COMP3500 – Frequently Asked Questions
Project 5 – Processes and System Calls

1. The files kern/include/file.h and kern/userprog/file.c do not currently exist. Am I to create these files or is there supposed to be existing files I should stub out?
   Answer: You need to create these two files.

2. What exactly I have to write in the structure of open file table for OS161?
   Answer: You may implement the openfile table in form of an array or singly linked list.

3. Do we create a system call implementation file for each system call?
   Answer: You may put all the implementations of the system calls related to files into file_syscall.c

   Similarly, we may place all the implementations of the process related system calls into a file named proc_syscalls.c

4. Where does the openfile struct go?
   Answer: The openfile struct can be placed in file.h

5. **Makefile Prolems.** We went back and redid all of the instructions in the "Project 4 Adding System Calls.pdf" one more time to make sure we didn't accidentally skip over anything. After doing that all over again we are still getting the same error. Also as a sanity check, we followed the instructions for setting up ASST2 based off of the method used for ASST1. This is the final error that we keep getting:

```
[group4@localhost ASST2]$ cd ~/cs161/src/testbin
[group4@localhost testbin]$ cp -r forktest getpidtest
[group4@localhost testbin]$ cd getpidtest
[group4@localhost getpidtest]$ mv forktest.c getpidtest.c
[group4@localhost getpidtest]$ gedit getpidtest.c
[group4@localhost getpidtest]$ gedit depend.mk
[group4@localhost getpidtest]$ gedit Makefile
[group4@localhost getpidtest]$ make
make: *** No rule to make target `/home/group4/cs161/root/include/unistd.h', nee
ded by `getpidtest.o'.  Stop.
[group4@localhost getpidtest]$
```

   It is looking in the root folder because that is where the OSTREE constant is based. Within the root folder, there is no "include" folder or any of the other subdirectories or files that should be in this location as depicted by the depend.mk file for getpidtest (based off of forktest).
   Answer: After create the new getpidtest directory from the old "forktest" folder, you MUST modify the makefile inside the new getpidtest directory.

   There are a total of six steps, in which you missed Step 4 (see instruction below).
   Step 4: Modify Makefile and depend.mk by replacing forktest with getpidtest

Other possible causes of this problem may include:

- File getpidtest.c doesn't exist
- You're in the wrong directory when you run make
- Check your Makefile

```
getpidtest.o: getpidtest.c getpidtest.h
        gcc -c getpidtest.c
```

Do not place any space before gcc

Please also update the makefile in the main testbin folder accordingly.

6. **Requirements Clarification.** Are we responsible for implementing the shell prompt from Project 4 Specification 6.1? Also, are we responsible for implementing a second scheduling algorithm as described in Project 4 Specification 6.2?

   Answer: The shell prompt has been implemented, meaning that you don't need to implement the shell. However, you should implement the following system calls (see also Section 4):

   - `open, read, write, lseek, close, dup2`
   - `getpid`
   - `fork, execv, waitpid, _exit`
   - `open(), read(), write(), lseek(), close(), dup2()`

   You should implement a second scheduling algorithm. If you have no time to implement this algorithm, please design your new scheduling algorithm. In your design document, you need to explain why you selected the algorithm, and under what conditions you expect your scheduling solution to perform well.

7. **Calling `thread_fork`.** I'm trying to implement fork by calling thread_fork, and looking at the slides for reference. In the slide the call is:

   ```
   thread_fork(curthread->t_name, ntf, 0, child_thread, retval);
   ```

   I'm assuming that ntf is the newly-created trap frame, but what is child_thread? The header file thread.h says that this argument is the function that the new thread starts execution in, but I don't know what that should be.

   Answer: `child_thread` in the parameter list of `thread_fork()` is a function, in which the newly created child thread starts execution. The source code of child_thread() function is given below.

```
static
void
child_thread(void *vtf, unsigned long junk)
{
 struct trapframe mytf;
 struct trapframe *ntf = vtf;
```

```
 (void)junk;

 /*
  * Now copy the trapframe to our stack, so we can free the one
  * that was malloced and use the one on our stack for going to
  * userspace.
  */

 mytf = *ntf;
 kfree(ntf);

 md_forkentry(&mytf);
}
```

8. **Where to put `sys_fork()`?** Do we put `sys_fork` and `sys_execv` and the like in `syscall.c`, and the `fork()`, `execv()` and such somewhere else?

