Intrusiveness | 9.1 Debugging Theory | UT.6... https://courses.edx.org/courses/UTAustinX/UT...
(https://www.edx.org)
UTAustinX: UT.6.01x Embedded Systems - Shape the World
KarenWest (/dashboard) ▼
Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)
Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)
Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Every programmer is faced with the need to debug and verify the correctness of his or her software. A debugging **instrument** is hardware or software used for the purpose of debugging. In this class, we will study hardware-level probes like the logic analyzer, oscilloscope, and Joint Test Action Group (JTAG standardized as the IEEE 1149.1); software-level tools like simulators, monitors, and profilers; and manual tools like inspection and print statements. **Nonintrusiveness** is the characteristic or quality of a debugger that allows the software/hardware system to operate normally as if the debugger did not exist. Intrusiveness is used as a measure of the degree of perturbation caused in system performance by the debugging instrument itself. For example, a print statement added to your source code is very intrusive because it significantly affects the real-time interaction of the hardware and software. It is important to quantify the intrusiveness of an instrument. Let *t* be the average time it takes to run the software code comprising debugging instrument. This time *t* is how much less time the system has to perform its regular duties. Let *Δt* be the average time between executions of the instrument. A quantitative measure of intrusiveness is

Help

$$t/\Delta t$$

which is the fraction of the time consumed by the process of debugging itself. A debugging instrument is classified as **minimally intrusive** if it has a negligible effect on the system being debugged. In other words, if *t/Δt* so small that the debugging activities have a finite but inconsequential effect on the system behavior, we classify it as minimally intrusive. In a real microcomputer system, breakpoints and single-stepping are intrusive, because the real hardware continues to change while the software has stopped. When a program interacts with real-time events, the performance can be significantly altered when using intrusive debugging tools. On the other hand, we will learn later in this chapter that dumps, dumps with filter, and monitors (e.g., output strategic information on an LED or LCD) are much less intrusive. A logic analyzer that passively monitors the activity of the software is completely **nonintrusive**. Interestingly, breakpoints and single-stepping on a mixed hardware/software simulator are often nonintrusive, because the simulated hardware and the simulated software are affected together.

CHECKPOINT 9.1

What does it mean for a debugging instrument to be minimally intrusive? Give both a general answer and a specific criterion.

Hide Answer

In general, the presence of a minimally intrusive debugging instrument itself only has minimal effect on the parameter being measured. One criterion is the total execution time required to perform the instrumentation is small compared to the execution times of the original target operation.

Research in the area of program monitoring and debugging mirrors the rapid pace of developments in other areas of computer architecture and software systems. Because of the complexity explosion in computer systems, effective debugging tools are essential. The critical aspect of debugging an embedded system is the ability to see what the software is doing, where it is executing, and when it executed, without the debugger itself modifying system behavior. Terms such as program testing, diagnostics, performance debugging, functional debugging, tracing, profiling, instrumentation, visualization, optimization, verification, performance measurement, and execution measurement have specialized meanings, but they are also used interchangeably, and they often describe overlapping functions. For example, the terms profiling, tracing, performance measurement, or execution measurement may be used to describe the process of examining a program from a time viewpoint. But, tracing is also a term that may be used to describe the process of monitoring a program state or history for functional errors, or to describe the process of stepping through a program with a debugger. Usage of these terms among researchers and users vary.

Furthermore, the meaning and scope of the term debugging itself is not clear. In this class the goal of debugging is to maintain and improve software, and the role of a debugger is to support this endeavor. The debugging process is defined as testing, stabilizing, localizing, and correcting errors. Although testing, stabilizing, and localizing errors are important and essential to debugging, they are auxiliary processes: the primary goal of debugging is to remedy faults or to correct errors in a program. **Stabilization** is process of fixing the inputs so that the system can be run over and over again yielding repeatable outputs.

Although, a wide variety of program monitoring and debugging tools are available today, in practice it is found that an overwhelming majority of users either still prefer or rely mainly upon “rough and ready” manual methods for locating and correcting program errors. These methods include desk-checking, dumps, and print statements, with print statements being one of the most popular manual methods. Manual methods are useful because they are readily available, and they are relatively simple to use. But, the usefulness of manual methods is limited: they tend to be highly intrusive, and they do not provide adequate control over repeatability, event selection, or event isolation. A real-time system, where software execution timing is critical, usually cannot be debugged with simple print statements, because the print statement itself will require too much time to execute.

Help

Help