Overview

In this homework we extended the xv6 operating system by adding three new system calls that expose process information from kernel space to user space. Using these system calls, we implemented a user-space program ps, similar to the Linux ps command, which displays information about all active processes in the system.

The solution includes kernel modifications, syscall plumbing, and a new user program.

Added System Calls

We implemented the following system calls:

1. getNumProc()
   Returns the total number of active processes in the system (processes that are not in the UNUSED state).
2. getMaxPid()
   Returns the maximum PID among all currently active processes.
3. getProcInfo(pid, &processInfo)
   Fills a processInfo structure with information about the process whose PID is pid. Returns 0 on success, or a negative value if the process does not exist.

processInfo Structure

A new structure processInfo was defined in a new header file:

- processInfo.h

This structure contains:

- process state
- parent PID
- process memory size (bytes)
- number of open file descriptors
- number of context switches (nrswitch)

The structure is shared between kernel space and user space.

Kernel Changes

proc.h / proc.c

- Added a new field nrswitch to struct proc to count the number of context switches.
- Updated the scheduler to increment nrswitch every time a process is scheduled.
- Implemented kernel helper functions that:

    - count active processes
    - find the maximum PID
    - fill a processInfo structure for a given PID
- All accesses to the process table are protected by ptable.lock.

System Call Infrastructure

syscall.h / syscall.c

- Added syscall numbers for the three new system calls.
- Registered the system calls in the syscall table.

sysproc.c

- Implemented syscall handlers that:
    - parse arguments from user space
    - call the corresponding kernel helper functions
    - return results or error codes

usys.S

- Added user-space syscall stubs so that user programs can invoke the new system calls.

user.h

- Declared the new system calls for user space.
- Exposed the processInfo structure to user programs.

User Program – ps

ps.c

- Implemented a user-space program ps.
- The program:
    - prints the total number of active processes
    - prints the maximum PID
    - iterates over all PIDs and retrieves process information using getProcInfo
    - displays the process list sorted by PID
- Output is formatted as a table containing:
  PID, state, PPID, memory size, number of open file descriptors, and number of context switches.

Build System

Makefile

- Updated to include ps in the list of user programs.
- Ensures ps is built and available in the xv6 file system.

Testing

After copying the modified and new files over a clean xv6-public tree:

- The kernel compiles successfully using make.
- xv6 boots correctly using make qemu-nox.
- Running the ps command inside xv6 displays the correct process information.

Summary

This assignment demonstrates how kernel information can be safely exposed to user space using system calls. The solution follows proper locking discipline, maintains xv6 coding conventions, and provides a functional user-space tool for process inspection.