Concepts of Programming Languages, CSCI 305, Fall 2021
Storage Management, Nov. 8
Section 3.2 Storage Management

Distinguish between names and the objects to which they refer

Object Lifetime

Typical order of events in the life of an object:

creation of object
creation of name-to-object bindings
references to variables (which use bindings)
(temporary) deactivation of bindings
reactivation of bindings
destruction of bindings
destruction of objects

Lifetime of an object - the time between when it is created and when it is destroyed
Lifetime of a name-to-object binding – time between when bound and when unbound

Storage Allocation

Three primary mechanisms for managing the lifetime of an object in memory:

- Static allocation
- Stack allocation
- Heap allocation

Typical memory layout (Figure 15.8, page 793)

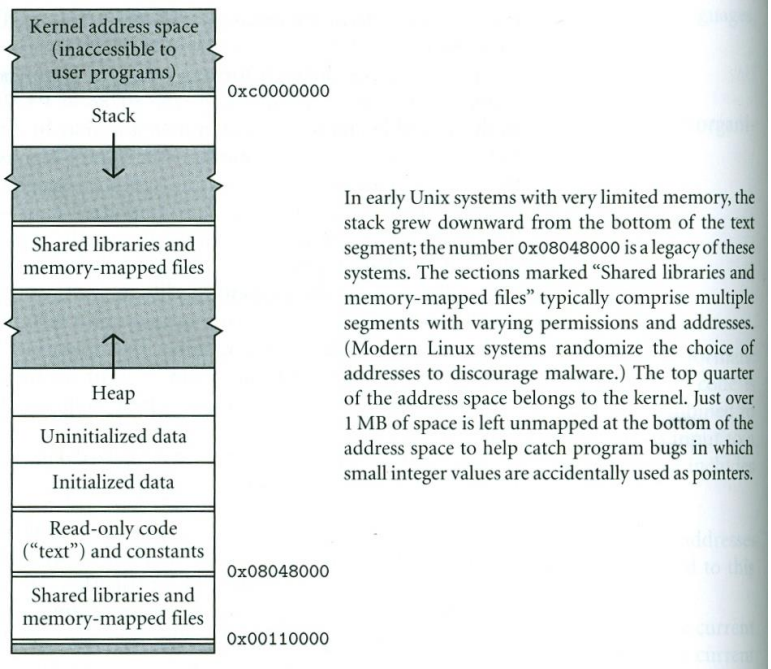

Figure 14.8 Layout of process address space in x86 Linux (not to scale). Double lines separate regions with potentially different access permissions.

Scalar variables - hold a single data item
Non-scalar, or compound variables – holds multiple elements, array, vector, structure, object

Scalar variables can be divided into three categories by considering their lifetimes and where they are stored:

1. Static variables -bound to storage before execution begins and this binding is never changed
2. Stack variables - objects are allocated and deallocated in last-in, first-out order as stack frames are added and removed from the stack

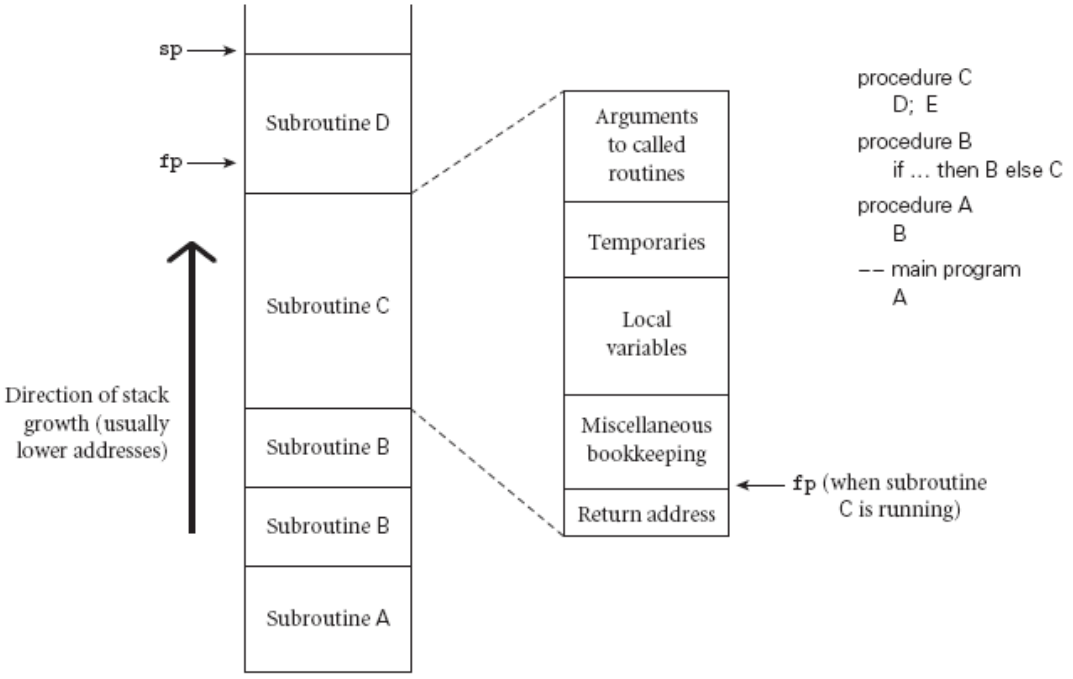

Figure 3.1 Stack-based allocation of space for subroutines. We assume here that subroutines have been called as shown in the upper right. In particular, B has called itself once, recursively, before calling C. If D returns and C calls E, E's frame (activation record) will occupy the same space previously used for D's frame. At any given time, the stack pointer (`sp`) register points to the first unused location on the stack (or the last used location on some machines), and the frame pointer (`fp`) register points to a known location within the frame of the current subroutine. The relative order of fields within a frame may vary from machine to machine and compiler to compiler.

3. Heap variables - unorganized storage accessed via pointers or reference variables, flexible but requires lots of overhead (may be garbage collected)