HW3 External Documentation

Adding system calls in xv6

Asaf Schwartz, i.d., 208436048

This exercise goal is to add system calls in xv6.

My method of doing so is to choose two similar system calls, one without arguments and the second with, and for adding the new command, also choosing an existing command, and then by grep command in the xv6 directory, find what files should be created/adjusted and how.

The chosen examples:

- The system call exit, which has no arguments, and does iterates the process table.
- The system call fstat (from sysfile.c), which has arguments, and one of them is a struct pointer.
- The command grep.

The following files were added/adjusted:

proc.c - the implementation of the process-related function, to be later used to implement the process-related system calls.
Added the functions:

- getNumProc - the function loops over the process table and count all of the processes which are not in a "unused" state.
  to loop over the process table without any synchronizations issues, the function acquire a lock before looping and releases it when it's done.
- getMaxPid – the function loops over the process table and look for the maximum pid among the processes which are not in a "unused" state.
- getProcInfo – the function checks the arguments are valid, and if so it searches the process in the process table.
  if not found – return -1
  if found – update the processInfo fields:
  state, sz, and ppid - already set in the process structure (just need to check if the pid == 1, then it's ppid is 0, as defined in the exercise)
  nfd – the process structure hold an array of file descriptors, but not necessarily the cells are not NULL, a function was added in proc.c to count the number of open FDs.
  nrswitch – the field is added to the process struct, it initialized as 0 in the process creation (in the allocproc() function) and it is updated every time the scheduler context switches into the current process (in the scheduler() function).

proc.h – header file contains process-related structures.
Added the field nrswitch to the proc structure.

processInfo.h – new header file which contains the new required structure.

syscall.c – a helper function to extract the system call's arguments, and system call declarations.
Added the declarations to the 3 added system calls.

syscall.h – the mapping of a system call to an index.
Added the new system calls, and their mapping is the continuation of the indices incrementation.

sysproc.c – the implementation of the system call's function, using the proc.c functions.
Added the three new system calls.

defs.h – header file which contains all function declarations in the kernel.
Added declarations of the three new functions, in the proc.c section.

usys.S -contains the system calls list, exported by the kernel and contain the corresponding traps.
Added the three new system calls.

ps.c – new c file, contains the function required ps function, it uses the 3 new system calls, for the two first prints getNumProc() and getMaxPid() are needed.
then to print the information of the active processes, a loop is iterating the incremented PIDs and the getProcInfo() system scall is used to get their data.

user.h – contains the list of system calls definitions.
Added the definitions of the new 4 system calls.

Makefile – added ps function, in the extra files and in UPROGS.

Results and Verifying Steps:

- When no background command are being held, when setting "ps", it present the 3 current processes – init, shell and the process running the "ps" command.
  They all have 3 open files – stdin, stdout and stderr.
  while "ps" process in RUNNING state, the two others are in SLEEPING mode.

```
Total number of active processes: 3
Maximum PID: 7

PID     STATE           PPID    SZ      NFD     NRSWITCH
1       SLEEPING        0       12288   3       17
2       SLEEPING        1       16384   3       61
7       RUNNING         2       12288   3       3
```

- Setting the command "ls > filelist.txt" in the background allows a check for the RUNNABLE state and for number of open files.

```
Total number of active processes: 4
Maximum PID: 5

PID     STATE           PPID    SZ      NFD     NRSWITCH
1       SLEEPING        0       12288   3       16
2       SLEEPING        1       16384   3       23
4       RUNNABLE        1       12288   4       1305
5       RUNNING         2       12288   3       64
```

Indeed the NFD incremented, and the state is RUNNABLE.

- For checking the ZOMBIE state, I added a command called zombiecreate, the function forks, the child is exiting immediately, and the father "picks him up" after sleeping for a period.
  creating a manually zombie allows the checking, since when a zombie is detected usually there's a policy of the init process to handle it.

zombiecreate.c > main()

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(){
    printf(1, "printingggg\n");
    int pid = fork();
    if(pid < 0) {
        printf(1, "zombiefailed\n");
        exit();
        }
    if(pid == 0){
        printf(1, "child exit\n");
        exit();
        }
    else {
        printf(1, "father sleep\n");
        sleep(1000);
        printf(1, "father wake\n");
        wait();
    }
    exit();

}
```

```
$ zombiecreate &
printingggg
father sleep
child exit
$ ps
Total number of active processes: 5
Maximum PID: 8

PID     STATE           PPID    SZ      NFD     NRSWITCH
1       SLEEPING        0       12288   3       25
2       SLEEPING        1       16384   3       24
6       SLEEPING        1       12288   3       117
7       ZOMBIE          6       12288   0       1
8       RUNNING         2       12288   3       10
$ father wake
zombie!
```

Added the command in the Makefile in the correct places, reminder – needs to exit from the function, rather than return, and the function is defined as int returning.

- Furthermore, for verifying during the implementation, I added prints that shows fields of the process structure, the pid number, state and the name of it.