Αλέξανδρος Συμπέθερος

AM: 1115200900261
Προγραμματισμός Συστήματος
Άσκηση 3
README

Run on personal computer: Ubuntu Linux (10.10,12.04) and on a few Linux machines in the Unix Lab(linux02, linux03)
Code is 100% my own,
Helpful sources for coding problems: manpages, reference pages for standard library functions online(cplusplus.com for example), and various unknown blogs for valgrind error and ideas.
Code has been checked with valgrind and 0 leaks possible(for parent and children). There is 1 error, but some searching shows that the error is due to a valgrind bug.
A makefile exists, compiles everything into .o first(allowing following compiles to skip unchanged files), and has separate compilation for main.c and worker.c. A clean function is also available.
Also supplied is a script(go.sh) for automatically running make + the executable with a preset of parameters. The first argument is automatically set to 1, but if -d is given, the process is run in debug mode, if a number is given, the program is run that many times. The second argument tells the script whether to read from file(1) or given url(2).

About the code:
-Inline Arguments:
The program follows the exercise criteria.
User must give all arguments requested, otherwise error will appear.
If any arguments are written incorrectly, do not have the necessary paths(-f without file following), or two arguments that clash are given at once(-f and -u), then error will appear.

Arguments:
-f input_file
-u url
-p number_processes
-n number_urls
-s output_file

Structure of program:
In the manager part of the code(main.c), there exist the following structures:
-hash table
Hash table is created to allow program to ask quickly if a certain url has already been crawled. Each cell has a list(allowing overflow), that increases when an url is added. When the program has added to the hash an url, it is also added to the queue. If the total number of url's requested have been added to the hash, no more url's are allowed(this saves a huge amount of space).
-queue
A simple queue that holds urls and removes them one by one, when the manager wants to give an url to a child. Queue has maximum number of urls equal to the max requested, keeping the size of it at a minimum.
-various small simple tables, structs, integers
These are used for pipes, for checking if processes are busy, for keeping information(which process is working on which url) and various other uses.

About the Manager(main.c):
The manager, besides setting various variables, reading and using the arguments, starts by creating

2 pipes for every process(child) and forking the process into existence. The processes then exec's itself into worker.c(the child code). After creating the pipes/processes, the manager freeze's until all children send a signal alerting the processes that they are ready. This happens because if the manager started right away, it may finish its work and send a termination signal before some child set up its trap and in this way the processes would wait for a signal that will never come.
After receiving the green light, the manager starts the main part of its code. Inside of an unconditional loop, the manager does a few important things. Firstly, the manager sends as many url's as possible to as many available processes itcan. Secondly, if in total, the requested amount of url's have been sent, the manager signals the processes that it is time to finish up and terminate. Thirdly, the manager freezes(poll's) until any child has written something to its pipe. In this case, the manager reads all the url's parsed from the given url, plus a struct with various statistics. The above three actions will continue to happen until the requested amount of url's have been given to the processes, computed, and collected from the manager.
Afterwards, the manager awaits a termination signal from all children and then sort's alphabetically the url's given, sort's the url's parsed from these files and writes all the above plus the statistics for each url given and the total statistics into a output file.

A few small notes for the manager:
-Because poll unfreezes if a child has closed his end of a pipe, when the manager has sent the requested amount of url's, any child not busy, gets a different pipe, a temporary pipe with no information, allowing poll to work correctly for the remaining processes. (This was a tricky case where busy waiting occurred, even though poll existed.)
-The program works with url files of any size(0 url's will result in an error).
-If not enough url's exist to satisfy the request(from the input file/url plus all crawled url's), an error is sent.

About the Child(worker.c):
The child, besides setting various variables and actions(one of which is to handle the signal the parent will send to terminate), sends a signal to the parents alerting him that it is ready. Afterwards begins the main loop of the child. In each loop, the child begins by polling two pipes. The first pipe is the read end of the pipe given from the parent(the url's the child will be given go here) and the second pipe is the special pipe the signal uses(Explanation for this action is in the notes below). After unfreezing, the child reads a size and url from the parent, calls wget, and parses the url based on a few rules. Depending on whether the url successfully was downloaded or not, the program parses or sends empty results. For every url parsed, it is sent right away to the parent and when no other url's available, a struct with statistics are sent. If the processes unfreezes due to the pipe from the signal, it sends a termination accomplished signal to the parent and then finishes.

-Notes:
-About the extra pipe, when the child is running, at some point a signal will occur. In order to handle this signal without any critical section problems, the signal writes to a special pipe, and then when poll is called again, it checks both the read pipe from the parent and this pipe.
With this “trick”, no matter than the signal happens, no data is lost(from read for example) and the program will not lock in read if the signal is sent.
If for example, the signal increased a global int, there is a small critical section between if(global! =0) and read, that will make the code freeze forever.

All other files are rather self-explanatory.