1 Old memory layout

PSL was developed back in the 1980s and has mostly kept the memory layout of the times: linear memory with text (code), data (initialized data), and bss (uninitialized data) areas, plus heap. On Unix and similar operating systems, the heap area was traditionally allocated either statically at link time or dynamically using the sbrk system call.

The code area consists of simple C Code for the most basic operating system interfaces: I/O and very simple memory management plus a layer of assembler code t hat is translated from Lisp. All the rest of PSL is loaded at runtime into the so-called binary program space (BPS) residing in the bss area.

A running PSL system can be dumped to a so-called image file. This image file can be loaded later at startup to restart the previously running system. Originally the whole memory image of the process was written to disc as an executable to be started again. In the course of time this was replaced with code that dumps (and loads) only the relevant data. In order for this method to work, the important memory areas like symbol table, BPS, and heap must be allocated at the very same memory address. Since more recent operating systems cannot always guarantee the memory address of a dynamically allocated heap, all memory references within the heap need to be checked and adjusted when the image file is loaded. However, this is not possible for the BPS: The compiled lisp code area includes constant lisp objects whose addresses are referenced in the actual code.

2 Problems of the old model

Current operating systems employ several security techniques that comflict with the simple old model:

- Address space layout randomization: a specific address for the various areas cannot be guaranteed[1].
- WˆX protection: a memory area way be written to, or the code in it may be executed but not both. Code loaded at runtime can therefore not easily be run. There are ways around this restriction, but so far they work only with dynamically allocated memory areas - which conflicts with the requirement that the addresses in the BPS do not change.

3 The new memory model

To get around the problems of the old model the BPS area must be made relocatable. This means:

1. The constant lisp objects need to be stored separately. To this end a new static memory area is allocated in the bss segment. This addresses of the lisp objects stored there are not changed across program starts.

[1]In addition, the lisp compiler for the x86_64 platform generates memory references as 32 bit, not 64 bit offsets. This could be changed, of course.

2. The code must be position-independent. There must be no explicit memory references, instead all accesses to other memory areas, i.e. the symbol table and the static lisp area must be relativ to their base addresses[2].

3. Since code and constant lisp objects are now separate the fast loadable (FASL) file format needs to be changed.

[2]This is most easily done by keeping the base addresses of the symbol table and the static lisp area in processsor registers.