IA-64 gABI Issue 72: COMDAT

Revised 19 October 1999

Revisions

Introduction

C++ has many situations where the compiler may need to emit code or data, but may not be able to identify a unique compilation unit where it should be emitted. The approach chosen by the C++ ABI group to deal with this problem, is to allow the compiler to emit the required information in multiple compilation units, in a form which allows the linker to remove all but one copy. This is essentially the feature called COMDAT in several existing implementations.

Various other implementations (notably Windows NT) and proposals obtain more generality by varying the duplicate removal semantics. The most obviously useful variant supports grouping of sections for removal purposes, but treats duplication as an error, using it to support link-time removal of unreferenced sections. The proposal below treats this simple grouping as the default semantics, and provides duplicate removal as an option.

Our objectives include:

- Use existing structures as far as possible, to minimize impact on existing tools.
- Minimize impact on the linker by defining the unit of duplication and removal as a full section.
- Maximize generality. The C++ needs are rather varied, and similar needs from other languages should also be handled.
- In general, duplicated code or data sections are accompanied by additional duplicated sections, e.g. containing debug information. We want to define a mechanism which can deal with arbitrary such associations, without predicting them in advance.

Proposal

The proposal below is based on the HP definition, with minor modifications and more precise definitions.

SHF_GROUP: Group Member Sections

A section which is part of a group, and is to be retained or discarded with the group as a whole, is identified by a new section header attribute:

SHF_GROUP
: This section is a member (perhaps the only one) of a group of sections, and the linker should retain or discard all or none of the members. This section must be referenced in a SHT_GROUP section (see below).

This attribute flag may be set in any section header, and no other modification or indication is made in the grouped sections. All additional information is contained in the associated SHT_GROUP section (see below).

SHT_GROUP: Section Group Definition

Some sections occur in interrelated groups. For instance, an out-of-line definition of an inline function might require, in addition to its .text section, a read-only data section containing literals referenced, one or more debug information sections, and/or other informational sections. Furthermore, there may be internal references among these sections that would not make sense if one of them were removed or replaced by a duplicate from another object. Therefore, we assume that such groups are to be included or omitted from the linked object as a unit. (Except for the GRP_COMDAT flag described below, this definition does not specify the circumstances under which the members of a group might be discarded from the linked object.)

To facilitate this, we define a SHT_GROUP section:

The section header attributes of a Group Section are:

name	unspecified
`sh_type`	`SHT_GROUP`
`sh_link`	`.symtab` section index
`sh_info`	symbol index
`sh_flags`	none
`sh_entsize`	size of section indices (`4`)
requirements	may not be stripped

The section group's `sh_link` field identifies a symbol table section, and its `sh_info` field the index of a symbol in that section. The name of that symbol is treated as the identifier of the section group.

The section data of a SHT_GROUP section is a flag word followed by a sequence of section indices. The flag word may contain the following flags:

GRP_COMDAT (0x1)
: This is a COMDAT group. It may duplicate another COMDAT group in another object file, where duplication is defined as having the same identifying symbol name. In such cases, only one of the duplicate groups should be retained by the linker, and the remaining groups should be discarded.

The section indices in the SHT_GROUP section identify the sections which make up the group.

The `sh_size` value is `sh_entsize` times one plus the number of sections in the group.

The linker may choose to discard a section in a group, i.e. not include its data in the linked object, based on COMDAT duplicate semantics (above), or for other implementation-defined reasons (e.g. removing unreferenced code). If it does so, the group semantics requires that all of the group members be removed as a unit.

(Note, however, that this is not intended to imply that special-case behavior like removing debug information requires removing the sections to which it refers, even if they are in a group. We could

clarify this issue by tying the removal semantics to the section which contains the identifying symbol, but this seems overly restrictive and unnecessary.

Requirements

- References to the sections comprising a group, from sections outside the group, must be made via global UNDEF symbols, referencing global symbols defined as addresses in the group sections. They may not reference local symbols for addresses in the group's sections, including section symbols.
- There may not be non-symbol references to the sections comprising a COMDAT group from sections outside the group, e.g. in sh_link fields. For example, relocations of one of the group's sections must be in a relocation section which is also part of the group.

  The above rules allow a group to be removed without leaving dangling references, with only minimal processing of the symbol table.
- An entry in a symbol table section not in the group, with a definition that is relative to one of the group's sections, should be removed if the group is discarded.
- The SHT_GROUP section must precede the sections in the group (in the section table).

Questions

- *Do we want flags to specify checking prior to removal of duplicates, e.g. for identical sections, same defined global symbols, etc.? If so, should there be one flags word per section index, instead of per group?*
  Answer: No flags besides GRP_COMDAT. Only one flags word needed.
- *Do we want more control over when global symbols are removed vs. being converted to UNDEF? Alternatively, should we simply require that all symbols defined as addresses in the group be removed, and that references to them from outside do so via distinct UNDEF global symbols?*
  Answer: Just remove symbols defined relative to removed sections.
- *Do we want to replace the symbol rule by simply requiring that any symbols defined as addresses in the group be defined in a .symtab section that is itself in the group?*
  Answer: No. Removing symbols is not hard, and is already done in other circumstances in many implementations.
- *Why use a symbol to identify a (COMDAT) group instead of the group section name?*
  Answer: These will be very common in C++, and the typical names in C++ can be extremely long (for template instantiations often multiple kilobytes in current implementations, and probably 100+ characters on average even if vastly better mangling is adopted). Virtually all such sections must already define a real global symbol, and duplicating such long names in the section name string table is undesirable space overhead.
- *What is the name of a group section?*
  Answer: It is unspecified, because it has no significance to the linker. It is recommended that it be something like .group, possibly with a short suffix to distinguish multiple groups in an object if necessary, but we place no requirement on the name.