ENCE 360 Lab 8 – simple operating system xv6

For this weeks lab we're going to experiment with a toy operating system called xv6, designed for teaching purposes and written in ANSI C.

Compiling and running

Download and unzip the lab from learn. To compile simply use '*make*'

In order to run the operating system we use an emulator called qemu which emulates a processor, in our case since xv6 is compiled for an i386 processor we use qemu-system-i386 which we can specify like this:

export QEMU=qemu-system-i386

This sets a variable “QEMU” in bash that can be used by a subprocess like make.
To run the emulator use the command:

make qemu

If you make modifications to the user programs (e.g. cp.c) you'll need to re-run the emulator so that they're included in the filesystem.

There's a number of familiar shell commands and utilities present in xv6, as well as many which are not present! Here's an example of a shell session from qemu with xv6:

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 1929
cat            2 3 9548
echo           2 4 9081
forktest       2 5 5865
grep           2 6 10696
init           2 7 9402
kill           2 8 9077
ln             2 9 9063
ls             2 10 10643
mkdir          2 11 9150
rm             2 12 9131
sh             2 13 164 19
stressfs       2 14 9617
usertests      2 15 37440
wc             2 16 9886
zombie         2 17 8863
cp             2 18 9670
pipetest       2 19 9221
console        3 20 0
$ wc README
50 282 1929 README
$
$ cat README | wc
50 282 1929
```

```
$
$ mkdir Example
$
$ ls
.              1 1 512
..             1 1 512
README         2 2 1929
cat            2 3 9548
echo           2 4 9081
forktest       2 5 5865
grep           2 6 10696
init           2 7 9402
kill           2 8 9077
ln             2 9 9063
ls             2 10 10643
mkdir          2 11 9150
rm             2 12 9131
sh             2 13 164 19
stressfs       2 14 9617
usertests      2 15 37440
wc             2 16 9886
zombie         2 17 8863
cp             2 18 9670
pipetest       2 19 9221
console        3 20 0
Example        1 21 32
$
$ rm Example
$
$ echo "hello world"
"hello world"
$
$ echo "hello world" | wc
1 2 14
```

Xv6 comes with a vastly reduced number of system calls to Linux – and often simplified, a list of them can be seen inside 'user.h' which has the following contents:

```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(char*, int);
int mknod(char*, short, short);
int unlink(char*);
int fstat(int fd, struct stat*);
int link(char*, char*);
int mkdir(char*);
int chdir(char*);
int dup(int);
int getpid(void);
char* sbrk(int);
```

```
int sleep(int);
int uptime(void);

// ulib.c
int stat(char*, struct stat*);
char* strcpy(char*, char*);
void *memmove(void*, void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, char*, ...);
char* gets(char*, int max);
uint strlen(char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
```

Task 1 – implement cp

Xv6 comes with no command to copy files. Though you can copy files by doing something like this: '*cat src_file > dest_file*'.

We have setup a skeleton for cp which is compiled from '**cp.c**' in the base directory, if you need hints you can look at how the 'cat.c' is implemented – or any of the examples inside 'usertests.c'. Xv6 has no implementation of buffered IO or the stdio, so you have the lower level open/close and read/write system calls to use.

You can test your implementation by using 'cat' to examine the contents of a file. e.g.

```
$ echo "hello world" > my_file
$
$ cp my_file my_file2
$
$ cat my_file2
"hello world"
```

Or on larger files such as the 'README'.

Task 2 – pipes!

Just like linux, xv6 has pipes. We've written a sample pipes program in 'pipetest' which sits in the base directory as '**pipetest.c**' which shows how to use 'dup' to redirect input to the wc program.

Notice that there's no dup2 command, instead we close stdin using *close(0)* then when we call *dup(fd)* because have have no open file descriptor in slot 0 it becomes stdin similar to the way we used dup2 in Linux.

Modify this program so that it uses two way pipes to print the result of running wc in the parent process instead of letting it output to stdout directly.