Project 3 Discussion

Multi-Process and Multi-Threaded Web Servers

Setup

- Use the same virtual machine as in Project 2
  - Login as *root*

- If you want a new copy, grab the appropriate software and disk image from the assignment page

Setup

As the root user:

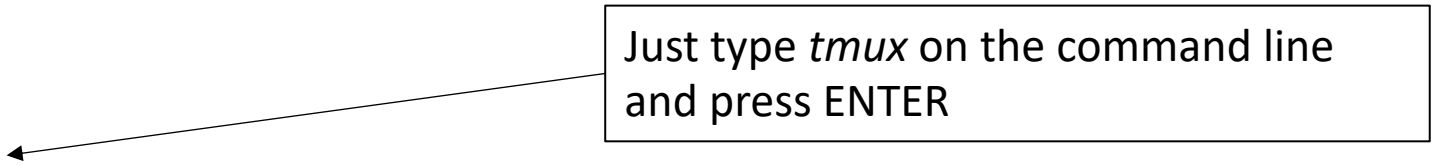
```
apt-get update

apt-get install tmux links
```

To get the single-threader server code from thoth:

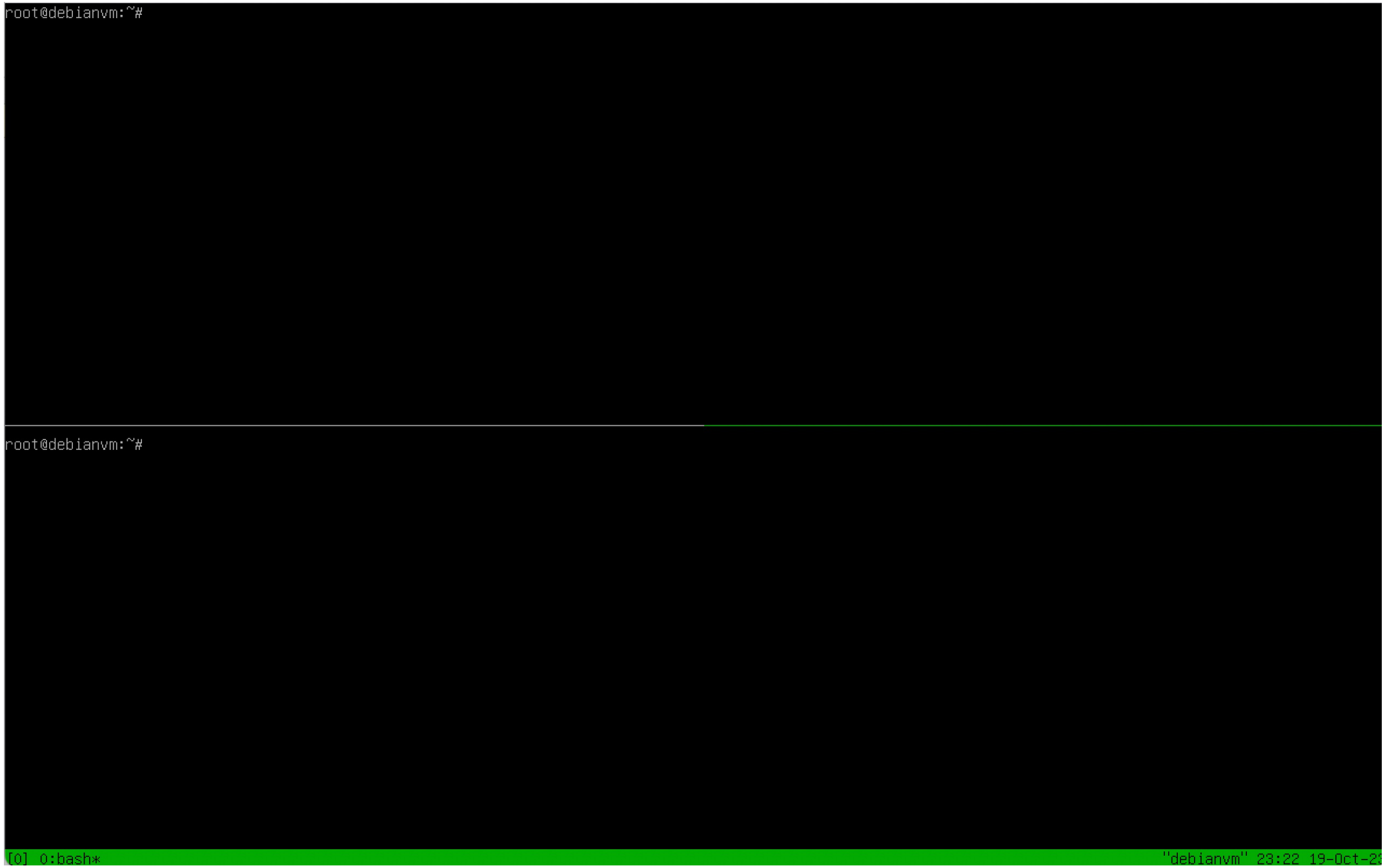
```
scp USERNAME@thoth.cs.pitt.edu:/u/OSLab/original/server.c .
```

