Bab 10

Runtime Environment dan Memory Management

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 5.1:** Mendesain runtime environment untuk bahasa pemrograman

10.1 Layout Memori dan Segmentasi

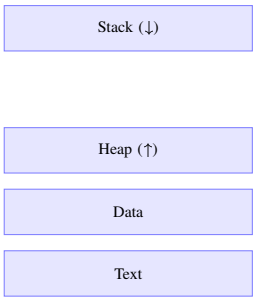
Runtime Environment mengelola bagaimana program menggunakan memori saat dieksekusi. Manajemen memori saat runtime merupakan salah satu aspek yang paling krusial dalam performa dan keamanan sebuah program hasil kompilasi [1].

10.1.1 Segmen Memori Utama

- **Text/Code Segment**: Menyimpan instruksi biner hasil kompilasi. Biasanya bersifat *read-only*.
- **Data Segment**: Menyimpan variabel global dan statis yang sudah diinisialisasi.
- **Stack**: Segmen yang tumbuh ke bawah untuk mengelola pemanggilan fungsi dan variabel lokal.
- **Heap**: Segmen yang tumbuh ke atas untuk alokasi memori dinamis (*runtime allocation*).

10.2 Activation Records dan Stack Frame

Setiap kali sebuah fungsi dipanggil, struktur data yang disebut ***Activation Record*** (atau *Stack Frame*) dibuat di puncak stack.



Gambar 10.1: Model konseptual organisasi memori runtime

10.2.1 Isi Activation Record

Sebuah stack frame biasanya menampung informasi kritis:

1. **Return Address**: Alamat instruksi berikutnya di penyeru (*caller*) setelah fungsi selesai.
2. **Control Link**: Referensi ke stack frame milik penyeru.
3. **Saved Registers**: Nilai register prosesor yang harus dipulihkan nantinya.
4. **Local Variables**: Ruang untuk variabel yang dideklarasikan di dalam fungsi.

10.2.2 Manajemen Frame Pointer

Frame Pointer (FP) digunakan sebagai referensi tetap untuk mengakses parameter (offset positif) dan variabel lokal (offset negatif) di dalam frame saat ini.

10.3 Mekanisme Panggilan: Prologue dan Epilogue

10.3.1 Prologue

Saat fungsi dipanggil, sekumpulan instruksi *prologue* dijalankan untuk menyiapkan lingkungan baru:

```
push rbp          ; Simpan frame pointer lama
mov rbp, rsp      ; Set frame pointer baru
sub rsp, 16       ; Alokasi ruang untuk lokal
```

10.3.2 Epilogue

Sebelum kembali ke penyeru, instruksi *epilogue* dijalankan untuk membersihkan stack:

```
mov rsp, rbp      ; Dealokasi ruang lokal
pop rbp           ; Pulihkan frame pointer lama
ret               ; Kembali ke return address
```

10.4 Manajemen Dinamis dan Heap

Berbeda dengan stack yang dikelola secara otomatis (LIFO), ***Heap*** memungkinkan alokasi memori yang waktu hidupnya (*lifetime*) tidak terikat pada durasi pemanggilan fungsi.

10.4.1 Alokasi dan Dealokasi

Dalam bahasa C++, ini dikelola melalui operator `new` dan `delete`. Kompilator bertanggung jawab menyisipkan panggilan ke sistem operasi atau pengelola memori (*allocator*).

10.4.2 Masalah Manajemen Heap

- **Fragmentasi**: Tersebarnya blok-blok memori kosong yang kecil sehingga tidak bisa menampung alokasi besar.
- **Memory Leak**: Kegagalan membebaskan memori yang sudah tidak digunakan, menyebabkan konsumsi RAM terus meningkat.

10.5 Praktikum: Simulator Runtime Stack

Mahasiswa akan mempelajari cara kerja stack frame melalui implementasi simulator sederhana. Simulator ini melacak pembuatan dan penghapusan *Activation Record* setiap kali ada simulasi pemanggilan fungsi.

```
struct Frame {
    string funcName;
    Frame* caller;
    map<string, int> locals;
};

class StackSimulator {
    Frame* top = nullptr;
public:
    void call(string name) {
        top = new Frame{name, top};
    }
    void returnFunc() {
        Frame* old = top;
        top = top->caller;
        delete old;
    }
};
```

Simulator ini membantu memvisualisasikan bagaimana variabel lokal diisolasi antar fungsi dan bagaimana rekursi dapat menyebabkan *Stack Overflow* jika tidak terkendali.

