Assignment 3: Multi-process Merge Sort

Announce Day: Wednesday, November 23, 2011
Due Day: Tuesday, December 13, 2011

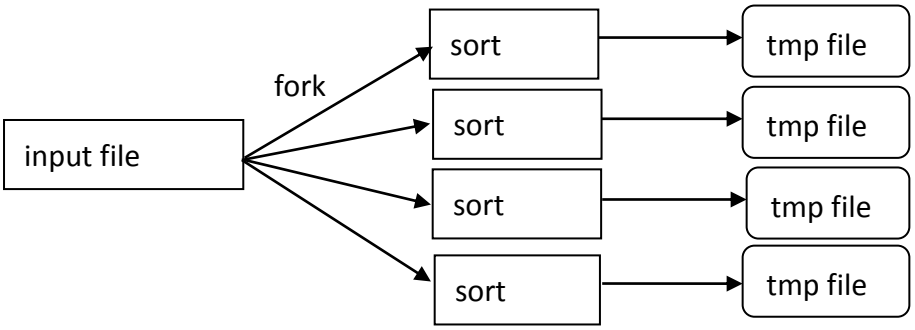
Goal

The goal of this assignment is be familiar with process control mechanism in multitasking environment and understand the performance trade-off among different process control mechanisms. You will implement a multi-process merge sort program, using fork(), vfork(), and related functions.

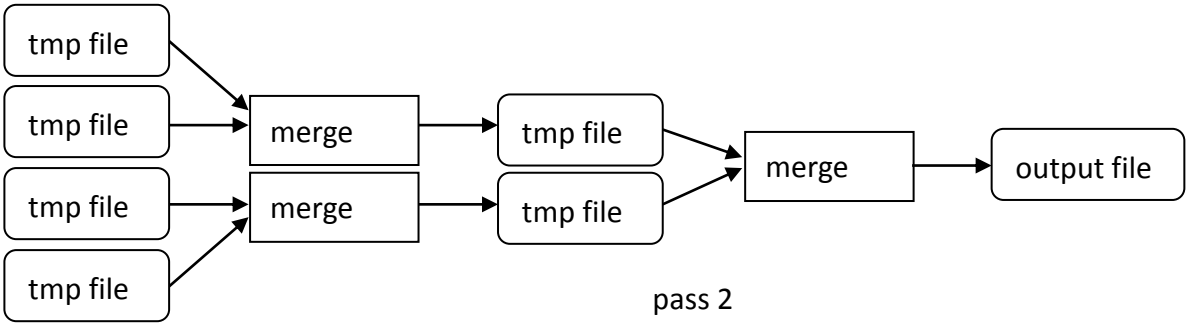
Description

Merge sort algorithm is an effective sorting algorithm when it is not practical to store all the targeted objects in process memory. The merge sort algorithm mainly consists of two steps.

1. Divide and sort:
   The given input file contains large number of objects, which cannot be read and stored as a whole in a process memory. Hence, the program creates several child processes (We call them sorting processes.) using fork() or vfork(). Each of the sorting process reads a chunk of input file, sorts the numbers, and outputs them into a temporary file.



2. Merge
   After sorting processes terminate successfully, the parent process creates child processes again (We call them merging processes.) to merge files. Each merging process merges certain number of temporary files into a bigger one. It may take multiple merge iterations to generate the complete sorting results. The parent process is responsible for taking care of the progress of these merging processes.



In this assignment, you have to implement merge sort in two versions using fork() and vfork(). To focus on learning process control, the functions for sorting, merging and R/W temporary files are provided. Please use the functions and programs given by TA. For fork() version, your program calls TA's functions; for vfork() version, you are required use vfork() and exec() to run TA's programs.

File format

The given input file, named "num.txt", contains large number of objects to be sorted. Each object has 9 digits (characters). Preceding zeros are filled if the number does not have nine digits (i.e. less than 100,000,000). For example, 15 will be represented by 000000015 in the file. A new line character ('\n') follows each number, including the last one.

The temporary and output files are in the same format as the input, containing numbers sorted in ascending order. The temporary file names are in the form of "merge_X_Y" which means the X-th object to the Y-th object (including X, not including Y) in "num.txt" are sorted and stored in this file. The output file name is "merge_0_N" where N is the number of objects in "num.txt".

I/O format:

merge_sort_fork and merge_sort_vfork

The main programs you need to write are "merge_sort_fork" and "merge_sort_vfork" which receive 3 command-line arguments:

```
$ merge_sort_fork MAX_PROCESS    LENGTH_OF_CHUNK    MERGE_DEGREE
$merge_sort_vfork MAX_PROCESS    LENGTH_OF_CHUNK    MERGE_DEGREE
```

- MAX_PROCESS indicates maximum number of simultaneously running child

(sorting and merging) processes. If number of child processes reaches MAX_PROCESS, the parent process should wait for some children to terminate before creating new process.

