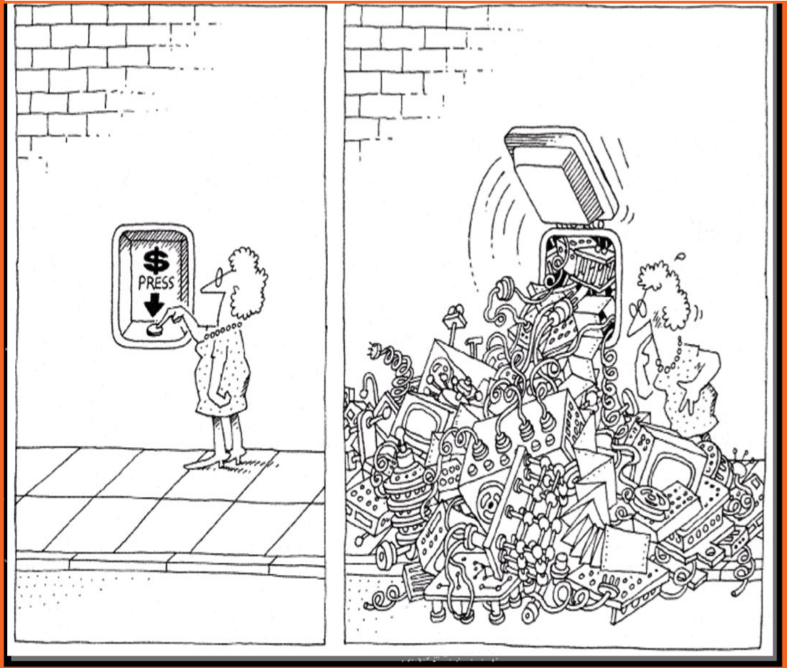
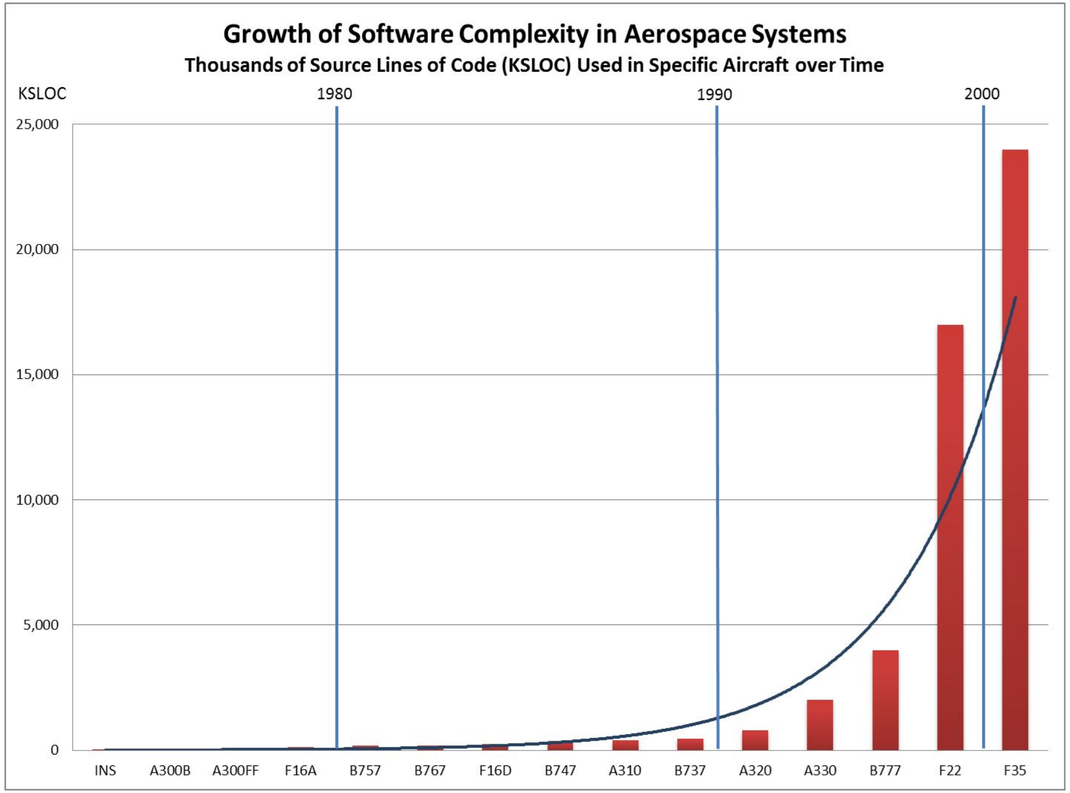
GDB Tutorial



