CSCI 6730 Operating Systems

Project #1: Multi-process and IPC

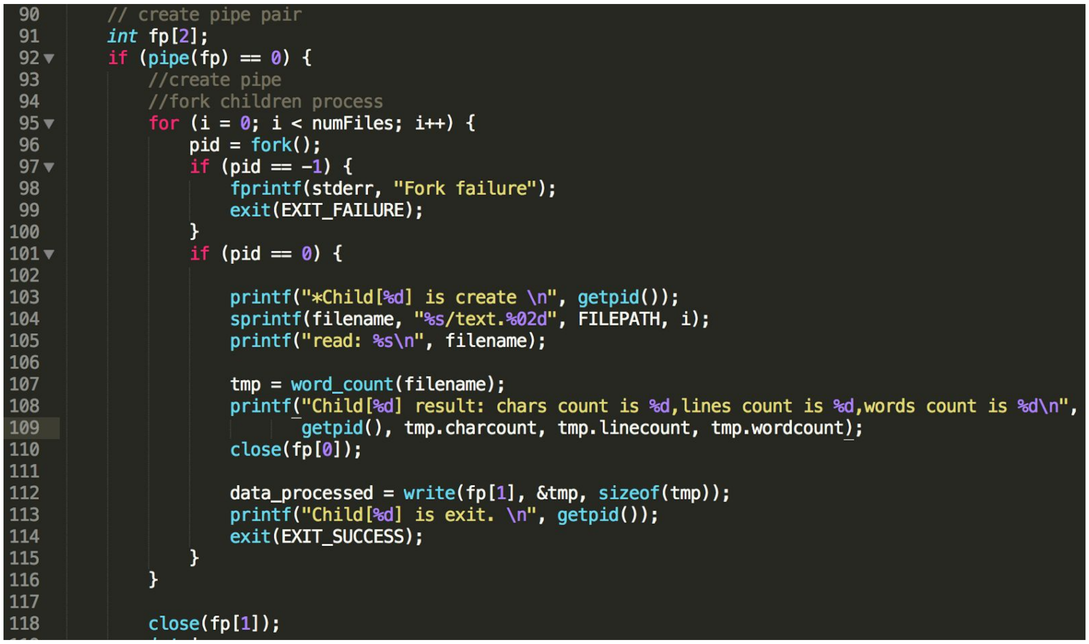
Xiaodong Jiang
Department of Statistics
Feb. 6th, 2017

Introduction

In this project, we design and implement a multiprocess word counting program by converting a given single-process word counting program in C language, the key idea is use ***fork()*** to create child and parent processes to read and count the number of word/char/lines simultaneously, while employing ***pipe()*** to achieve the inter-process communication (***IPC***). In the end, we compare the running time of single-process and multiprocess versions, and observed a great advantage to use multiprocess program, which runs much faster while achieving the same results.

Main idea on fork and pipe (IPC)

