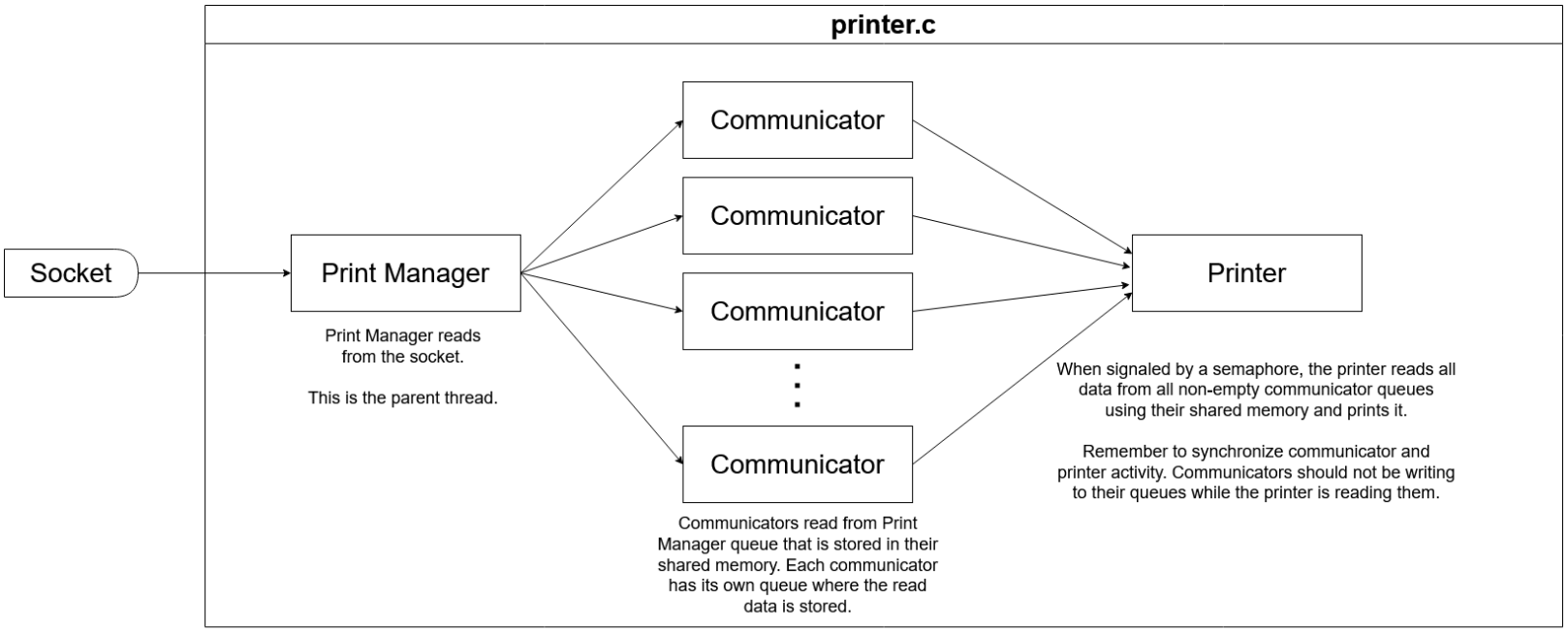
CS 4348/5348 Operating Systems -- Project 3

You will extend your existing C files and create new ones to simulate multiple instances of the operating system that you have developed. These multiple OS instances will communicate with a print manager using sockets. The print manager uses a shared memory space to pass data to a series of communicators, and the printer reads this data from the communicators. The C files of interest for this project are as follows:

printer.c

You will update your *printer.c* file from Project 2. This file will instantiate 2 child threads to handle printing and message communication, while the parent thread reads external print request coming in from a socket. This relationship is visualized in the following image:



