Suggesting webpages with uniform online sampling of text

Saad Mneimneh
Computer Science
Hunter College of CUNY

General description

We have learned in one of the homework assignments a technique to sample words from a text uniformly at random. For instance, by simply maintaining a buffer of n words, all subsets of size $\min(n, m)$ are equally likely to end up in the buffer (with probability $1/C_n^m$), and this can be achieved without knowing in advance the size m of the text. This is particularly important when dealing with online (realtime) input, thus the name online sampling. Recall the algorithm:

$m = 0$
upon receiving a word w
  $m = m + 1$
  with probability $\min(1, n/m)$
    if buffer is full
      drop one of the n words uniformly at random
      shift left the words in the buffer
      place w in the last position of the buffer
    else
      place w in the first empty position of the buffer

The purpose of this project is to uniformly sample words from text, and use the sample to suggest webpages that may contain useful information on the subject described by the text. Due to the online nature of the sampling, the webpages may be updated periodically (when the user is typing the text). The main features of the program will be as follows:

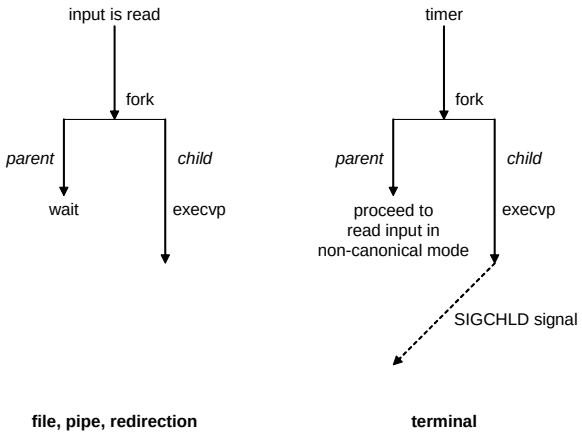
- use of online sampling of words, the text is assumed to be received from a file, or the standard input (pipe, redirection, terminal)
- use of fork and execvp to launch a web search using the words sample
- use of signals and timers to periodically update the webpages as text evolves (when input is received from terminal)

Online sampling

Regardless of the input source, words will be uniformly sampled from text as described in the algorithm above. The parameter n is chosen by the user. To avoid sampling words such as 'a', 'in', and 'the', only words larger than a certain size will be considered. The size threshold may also be provided by the user.

Web search, e.g. google

The web search will be invoked by a fork and the execvp function. The behavior of the fork will depend on the source of the input, as illustrated below:



When input is received from a file, a pipe, or a redirection, the fork is invoked after the input is read and the sample is obtained. The child process executes a web search, while the parent process waits for the child to be done. The result of the web search is then accessed by the parent process.

When input is received from the terminal, a fork is invoked periodically, and only when the sample has changed enough (it is up to the programmer to determine the criteria). As before, the child process executes a web search; however, a major task of the parent process is to continue to read input to guarantee appropriate responsiveness to the user. The parent process will therefore be notified by a signal, SIGCHLD, that the child process is done.

The web search can be done by executing the wget command, e.g.

```
wget -U "" "http://www.google.com/search?hl=en&q=word_1+word_2+...+word_k&aq=f&oq="
```

To avoid using all n words in the search, a number of searches will be performed with the wget command, and only k words (e.g. 5) will be used for each. The k words will be consecutive words taken from the buffer (it is important for the search that words are close rather than distant).

The wget command will place the result in a file that the parent process can access and parse to obtain relevant web pages. This file can then be removed.

Timers and signal

Special care is needed in the design of the handlers and the interaction of the different parts of the program. In particular, the program should be responsive and not place the user on hold when accessing the network if the source of the input is the terminal. On the other hand, asynchronous input can be used to avoid polling for input in the parent process.

Further detail

Further detail on the project and implementation can be obtained through discussion between the student and the professor.