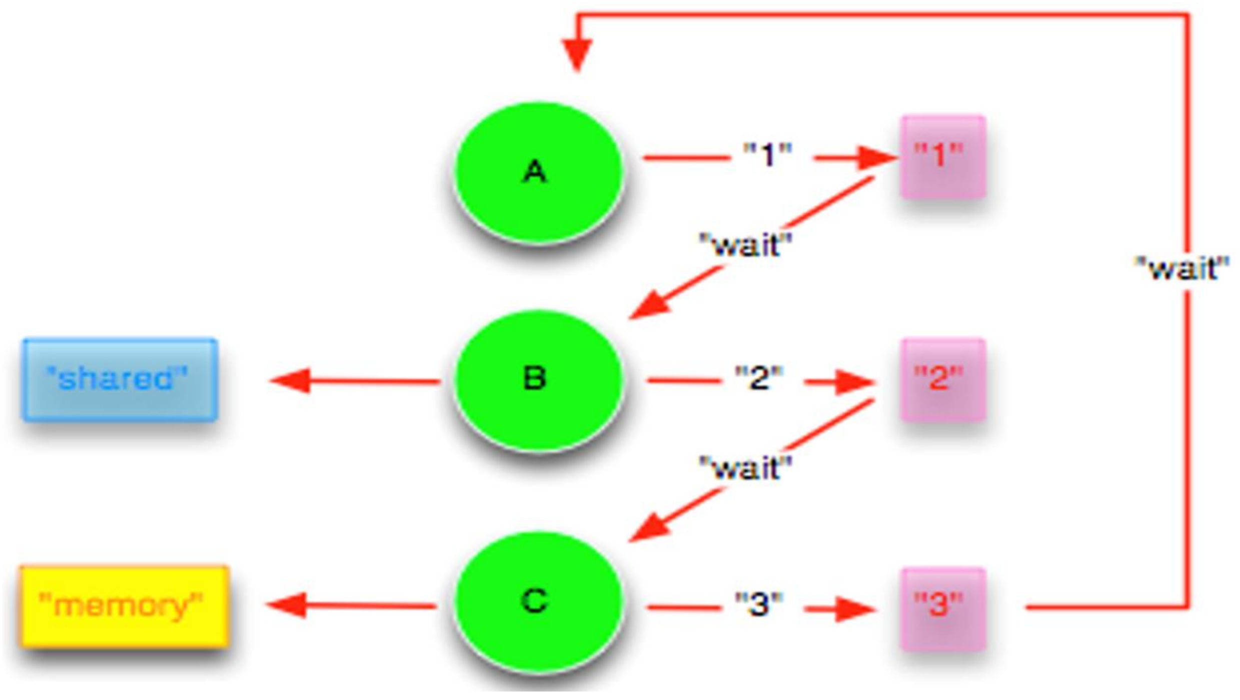
CMSC 312: Assignment 1: IPC Mechanisms

Question 1: Shared Memory (15 pts):

Write a simple program (example code available on BlackBoard; can be used as starting point) where *processes synchronize via polling* such that a process A prints out the strings of two other separate writing processes (B first and then C second) from *shared memory*. Process A needs to 'wait' by polling until B and C finish writing their strings to memory. Each of processes A, B and C should be in different code files. Create TWO different shared memory locations: (i) one for storing the integer identifier and (ii) for storing the string.

Here is the sequence of events that needs to be implemented:

1. Process A writes “1” to the integer shared memory location and then waits until B and C completes.
2. Process B writes the string "shared" into the string shared memory location and then signals A & C that it is complete by writing “2” into the integer shared memory location (note B should wait to write into position the integer shared memory location until after process A writes “1” there). Then process B quits.
3. Process C writes the string "memory" into the string shared memory and then signals A that it is complete by writing “3” into the integer shared memory location (note C should wait until process B writes “2” into the integer shared memory). Then process C quits.
4. Process A needs to keep polling the integer shared memory location continuously. After process B writes “shared”, process A should be able to print it out. Similarly, after process C writes “memory”, process A needs to be able to print it out. Here some ad-hoc synchronization is needed. Specifically, consider using usleep() function to delay process B (and process C) between writing the string and the integer to give process A a chance to poll and print. Adjust the delay properly such that prints happen properly.
5. Process A is the last one to quit and prints out a “GoodBye” message at the very end before quitting.



Question 2: Shared Memory (4 pts)

Create processes A, B and C from the same file using fork.

Question 3: Message Queues (6 pts)

Review the programs (spock.c and kirk.c).

Answer (or discuss) questions listed below:

a) Discuss and evaluate what happens when you're running both in separate windows and you kill one or the other.

b) Discuss what happens (and why) when you run two copies of kirk.

c) Discuss what happens (and why) when your run two copies of spock.

Remote Procedure Calls

Note that you will need a .rhost file under your home directory, it includes authority to the machine that the 'server' accepts clients <hostname> and from what login id <loginid>.

The format for a .rhost file is:

```
<hostname1> <loginid1>
<hostname2> <loginid2>
.
.
.

<hostnameN> <loginidN>
```

The instructor's .rhost file may look like the below (login id is 'pghosh'):

```
172.18.233.71 pghosh
```

Great RPC tutorial - Prof David Marshall @ Cardiff University (optional reading):
http://www.cs.cf.ac.uk/Dave/C/node33.html

On rpcgen: http://www.cs.cf.ac.uk/Dave/C/node34.html#ch:rpcgen (note: you may not have permission to print on another server's Console for the printmessage tutorial).

Wikipedia: http://en.wikipedia.org/wiki/Remote_procedure_call (also check the link listed under 'external link" to the SGI Tutorial "Introduction to RPC Programming").

Question 4: Compute the median (10 pts)

Modify the average RPC programs (avg.x, avg_proc.c and ravg.c) so that it computes the **median** of a maximum of 100 numbers instead of the average of a maximum of 200 numbers. Test your program on the server.

The average RPC program is available on BlackBoard - a Makefile is included). Just type **make** (make sure you have an .rhost file under your home directory, and that 'white' space works correctly in the makefile (i.e., cut and paste the makefile will not work).

For the avg program: Run the RPC server program by starting the server: ./avg_svc

For the avg program: Run the RPC client program by: ./ravg 127.0.0.1 1 2 3 4

Try to use a different 'port number' than other classmates by changing the 22855 number given in the avg.x file. Suggestion, use the last 4 digits of your student ID number or some other unique number.

Question 5: Remote Procedure Calls (RPC): Echo RPC server (5 pts)

Write a program using remote procedure calls where the server simply echoes the word back to the client.

Question 6: Remote Procedure Calls (RPC): Sort numbers (10 pts)

Modify the average RPC program so that the server sends back a list of 'sorted' numbers to the client either in descending or ascending order given by an input parameter when invoking the client, e.g., the word "ascending:, "descending" (or just -a, -d).

In this case, the entire array needs to be returned instead of a single variable. Consider changing the return type of the average_1 function on the server from double * to input_data *. Sort the numbers in the input_data structure itself. Then the client can access the sorted numbers through the input_data structure.

Deliverables:

Upload your code and write-up (answers to the questions) on BlackBoard.

Late Policy:

Late assignments will incur a penalty of 5 points per day for a maximum of 2 days.