   Answer: (1) You should place process related system calls (e.g., sys_getpid(), sys_fork() and sys_waitpid()) in proc_syscalls.c
   (2) Add "file userprog/getpid_syscall.c" into src/kern/conf/conf.kern. If you add other new source code files, please modify conf.kern in a similar way.
   (3) The prototype sys_getpid() and the other function prototypes starting with "sys_" should be declared in src/kern/include/syscall.h
   (4) sys_getpid() and the like are invoked in src/kern/arch/mips/mips/syscall.c
   (5) The implementation of fork() is accomplished by sys_fork() in the kernel.

9. **Edit Parameters for sys_fork()?** Are we allowed to edit the function parameters for some of the system calls, namely `sys_fork()` and `sys_execv()`?

   Answer: You can edit the function parameters. Please provide reasons in your design report.

10. **Undefined Reference.** I added the system call prototypes to the syscall.h file but I still get the following errors:

```
syscall.o(.text+0x138): In function `mips_syscall':
../../arch/mips/mips/syscall.c:84: undefined reference to `sys_execv'
syscall.o(.text+0x180):../../arch/mips/mips/syscall.c:88: undefined
reference to `sys_waitpid'
syscall.o(.text+0x1a8):../../arch/mips/mips/syscall.c:92: undefined
reference to `sys__exit'
```

    Answer: If `makefile` doesn't include your newly added source file that implements `sys_waitpid()`, you will receive the following error
    `"syscall.o(.text+0x180):../../arch/mips/mips/syscall.c:88: undefined reference to `sys_waitpid'`

After you add the new source code file, you must modify `conf.kern` and rebuild your kernel correspondingly. In particular, you must run `./config ASST2` to update the important `makefile`, which drives the make utility program.

11. **How to properly test our functions?** Is there a way we can properly test our functions such as `execv()`, `fork()`, and `waitpid()` so that we can know we wrote them correctly?

    Answer: Let's use the `getpid` system call as an example.

    (1) Create a User Program for the New System Call

    We place all the test programs in the following directory:

    ```
    ~/cs161/src/testbin
    ```

    Each test program and its associated files (e.g., Makefile) are organized in a dedicated directory. For example, test program `forktest.c` and its Makefile can be found in:

    ```
    ~/cs161/src/testbin/forktest
    ```

    In what follows, let us use `forktest` as a template to create a test driver for the `getpid` system call.

    Step 1: Create a new directory using forktest as a template:

    ```
    %cd ~/cs161/src/testbin
    %cp –r forktest getpidtest
    ```

    Step 2: Change source code name:

    ```
    %cd getpidtest
    %mv forktest.c getpidtest.c
    ```

    Step 3: **Important!** Modify getpidtest.c as follows. This program is quite simple; it calls the getpid system call and then shuts down OS/161.

    ```
    #include <unistd.h>
    #include <stdio.h>

    int main() {
          int mypid;

          mypid = getpid();
          /*
           * printf() does not work unless you have
           * implemented sys_write() */
          /* printf("My PID is: %d\n", mypid); */
          reboot(RB_REBOOT);
          return 0;
    }
    ```

    Step 4: Modify Makefile and depend.mk by replacing forktest with getpidtest

Step 5: Compile `getpidtest.c` using cs161-gcc. This can be done through running Makefile as below.

```
%make
```

The make utility program compile `getpidtest.c` and generate an execute file called `getpidtest`

Step 6: Copy the executable file `getpidtest` into `~/cs161/root/testbin`

```
%cp getpidtest ~/cs161/root/testbin/getpidtest
```

The above executable file will be loaded by OS/161 through the `p` command in the main menu.

(2) Run the User Program in OS/161

You can follow the instructions below to run the testing program created in Step 3.1:

```
%cd ~/cs161/root

%./sys161 kernel
```

Important! In the menu prompt type:

```
p /testbin/getpidtest
```

12. **`error: 'errno' undeclared`.** When compiling my code, I keep receiving a message "`error: 'errno' undeclared (first use in this function`". I have included errno.h. Is there another file I need to include in order to set the errno variable?

    Answer: All the error numbers (a.k.a., IDs) are defined in
    `/src/kern/include/kern/errno.h`
    If you try to use any error number that is not defined in the above file, you will receive a message "error: 'errno' undeclared (first use in this function". When you add a new error number into errono.h, please update the corresponding error messages in:
    `/src/kern/include/kern/errmsg.h`

13. **`addrspace.c`.** The file addrspace.c contains methods that are necessary to finish Project 4 (e.g., `as_copy, as_create`). but was never talked about in class. How do I implement this file?

    Answer: You don't have to implement these functions, because the address space related functions already exist. The prototype of `as_copy()` can be found in
    `kern/include/addrspace.h`
    The implementation of `as_copy()` is in the following source code file (see Line 277).
    `kern/arch/mips/mips/dumbvm.c`