Network cat (netcat)

For this project you will write a netcat-like application. At its most basic, netcat establishes a TCP connection to a remote computer, then reads from stdin and writes to the network socket while simultaneously reading from the network socket and writing to stdout.

Write a 64-bit Linux assembly language program that behaves as described below. You may not use any external libraries.

Parent process behavior:

1. Your program will receive two command line arguments. argv[1] will contain a host name (not an IP address). argv[2] will specify a port number.
2. If your program does not receive exactly 3 command line arguments, you should exit with status code 1.
3. Resolve the hostname to an IP address (see below), then create a TCP socket and connect it to the indicated host on the indicated port. Read below for more information on connecting your socket. If the host name cannot be resolved, print exactly "unable to resolve host\n" and exit with status code 2.
4. If the connection attempt fails, print exactly "connection attempt failed\n", close the socket, and exit with status code 3.
5. If you successfully connect, create two child processes referred to here as child A and child B.
    - Fork the first child (child A).
    - In child A, duplicate the connected socket to stdin (fd 0) of child A, then close the socket and *execve* assign3 (your assignment 3 binary).
    - In the parent, fork a second time to create child B.
    - In child B, duplicate the socket onto stdout (fd 1), then close the socket and execve assign3 from within child B.
6. At this point the parent process should close the socket descriptor.
7. In the parent process, wait for child A to complete.
8. In the parent, once child A completes, send a SIGTERM signal to child B.
9. In the parent, wait for child B to complete.

Child process behavior

Because of the way the socket descriptor was duplicated for each child process in step 5 above, each of the child processes is really just cat (read from 0, write to 1) at this point, so you should be able to reuse your code from assignment 3 to implement the child behavior. When a child sees EOF or any error condition while reading from 0, or any error condition while writing to file descriptor 1, the child should terminate by using the exit syscall.

When you execve assign3, you can expect the assign3 binary to be in the current working directory of the parent process ("./assign3"). You should pass in an argv that contains only an argv[0] and a NULL pointer. The third argument to execve (i.e., envp) should provide the parent's environment to the child process. You need to understand how to derive envp in the parent assign5 process.

To wait for child processes, you should research the *wait4* system call. To send a signal to a child process, you should research the *kill* system call.

Parsing the port number argument:

You need to convert the ASCII port number to an internal two-byte representation (perhaps using l_atoi). You will need to convert the resulting port number to network byte order by swapping the high and low byte of the 16-bit result. For example, if the port number is in ax, you could swap the bytes using "xchg ah, al".

Host name resolution:

To assist with host name resolution, you may link to the provided dns64.o, which contains the following function:

```
unsigned int resolv(const char *hostName);
```

This function uses the C calling convention and returns a network-byte-ordered IP address of the named host or 0xffffffff if the host name can't be resolved. You should declare *resolv* as an extern in your asm file. Note that this function will not resolve dotted-quad-style IP addresses.

Armed with a network-byte-ordered port number and host IP address, you will need to fill in the fields of a sockaddr_in struct that you have allocated (this can be a global). This initialized sockaddr_in will be passed to *connect* to create a connected socket. Two syscalls are required to create a connected socket (*socket* and *connect*).

Requirements:

1. Your assembly language source file implementing the parent process behavior which must be named **exactly** assign5.asm.
2. Your assembly language source file implementing the cat behavior, which must be named **exactly** assign3.asm. You are welcome to make any changes to your original assign3.asm that you deem necessary.
3. Using comments, include your name, class number, project number, and document the commands required to create an executable file from assign5.asm and the provided dns64.o.
4. Submit your source code to the assignment submission page in Sakai.