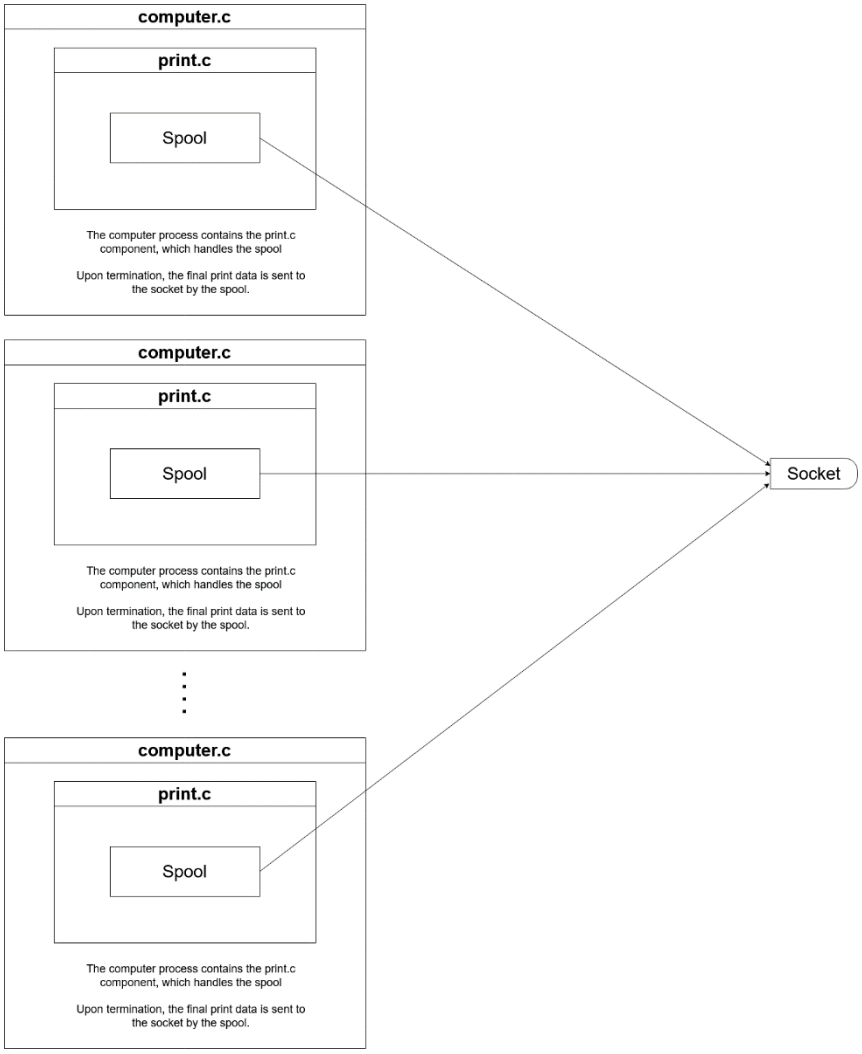
Note: Multiple semaphores are required to ensure correctness of operations on the shared memory. They are not pictured here, because you are required to implement them yourself. Review the textbook for more information on semaphore usage.

All the functions for these threads will be defined in the printer.c file. Some of these functions are:

printer_manager_init(…)	Initializes the Printer Manager and its components such as the child threads, its shared memory, relevant semaphores, and its listening socket.
printer_manager(…)	Continuously listens on the listening socket and places data into the Print Manager queue.
communicator(…)	Thread function that acts as a single communicator. Initializes a queue and relevant semaphores, then continuously reads data from the Print Manager's queue in shared memory, and writes this data to its internal queue.
printer_init(…)	You will update the old printer_init(…) function such that it has necessary semaphores and so that it prints a message indicating that initialization is complete.
printer_main(…)	You will update the old printer_main(…) function such that it reads data from the communicator queues in shared memory, such that the read operations are synchronized with sempahores.

computer.c

Modify computer.c so that it takes a new command line argument that holds a numerical identification number for the computer, called a CID. When you want to test multiple computer instances running concurrently, you can do so using multiple console windows, and each computer instance should have a unique CID. This is visualized in the following graph:



Whenever you call the print component from a computer instance, you must pass the PID+CID in lieu of the PID in the printer request messages. Only start a computer instance after the printer instance has started.

print .c

You will update your *print.c* file from Project 2. The functions in this file will need to be updated to account for new printer setup. Some of these functions are:

print_init(…)	Set up a socket connection to the printer instance.
print_init_spool(…)	Initialize the spooling for a process when it is created. The process should pass the PID+CID to this function.
print_end_spool(…)	When a process terminates this function sends the print data and the PID+CID through the socket to the Print Manager.
print_print(…)	This function takes a buffer to be printed and the PID+CID, before sending them through the socket to the Print Manager. This is called by the CPU.

Requirements

Your C files will need to work together to simulate many concurrent operating systems that run an instruction set program with print instructions. When code is compiled and run, the following sequence of tasks should occur:

1. The Print Manager should initialize itself and its child threads.
2. Each computer instance will run an instruction set program that has both mathematical and print commands. These programs will be run inside the computer instances, such that the instances will all be sending print data through a socket concurrently.
3. The Print Manager will read incoming data from the socket, and place the data in its queue for consumption by the communicators.
4. The communicators will read the print data and place it within their own queues.
5. The printer will read the print data from the communicators and "print" it like in Project 2.

Compiling and Executing Programs in Unix

You need to prepare a **makefile** for automatic compiling of your source code. You can read about makefiles here: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Note that you should not use #include to include any .c code files (.h header files are ok). You should compile each C code file into an object code file (".o") separately and then link the ".o" files together into the executable. You can read about using makefiles in this way here: https://cs.gmu.edu/~huangyih/310/makefile.pdf

Project Submission

You need to submit your program **before** midnight (11:59pm) of the due date listed on eLearning. Your submission should be a .ZIP file which contains the following:

- All source code files and header files making up your solutions to this assignment.
- makefile
- A "readme" file that describes what each .c file does, and how to run the program.
  - If you did not finish the project, you need to specify which part(s) you have completed and which part(s) are missing.
  - If you submit an unfinished project without this specification, you will not receive partial credit.

Failure to follow these submission requirements will result in an automatic -10 point penalty.