CSE 410 Operating Systems
Project 2: Experience with Unix/Linux Signals
Due: 11:59 p.m., Wednesday, June 4, 2013

Project Goals

1. To understand the use and operation of signals in the Linux operating system, including
    1.1 ways to send signal to a process
    1.2 various types of signal
    1.3 how a signal handler works
2. To gain experience with signal handler implementation and registration
3. To learn/review C programming

Background

You can find information on Linux signals at: http://man7.org/linux/man-pages/man7/signal.7.html and many on-line tutorials and information pages.

Signals are a form of interprocess communication to notify a (target) process when certain events happen. Some signals (e.g., SIGINT, SIGBUS, SIGSEGV, SIGALRM) are associated with interrupts, while others are initiated in software. The most interesting feature of signals is that they are handled asynchronously, interrupting normal execution flow of the target process. When delivering a signal, the kernel simply sets a bit in the process control block (task_struct in Linux) of the target process. We say that the signal is "posted" against the process. The next time the target process is scheduled to execute, it will check these signal bits before returning to user level. The severity of the signal depends on whether and how it is "handled" by the target process. Upon receiving a signal, a process may:

1. Perform a default signal handler, as defined by the kernel (for example, terminating itself).
2. Ignore the signal or temporarily block the signal (only allowed for some signals).
3. Execute a user-defined signal handler. A signal handler is a pre-defined function to be executed (at user level and in the context of the target process) upon receipt of the signal. We say that such a signal is "caught." In this project, we will implement a few signal handlers. Three features of signal handlers are worth noting:

• Not all signals can have a user-defined signal handler. For example, the default action of a SIGKILL signal (terminating the process) cannot be overridden.
• To make a signal handler effective, it must be registered with the kernel, using the signal() or sigaction() system call. See the example below.
• A signal handler *does not* take any parameter except a signal number. In another words, in this project you will need pass data to the signal handler in a different way.

As noted, some signals are posted against a process by the kernel in response to an interrupt, such a as a bus error or segmentation fault. Others are under control of the user. For example, when the user presses ctrl-c, generating a keyboard interrupt, a SIGINT signal is posted against the currently running process. By default, the process will terminate. Alternatively, this signal can be caught, with control transferred to a user-defined signal handler. Two signals (SIGUSR1 and SIGUSR2) are even available for the programmer to use in a custom manner.

Signals can also be sent from one process to another using the (poorly named) kill() system call. Unless the process has superuser privileges, generally the two processes have to be owned by the same user. A special case is raise(), a library routine that can be invoked by a process to send a

signal to itself. Finally, a user can invoke kill at the command line, which sends a specified signal to the target process by use of the kill() system call. Follows is a simple example, written in C, of handling the SIGALRM signal.

alarm_example.c

```
#include<stdio.h>
#include<signal.h>
void alarm_handler(int signo)
// user defined signal handler for alarm.
{
    if (signo == SIGALRM)
    {
        printf("alarm goes off\n");
    }
}

int main(void)
{
    // register the signal handler
    if (signal(SIGALRM, alarm_handler) == SIG_ERR)
    printf("failed to register alarm handler.");
    alarm(3);
    while(1);
    // set alarm to fire in 3 seconds.
    // wait until alarm goes off
}
```

Project Description

The basic problem to solve in this project is very simple, and perhaps some of you CSE231. You are asked to write a program that counts instances of specified words in a given text file. You are to calculate counts for five stop words:

1. "a"
2. "an"
3. "the”
4. "is"
5. "are"

as well as a special keyword,

6. “sparty”

Here, you do not need to match any words that contain upper-case letters. We are matching only lower-case words. A function strtok(), might be helpful for parsing a line of text into words. An example is available at the following URL: http://www.cplusplus.com/reference/cstring/strtok/. For simplicity, we only use white space as delimiter. For debugging of the text processing part of your program, you will probably want to use relatively small test files.

For this assignment, more important than the text processing is the handling of signals. During file processing, which may last for several seconds, your program should output the current line number being processed *every 2 seconds*. This can be done by defining your own SIGALRM handler, which also needs to schedule the next alarm. The alarm handler is invoked every 2 seconds and the current line number is displayed. For testing purposes, you will need to use a large text file because with a small input file, your program will terminate even before the first alarm signal is received. A sample test file is available at http://www.cse.msu.edu/~cse410/data/mail.data.zip.

NOTE: You must copy this file to /tmp on your lab machine and unzip it there. This will

ensure that you are reading from the local disk, instead of across the network.

In addition, your program should handle signals generated by keyboard interrupt. When ctrl-c is pressed, instead of by default terminating the process, your program should ask user whether she wants to quit or not. User can input “y” or “n” characters. If the user's choice is “y,” then your program should print the statistics so far and quit. Otherwise it should continue to run.

Finally, your program should count instances of the keyword 'sparty'. For every 10,000 occurrences, a message should be displayed, reporting the current number of occurrences. **Please note that, this message must be displayed in a signal handler of user-defined signal (SIGUSR1).** In other words, for every 10,000 occurrence of 'sparty', your program should send itself a SIGUSR1 signal, telling itself to output the message. **You will not receive credit for this functionality if you print the message in your main function.**

Project Specifications

In summary, your program should :
1. Take one parameter, the input filename
2. **In a signal handler**, output the “sparty” count for every 10,000 occurrences. **You will not receive credit if you print it in your main function.**
3. While reading and processing the input file, at intervals of 2 seconds, print out the number of the current line being processed.
4. Handle keyboard interrupts. When the user presses ctrl-c, your program should ask the user if she/he really wants to quit. Your program quits if the choice is yes, or continues to run otherwise.
5. At the end of your program, output the number of occurrences of all 6 words (5 stop words + sparty).

Deliverables:

proj01.c
Sample output:

```
$ ./proj01 input.txt
reading line #1293054 ...
keyword sparty occurs 10000 times.
reading line #2629944 ...
reading line #3900384 ...
keyword sparty occurs 20000 times.
reading line #5317423 ...
keyword sparty occurs 30000 times.
reading line #6502010 ...
reading line #7804032 ...
keyword sparty occurs 40000 times.
reading line #9136584 ...
keyword sparty occurs 50000 times.
reading line #10314624 ...
reading line #11311500 ...
keyword sparty occurs 60000 times.
reading line #12465676 ...
reading line #13714569 ...
keyword sparty occurs 70000 times.
reading line #14934328 ...
keyword sparty occurs 80000 times.
reading line #16248760 ...
```

a: 496262
an: 92268
the: 1382135
is: 305867
are: 156795
sparty: 85345
Another sample output:
reading line #1277628 ...
keyword sparty occurs 10000 times.
reading line #2647584 ...
reading line #3948576 ...
keyword sparty occurs 20000 times.
reading line #5396183 ...
keyword sparty occurs 30000 times.
^C Exit? (Y/N)
y
$
One more sample output
reading line #1418102 ...
keyword sparty occurs 10000 times.
reading line #2775844 ...
keyword sparty occurs 20000 times.
reading line #4097432 ...
reading line #5579241 ...
keyword sparty occurs 30000 times.
^C Exit? (Y/N)
n
keyword sparty occurs 40000 times.
reading line #8912404 ...
keyword sparty occurs 50000 times.
reading line #10135674 ...
reading line #11193550 ...
keyword sparty occurs 60000 times.
reading line #12280540 ...
reading line #13552120 ...
keyword sparty occurs 70000 times.
reading line #14749386 ...
reading line #15915158 ...
keyword sparty occurs 80000 times.
a: 496262
an: 92268
the: 1382135
is: 305867
are: 156795
sparty: 85345