Ein-/Ausgabe

Praktikum „C-Programmierung“

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eugen Betke, Nathanael Hübbe, Michael Kuhn, Jakob Lüttgau, Jannek Squar

2020-01-13

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Motivation

- Ein-/Ausgabe wichtig in realen Programmen
  - Konfigurationsdaten etc. müssen eingelesen werden
  - Berechnungsergebnisse etc. müssen geschrieben werden
- Bisher nur Bildschirmausgabe mithilfe von `printf`
  - Funktioniert intern analog zur heute betrachteten Ein-/Ausgabe
- Es gibt unterschiedliche Abstraktionsebenen
  - Ein-/Ausgabe mit Dateideskriptoren (heute betrachtet)
  - Gepufferte Ein-/Ausgabe
  - Implizite Ein-/Ausgabe bzw. Memory Mapping

```c
int main(void) {
    int fd;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    assert(fd != -1);
    close(fd);
    return 0;
}
```

- Mit open können Dateien erstellt und geöffnet werden
  - Alternativ kann `creat` benutzt werden
  - Es wird ein Dateideskriptor zurückgegeben
- Die Flags geben an, wie die Datei geöffnet werden soll (siehe `man open`)
  - Der Modus ist nur bei `O_CREAT` und `O_TMPFILE` relevant

```c
int main(void) {
    int fd;
    ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    nb = write(fd, &fd, sizeof(fd));
    assert(nb == sizeof(fd));
    close(fd);
    return 0;
}
```

- Mit `write` können Daten in die Datei geschrieben werden
  - Eine Datei besteht aus einem Bytestrom, Format ist beliebig
- Es werden nicht notwendigerweise alle Daten auf einmal geschrieben
  - Rückgabewert muss überprüft und `write` u. U. erneut aufgerufen werden

read v0

```c
int main(void) {
    int fd, dummy = 0;
    ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    nb = read(fd, &dummy, sizeof(fd));
    assert(nb == sizeof(fd));
    printf("Inhalt: %d\n", dummy);
    close(fd);
    return 0;
}
```

- Mit `read` können Daten aus einer Datei gelesen werden
  - Funktioniert analog zu `write`, es wird ein Bytestrom gelesen
- Es werden wieder nicht notwendigerweise alle Daten gelesen

read v1

```
1  int main(void) {
2      int fd, dummy = 0;
3      ssize_t nb;
4      fd = open("myfile", O_RDWR | O_CREAT, 0644);
5      nb = read(fd, &dummy, sizeof(fd));
6      assert(nb == sizeof(fd));
7      nb = read(fd, &dummy, sizeof(fd));
8      assert(nb == 0);
9      close(fd);
10     return 0;
11 }
```

- Jede Lese- oder Schreiboperation verändert den Dateizeiger
  - Dateizeiger ist dem Dateideskriptor zugeordnet
- Ein Rückgabewert von 0 zeigt das Dateiende an

```c
int main(void) {
    int fd, dummy = 0;
    ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    nb = read(fd, &dummy, sizeof(fd));
    assert(nb == sizeof(fd));
    lseek(fd, 0, SEEK_SET);
    nb = read(fd, &dummy, sizeof(fd));
    assert(nb == sizeof(fd));
    close(fd);
    return 0;
}
```

- Mit `lseek` kann innerhalb der Datei positioniert werden
  - Absolut (`SEEK_SET`), relativ (`SEEK_CUR`) oder relativ zum Ende (`SEEK_END`)

```c
int main(void) {
    int fd, dummy = 0;
    ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    nb = pread(fd, &dummy, sizeof(fd), 0);
    assert(nb == sizeof(fd));
    nb = pread(fd, &dummy, sizeof(fd), 0);
    assert(nb == sizeof(fd));
    close(fd);
    return 0;
}
```

- `pread` und `pwrite` erlauben Angabe von Offsets
  - Der Dateizeiger wird dabei nicht verändert
  - So kann Dateideskriptor von mehreren Threads gleichzeitig benutzt werden

```c
int main(void) {
    write(STDOUT_FILENO, "Hallo Welt!\n", 13);
    return 0;
}
```

- Ein-/Ausgabe im Terminal findet auch über Dateideskriptoren statt
- Standardeingabe, -ausgabe und -fehlerausgabe sind standardmäßig geöffnet
  - Belegen üblicherweise die Dateideskriptoren `0` bis `2`
  - Zugriff über `STDIN_FILENO`, `STDOUT_FILENO` und `STDERR_FILENO`

```c
int main(void) {
    int ret;
    struct stat st;
    ret = stat("myfile", &st);
    assert(ret == 0);
    printf("Groesse: %lu\n", st.st_size);
    return 0;
}
```

- Mit `stat` können Informationen über Dateien abgerufen werden
  - U. a. die Berechtigungen, der Besitzer, die Größe und diverse Zeitstempel

```c
int main(void) {
    int fd, ret; struct stat st; ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    lseek(fd, 1000000, SEEK_SET);
    assert(write(fd, &fd, sizeof(fd)) == sizeof(fd));
    assert(fstat(fd, &st) == 0);
    printf("Groesse: %lu\n", st.st_size);
    printf("Echte Groesse: %lu\n", st.st_blocks * 512);
    close(fd);
    return 0;
}
```

- Sparse-Dateien belegen nicht den gesamten Platz
  - Leere Bereiche liefern 0en zurück

```c
int main(void) {
    char dummy[999996];
    int fd;
    ssize_t nb;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    nb = pwrite(fd, dummy, sizeof(dummy), sizeof(fd));
    assert(nb == sizeof(dummy));
    fsync(fd);
    close(fd);
    return 0;
}
```

- Nach `write` befinden sich Daten noch nicht auf dem Speichergerät
  - Üblicherweise in einem Cache des Betriebssystems
  - Cache kann mit `O_DIRECT` umgangen werden

Gepufferte Ein-/Ausgabe

- Außerdem noch gepufferte Ein-/Ausgabe
  - `fopen`, `fread`, `fwrite`, `fclose` etc.
  - Durch Pufferung u. U. explizites `fflush` notwendig
- Benutzung etwas komfortabler, da keine Flags notwendig sind
  - Modus wird als String angegeben, z. B. `"r"` zum Lesen
  - Dafür nicht alle Kombinationen möglich

```c
struct foo { int fd1; char bar[999996]; int fd2; };
int main(void) {
    int fd;
    fd = open("myfile", O_RDWR | O_CREAT, 0644);
    struct foo* map = mmap(NULL, sizeof(struct foo),
                           PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    printf("Inhalt: %d %d\n", map->fd1, map->fd2);
    munmap(map, sizeof(struct foo));
    close(fd);
    return 0;
}
```

- Mit `mmap` kann eine Datei in den Arbeitsspeicher eingeblendet werden
  - Änderungen wie bei normalen Speicherbereichen, z. B. mit `memcpy`
  - Lesen und Schreiben wird vom Betriebssystem übernommen

Zusammenfassung

- Ein-/Ausgabe ist ein wichtiges Thema in echten Anwendungen
  - Überprüfen der Rückgabewerte hier besonders wichtig
- Es stehen unterschiedliche Abstraktionsebenen zur Verfügung
  - Ein-/Ausgabe mit Dateideskriptoren
  - Gepufferte Ein-/Ausgabe
  - Implizite Ein-/Ausgabe bzw. Memory Mapping