```
pid_t fork(void);
```

Description: `fork` creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

Returns: On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution.

```
int execve(const  char  *filename,  char  *const  argv [], char *const envp[]);
```

Description: `execve()` executes the program pointed to by filename. filename must be either a binary executable, or a script. `argv` is an array of argument strings passed to the new program. `envp` is an array of strings, conventionally of the form key=value, which are passed as environment to the new program.

Returns: `execve()` does not return on success, and the text, data, bss, and stack of the calling process are overwritten by that of the program loaded.

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Description: The `wait` function suspends execution of the current process until a child has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function.

The `waitpid` function suspends execution of the current process until a child as specified by the pid argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function.

If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

The value of pid can be one of:

- <-1, which means to wait for any child process whose process group ID is equal to the absolute value of `pid`.
- -1, which means to wait for any child process; this is the same behaviour which wait exhibits.
- 0, which means to wait for any child process whose process group ID is equal to that of the calling process.
- >0, which means to wait for the child whose process ID is equal to the value of `pid`.

The value of `options` is an OR of zero or more of the following constants:

- `WNOHANG`, which means to return immediately if no child has exited.
- `WUNTRACED`, which means to also return for children which are stopped (but not traced), and whose status has not been reported. Status for traced children which are stopped is provided also without this option.

If `status` is not `NULL`, `wait` or `waitpid` store status information in the location pointed to by `status`.

Returns: The process ID of the child which exited, or zero if `WNOHANG` was used and no child was available, or -1 on error (in which case `errno` is set to an appropriate value).

```
int kill(pid_t pid, int sig);
```

Description: The kill system call can be used to send any signal to any process group or process.

Returns: On success, zero is returned. On error, -1 is returned, and `errno` is set appropriately.

```
int open(const char *pathname, int flags);
```

Description: The `open()` system call is used to convert a pathname into a file descriptor (a small, non-negative integer for use in subsequent I/O as with read, write, etc.). When the call is successful, the file descriptor returned will be the lowest file descriptor not currently open for the process.

The parameter flags is one of `O_RDONLY`, `O_WRONLY` or `O_RDWR` which request opening the file read-only, write-only or read/write, respectively

Returns: `open` returns the new file descriptor, or -1 if an error occurred (in which case, `errno` is set appropriately).

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

Description: `dup` and `dup2` create a copy of the file descriptor `oldfd`.

After successful return of `dup` or `dup2`, the old and new descriptors may be used interchangeably. They share locks, file position pointers and flags; for example, if the file position is modified by using `lseek` on one of the descriptors, the position is also changed for the other.

`dup2` makes `newfd` be the copy of `oldfd`, closing `newfd` first if necessary.

Returns: `dup` and `dup2` return the new descriptor, or -1 if an error occurred (in which case, `errno` is set appropriately).

```
off_t lseek(int fildes, off_t offset, int whence);
```

Description: The `lseek` function repositions the offset of the file descriptor `fildes` to the argument offset according to the directive whence as follows:

- `SEEK_SET` The offset is set to offset bytes.
- `SEEK_CUR` The offset is set to its current location plus offset bytes.
- `SEEK_END` The offset is set to the size of the file plus offset bytes.

Returns: Upon successful completion, `lseek` returns the resulting offset location as measured in bytes from the beginning of the file. Otherwise, a value of `(off_t)-1` is returned and `errno` is set to indicate the error.

`/dev/null` - The null device is a special file that discards all data written to it (but reports that the write operation succeeded), and provides no data to any process that reads from it (yielding EOF immediately)

`/dev/zero` - All reads on `/dev/zero` return as many null values (ASCII NUL, 0x00) as characters requested. Like `/dev/null`, `/dev/zero` acts as a source and sink for data. All writes to `/dev/zero` succeed with no other effects.