34 Organisation of Memory

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Summer Term 2024

http://sys.cs.fau.de/lehre/ss24
int a; // a: global, uninitialized
int b = 1; // b: global, initialized
const int c = 2; // c: global, const

void main(void) {
  static int s = 3; // s: local, static, initialized
  int x, y; // x: local, auto; y: local, auto
  char *p = malloc(100); // p: local, auto; *p: heap (100 byte)
}

Where does the memory for these variables come from?
Organisation of Memory

```c
int a; // a: global, uninitialized
int b = 1; // b: global, initialized
const int c = 2; // c: global, const

void main(void) {
    static int s = 3; // s: local, static, initialized
    int x, y; // x: local, auto; y: local, auto
    char *p = malloc(100); // p: local, auto; *p: heap (100 byte)
}
```

Static allocation – allocation during compilation / linking

- Concerns all global/static variables and the code itself
- Allocation by getting placed into a section
  - .text – contains program code
  - .bss – contains all variables initialized with 0
  - .data – contains all variables initialized with other values
  - .rodata – contains all constant variables

© klsw System-Level Programming (ST 24) 34 Organisation of Memory – Introduction
Organisation of Memory

```c
int a; // a: global, uninitialized
int b = 1; // b: global, initialized
const int c = 2; // c: global, const

void main(void) {
    static int s = 3; // s: local, static, initialized
    int x, y; // x: local, auto; y: local, auto
    char *p = malloc(100); // p: local, auto; *p: heap (100 byte)
}
```

Static allocation – allocation during compilation / linking
- Concerns all global/static variables and the code itself
- Allocation by getting placed into a section
  - `.text` – contains program code
  - `.bss` – contains all variables initialized with 0
  - `.data` – contains all variables initialized with other values
  - `.rodata` – contains all constant variables

Dynamic allocation – reserved during runtime
- Concerns all local automatic variables and explicitly allocated memory
  - Stack – contains all auto variables that are currently alive
  - Heap – contains with `malloc()` explicitly allocated memory areas

© klsw  System-Level Programming (ST 24)  34 Organisation of Memory  – Introduction
int a;
int b = 1;
const int c = 2;

void main(void) {
    static int s = 3;
    int x, y;
    char *p = malloc(100);
    
}
Organisation of Memory with an OS

```c
int a;
int b = 1;
const int c = 2;

void main(void) {
    static int s = 3;
    int x, y;
    char *p = malloc(100);
}
```

- **Stack**
- **Heap**
- **RAM (read/write)**
- **RAM (read-only)**

### Symbol Table
- `.data`
  - `s=3`  
  - `b=1`
- `.rodata`
  - `c=2`
- `.text`
  - `main`
- `.bss`
  - `a=0`

### Compile Link
- Source program (file)
- ELF binary (file)
- PC (memory)

### Exec
- ```
Organisation of Memory with an OS (continued)

[Diagram showing memory organisation and stack/heap usage]

- Source program (file) → Compile
- ELF binary (file) → Link
- ELF binary (file) → Boot
- Source program (file) → Compile
- ELF binary (file) → Execute

- Stack
- Heap
- .bss
- .data
- .rodata
- .text
- Header

- RAM (read/write)
- RAM (read-only)