10.6 Garbage Collection

10.6.1 Reference Counting

Simple garbage collection:

```c
typedef struct Object {
    int ref_count;
    void *data;
    struct Object **references;
    int ref_count_size;
} GCObject;

void gc_retain(GCObject *obj) {
    if (obj) obj->ref_count++;
}

void gc_release(GCObject *obj) {
    if (obj && --obj->ref_count == 0) {
        // Release all references
        for (int i = 0; i < obj->ref_count_size; i++) {
            gc_release(obj->references[i]);
        }
        free(obj);
    }
}
```

10.6.2 Mark and Sweep

```c
void mark_and_sweep(HeapManager *heap) {
    // Mark phase
    mark_phase(heap);

    // Sweep phase
    sweep_phase(heap);
}

void mark_phase(HeapManager *heap) {
    // Mark all reachable objects from roots
    Object **roots = get_gc_roots();
    for (int i = 0; i < num_roots; i++) {
        mark_object(roots[i]);
    }
}

void sweep_phase(HeapManager *heap) {
    MemoryBlock *block = heap->allocated_list;
    while (block) {
        if (!is_marked(block->object)) {
            // Free unreachable object
            free_object(block->object);
            remove_from_allocated(block);
            add_to_free(block);
```

```
25          } else {
26              // Clear mark for next GC
27              clear_mark(block->object);
28          }
29          block = block->next;
30      }
31  }
```

Aktivitas Pembelajaran

1. **Memory Layout**: Implementasikan simulator memory layout program.
2. **Activation Records**: Bangun activation record management system.
3. **Parameter Passing**: Implementasikan berbagai parameter passing methods.
4. **Heap Management**: Implementasikan heap allocator dengan first-fit algorithm.
5. **Garbage Collection**: Bangun simple mark-and-sweep garbage collector.

Latihan dan Refleksi

1. Gambarkan memory layout untuk program dengan recursive functions!
2. Implementasikan activation record untuk function dengan nested scopes!
3. Analisis perbedaan pass by value, reference, dan value-result!
4. Implementasikan heap manager dengan fragmentation handling!
5. Desain garbage collection algorithm untuk object-oriented language!
6. **Refleksi**: Bagaimana runtime environment mempengaruhi performance program?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 5.1

A. Pilihan Ganda

1. Activation record disimpan di:

   (a) Heap
   (b) Stack
   (c) Static area
   (d) Code segment

2. Pass by reference mengirimkan:
   (a) Nilai variabel
   (b) Alamat variabel
   (c) Copy variabel
   (d) Pointer ke pointer

3. Garbage collection menghapus:
   (a) Semua objek
   (b) Objek yang tidak reachable
   (c) Objek yang besar
   (d) Objek yang lama

B. Essay

1. Jelaskan prosedur call mechanism lengkap dengan activation record management!
2. Desain runtime environment untuk bahasa dengan support untuk dynamic arrays dan garbage collection!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat mendesain runtime environment untuk bahasa pemrograman

☐ Saya dapat mengimplementasikan memory layout yang efisien

☐ Saya dapat membangun activation record management

☐ Saya dapat mengimplementasikan berbagai parameter passing methods

☐ Saya dapat mendesain heap management system

☐ Saya dapat mengimplementasikan garbage collection

Rangkuman

Bab ini membahas runtime environment dan memory management, termasuk memory layout, activation records, parameter passing, heap management, dan garbage collection. Mahasiswa belajar mendesain infrastruktur runtime yang efisien.

Poin Kunci:

- Runtime environment menyediakan infrastruktur untuk eksekusi program
- Memory layout terdiri dari stack, heap, static area, dan code segment
- Activation records mengelola function calls dan local variables
- Parameter passing methods memiliki trade-off berbeda
- Heap management mengelola dynamic memory allocation
- Garbage collection otomatis menghapus objek yang tidak digunakan

Kata Kunci: ***Runtime Environment***, ***Memory Layout***, ***Activation Record***, ***Parameter Passing***, ***Heap Management***, ***Garbage Collection***, ***Stack Frame***

Daftar Pustaka

[1] UC San Diego CSE Department. *CSE 231: Compiler Construction / Advanced Compiler Design*. Course materials and syllabus. 2024. URL: https://catalog.ucsd.edu/archive/2024-25/courses/CSE.html.