Besprechung der `halde`[1]

Tafelübung Systemprogrammierung

Andreas Schärtl <andreas.schaertl@fau.de>

SS 2021

[1] https://www.schaertl.me/sp1

Besprechung

Besprechung

- Jetzt Besprechung einer beispielhaften Aufgabe.
- Be excellent to each other.

Overflow

Overflow

- Selbst `size_t` mit seinen 64 Bit (auf amd64) kann überlaufen.
- Zum Beispiel passiert das in `calloc`!

Behandeln[2]

```
void *calloc(size_t m, size_t n) {
    void *p;
    size_t *z;
    if (n && m > (size_t)-1/n) {
        errno = ENOMEM;
        return 0;
    }

[...]
```

[2] Aus: musl libc: `https://git.musl-libc.org/cgit/musl/tree/src/malloc/calloc.c?h=rs-1.0`

Fehlerbehandlung mit `errno`

Fehlerbehandlung mit `errno`

- Viele Funktion geben ihren genauen Fehler nicht genau zurück.
  - Zum Beispiel `fgets` mit `NULL` oder `chdir` mit `-1`.
- Was genau passiert ist steht in der globalen[3] Variable `errno`. Der Typ von `errno` ist `int`.
- Viele vordefinierte Fehlercodes wie `ENOMEM`, `EINVAL` oder `ENOENT`
  - Nachschlagbar zum Beispiel mit `man 3 errno`.
- Der Ansatz mit der globalen Variable ist heute nicht bei allen beliebt[4] aber auch nicht unüblich.

[3] und *thread-local*

[4] `https://stackoverflow.com/a/42622258`

Fehlerbehandlung mit errno

```
void perror(const char *prefix) {
    int my_errno = errno;  // read global variable
    const char *error_string = strerror(my_errno);
    fprintf(stderr, "%s: %s\n", prefix, error_string);
}
```

- Funktionen wie perror lesen errno selber auch nur aus.
- Fehlerbehandlung so früh wie nur möglich! Aufruf von Bibliotheksfunktionen könnte die errno ändern.

Fehlerbehandlung mit `errno`

- Viele Funktionen garantieren nur, dass sie *im Fehlerfall* die `errno` setzen. Was im Erfolgsfall passiert ist gerne mal nicht definiert. Au weia!
- Im Zweifelsfall vorher `errno = 0` setzen.

```
const struct passwd *get_user(uid_t uid) {
    errno = 0;  // required!
    const struct passwd *user_info = getpwuid(uid);
    if (!user_info) {
        if (errno == 0) {
            fprintf(stderr, "user with uid=%d not found\n", uid);
            exit(EXIT_FAILURE);
        } else {
            perror("getpwuid");
            exit(EXIT_FAILURE);
        }
    }
    return user_info;
}
```

Coole Typen: `bool`

Coole Typen: `bool`

- Sehr alte Versionen von C hatten keinen Typ `bool`. Man nahm stattdessen einfach `int` o.Ä.
- Seit C99 gibt es aber den Typ `bool` mit den bekannten Werten `true` und `false`.
- Header `stdbool.h` muss inkludiert werden.

Coole Typen: `size_t`

Coole Typen: `size_t`

- Es gibt mehr als nur einen schnöden `int`.
- Besonders Sinnvoll: `size_t` ist ein `unsigned` Typ für Größenangaben jeder Art.
  - Funktionen wir `strlen` oder `strcpy` arbeiten alle mit Typ `size_t`.

Das ist wirklich wichtig!

- `INT_MAX` auf amd64 ist 2.147.483.647! Das reicht gerade mal um 2 GiB zu adressieren. Der `int` ist also total ungeeignet für Speicheroperationen in portablen Programmen.

Lokale `static` Variablen

Lokale `static` Variablen

```
int hashmap_get(keytype k) {
  static bool is_initialized = false;
  if (!is_initialized) {
        initalize();
        is_initialized = true;
  }

  return get(k);
}
```

- Lokale Variablen können `static` sein.
- Sie verhalten sich dann wie globale Variablen, sind aber im Scope auf eine Funktion begrenzt.

`make` und die Flags

`make` und die Flags

- Im `Makefile` beschreibt man die einzelnen Schritte, die zum Bauen eines Projektes notwendig sind.
  - Typischerweise teilt man das ganze in zwei Kategorien ein.
  - `*.c`-Datei $\rightarrow$ `*.o`-Datei (*Übersetzen*)
  - `*.o`-Dateien $\rightarrow$ Programm, auf Windows `*.exe` (*Linken*)
- Für das Übersetzen sind die `CFLAGS` zuständig.
- Für das Linken die `LDFLAGS`.

