CS 3204 Project 4 Spring 2002

Overlay Memory Management **Due: 29 April**

1 Introduction

We have discussed several memory management techniques that have been used by operating systems. Early and simple operating systems treated memory as a single contiguous block, and applications were compiled to access absolute (physical) memory addresses. A disadvantage to this approach was that processes were limited by the size of memory available.

An attempt to overcome this limitation was *overlays*, the ability to overlay memory locations with independent software modules. A process is broken up into several software modules, and dependencies between the modules are modelled with a tree. For example, suppose a process has 5 modules A, B, C, D, E such that B and E require A to be in memory, C and D require B to be in memory, but no other restrictions on the modules exist. These relationships can be modelled with the tree in Figure 1. The dependencies as

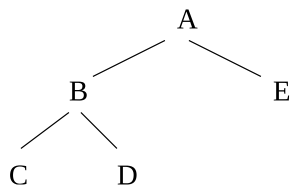

Figure 1: A tree modelling the relationships between program modules.

indicated in such a tree define which modules can be assigned the same memory addresses. If the modules above used $40, 30, 10, 10$, and 40 memory units respectively, then Figure 2 contains the corresponding memory layout for the process using overlays.

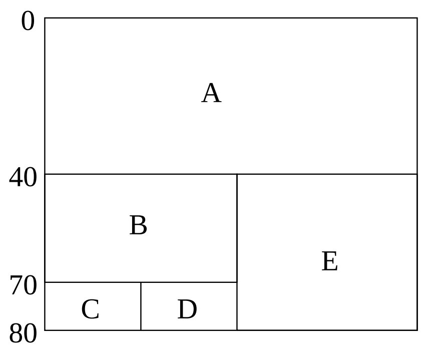

Figure 2: The memory layout for the sample process.

2 Specification

In this assignment, you will implement a simplified linker named `overlay` that takes an overlay specification and determines the memory layout of the program. Your program should determine the (minimum) amount of memory required by the process for execution, and the absolute addresses used by each module.

A sequence of accesses to *relative* addresses in program modules will also be given. For each access your program should determine

- which modules are unloaded as a result of the access,
- which modules are loaded as a result of the access, and
- the absolute address corresponding to the relative address being accessed.

Your program is to read from standard input and write to standard output output (e.g., `cin` and `cout` respectively). You should use command line input and output redirection to read from a file and to print to a file.

3 Input

The input file consists of two parts: the module specification and the access sequence. A sample input file is contained in Figure 3.

The first line in the module specification is the number of modules n for the program. The next n lines are in the following format:

```
module mem numchild child1 child2 ...
```

where `module` is the name of the module, `mem` is the amount of memory used by the module, `numchild` is the number of modules that require this module to be loaded, and `child1`, `child2`, ... are the names of the modules that require this module to be loaded. Module names are strings without spaces. The memory usage and number of children are integers.

```
5
A 40 2 B E
B 30 2 C D
C 10 0
D 10 0
E 40 0
D 5
C 7
E 16
A 32
```

Figure 3: A sample input file.

Following the module specification, the remainder of the file contains memory accesses in the form

```
module address
```

where `module` is the module containing the relative address and `address` is a *relative* address within that module. Your program should then generate commands to unload any modules that are not required to be in memory and load any modules that are required to be in memory. The modules loaded as the result of a memory access are used as the starting point for the next memory access.

The following will be true of the input.

- The input file will be in the proper format and contain only valid information;
- The first module in the module specification is the root of the tree;
- The description of the parent module will appear before the description of any of its children;
- All memory accesses in the file will be valid.

4 Output

An output file corresponding to the sample input file is shown in Figure 4. The first part of the output is a table listing each module and its memory allocation. The table should be sorted first by ascending starting address of each module, then by ascending ending address, and then by ascending module name to break ties. Following the table is the total amount of memory required by the process.

After the module summary table, a list of load, unload, and access commands are displayed for each access specification in the input file. Modules should be unloaded (if necessary) before loading new modules (if necessary). The access line contains the physical address of the memory location.

5 Submission

We will use the Curator at `http://spasm.cs.vt.edu:8080/curator01/` to collect program submissions. No grading will be done by the Curator.

You are to submit a single tarred (`man tar`) and gzipped (`man gzip`) archive containing

- A text file named `README` describing the program, describing the contents of the archive, providing building instructions (including the platform you used for development), a user's guide (including how to start the program), and examples of usage with your test files;
- The source code for your programs;
- Sample output for 3 executions of your programs;
- A script named `build` or a suitable `Makefile` for building your programs.

```
Module              Start     End
---------------------------------
A                       0      39
B                      40      69
E                      40      79
C                      70      79
D                      70      79
---------------------------------
Memory Required: 80

Commands
--------
load A
load B
load D
access 75
unload D
load C
access 77
unload C
unload B
load E
access 56
unload E
access 32
```

Figure 4: A sample output file.

Your files must be a directory named `p4-pid` (where `pid` is your university pid) of the archive. Be sure to include only the files listed above. Do not include extra files from an integrated development environment such as `configure` scripts, automake related files, etc. This is primarily an issue if you are using KDevelop.

Be sure to include your name in all files submitted. **DO NOT** include executables or object files of any type in the archive. **Submissions that do not gunzip and/or untar will not be graded. Be careful to FTP in binary mode if you are transferring your file to a Windows machine before submitting to the Curator.**

Failure to follow the submission rules will result in a grade of zero (0) for this assignment. There will be no exceptions.

6 Programming Environment

As stated in the syllabus, you must use Linux and `gcc/g++` to implement this project. Your program must compile and run properly in the lab.

Your solution may be either object-oriented or procedural, but your design must use reasonable decomposition of the computational task.

You may use standard library (e.g., STL) classes, but third party libraries are not to be used. You may want to implement a general tree data structure — in which case, you may use code from a data structures textbook, provided you properly cite the source in your comments.