Simulated Operating Systems
Phase 4

Multiple Clients Submitting Programs via Sockets

Another change to simOS is to make it support multiple clients. Each client can submit his/her program to simOS via his/her own window. Even though simOS is now executing processes in a large number of iterations,the submit request from clients should be taken care of without much delay. The needed changes to simOS are as follows.

(A) Now you need to remove the "s" command from process_admin_command in admin.c and let client submissions be handled by submit.c.

Modify submit.c to run it as a submit thread in simOS (similar to what you do for swap.c and term.c). The submit thread should be able to receive requests from multiple clients. We use socket to achieve this task. In the submit thread, a server socket is created. It listens on a port. When a submission request is sent from any client to the port, the server socket receives the submission requests (input message) and process it. For each submission request, the submit thread raises the **submitInterrupt** (or any name you want to use). The request isthen processed by the interrupt handler, which calls "**submit_process**" (provided in process.c) to create a PCB for the process and to load the program into swap space and memory.

Note that you need to keep the submission information somewhere so that upon interrupt handling, you can access the corresponding submission information. Since there may be multiple clients, multiple submission requests may be issued at the same time. But submitInterrupt is only one bit and multiple submissions will only raise it once. This is different from admin's case since there will only be one admin. Thus, when handling submitInterrupt, you need to consider processing multiple submissions. Specifically, you need to maintain all the awaiting submissions in some data structure. You can implement the data structure by yourself, or copy andmodify one of the queues implemented in simOS, or use a library for the purpose.

For the program submission requests, you are not supposed to use signal to handle the request in a signal handler. You need to use socket and a **single** submit thread to achieve the task. To make your code flexible, andto make TA testing easier, you need to give the port number for your server socket as a command line input. After Project 2, your main program becomes admin.c and we run admin.exe for the simOS system. Rememberthat admin.exe had a numR command line input. Now, you add the port number as the second command line input to admin.exe. E.g.,

./admin.exe 1000000 21234

(B) similar to the change for admin, you need to add the definition for submitInterrupt in simos.h and addthe call to the submission interrupt handler in handle_interrupt().

(C) You also need to implement a separate submission client program, **client.c** (compiled to **client.exe**), to interface with the clients for program submissions. Each client runs the client program on its own window. The submission client program reads in the client submission information and wraps it as a request and sends the submission request to the server socket established by the submit thread (in A). Note that since the submission client can run on any computer, the program a client submits should be transferred through the communication channel to the submit thread. You cannot just send a file name and assume that the file is accessible by simOS.

The submission client program can be implemented in any language, as long as you can communicate withthe server socket properly. In Java and Python, some types of output streams may be formatted differently fromthose in c or c++ and you need to use the proper data stream and you need to make sure that the messages communicated between the client and simOS are compatible.

(D) With the submission client program, each client can have its own window for submission. At the same time, any printout from the program should be sent back to the client and get printed in the client window. Thus, you need to change the simulated terminal in term.c also. First, the "**terminal_output**" function, which originally outputs the output strings from programs to file "terminal.out", now should send the output string to each specific client via a specific socket. Upon accepting a client connection, the submit thread should retain thesocket information that is dedicated to the client. The client and socket information can be maintained in PCB (remember

PCB has a pointer pointing to files and devices in use). This way, terminal manager can easily accessthe socket information and use it for the corresponding client.

(E) Your client program needs to know the IP address and port number of the server socket in simOS (or admin.exe). Thus, the IP address and port number should be provided as command line input to client.exe. The sample client execution and program submission for client.exe is as follows:

```
> ./client.exe 129.110.242.180 21234          (IP and port are samples)
> s prog1                                     (s: submit, prog1 is the program file name)
  pid=2, M[10]=10.00pid=2, M[11]=60.00
  Process 2 had completed successfully: Time=15, PF=5
```

The blue text shows the sample outputs from simOS, sent from term.c. Only outputs for the program shouldbe returned to the client via the established socket. The client program should wait on its socket for outputs andprint the outputs to monitor. But the client program cannot have an infinite loop to wait on the socket for the outputs. You need to find out all the termination outputs from process.c and detect them to terminate the outputwaiting loop. After received all the outputs, the client can close its connection and terminate. The server, knowing that the client program has terminated, will close the specific client socket and clean up for process exiting.

Continuous Client Submission and Select System Call

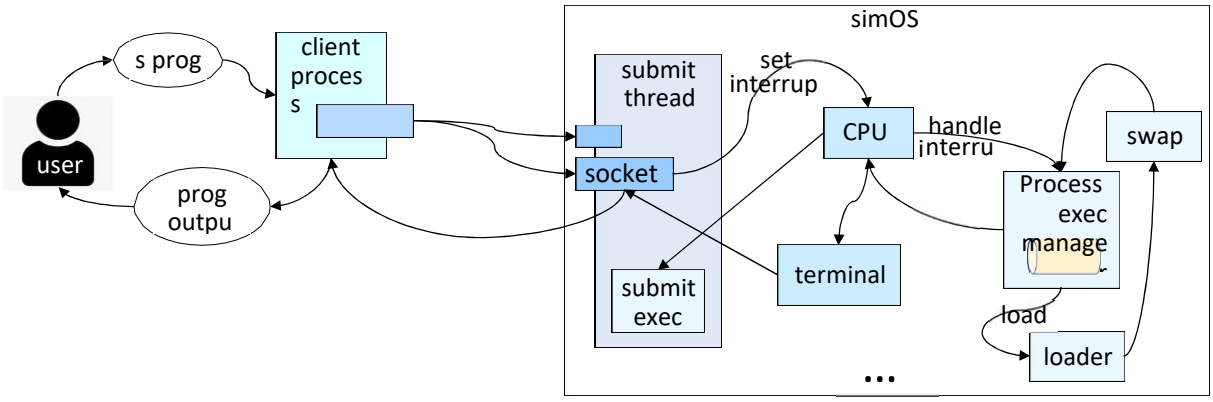
For convenience, we will allow each client connection to act like a session and allow the user to submitmultiple programs. The connection should be closed after the user decides not to submit any new programs.Thus, your client program should accept commands as follows:

```
> ./client.exe 129.110.242.180 21234          (IP and port are samples)
> s prog1                                     (s: submit, prog1 is the program file name)
  pid=2, M[10]=10.00pid=2, M[11]=60.00
  Process 2 had completed successfully: Time=15, PF=5
> s prog1.err
  Program prog.err has loading problem!!!
> s prog2.err
  pid=3, M[10]=10.00
  Process 4 had encountered error in execution!!!
> T       (terminate, the client will no longer submit any more programs)
```

After a user submits a program, the client program will not prompt to the user till the received output indicates the termination of the program. The client uses command “s” for program submission, and “T” forterminating the submission activities.

In Phase 3, we only consider one submission from each client. Thus, after accepting a client connection, the submit thread can go ahead and read the client submission message. Since it only needs to read once from the client, we let the thread be blocked by the receive function. In other words, the submit thread can only receive one client submission at a time, even though the main thread may be executing other programs and/or responding to other programs in parallel.

We do not want a multi-thread solution for this case. The required solution is to use the **select()** system call, which allows you to wait on multiple sockets (or any other types of file descriptors) at the same time and receives some message(s) when one or more are available. Now you have a complete simOS that can handle multiple clients.


simOS
s prog
client
proces
s
user
prog
outpu
submit
thread
set
interrup
socket
CPU
handle
interru
swap
Process
exec
manage
terminal
submit
exec
load
loader
...