CS 4348/5348 Operating Systems -- Project 1

The goal of this project is to let you get familiar with how to create processes and how to communicate between processes. The project has to be programmed in C or C++ and executed under Unix (or Linux). We separate the project into multiple phases to help you build a working program. You only need to submit your final program. If you cannot get the final program to work, you can submit a working program for a certain phase. Non-working code will get 0 point.

1 Phase 1: Process Creation

Try to familiarize yourself with the Unix "fork" system call for process creation. Let the original process be the "Admin" process. From the "Admin" process, fork one process called the "Computer" process. In each process, print out its

ID, returned-pid, PID

ID of the processes should be "Admin" or "Computer", corresponding to the definitions given above. The returned-pid is the value returned from fork(). PID is the actual process ID and you can use getpid() to obtain it. Both returned-pid and PID are integers.

2 Phase 2: Process Communication via Socket

The Computer process can perform some computing for the Client. The Client sends a file name to the Computer. The file contains M+1 or more integers. The first integer is M. Following M, there should be at least M integer values. When the Computer process receives the computing request from a client (the file name), it reads from the file (with the given name) the M value, and then read M integers and sum them together. The sum is then sent back to the Client process. The client process subsequently prints the summed result.

The Computer process should create a server socket to accept the Client connections. The server port number should be the same as the one used by the Client process. We will use command line input for the Admin process to provide the **port number**. When the Admin process fork the Computer process, the port number should be passed along. Note: If two students use the same port number on the same host, there will be conflict and may not be easy to detect. You need to watch out for this potential problem. We recommend that each student uses <1..4>+ <last 4 digits of your student id> to avoid most of the potential conflicts. Also, please check the return values of socket related calls to make sure that server socket initiation is successful.

The Computer process needs to create the server socket before running the Client program to avoid having failed connection attempts. You can achieve this sync manually by starting the Client program only after the Computer process prints out a message:

Computer process server socket ready.

After this message, you can start your Client program. When the server socket receives a connection request from a Client process, it accepts the connection which returns a new socket. The Computer process can communicate with this specific Client using this returned socket (receives the file name sent by the Client and sends back the summing result).

You should write a separate program for the Client. Each Client process should have a unique Client-ID which is an integer value assigned as the command line input. The Client process starts by connecting to the Computer process. After the connection, it goes into a loop. In the loop, it reads in a file name from stdin, sends the file name together with its Client-ID to the Computer process, wait to receive the summing result

from the Computer process, and print the summing result on stdout after receiving it. To make it easier, we assume that the file name will always be 8 bytes. If the input file name is "nullfile", then the Client process terminates. If you have flexible file name length, it is not a problem. We will use our own test cases during testing.

Besides the Client-ID, the Client process also need to know the server name and server port number (server is the Computer process), which should also be provided via the command line input. Thus, you need to start your Client program as follows:

Client.exe Client-ID Server-host-name Server-port-number

When the Client sends the request to the Computer, and when the Computer process replies to the Client, the format of the messages should be:

Client-ID space file-name (integer, a single space, 8 characters of file name; e.g., **103 cinput.1**)

Client-ID space sum (2 integers separated by a space; e.g., **103 515**)

When a Client prints out the summing result, it should be in the form of

Client-ID, file name, summing result

You can work on this phase in two sub-phases. In the first sub-phase, the Computer only works with one Client. Client can send a file name to the Computer and Computer can process it and send back the sum. Then Client can send another file name, and so on, till the "nullfile" terminates the Client and the connection with Computer.

In the second sub-phase, you can extend your program to multiple Clients. Since the Computer only has a single thread (no need to work on multiple threads here), you will not be able to communicate with multiple Clients at the same time. Thus, Computer needs to interact with the Clients in rotation. For example, if you want to establish connections with 3 Clients, then you can first establish 3 connections with these 3 Clients (wait on accept 3 times). Then, you can rotate over the Client-specific socket descriptors to communicate with the Clients, i.e., get one file name from the first Client, then get one file name from the second Client, then get one file name from the third Client, then get back to the first Client, and so on. Once a Client sends the "nullfile" string, Computer should close the connection to that client and continue the rotation on the remaining Clients. Note that Computer should maintain the active Clients in an ActiveClient list in order to properly poll the Client inputs.

3 Phase 3: Process Communication via Pipe and Signal

In this phase, we need to change the Computer process to perform actions under the control of the administrator (Admin process).

The Admin process, before forking the Computer process, should create pipes for inter-process communication. After forking, the Admin and the Computer processes should close the irrelevant pipe descriptors and use the correct ones for communication. After creating the pipes and forking the Computer process, the printout required in Phase 1 is required in your final project.

