Debugging

1. Gmake:
   1. Run but don’t execute commands: gmake –n
2. If executable, is up to date but you still want to see what gmake would do:
   1. Use touch file to update last-right-time of
3. Basic methods
   1. Stare at source
   2. Explain/demo
   3. Use debugger
4. Print statement:
   1. Easy to trace back what’s happening with your code
5. Debuggers:
   1. Control pace of execution
   2. read and write any variables
   3. breakpoints/ watchpoints
6. Rarely needed for simple programs but…
   1. Crucial when needed
   2. Always helpful
7. Gdb:
   1. –ggdb when compiling
   2. dangerous to compile –O with –ggdb or –g
   3. debugger needs direct mapping between source statements and machine instructions
8. can also append 1, 2, 3 to –g
   1. minimal info-unable to view local variables, set breakpoints or trace stack
   2. default
   3. maximal-default plus preprocessor definitions
9. gdb:
   1. gain control before execution
      1. gdb <exec file>
   2. attach to process a lready running gdb
      1. <exec file> PID
   3. port mortem on core file
      1. gdb –c <core file> or
      2. gdb<exec file <core file>
10. cheat sheets on gdb online! Find them!
11. gdb(help) gives you classes of commands—not the best
12. Breakpoint commands:
    1. Break <line> sets a breakpoint at that line
    2. Break <function> sets breakpoint before that function
    3. Print <var> displays the value of the variable
    4. Print <expr> displays value of any c/c++ expressions
    5. Print /x
    6. List<line> prints 10 source lines around line <line> in current file
    7. Run <args> runs the program
    8. Step executes one line at a time
    9. Next is like step but will do step over—execute whole call
    10. Cont is continue to next breakpoint
    11. Backtrace (bt) prints sequence of calls active
    12. Frame N (see local variable in function in stack)
13. Watchpoints
    1. Data-oriented (rather than control-oriented)
    2. Watch<var>
    3. Rwatch<var> stop when read
    4. Awatch <var> stop when read or written
    5. Display<var>
14. Other uses
    1. Construct hard to produce test cases—stresstesting
    2. Unit test without writing extra debug code
    3. Exercise every path through code
15. DDD—visual interface to make gdb better
    1. Data display debugger
       1. A GUI interface to a number of popular unix debuggers
          1. http://www.gnu.org/software/ddd
16. Technical help:
    1. Assert (3) – allows to assume the user isn’t an idiot and uses valid input
       1. When code blows up, you know it was something you did, not invalid input.
    2. Exceptions – jump from one section of the code and back
    3. Debug subclass
    4. Debug malloc libraries

MULTIPROCESSING:

1. The process:
   1. Program means either executable file or source code, depending on context
   2. Process is program in execution
   3. Many-to-many relationship
      1. Any given time, many processes may be executing the same program
      2. Over time, same process may execute several programs
2. Fork:
   1. Fork called once, returns twice(parent and child)
   2. Child can learn its PID with getpid(2)

pid\_t pid;

if((pid= fork()) < 0)

{

//error

}

else if (pid == 0)

{

//child

}

else

{

//parent

cout << “child’s pid is” << pid;

}

* 1. Child has its own copy of parents address space, file descriptors, etc

1. Process termination
   1. Process may end normally or abnormally
   2. Normal: return form main, exit(2), exit(3)
   3. All three methods
   4. Exit status: value returned by ended process
   5. Terminations status: indicates whether it was normal or abnormal termination
2. Early parent termination
   1. Function call return is analogy for process fork-wait-exit
   2. Parent must wait(2) for child’s exit status
3. Concurrency
   1. Hard to handle correctly
   2. Most common error: assuming that one process sleep(3)ing for a certain time gives another process enough time to do something
4. To show processes: ps a/ax/aux
5. To kill a program: sudo kill <process number>
6. Do not fork in a loop EVER
   1. Linux lab will crash and your access will be suspended
7. Early child termination
   1. OS saves child’s exit status
   2. OS retains data structure for child in zombie state
   3. Zombie state will stay until process is killed
8. Exec
   1. Execl, execlp, execle, execv, execvp, execve
      1. L means list rather than vector
      2. V means vector rather than list
      3. P means searches path for named executable file
      4. E meats caller provides environment as an argument
9. Argv
   1. Argv is a variable length, NULL terminated vector
      1. ./foo a 22 c
         1. argc: 4
         2. argv[0]: foo
         3. argv[1]: a
         4. argv[2]: 22
         5. argv[3]: c
         6. argv[4]: \0
10. changes across exec:
    1. almost everything is the same
    2. differences:
       1. contents of text segment
       2. signal handling
       3. file descriptors closed only if they were opened
       4. setuid—effective uid is set accordingly