Home work - Set2

August 29, 2017

1. Programming Assignment : A simple shell
   For this programming assignment, you need to use fork and execcvp. You can find detailed information of them in the man page. For example, to see how fork works, type man fork in the Linux terminal.

   Problem Statement:

   (a) Each time the shell is ready to read a command it should print a simple prompt. Make the prompt short and simple.

   (b) Command cd dir should set the current working directory to dir. Use system function chdir.

   (c) A command other than cd should be run by running the specified file. For example, command ls -l should run file /bin/ls with argument vector ["ls","-l"]. Do a fork and then an exec. Here are some particulars.

   - Break up the command at spaces or tabs. Several spaces in a row are equivalent to one space.
   - You need to look in more than one directory for commands. For this simple shell, only look in directory /bin and in the current directory for commands. But design your program to make the list of directories to search as easy to change as possible.
   - When you look for a command, use system function stat to check for the presence of the file, and to get its properties. Only take a file if it exists and has execute permission for the user (and is owned by the current user) or has execute permission for all users. Ignore the groups. Use function getuid to get the current user's id.

   (d) If a command ends on &, then run the command in background. Just fork it and continue running the shell. If the command does not end on &, then wait for the child process to finish before printing the next prompt.

2. Consider the following lines of code in a program running on xv6.
   int pid = fork();
   if(pid==0) //do something in child
   else //do something in parent

(a) When a new child process is created as part of handling fork, what does the kernel stack of the new child process contain, after fork finishes creating it, but just before the CPU switches away from the parent?

(b) How is the kernel stack of the newly created child process different from that of the parent?

(c) The EIP value that is present in the trap frames of the parent and child processes decides where both the processes resume execution in user mode. Do both the EIP pointers in the parent and child contain the same logical address? Do they point to the same physical address in memory (after address translation by page tables)? Explain.

(d) How would your answer to (c) above change if xv6 implemented copy-on-write during fork? When the child process is scheduled for the first time, where does it start execution in kernel mode? List the steps until it finally gets to executing the instruction after fork in the program above in user mode.