1 User Level System Calls

1.1 Create a new Process

```
int p_spawn(char *name, void *func, int argc, char **argv);
```

Creates a new process.
Takes in the name of the process, the function it will run, the number of arguments that function will take in, and an array of char*s which holds the function arguments.
If the name is null, it will set the *os_errno* to ENAME.
If it runs successfully it will return the pid of the spawned process.

1.2 Send a Process Signals

```
void p_kill(int pid, int sig);
```

Sends a signal to a process. The call takes in the process ID pid of the process that we are sending the signal to and the signal number pid. The signals are:
S_SIGSTOP : Stops the process.
S_SIGTERM : Termiantes the process.
S_SIGCONT : Continue the process if stopped.
These signals will be ignored if the process is masking them. If a signal other than the above are specified, then it will set the `os_errno` to `ESIGNAL`.

1.3 Wait for Children

```
int p_wait(int mode, int *status);
```

Waits for a child process' status to change. Takes as arguments the mode: `0` to block this process until a child has changed, or 1 to harvest a zombie child without blocking. The other argument it takes is a pointer to a status that will be set once a child changes state to indicate what happened.

The process will first see if there are any zombie children to harvest, and if so, will terminate one and return its pid. Otherwise, if the NOHANG flag is set, it returns. If not, it waits for a child to change state, has the status set (to indicate how the child changed state), and returns the pid of the child that changed state.

os_errno will be set to ESTATE if once the process unblocks, the pid of the child that unblocks it is not consistent in the state.

1.4 Ending a Process

```
void p_exit();
```

Exits the current thread. If the process has any children, it makes them orphans and terminates them. If the process's parent is the OS, then terminate the process. Otherwise, make it a zombie (and put it on its parent's zombie queue).

1.5 Changing the Priority of a Process

```
void p_nice(int pid, int priority);
```

Changes the prioirity of a process for scheduling. Removes it from the queue it used to be in and enqueues it on the proper prioirity queue. At the end of this function, swaps contexts to the OS for scheduling.

1.6 Getting Information about a Process

```
info_t *p_info(int pid);
```

Takes in the pid of a process and returns a struct with its information (the name, pid, status, and priority).

1.7 Making a Process Sleep

```
void p_sleep(int ticks);
```

Blocks the process until the given number of clock ticks has passed. If the process is stopped while sleeping, it will continue to count down until the allotted number of ticks. If it is continued after the number of clock ticks is up, then it will immediately unblock.

1.8 Getting all pids

```
int *p_getpids();
```

Returns an array of all the pids of the currently running processes.

1.9 Getting information from errno

```
void p_error();
```

Prints the *os_errno* as a human readable message.

1.10 Opening A File

```
void f_open(const char *fname, int mode);
```

Open file fname with mode and return a file descriptor. mode can be: F_WRITE, F_READ, F_APPEND.

1.11 Reading a File

```
int f_read(char *buffer, int fd, int n);
```

Reads n bytes from file referenced by fd into buffer. Will read until null terminator or n bytes. If no EOF encountered, will not return null terminated string.

1.12 Writing A File

```
int f_write(int fd, const char *str, int n);
```

Writes n bytes of str to file described by fd. Requires null terminated string str.

1.13 Closing A File

```
int f_close(int fd);
```

CLoses the file descriptor associated with this file.

1.14 Removing A File

```
void f_unlink(const char *fname);
```

Removes the file from the directory and frees its memory.

1.15 Seeking File Pointer

```
void f_lseek(int fd, int offset, int whence);
```

Repositions fd to the offset relative to whence.

1.16 Gets all File Names

```
void f_get_files(char **file_array);
```

Given a char**, populates it with the names of all files in the directory

1.17 Gets Size of File

```
int f_get_size(char *fname);
```

Get the size of a file given the name of the file. -1 if no such file.

1.18 Get Number of Files

```
int f_get_num_files();
```

Get the number of files in the file directory