Secure Systems Engineering
Lab session-1
Patanjali SLPSK

Why GDB?

- Increasing need for automation means that software is becoming **ubiquitous**.
- Some of the applications are **safety-critical** (need for reliability).
- Software/Program is no longer monolithic.
- Modern programs are more of blocks put together to achieve desired goal.
- Software complexity => increasing over time **(lines of code as well as interacting components).**
- **How do we debug?**



24million lines of code and 1 million interacting parts ! [1][2]

GDB

- GNU Debugger
- Helps trace and control execution of programs.
  - Works well for C, C++.
- Provides a safe environment for the debugger to :
  - Monitor/modify the value of program variables.
  - Call functions independently without altering program behaviour.
- Support for wide variety of architectures.
- Can be used in conjunction with various other tools like valgrind to create effective debug environments.

Outline

- Invoke gdb.
- Executing a program on gdb.
- Setting breakpoints and watchpoints.
- Single Stepping
- Execution Control.
- Miscellaneous.

System Requirements:

OS: Linux (32-bit or 64-bit variant).

Sample programs:

Download from moodle.

Invoking Gdb

- gdb <program-name> - begin debugging program.
- set debug commands {insert breakpoint, watchpoint}.
- run [argslist].
- quit <Ctrl-D> - ends the program.
- To enable debugging symbols we need to use "-g" flag in gcc/g++.
  - $gcc -g -o test.out test.cpp
  - $gdb ./test.out
  - (gdb) run infile.txt.
  - (gdb)

Setting breakpoints and watchpoints

- Needed to temporarily halt the execution of a program and monitor the status of variables.
  - b , break - sets breakpoint
  - Info breakpoint - prints information about the breakpoints
  - b main - breakpoint at main
  - b - breakpoint at current line
  - b+N - breakpoint at N lines from current line
  - b file:line_num - breakpoint at line_num in file.
  - b file:func - breakpoint at function in file.
  - tbreak - temporary breakpoint.
  - watch expr - monitors the change in variable { analogous to $monitor in verilog }.

Single Stepping

- s, step - executes program line by line.
- n, next - executes program line by line but does not step into functions.
- s [count] - executes <count> lines at a time.
- si - executes instruction by instruction.
- nexti - executes instruction by instruction
- until [location] -runs until next instruction (or location).
- finish - runs until selected stack frame returns.

Execution Control

- jump <line> - resumes execution at specific line.
- print [format] [expr] - print value of expression in desired format.
    - x - hexadecimal
    - d - signed decimal.
    - u - unsigned decimal.
    - o - octal.
    - t - binary.
    - a - address, absolute and relative.
    - c - character.
    - f- floating point.
- bt, backtrace - print trace of all frames in stack
- bt [n] - print trace of n frames in stack.

Miscellaneous

- Info args - prints arguments of the current stack frame.
- Info locals -prints local variables of selected stack frame.
- info frame [addr] - prints info of stack frame at [addr]
- info regs - prints info of all registers.
- Info all-regs - prints info of all registers including floating point.
- whatis [expr] - prints the datatype of expr.

When not to use gdb.

- Heisenbugs.
- Race conditions
- **Deploying attacks. - stack frame pointer, base pointers change when debug hooks are used, Compiler optimizations change the addresses too!.**