Summary:
- what is the code doing and why? How would I fix it? What happens if I change this.

1 Terminal Commands

Summary:

Terminal Command
`readelf -a <FILE>`   • See the ELF header of a file.
`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 C Programming

Summary:

C Programming Syntax

`static`

- Only able to use the global variable in the current C file.

`static`

- Only able to use the global variable in the current C file.

`static`

- Only able to use the global variable in the current C file.

3 Functions

Summary:

```c
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.

```c
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.

```c
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.

```c
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.

```c
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.

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

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

```c
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.

```c
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.

```c
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.

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

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

```c
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.

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

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

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

- Register functions to call on program exit.

```c
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.

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

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

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

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

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

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

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

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

```c
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.

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

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

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

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

4 Operating System Structure

4.1 3 Operating System Concepts

Definition:
1. **Virtualization:** Share one resource by mimicking multiple independent copies.
2. **Concurrency:** Handle multiple things happening at the same time.
3. **Persistence:** Retain data consistency even without power.

4.2 Different Types of Kernels

5 Processes

Definition: An instance of running a program.
- **Program:** A file containing all the instructions and data required to run.

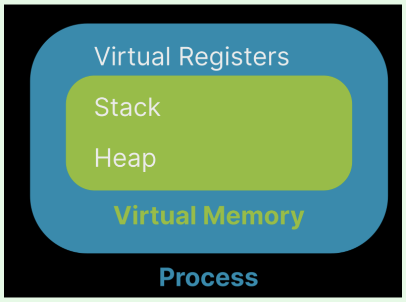

Figure 1: Process

5.1 Hello World

Example:

6 Threads

7 Synchronization

8 CPU Scheduling

9 Memory Management

10 File Systems

11 I/O