# 2.11 Parallelism and Instructions:

# Synchronization

* Cooperation means some tasks need to be writing new value and some tasks must read. And those tasks need to be synchronized.
* data race Two memory accesses form a data race if they are from different threads to same location, at least one is a write, and they occur one after another.

# Compiler

* The compiler transforms the C program into an assembly language program, a symbolic form of what the machine understands. High-level language programs take many fewer lines of code than assembly language, so programmer productivity is much higher.
* Previously, many operating systems and assemblers were written in assembly language because memories were small, and compilers were inefficient.
* Optimizing compilers today can produce assembly language programs nearly as good as an assembly language expert, and sometimes even better for large programs.

# Assembler

* The assembler can also treat common variations of machine language instructions as their own instructions is called pseudoinstructions.
* pseudoinstructions give MIPS a richer set of assembly language instructions than those implemented by the hardware. The only cost is to reserve one register for the assembler to use.

# Linker

* a single change to one line of one procedure requires recompiling and reassembling the whole program.
* Complete retranslation is a terrible waste of computing resources.
* An alternative is to compile and assemble each procedure independently, so that a change to one line would require compiling and assembling only one procedure. Which requires a new system program which is called linker or link editor.
* The linker produce an executable file which has a similar format type as an object file except it does not contains any unresolved references.

# Loader

A systems program that places an object program in main memory so that it is ready to execute.

The loader follows these steps in UNIX systems:

* Reads the executable file header to determine size of the text and data segments.
* Creates an address space large enough for the text and data.
* Copies the instructions and data from the executable file into memory.
* Copies the parameters (if any) to the main program onto the stack.
* Initializes the machine registers and sets the stack pointer to the first free location.
* Jumps to a start-up routine that copies the parameters into the argument registers and calls the main routine of the program. When the main routine returns, the start-up routine terminates the program with an exit system call.

# DLL

static approach is the fastest way to call library routines, it has a few disadvantages:

* The library routines become part of the executable code. If a new version of the library is released that fixes bugs or supports new hardware devices, the statically linked program keeps using the old version.
* It loads all routines in the library that are called anywhere in the executable, even if those calls are not executed. The library can be large relative to the program; for example, the standard C library is 2.5 MB.
* Both the program and library routines keep extra information on the location of nonlocal procedures and their names.
* In the initial version of DLLs, the loader ran a dynamic linker, using the extra information in the file to find the appropriate libraries and to update all external references.
* The downside of the initial version of DLLs was that it still linked all routines of the library that might be called, versus only those that are called during the running of the program.
* This observation led to the lazy procedure linkage version of DLLs, where each routine is linked only after it is called.

# Start JAVA

# Translating and Starting a Program

Graphical user interface

Description automatically generated

In summary, DLLs require extra space for the information needed for dynamic linking, but do not require that whole libraries be copied or linked. Microsoft’s Windows relies extensively on dynamically linked libraries, and it is also the default when executing programs on UNIX systems today.

**Starting a Java Program**

It is possible to execute Java programs just like C. Java was invented with a different set of goals. One was to run safely on any computer, even if it might slow execution time. Diagram, text

Description automatically generated

Rather than compile to the assembly language of a target computer, Java is compiled first to instructions that are easy to interpret: the Java bytecode instruction set. Like the C compiler, the Java compiler checks the types of data and produces the proper operation for each type.

A software interpreter, called a Java Virtual Machine (JVM), can execute Java bytecodes. An interpreter is a program that simulates an instruction set architecture.

The upside of interpretation is portability. The downside of interpretation is lower performance. The incredible advances in performance of the 1980s and 1990s made interpretation viable for many important applications, but the factor of 10 slowdown when compared to traditionally compiled C programs made Java unattractive for some applications.

Which of the advantages of an interpreter over a translator do you think was most important for the designers of Java?

1. Ease of writing an interpreter
2. Better error messages
3. Smaller object code
4. Machine independence