1   Terminal Commands

Summary:

Terminal Command

`strace <PROGRAM>`

- Trace all the system calls a process makes on Linux.

`valgrind <executable>`

- Detect memory leaks from `malloc` and `free`.

`-Db_sanitize=address`

- Detect memory leaks by adding this flag to Meson.

`htop`

- Process tree. Use F5 to swtich b/w tree and list view.

2   Functions

Summary:

```
int fork();
```

- Creates a new process that's a clone of the currently running process. In the original process, it returns the process ID (pid) of the newly created child process. In the child process, it returns 0.

```
int execlp(const char *file, const char *arg, ...);
```

- Replaces the current process with a new program specified by `file`. The new process is given the command-line arguments specified by `arg` and any additional arguments. Returns only if there is an error.

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

- Duplicates the file descriptor `oldfd` to `newfd`. If `newfd` is already open, it is first closed. Returns the new file descriptor on success.

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

- Waits for a specific child process (`pid`) to change state. The state change is stored in `status`. The `options` argument can modify the behavior of `waitpid`, use 0 for the defaults (blocking). Returns the pid of the child process on success.

```
int pipe(int pipefd[2]);
```

- Creates a unidirectional data channel. `pipefd[0]` is set up for reading, and `pipefd[1]` is set up for writing. Returns 0 on success.

```
void exit(int status);
```

- Terminates the calling process with an exit status of `status`.

```
ssize_t write(int fd, const void *buf, size_t count);
```

- Writes `count` bytes from `buf` to the file or device associated with `fd`. Returns the number of bytes written.

```
ssize_t read(int fd, void *buf, size_t count);
```

- Reads up to `count` bytes from the file or socket associated with `fd` into `buf`. Returns the number of bytes read.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine)(void *), void *arg);
```

- Creates a new thread with attributes specified by `attr`. The new thread starts execution by invoking `start_routine` with `arg` as its argument. Returns 0 on success.

```
void pthread_exit(void *retval);
```

- Terminates the calling thread, returning `retval` to any joining thread.

```
int pthread_join(pthread_t thread, void **retval);
```

- Waits for the thread specified by `thread` to terminate. The thread's return value is stored in `retval`. Returns 0 on success.

```
int pthread_detach(pthread_t thread);
```

- Detaches the specified thread, so its resources can be reclaimed immediately upon termination. Returns 0 on success.

```
atexit(void (*function)(void));
```

- Register functions to call on program exit.

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

- Replaces the current process with a new program and resets.

- pathname: Full path of the program to load.
- argv: Array of strings (array of characters), terminated by a null pointer. Represents arguments to the process.
- envp: Array of strings (array of characters), terminated by a null pointer. Represents the environment variables of the process.
- Returns only if there is an error.

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

- Opens a file specified by pathname with the specified flags. Returns a file descriptor on success.

```
int close(int fd);
```

- Closes the file descriptor fd. Returns 0 on success.

```
DIR *opendir(const char *name);
```

- Opens the directory specified by name for reading. Returns a pointer to a DIR structure on success.

```
int closedir(DIR *dirp);
```

- Closes the directory stream pointed to by dirp. Returns 0 on success.

```
ssize_t read(int fd, void *buf, size_t count);
```

- Reads up to count bytes from the file descriptor fd into the buffer buf. Returns the number of bytes read on success or -1 on error.

```
void perror(const char *s);
```

- Prints a descriptive error message to stderr, prefixed by the string s, based on the current value of errno.

```
void exit(int status);
```

- Terminates the calling process with the specified status. Use EXIT_SUCCESS or EXIT_FAILURE for standard status codes.

3 Operating System Structure

3.1 Different Types of Kernels

4 Processes

> **Definition**: An instance of running a program.

5 Threads

6 Synchronization

7 CPU Scheduling

8 Memory Management

9 File Systems

10 I/O