The basics of programming in C

```
for (;;) {
    n = readlink(pathname, buff->mem, buff->cap - 1);
    if (n != (ssize_t)(buff->cap - 1))
        break;
    string_buff_grow(buff);
}
```

```
for (;;) {
    n = readlink(pathname, buff->mem, buff->cap - 1);
    if (n != (ssize_t)(buff->cap - 1))
        break;
    string_buff_grow(buff);
}
```

Do not make things more complicated than needed.

Read `/src/linux/fs/namei.c:getname_flags()`. In Linux there is no way to create a symlink that is longer than `4096 (= PATH_MAX)` bytes.

```
assert(close(currdir) >= 0);
```

```
assert(close(currdir) >= 0);
```

```
close(curdir);
```

Second, it is simpler.

```
assert(close(currdir) >= 0);
```

```
close(curdir);
```

Second, it is simpler.

First, `assert()` is a macros that can expand to an empty string.

```
% cat assert.c
#include <stdio.h>
#include <assert.h>

int main()
{
   assert(printf("hello, world\n") >= 0);
   return 0;
}

% gcc -DNDEBUG assert.c

% ./a.out

%
```

```
char* token = strtok(temp_path, "/");
while (token != NULL) {
    …
```

```
char* token = strtok(temp_path, "/");
while (token != NULL) {
    …
```

This is not thread-safe.

Glibc has `strtok_r()`, but it is better to avoid this family of functions altogether.

```
for (i = 0; i < QUEUE_DEPTH; i++) {
    sqe = io_uring_get_sqe(&ring);
    if (!sqe) {
        ret = -EAGAIN;
        goto cleanup;
    }
    ...
```

```
for (i = 0; i < QUEUE_DEPTH; i++) {
    sqe = io_uring_get_sqe(&ring);
    if (!sqe) {
        ret = -EAGAIN;
        goto cleanup;
    }
    …
```

```
for (i = 0; i < QUEUE_DEPTH; i++) {
    sqe = io_uring_get_sqe(&ring);
    … sqe is guaranteed to be non-NULL …
```

Don’t make things more complex than needed.

`io_uring_get_sqe()` fails only if there are no more entries in the submission queue.

```
int blocks_count = file_sz / BLOCK_SZ;
if (file_sz % BLOCK_SZ)
    blocks_count++;
```

```
int blocks_count = file_sz / BLOCK_SZ;
if (file_sz % BLOCK_SZ)
    blocks_count++;
```

```
int blocks_count = (file_sz + BLOCK_SZ - 1) / BLOCK_SZ;
```

```
struct io_data {
    char   *buf;
    size_t size;
    off_t  offset;
    int    read_done;
};

...

for (i = 0; i < QUEUE_DEPTH; i++) {
    data[i] = malloc(sizeof(struct io_data));
    data[i]->buf = malloc(COPY_BLOCK_SIZE);
    data[i]->offset = i * COPY_BLOCK_SIZE;
    data[i]->read_done = 0;
}
```

```
struct io_data {
   char   *buf;
   size_t size;
   off_t  offset;
   int    read_done;
};

...

for (i = 0; i < QUEUE_DEPTH; i++) {
   data[i] = malloc(sizeof(struct io_data));
   data[i]->buf = malloc(COPY_BLOCK_SIZE);
   data[i]->offset = i * COPY_BLOCK_SIZE;
   data[i]->read_done = 0;
}
```

```
struct io_data {
   off_t  offset;
   int    size;
   bool   read_done;
   char   buf[];
};

...

for (i = 0; i < QUEUE_DEPTH; i++) {
   data[i] = xmalloc(sizeof(*data[i]) + COPY_BLOCK_SIZE);
   data[i]->offset = i * COPY_BLOCK_SIZE;
   data[i]->read_done = 0;
}
```

This is called **flexible arrays**.

```
struct io_data {
   char   *buf;
   size_t size;
   off_t  offset;
   int    read_done;
};

...

for (i = 0; i < QUEUE_DEPTH; i++) {
   data[i] = malloc(sizeof(struct io_data));
   data[i]->buf = malloc(COPY_BLOCK_SIZE);
   data[i]->offset = i * COPY_BLOCK_SIZE;
   data[i]->read_done = 0;
}
```

```
struct io_data {
   off_t  offset;
   int    size;
   bool   read_done;
   char   buf[] __attribute__((__element_count__(size)));
};

...

for (i = 0; i < QUEUE_DEPTH; i++) {
   data[i] = xmalloc(sizeof(*data[i]) + COPY_BLOCK_SIZE);
   data[i]->offset = i * COPY_BLOCK_SIZE;
   data[i]->read_done = 0;
}
```

GCC 15 is going to have bound checks on flexible arrays.
See also: https://lwn.net/Articles/908817/

```
static char is_directory(const char *path) {
   struct stat path_stat;

   if (stat(path, &path_stat) == -1)
      return 0;

   return S_ISDIR(path_stat.st_mode);
}

static char is_link(const char* child) {
   …
}

…

char is_link_result = is_link(child);

if (is_link_result > 0) {
   …
} else {
   …
   if (is_directory(state.path)) {
      append_dir();
   …
```

```
static char is_directory(const char *path) {
   struct stat path_stat;

   if (stat(path, &path_stat) == -1)
      return 0;

   return S_ISDIR(path_stat.st_mode);
}

static char is_link(const char* child) {
   …
}

…

char is_link_result = is_link(child);

if (is_link_result > 0) {
   …
} else {
   …
   if (is_directory(state.path)) {
      append_dir();
   …
```

These system calls are redundant. Avoid this because a system call is an expensive operation. It used to be a context switch. Nowadays, it switches the context, switches the page tables, flushes caches, etc.

It suffices to `stat()` the current file only once.

```
struct dirwalk
{
   char **components;
   int  len;
   int  cap;
   int  fd;
};

int dirwalk_enter_one(struct dirwalk *d, const char *name)
{
   int fd = openat(d->fd, name, O_RDONLY|O_NOFOLLOW);
   … this is enough to detect symlinks and –ENOTDIR …
}

int dirwalk_enter_path(struct dirwalk_d, const char *path)
{
   … split path and dirwalk_enter_one() …
}
```