Operating System Assignment 3

Faizan Safdar Ali

17100152

Linkers:

Linker is a [computer program](https://en.wikipedia.org/wiki/Computer_program) that takes one or more [object files](https://en.wikipedia.org/wiki/Object_file) generated by a [compiler](https://en.wikipedia.org/wiki/Compiler) and combines them into a single [executable](https://en.wikipedia.org/wiki/Executable) file, [library](https://en.wikipedia.org/wiki/Library_(computing)) file, or another object file. In computers, each long program is divided into modules and different executable files by compilers. These files are tested individually. Linker has three functions: collect all the pieces of program, figure out new memory organization so that all the pieces fit together and touch up addresses so that the program can run under the new memory organization. The result generated by the linker is an executable file. Compiler doesn’t know where the things, it’s compiling, will go and let linker rearrange the files. Also compiler doesn’t know addresses of external objects when compiling files separately so compiling just puts zero in the object file. The linker should be able to solve these problems. Each object file consists of segments, symbol table, relocation records and some additional information for debugging. The addresses referenced in the object file’s relocation records must be fixed by the linker once it knows the final memory allocation. Linker executes in two passes. Pass 1: read in segment sizes, compute final memory layout. Also, read in all symbols, create complete symbol table in memory. Pass 2: read in segment and relocation information, update addresses and write out new file. Linking can be done in two ways, Static linking and dynamic linking. Static linking is the result of the linker copying all library routines used in the program into the executable image while a dynamic linker is the part of an operating system that loads and links the shared libraries needed by an executable when it is executed (at "run time"), by copying the content of libraries from persistent storage to RAM, and filling jump tables and relocating pointers.

Part 2:

Program halt.c is compiled during which the conversion of c-source file to MIPS source file is done. The first line (“$(CC) $(CFLAGS) -c halt.c “) tells the compiler to create MIPS code either by translating .c file into MIPS code or by using cross compiler while passing the CFLAGS as argument. Then compiler converts the MIPS source file into object file. Then compiler converts start.s into an object file. Finally a linker (LD) combines the two object files to make it into an executable C file. In this LDFLAGS, flag is also passed as arguments which links the first two module, ‘start.o’ & ‘halt.o’ into coff executable. Because nachos can’t run the coff format, it calls coff2noff function which converts the c-executable image to nachos-executable image done in the last line (“../bin/coff2noff halt.coff halt”).

Flow chart:

* The program initialized.
* Then the program enters the code block where USER\_PROGRAM is defined.
* Then it enters the code block where user program in run.
* There it goes to the StartProcess function defined in progtest.cc file.
* Then it goes to FileSystem:Open function in filesys.h where it opens the file using function OpenForReadWrite defined in sysdep.cc.
* Then the program goes to AddrSpace::AddrSpace defined in addrspace.cc file.
* From there it goes to OpenFile::ReadAt which direct program to Lseek defined in sysdep.cc which changes the location within an open file. Then ReadAt returns partial read file to the AddrSpace.
* Then the conversion to NOFF occurs. There we first check how big the address space is? Then we check if we are not trying to run anything too big. Then we set up the page table.
* Then we zero out the uninitialized data segment and the stack segment.
* Then we copy in the code and data segment into the memory.
* Then registers are initialized in AddrSpace::InitRegister function defined in machine.cc file.
* Then initialize program counter to start and tell MIPS where the next instruction is? Then set the stack pointer to the end of the address space making sure that we do not reference to the end.
* Then we load the page table registers in AddrSpace::RestoreState function.
* Then the program completes its execution and then the halt is called to end the program.