Now, let’s try to send requests to the single-threaded server (server.c)

Before that, do this



1. Run `tmux`
2. Execute your server
3. Hit **CTRL+B**, release, and then type " (a quotation mark -- this requires holding shift down to type)
4. You should see a window with a shell appear at the bottom
5. At any point, you can switch which shell has focus by typing **CTRL+B** and then pressing an arrow key to move up or down amongst the windows.
6. When you're done testing, type `exit` to leave the shell and the bottom window will go away. You'll probably want to use **CTRL+C** to kill your server. Then you can type `exit` again to leave `tmux`.



terminal 1

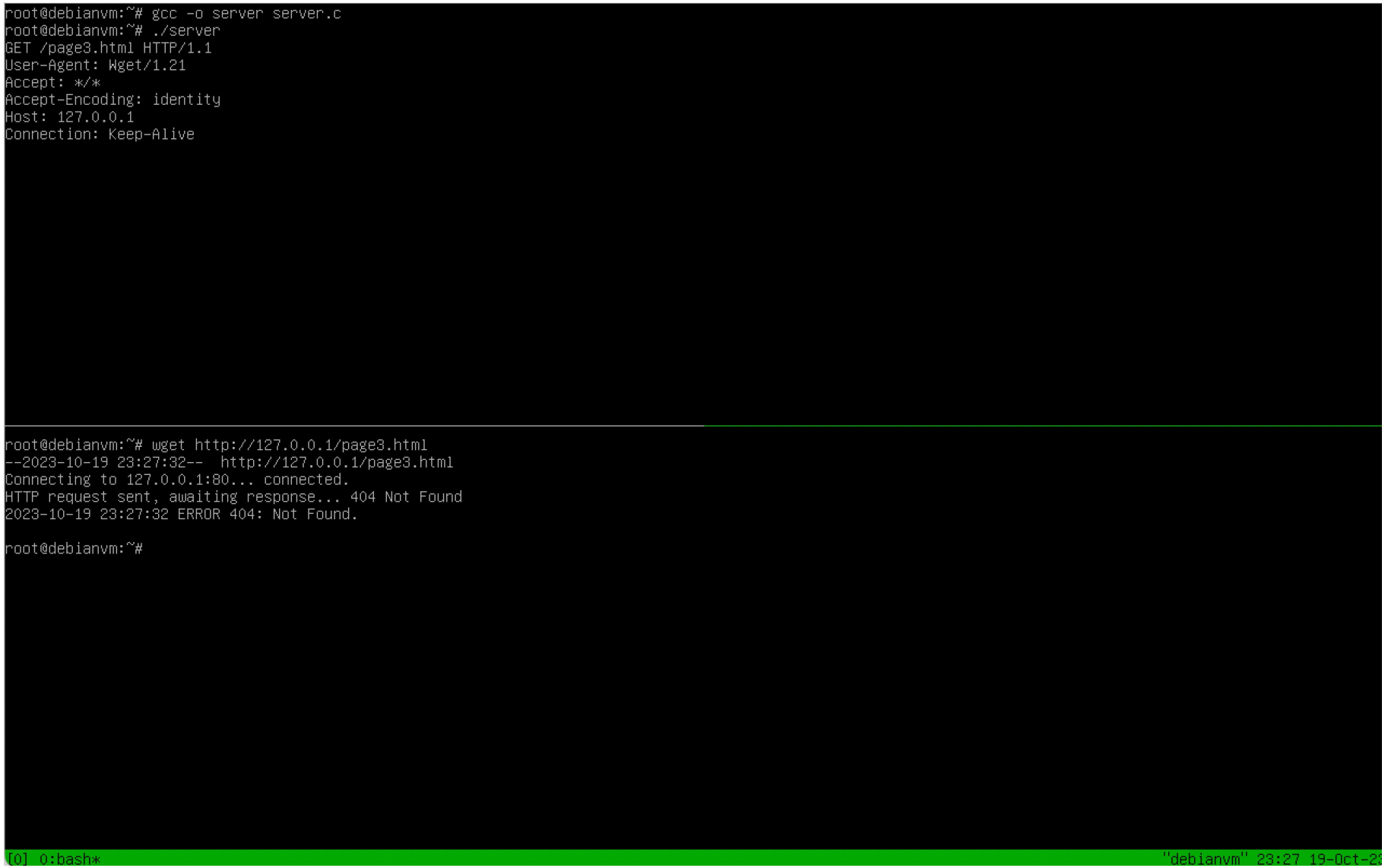
terminal 2

Running server and client

- Run the server on one terminal

- Run the client(s) on the other terminal

DEMO



- Server runs on this terminal.
- Server could not find the requested file, page3.html

- Client runs on this terminal.
- Client requested for page3.html
  - Server returned status 404(file not found).

What do you need to do?

- Write code for a multi-process version of the server in the file *server_proc.c*
- Write code for a multi-threaded version of the server in the file *server_thread.c*
- Write a 1-2 page paper
  - Answer this question
    - Which implementation of the server is faster?
  - Describe your experimental design, your results, and your conclusions in a 1-2 page paper.

Multi-process version of the server

server_proc.c

- Use *fork()* to create a worker process
- Worker Process
  - Handles sending the requested file over the network connection
  - When done sending the file, it should log the request for that particular webpage by appending it to a file named *stats_proc.txt*
    - Access to *stats_proc.txt* needs to be synchronized
      - Use *sem_wait()* and *sem_post()* from <semaphore.h>

