Other (UNIX) Scripting Languages, Daemons and Debugging

Benjamin Brewster

Material also from Wikipedia

Scripting Languages

- The bourne shell scripting language is not the only standard UNIX scripting language

- The only other required scripting language for a UNIX system is `awk`

`awk`

- `awk` was invented by
  - Alfred Aho
  - Peter Weinberger
  - Brian Kernighan
- It is commonly used for writing one-line programs
- Popular early on because it adds computational ability to the command line

awk example

```
#!/usr/bin/awk -f
BEGIN { print "Hello, world!"; exit }
```

- Hello world in awk

Associative Arrays

- `awk` features a kind of array called an associative array
- A normal array maps numbers to arbitrary objects (i.e., whatever you pick)
- Here are examples of a normal array mapping integers to strings:
  - 6 maps to "jones"
  - 2 maps to "Nahasapeemapetilon"

Associative Arrays

- An associate array maps aribitrary objects to arbitrary objects
- Here is an example of mapping strings to other strings:
  - "Nahasapeemapetilon" maps to "Apu"
  - "Eat more beef" maps to "Kick less cats"
- Here, an object called MyObject maps to integers:
  - myObj1 maps to 6
  - myObj2 maps to 7

Associative Arrays

- An associative arrays is also called a
  - map
  - hash
  - lookup table

Perl

- Perl is a general-purpose programming language
- Written by Larry Wall, released in 1987
- Borrows features from C, shell scripting, awk, sed, Lisp, and others
- Designed to be easy to use, not necessarily elegant

Python

- Similar philosophies as Perl, but now even more widespread than Perl
- In active development and usage

- If you’re going to pick one of these up, I’d recommend Python
  - Faster, with better support for Object Oriented Programming

Perl & Python

- Perl & Python are interpreted languages

- When you want to run code you've written, it is first read by an interpreter, slightly optimized ("compiled"), and then executed.

- Perl can only be interpreted by `perl` (the Perl interpreter), Python is interpreted by `python`

UNIX Daemons

- A daemon is a process running in the background – a backgroud process
  - `syslogd` maintains the system log

- In UNIX, daemons all have the `init` process (pid == 1) as their parent

A `ps -e a` on flip

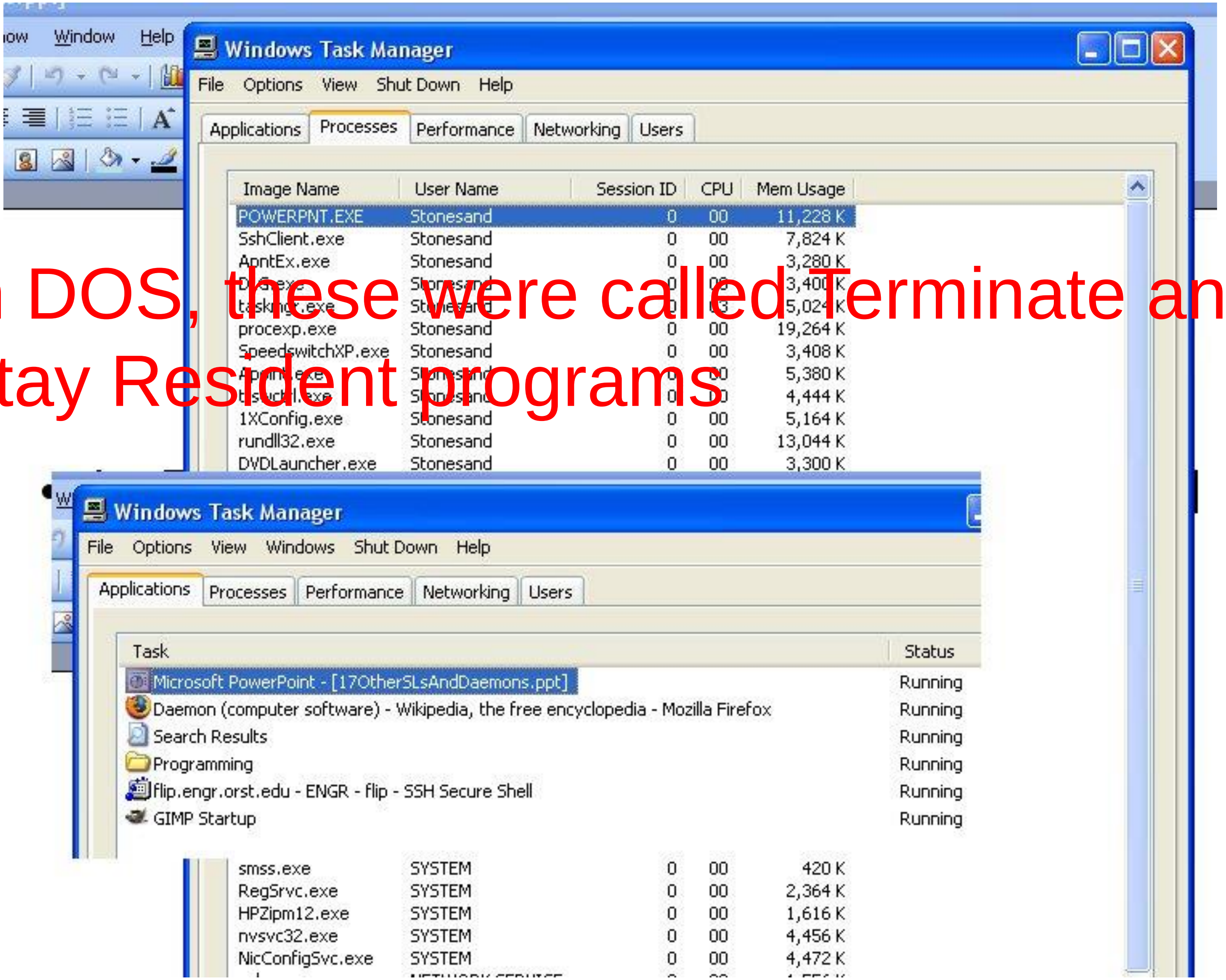
```
6067  ? S  0:00 /usr/libexec/gconfd-2 13
6107  ? S  0:00 /usr/bin/artsd -F 10 -S 4096 -s 60 -m …
31936 ? S  0:00 /usr/libexec/gconfd-2 10
21907 ? Ss 0:00 cupsd
21937 ? S  0:05 /usr/sbin/snmpd -Lsd -Lf /dev/null …
26638 ? Ss 0:00 sshd: brewstbe [priv]
26645 ? S  0:00 sshd: brewstbe@pts/0
```

