CS 333: Operating Systems Lab, Autumn 2018

3rd September 2018

Lab Quiz #2, 25 marks

1. **process parade** (10 marks)

   - Add a new system call `int getps(struct ps_entry *)`, which takes a pointer to an array of type `struct ps_entry`. This call writes the process related information such as `PID, PPID`, size, state, and name of process for all processes in the system, and returns the number of processes. An array of `struct ps_entry` variables will be required to get information of all processes and has to be declared in the user process. The information tuple for all process is to be copied into the output variable sent as an argument to the system call.
   - Write a program `ps.c` that uses the above system call to get the details of all processes in the system, and prints them to the screen.
     Note: The process details should be printed in the user program and **not** in the system call.

2. **just share it!** (15 marks)
   Inter-process communication can be implemented using different mechanisms like pipe, signals, shared memory etc. As part of this exercise, we are to setup a simple method to share memory between processes.

   Implement a memory sharing interface between processes. Assume that the kernel supports a set of 10 pages (in total) that can be shared across processes. Each page is referred with a key to get access, e.g., `shm_get(key)` called from a process should return a process virtual address which is mapped to the page referred to by the `key`. Exit of processes should be handled to release physical pages if no virtual pages are mapped to them.

   To share a page between two virtual pages or more, the corresponding page tables should be updated in such a way that the page table entries of both the virtual pages map to the same physical page.

   Following changes will be needed (code segments for some of these are already added to source files and are commented),

   - Add new structure `struct shm_entry` to store the meta-data information of the shared pages. Meta-data such as number of virtual pages mapped (reference count) and kernel virtual address of the shared page. The kernel virtual address is required as the shared page is allocated and maintained by the kernel and the corresponding physical page is shared via user virtual addresses.
   - Add a new system call `shm_get(int key)` which returns a user-space virtual address corresponding to the page referred to by the `key`. Note that two or more processes that call `shm_get` with the same `key`, should be mapped to the same physical page (maybe at different local user space addresses) and hence share a page!
     Important:
     - The invariant to be maintained is that if no user virtual address is mapped to a shared page (corresponding to a `key`), no physical page should be allocated to the corresponding `key`.
     For example, on boot up, when no `shm_get` call has been made yet, no pages are allocated and maintained by the kernel for sharing. The first request for each key should allocate a physical page. If a key has no references, it should hold no memory.
     - Note that if the current address (`proc->sz`) is not page aligned, skip to the next page-aligned address (i.e., the address of first byte in the next page). This is required as all page table mappings are done at page granularity.
   - When a process terminates, de-allocation (`deallocuvm(...)`) in xv6, the reference count of each key, if mapped to virtual addresses of the process, should be decremented and shared pages freed if no mappings remain.
     Note that a single process may have multiple virtual address mappings to the same shared page or to different pages. For example, the following calls in a process can occur,
     `shm_get(1); shm_get(5); shm_get(1)`, and each of them will be associated with a different virtual address in the user space, irrespective of the shared physical mappings.

- A system call `shm_stat()` prints shared memory statistics (reference count and the kernel virtual address of the mapped shared physical page).
  Implementation of this function is provided in the lab archive.

Refer to the `README` file for more details. Check the tar file for sample testcases and outputs.

Deadline: 3rd September 5.00 p.m.
Submission via moodle.

Submission Guidelines

- All submissions via moodle. Name your submission as: <**rollno_labquiz2**>**.tar.gz**
- Execute the following command in the xv6 directory to create the submission archive.
  `make clean; tar -cvzf ~/rollno-labquiz2.tar.gz ./`
- Verify submission archive exists in the home directory.
- Upload on moodle.