

COMP2310/COMP6310

Systems, Networks, & Concurrency

Convener: Prof John Taylor

Linking – 1

Acknowledgement of material: With changes suited to ANU needs, the slides are obtained from Carnegie Mellon University: <https://www.cs.cmu.edu/~213/>

Example C Program

```
int sum(int *a, int n);

int array[2] = {1, 2};

int main(int argc, char** argv)
{
    int val = sum(array, 2);
    return val;
}
```

main.c

```
int sum(int *a, int n)
{
    int i, s = 0;

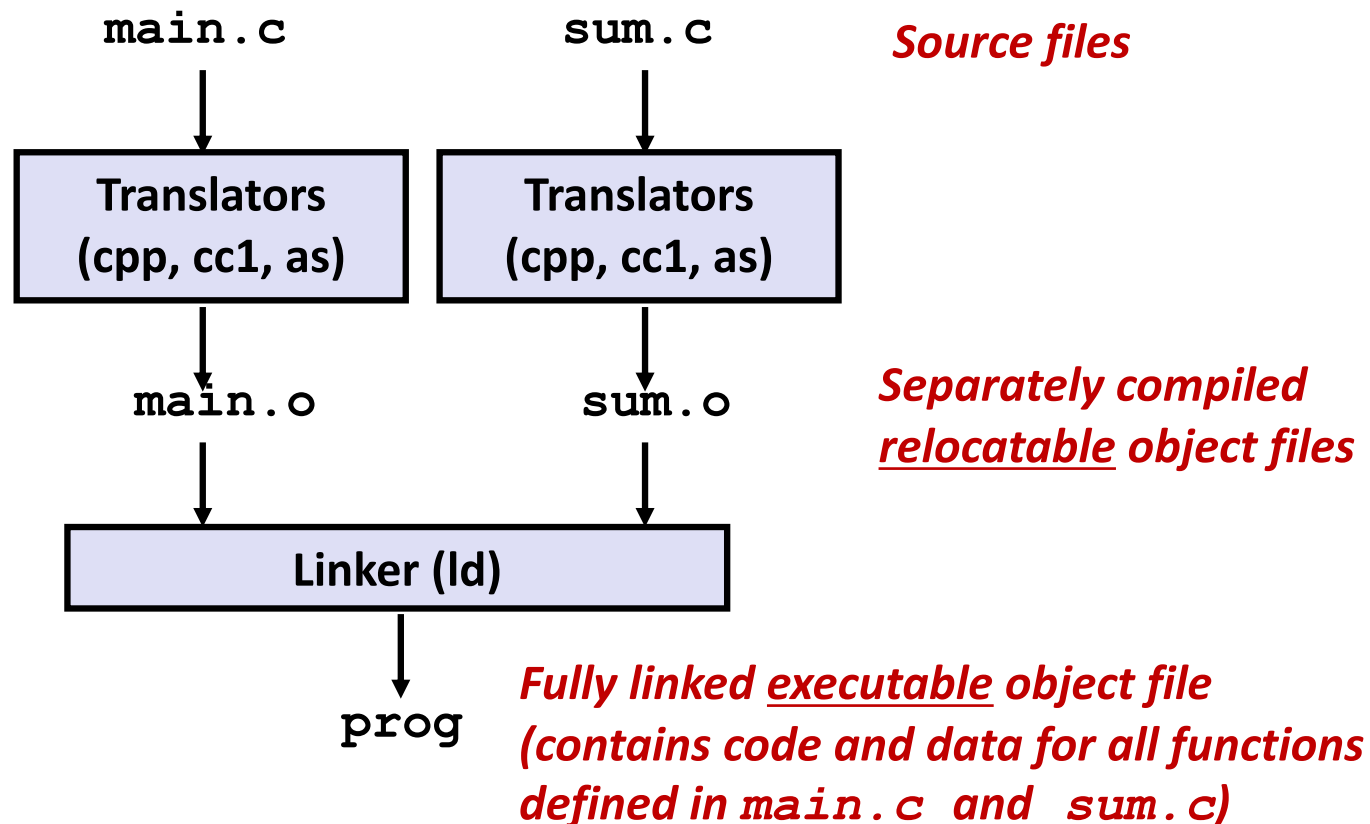
    for (i = 0; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

sum.c

Linking

- Programs are translated and linked using a *compiler driver*:

- `linux> gcc -Og -o prog main.c sum.c`
- `linux> ./prog`



Why Linkers?

■ Reason 1: Modularity

- Program can be written as a collection of smaller source files, rather than one monolithic mass.
- Can build libraries of common functions
 - e.g., Math library, standard C library
 - Header files in C declare types that are defined in libraries

Why Linkers? (cont)

■ Reason 2: Efficiency

- Time: Separate compilation
 - Change one source file, compile, and then relink.
 - No need to recompile other source files.
 - Can compile multiple files concurrently.
- Space: Libraries
 - Common functions can be aggregated into a single file...
 - **Option 1: *Static Linking***
 - Executable files and running memory images contain only the library code they actually use
 - **Option 2: *Dynamic linking***
 - Executable files contain no library code
 - During execution, single copy of library code can be shared across all executing processes

What Do Linkers Do?

■ Step 1: Symbol resolution

- Programs define and reference *symbols* (global variables and functions):
 - `void swap() {...} /* define symbol swap */`
 - `swap(); /* reference symbol swap */`
 - `int *xp = &x; /* define symbol xp, reference x */`
- Symbol definitions are stored in object file (by assembler) in *symbol table*.
 - Symbol table is an array of entries
 - Each entry includes name, size, and location of symbol.
- **During symbol resolution step, the linker associates each symbol reference with exactly one symbol definition.**

Symbols in Example C Program

Definitions

```
int sum(int *a, int n);  
int array[2] = {1, 2};  
int main(int argc, char** argv)  
{  
    int val = sum(array, 2);  
    return val;  
}
```

main.c

```
int sum(int *a, int n)  
{  
    int i, s = 0;  
  
    for (i = 0; i < n; i++) {  
        s += a[i];  
    }  
    return s;  
}
```

sum.c

Reference

What Do Linkers Do? (cont'd)

■ Step 2: Relocation

- Merges separate code and data sections into single sections
- Relocates symbols from their relative locations in the `.o` files to their final absolute memory locations in the executable.
- Updates all references to these symbols to reflect their new positions.

Let's look at these two steps in more detail....

Three Kinds of Object Files (Modules)

■ Relocatable object file (`.o` file)

- Contains code and data in a form that can be combined with other relocatable object files to form executable object file.
 - Each `.o` file is produced from exactly one source (`.c`) file

■ Executable object file (a `.out` file)

- Contains code and data in a form that can be copied directly into memory and then executed.

■ Shared object file (`.so` file)

- Special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run-time.
- Called *Dynamic Link Libraries* (DLLs) by Windows

More on Linking

- Entire lecture toward the end of the course