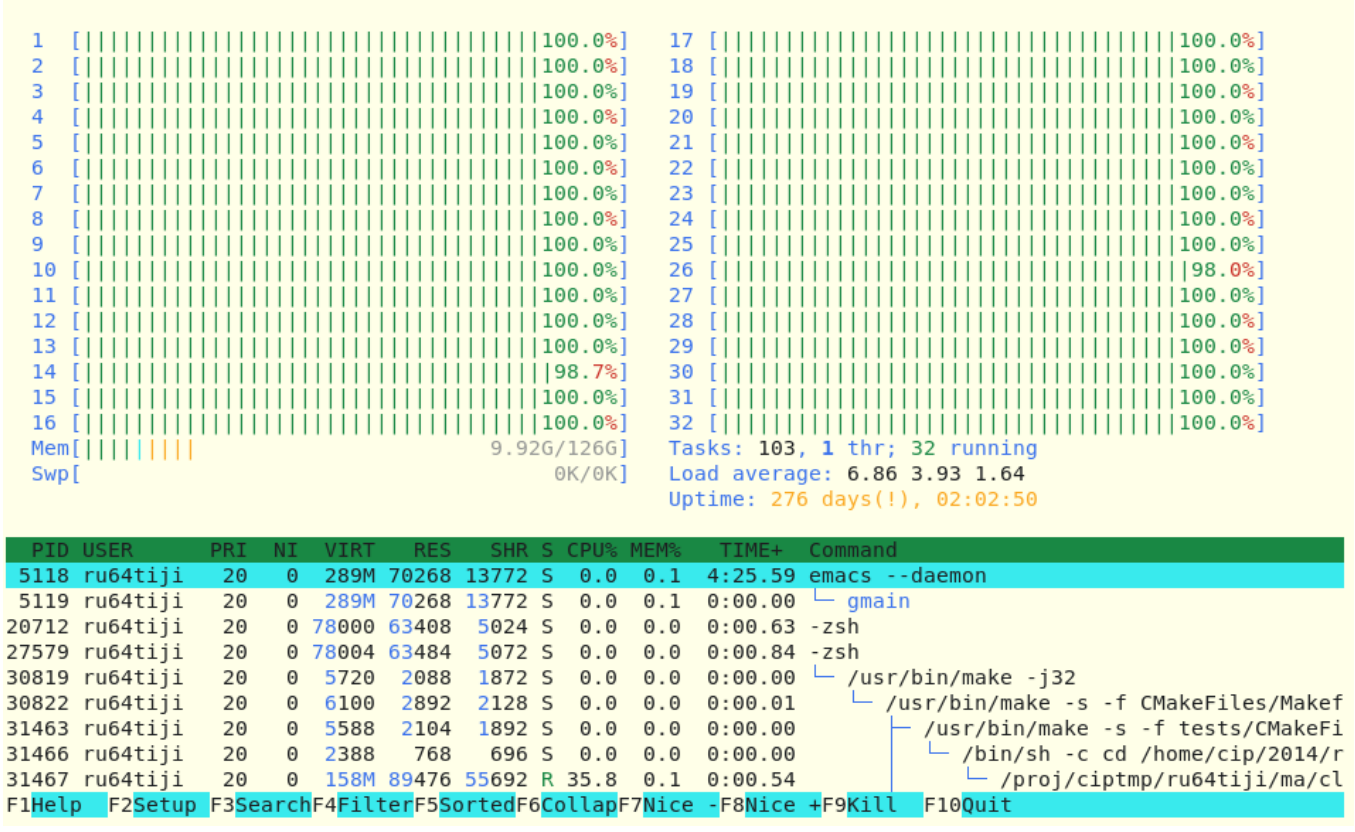
Aufgepasst

- `CFLAGS` braucht es also beim Linken nicht.
- `LDFLAGS` braucht es beim Übersetzen nicht.

`make` und die Flags

- Wichtig sind die Abhängigkeiten. Header nicht vergessen!
- `make` baut dann nur das neu, was neu gebaut werden muss.
- Sind alle Abhängigkeiten richtig, könnt ihr mit `make -jN` euer Projekt parallel mit `N` Threads ausführen. Cool!

make und die Flags



→ `make` und vergleichbare Tools sind clever und sparen viel Zeit.
Man muss sie aber richtig konfigurieren.

Fragen?