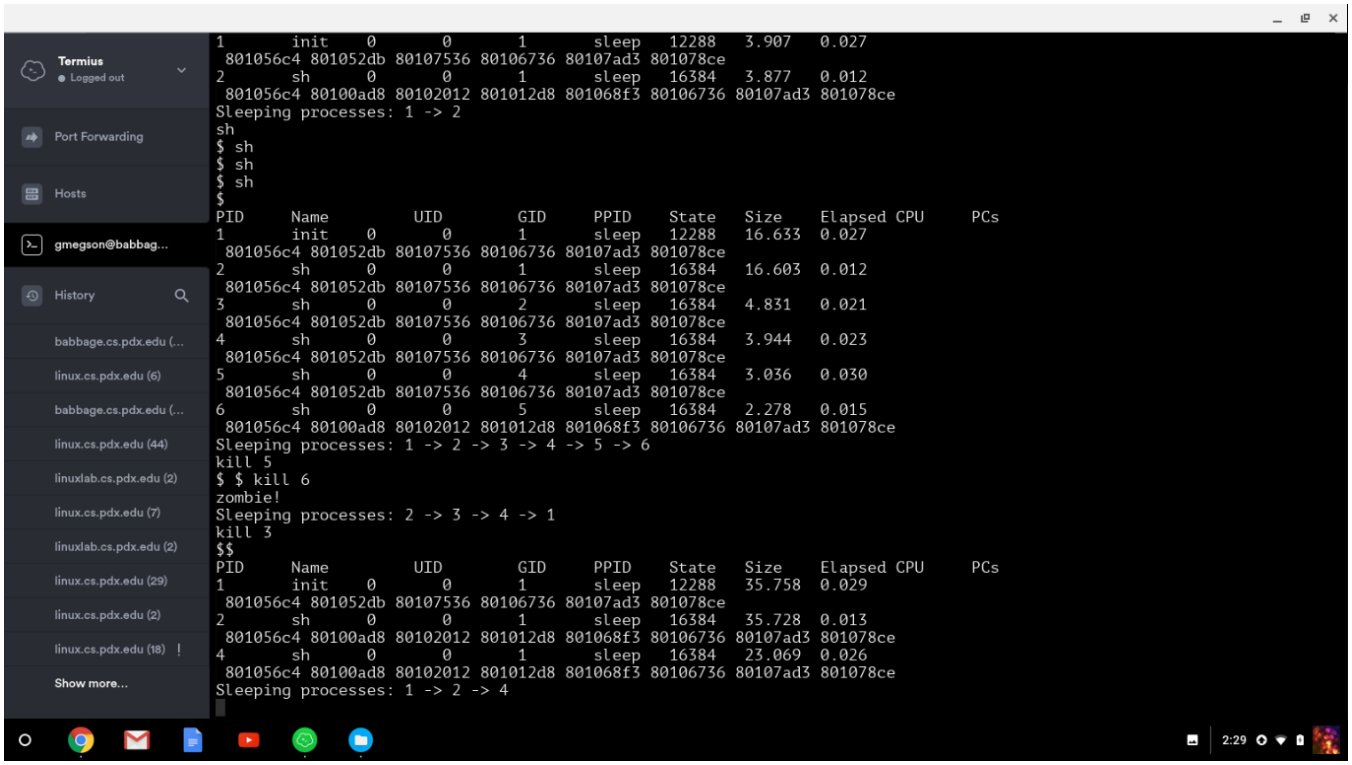

Figure 3: Sleep List

After creating and deleting new shells, the correct processes are shown to be sleeping at each step.

This test `PASSES`.