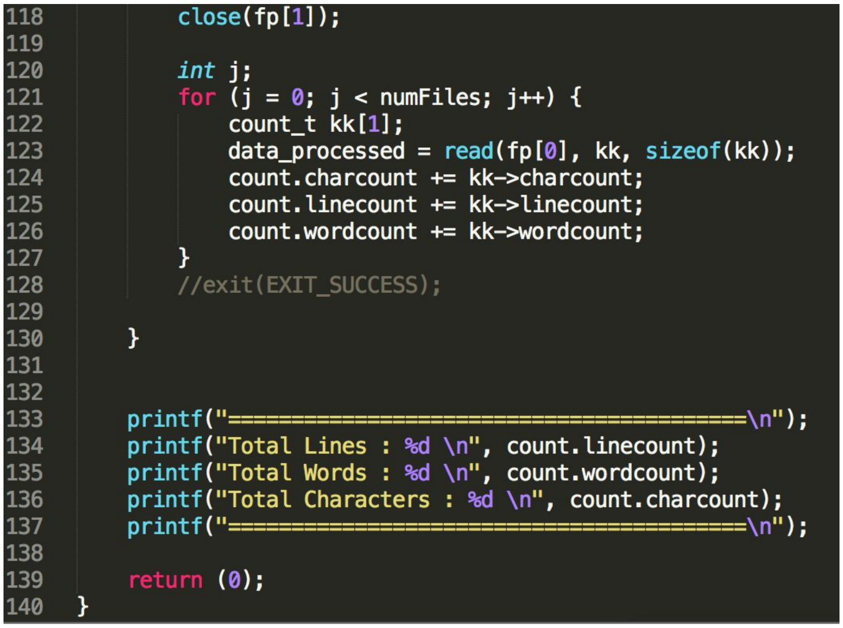
In the main function, instead of implementing a sequential loop to read the data one by one, we first create a pipe, which could return two file descriptors referring to the ends of the pipe, i.e., write end and read end.



```
90      // create pipe pair
91      int fp[2];
92      if (pipe(fp) == 0) {
93          //create pipe
94          //fork children process
95          for (i = 0; i < numFiles; i++) {
96              pid = fork();
97              if (pid == -1) {
98                  fprintf(stderr, "Fork failure");
99                  exit(EXIT_FAILURE);
100             }
101             if (pid == 0) {
102
103                 printf("*Child[%d] is create \n", getpid());
104                 sprintf(filename, "%s/text.%02d", FILEPATH, i);
105                 printf("read: %s\n", filename);
106
107                 tmp = word_count(filename);
108                 printf("Child[%d] result: chars count is %d,lines count is %d,words count is %d\n",
109                         getpid(), tmp.charcount, tmp.linecount, tmp.wordcount);
110                 close(fp[0]);
111
112                 data_processed = write(fp[1], &tmp, sizeof(tmp));
113                 printf("Child[%d] is exit. \n", getpid());
114                 exit(EXIT_SUCCESS);
115             }
116         }
117
118         close(fp[1]);
```

As shown in the screenshot above, we fork n pairs of processes at the read end of pipe with a loop in *line 95-116*, while first making some ***error message handling*** depends on the process id (pid), then call the child function - ***word_cout*** - in each child process when (pid==0). An important step here, is to close the read end before collecting the results from all child process, as we did in *line 118.*

Now, we make another loop to collect the results from all child process above, see the code below,

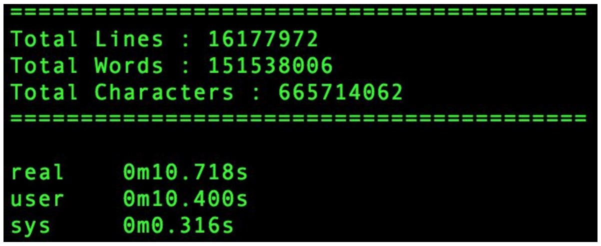
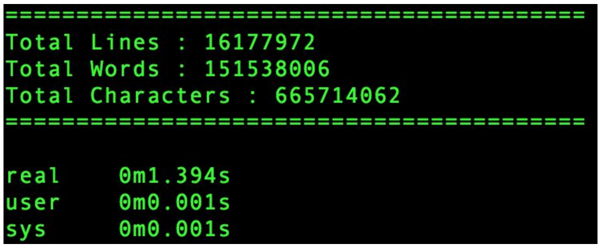


```
118         close(fp[1]);
119 
120         int j;
121         for (j = 0; j < numFiles; j++) {
122             count_t kk[1];
123             data_processed = read(fp[0], kk, sizeof(kk));
124             count.charcount += kk->charcount;
125             count.linecount += kk->linecount;
126             count.wordcount += kk->wordcount;
127         }
128         //exit(EXIT_SUCCESS);
129 
130     }
131 
132 
133     printf("=========================================\n");
134     printf("Total Lines : %d \n", count.linecount);
135     printf("Total Words : %d \n", count.wordcount);
136     printf("Total Characters : %d \n", count.charcount);
137     printf("=========================================\n");
138 
139     return (0);
140 }
```

In this loop, we create a count_t data structure and read the data/results from the pipe, and add three kind of count to the corresponding variables in ***count***.

Results Comparison on nike Machine

We can verified that, to read 10 files, the multiprocess program is much faster (almost 10 times) while obtaining the same results.

Single-Process Word Counting	Multiprocess Word Counting
	

```
=========================================
Total Lines : 16177972
Total Words : 151538006
Total Characters : 665714062
=========================================

real    0m10.718s
user    0m10.400s
sys     0m0.316s
```

```
=========================================
Total Lines : 16177972
Total Words : 151538006
Total Characters : 665714062
=========================================

real    0m1.394s
user    0m0.001s
sys     0m0.001s
```