- LENGTH_OF_CHUNK indicates number of objects that each sorting process should handle. The only exception is that the last chunk may be smaller.
- MERGE_DEGREE indicates number of temporary files a merging process should read. Only the last merging process in a merging pass may receive fewer than MERGE_DEGREE files.
- Note that MAX_PROCESS and MERGE_DEGREE specified by the arguments may exceed system limits (e.g. CHILD_MAX). If the parent process cannot create more child process, it should wait for some children to terminate and attempt to create again.
- If all of the 3 arguments are zero, the main program has to decide number of its child processes and merge degree by itself to minimize execution time. Please see the grading section for detail.

sort and merge:

One of the programs given by TA is “sort". You have to execute it in the vfork version program. In fork version, your program has to call the function “MP3_SORT”. Both the program and the function receive 3 arguments.

```
$ sort    BEGIN_INDEX    END_INDEX    TMP_FILE_NAME
MP3_SORT(int BEGIN_INDEX, int END_INDEX, char* TMP_FILE_NAME)
```

- BEGIN_INDEX and END_INDEX are nonnegative integers.
- TMP_FILE_NAME is the temporary file name.
- This program/function will read “num.txt” from the BEGIN_INDEX-th number to the END_INDEX-th number (include BEGIN, not include END), sort them, and output to TMP_FILE_NAME as stated in the first 2 pages.
- Buffer capacity of the program/function is at most 100000. That is, END_INDEX – BEGIN_INDEX cannot be greater than 100000.

Another program given by TA is “merge” which merges several temporary files into one file. The function for fork version is “MP3_MERGE”. The function and program receive at least 2 arguments.

```
$ merge OUTPUT_FILE_NAME TMP_FILE_NAME1 TMP_FILE_NAME2 ……
MP3_MERGE(int NUMBER_OF_FILE_NAME, char* FILE_NAME[])
```

- TMP_FILE_NAMEs are temporary files input to the program.
- OUTPUT_FILE_NAME is the output file.

- FILE_NAME is an array containing NUMBER_OF_FILE_NAME strings. FILE_NAME[0] is the output file name; FILE_NAME[1] to FILENAME[NUMBER_OF_FILE_NAME - 1] are input to MP3_MERGE function. All of the files follow the format described above.
- The merge program/function reads numbers from the temporary files and output them to the specified file in ascending order.

Remember that your vfork version program has to execute “sort” and “merge” by calling exec(). In fork version, your program has to call the functions “MP3_SORT( )” and “MP3_MERGE( )” declared in the header file “merge_sort.h”.

Sample

Sample 1

num.txt

```
000001000
481650182
000027151
000000333
```

Execute main program

```
$ ./merge_sort_fork 5 2 2
```

merge_sort_fork calls MP3_SORT

```
MP3_SORT(0, 2, ”merge_0_2”);
MP3_SORT(2, 4, “merge_2_4”);
```

Temporary file “merge_0_2”

```
000001000
481650182
```

Temporary file “merge_2_4”

```
000000333
000027151
```

merge_sort_fork calls TA’s function

```
char* name[3] = {“merge_0_4”, ”merge_0_2”, “merge_2_4”};
MP3_MERGE(3, name);
```

Final output “merge_0_4“

```
000000333
000001000
000027151
481650182
```

Sample 2

num.txt

```
333222111
111222333
001001001
000000000
```

Execute main program

```
$ ./merge_sort_vfork 1 3 5
```

exec() in merge_sort_vfork runs TA's program

```
$ ./sort 0 3 merge_0_3
$ ./sort 3 4 merge_3_4
```

Temporary file "merge_0_3"

```
001001001
111222333
333222111
```

Temporary file "merge_3_4"

```
000000000
```

exec() in merge_sort_vfork runs TA's program

```
$ ./merge merge_0_4 merge_0_3 merge_3_4
```

Final output "merge_0_4"

```
000000000
001001001
111222333
333222111
```

Submission:

Related files: https://sites.google.com/a/joen.cc/sp_hw/sp_mp3

Put your source codes, Makefile, and README.txt into a directory named your student ID. Compress the directory to gzip format and submit it to the course website.

http://ceiba.ntu.edu.tw/1001sp

http://ceiba.ntu.edu.tw/1001spcsie

Here are the commands to compress files.

```
$ mkdir b99902000
$ cp Makefile README.txt merge_sort_fork.c merge_sort_vfork.c b99902000
$ tar -zcvf b99902000.tar.gz b99902000
```

Before testing, TAs will copy their files into your directory. Please make sure

your Makefile and codes can be compiled to merge_sort_fork and merge_sort_vfork using the following commands.

```
# working directory is b99902000
# TA's codes are in ../lib/
$ cp –r ../lib ./
$ make
# you may need "make -C lib/" in your Makefile to compile TA's program
```

Grading:

Plagiarism is not tolerated. Your source codes will be checked by moss. (http://theory.stanford.edu/~aiken/moss/)

For submission after deadline, 5% of the credits will be deducted for every single day delay.

There are 5 tests in this assignment. None of them contain invalid input such as negative numbers, MERGE_DEGREE = 1, some arguments = 0, or wrong file format.

1. Compilation & execution (1 point)
   If TA's script can compile and execute your program, you get 1 point.
2. Correctness (1 point)
   Your merge_sort_fork has to call TA's function and merge_sort_vfork has to execute TA's program. Both of the programs have to produce correct temporary files and final output file as specified above.
3. Number of processes (1 point)
   Your program has to detect whether number of child processes reaches MAX_PROCESS or system limit.
4. Multiple process merge (1 point)
   Your program should perform process control in multi-pass merge.
5. Performance (2 points)
   If all arguments are zero, your programs should decide number of processes and merge degree by themselves. The TAs will measure your programs' running time under 2 input sets, one of which is public. If your programs runs faster than 80% of your classmates (including the other session), you get 2 points. And 1.5 points for 50%, 1 point for 20%, 0.5 point for 0%.

Last but not least, misuse of fork() will result in heavy system load or even crash. Please write and test your programs carefully.