In Phase 2, the Computer process performs computation right after receiving a file name from a client. Now, the Computer process, after receiving a file name, should only place it into a ReadyQueue. In the ready queue, we need to maintain the Client-ID, file name, the client socket information (so the response can be sent back to the correct client), etc.

The Admin process should request the administrator to input the control commands (in a loop). After receiving a command, it sends the control command to the Computer (a single character) through the pipe or signal. The control commands include

Action	Parameters	System actions

T (terminate)	-	Terminate the system
x (execute)	-	Execute requests from the ready queue
Q (queue)	-	Dump ready queue content, issued to Computer by a signal

If the Computer process reads the x command from the pipe, it will remove the Client requests (one at a time) from the ReadyQueue and process them. It will not stop till all the requests in the ReadyQueue finish execution.

If the Computer process reads the T command from the pipe, it will terminate. It should stop execution even if there are new requests in the ReadyQueue to be processed. We will not complicate the task by asking the Clients to terminate before terminating the Computer process (though this is not elegant). You can try to terminate the Client processes by issuing "nullfile" before terminating the Computer process.

Once Computer starts executing requests from the ready queue, it won't stop till all the requests in the queue finish execution. If we want to dump the ReadyQueue during Computer execution, then we need to use a signal to force the Computer process to do it. Admin, upon getting the Q command from the administrator, sends a signal to the Computer and Computer should have the corresponding signal handler to handle it. Let's use the lowest user signal (SIGRTMIN) for the purpose.

To allow easy control, the Computer process should sum the numbers in each file slowly. After performing each addition, the Computer process should sleep for **S** milliseconds and S should be given in the command line when executing the Admin program and passed to the Computer process. Together with the port number, your Admin process should be started by

Admin.exe server-port-number S

For dumping the ReadyQueue (command Q), the dump output should be sent to the Admin process and printed by the Admin process. ReadyQueue dump should be done immediately or at the end of the execution of the current request and it should not wait till all requests are finished. The output of the dump for each Client request in the ReadyQueue should be of the form:

Client ID, file name, client socket descriptor, client port number

In the dump output, you will have as many lines as the number of requests in the ReadyQueue. The Computer should send the dumped information to Admin line by line in the format given above. The Admin should directly print the received strings line by line to stdout without parsing each received string.

Note that the Computer process needs to read Admin commands from the pipe and read Client requests from the sockets, but it is not easy to achieve waiting on multiple ports at the same time. For this phase, you can do alternate waits, i.e., Computer waits for Clients' requests, then waits for Admin's commands, then goes back to Clients, then proceeds to Admin, … iteratively. For easy testing, you can try to wait for 3 Client requests and then switch to wait for 2 Admin's commands.

4 Phase 4: Threads

In Phase 3, your Computer process needs to interact with the Admin and the Clients. Specifically, it needs to wait on the pipe for Admin, wait on the server socket to accept Client connections, and poll the Client sockets for inputs. So, Computer process should create **two** new threads. Together with the original parent threads, Computer will have three threads. Among these threads, one should interact with the Admin (A-thread), one should accept Client connections (CC-thread), and one should interact with the Clients (C-thread). CC-thread, upon accepting a connection, needs to add the Client and its corresponding socket to an ActiveClient list. Since there may be multiple clients, C-thread needs to read the Client sockets in rotation (same as Phase 2). Each Client request, once received, should be added to the ReadyQueue. Once a Client sends the "nullfile" string, C-thread should remove the Client from the ActiveClient list, close the connection to that client, and continue the rotation on the remaining Clients. A-thread should work on the pipe establishment with the Admin and wait to receive the Admin requests. It may receive x, T, Q

commands from Admin. Upon receiving command “x”, the thread should remove the Client requests from the ReadyQueue (one request at a time), execute them, and sends the outputs to the Clients. Since both A-thread and C-thread need to access the ReadyQueue, they may run into a race condition. Similarly, both C-thread and CC-thread needs to access the ActiveClients, they also may run into a race condition. For this project, we will ignore this potential problem.

Another way for a process to wait on multiple input sources (the pipe and the sockets) is to use the “select” or “poll” system call or to set the input sources to nonblocking. But we will not explore these options for this project.

5 Project Submission

You need to submit your program **before** midnight (11:59pm) of the due date (check the web page for the due date). Use UTD elearning system for submission: "elearning.utdallas.edu".

Your submission should include the following:

- ✓ All source code files making up your solutions to this assignment.
- ✓ The *Makefile* that generates the executables from your source code files. Your Makefile should generate 2 executables. The Admin executable should be "**Admin.exe**" and the Client executable should be “**Client.exe**”.
- ✓ If you did not finish the project, you need to specify which phase you have completed in your *README* file.
- ✓ The *DesignDoc* file that contains the description of the major features of your program that is not specified in the project specification. This document is not needed if there are no additional designs but those specified.