init?

- "… The common method for launching a daemon involves forking once or twice, and making the parent … die while the child … process begins performing its normal function. The idiom is sometimes summarized with the phrase 'fork off and die'." - Wikipedia

These are common

- In DOS, these were called Terminate and Stay Resident programs



Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users

Image Name	User Name	Session ID	CPU	Mem Usage
POWERPNT.EXE	Stonesand	0	00	11,228 K
SshClient.exe	Stonesand	0	00	7,824 K
ApntEx.exe	Stonesand	0	00	3,280 K
[illegible]	Stonesand	0	[illegible]	3,400 K
taskmgr.exe	Stonesand	0	03	5,024 K
procexp.exe	Stonesand	0	00	19,264 K
SpeedswitchXP.exe	Stonesand	0	00	3,408 K
[illegible]	Stonesand	0	00	5,380 K
[illegible]	Stonesand	0	00	4,444 K
1XConfig.exe	Stonesand	0	00	5,164 K
rundll32.exe	Stonesand	0	00	13,044 K
DVDLauncher.exe	Stonesand	0	00	3,300 K
smss.exe	SYSTEM	0	00	420 K
RegSrvc.exe	SYSTEM	0	00	2,364 K
HPZipm12.exe	SYSTEM	0	00	1,616 K
nvsvc32.exe	SYSTEM	0	00	4,456 K
NicConfigSvc.exe	SYSTEM	0	00	4,472 K

Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users

Task	Status
Microsoft PowerPoint - [17OtherSLsAndDaemons.ppt]	Running
Daemon (computer software) - Wikipedia, the free encyclopedia - Mozilla Firefox	Running
Search Results	Running
Programming	Running
flip.engr.orst.edu - ENGR - flip - SSH Secure Shell	Running
GIMP Startup	Running

Daemons

- Daemons are typically launched at boot time to:
  - Respond to network requests
  - Monitor activities
  - Manage account billkeeping
  - Rotate/maintain/record logs
  - etc.

cron – system scheduler

- The `crontab` command is used to schedule UNIX programs to run periodically

- Commands are entered into a crontab – a a file that holds all of the commands to be run

cron

- The daemon `crond` runs in the background, and checks the cron tables once a minute to see if any of the listings need to be executed

- These are then called cron jobs

Making a cron job

```
% * * * * * commandToBeExecuted -flags
  ^ ^ ^ ^ ^
  | | | | |
  | | | | \---- day of week (0 - 6) (Sunday=0)
  | | | \------ month (1 - 12)
  | | \-------- day of month (1 - 31)
  | \---------- hour (0 - 23)
  \------------ min (0 - 59)
```

at

- `at` schedules jobs to run once at a specified time in the future

- These are called "at jobs", and are run by the `atd` daemon

Results

- Both `cron` and `at` can mail an admin the results of the jobs

Configuration

- Both `cron` and `at` have ways to list and remove jobs from the "to-run" list
- Also, `at` can be configured such that the job will only run if the system's load average is below a certain threshold

UNIX C Debugging

- Just a few notes on debugging…
  - http://en.wikipedia.org/wiki/Software_bug#Etymology

- See the Readings for a UNIX C debugging example

UNIX C Debugging

- Debugging refers to examining the state of a program, step by step, line by line, as it is running
- Typically you can also change the state of the program, i.e., change the values of variables
- Also, remember that you can examine UNIX core dumps when programs terminate in an ugly way

Using a debugger with `gcc`

- Compile with the "-g" option.

  -
    ```
    % gcc -g game.c libdb.a -o game
    ```
- Then start the debugger on the program

  -
    ```
    % gdb ./game
    ```
- In the debugger, some key commands:
  - run – (re)starts the program running; will stop at breakpoint
  - break - sets a break-point where the debugger will stop and allow you to examine variables or single step
  - step - executes a single line of C code - will enter a function call
  - next - executes a single line of C code - will not enter a function call
  - continue - continues execution again until another breakpoint is hit or the program completes
  - print - prints out a variable
  - quit – stop debugging

Debugging Demo Commands

```
1. % gdb ./game
2. (gdb) break main
3. (gdb) run
4. (gdb) print gameover
5. (gdb) step
6. (gdb) print ts
7. (gdb) run
8. (gdb) break game.c:30
9. (gdb) continue
```

Visual Studio Destroys `gdb`

- Any Integrated Development Environment destroys `gcc` and `gdb`
  - IDEs have code generation, compiling, optimization, organization, debugging, and documenting all built in
- Visual Studio 20XX rocks

The Best Debugging Technique

- Trace statements
  - Print out the value of your variables as you go

```
About to Begin Loop;   j = 0
Now At Start of Loop;  i = 0, j = 0
Incremented counter;   i = 1, j = 0
Executed file read;    i = 1, j = 2
Now at End of Loop;    i = 1, j = 2
```