Task Series 1

1.1 FIFO pipes

Implement a "library" of one-way FIFO pipelines, with the following programming interface:

int pipe_open(int size);	Create and open pipeline for writing and reading. Returns the ID of the pipeline created (positive integer).
int pipe_write(int p, char c);	Writing a byte to pipe p. Returns 1 on success, or -1 if there is no open pipe to write to with this identifier.
int pipe_writeDone(int p);	Closes the p pipeline for writing. Returns 1 on success or -1 if there is no open writeable pipe with this identifier.
int pipe_read(int p, char *c);	Remove one byte from pipeline p. Returns 1 if the read is successful, 0 if the pipe is empty and closed for writing, or -1 if there is no pipe open for reading with this identifier. If 0 is returned the pipeline is closed and destroyed automatically.

Assume that only one thread calls read and only one thread calls write at any time, but read and write can be called at the same time. Base the implementation on the ring buffer technique trying to avoid race conditions. Blocking, where/when needed, should be done with active standby. Check your implementation through a program that takes as an argument the name of a file, creates two threads and two pipes of size 64 bytes. The first pipeline is used to transfer the contents of the file from one thread to another, which will save the data to a new file (copy) with the same name and ending ".copy". The second pipeline is used to transfer the contents of the copy in the opposite direction. The second transfer starts after the first one is finished, and the data is saved in a second copy with the extension ".copy2".

1.2 Recognition of prime numbers

Implement a program that iteratively reads from its input integer values, for each of which it evaluates whether it corresponds to a prime number and prints 1 or 0 respectively. To speed up the calculation, the program uses N threads, in the spirit of the following pseudocode (N program argument):

main thread:	worker thread:

create N workers
while (input exists) {
  read next value
  wait for a worker to become available notify the
  worker to process the value
}
notify workers to terminate wait for all
workers to terminate
	while (1) {
	  notify main that I am available wait for
	  notification by main if notified to
	  terminate, break else process assigned
	  value
	}
	notify main that I will terminate

Use any prime finding algorithm you like. Analyze the performance as a function of N and the values accepted by the program. Run calculations for small and large prime numbers.

1.3 External merge sort

Implement a parallel, threaded version of external mergesort. At each recursion level, the current thread splits its assigned portion of the file into two, assigns the sorting of each portion to a separate thread it creates for that purpose, waits for both threads to terminate, merges the two portions into a single sorted section (without necessarily using any temporary intermediate file), and returns. If the remaining portion is small enough (64 integer values), the contents are sorted in memory (use any method you wish). Threads and the variables used to synchronize them must be created and destroyed**dynamically** in retrospect. Test your implementation through a program that sorts the contents of a binary file with integer values, the name of which is taken as an argument.

The implementation must be done in C using the pthreads library. Synchronization between threads should be implemented with simple variables**without** the use of synchronization mechanisms of pthreads or the operating system.

Delivery: SaturdayNovember 18, 2023, 11:59 p.m