exclusively accessed, so you'll need to do some sort of synchronization. Log the **name** of the file requested, its **size** in bytes, the **ending CPU time** of the request and the **elapsed CPU time** in seconds of the request, in the following tab-separated format:

```
file1.html    13452  0.5123 0.0123
```

server_proc.c

- If you need shared space to hold something between processes, use mmap() with the MAP_SHARED and MAP_ANONYMOUS flags.

- When you create a child process, you'll have two copies of the socket file descriptor and two of the connection file descriptors. The parent only needs one of those and the child only needs one. Make sure to close() the other.

Using fork() to create a child process

- DEMO

Using fork() to create a child process

```c
#include<stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{

    int x = 10;

    if(fork() == 0)
    {
            printf("[Child] pid %d from [parent] pid %d\n",getpid(),getppid());

            x = 20;
            printf("Child x : %d\n", x);

            exit(0);
    }

    wait(NULL);
    printf("Parent x: %d\n", x);


}
```

Multi-threaded version of the server

server_thread.c

- Use *pthread_create()* to create a worker thread
- Worker Thread
  - Handles sending the requested file over the network connection
  - When done sending the file, it should log the request for that particular webpage by appending it to a file named *stats_thread.txt*
    - Access to *stats_thread.txt* needs to be synchronized
      - Use *pthread_mutex_lock() and pthread_mutex_unlock() from <pthread.h>*

exclusively accessed, so you'll need to do some sort of synchronization. Log the **name** of the file requested, its **size** in bytes, the **ending CPU time** of the request and the **elapsed CPU time** in seconds of the request, in the following tab-separated format:

```
file1.html    13452  0.5123 0.0123
```

server_thread.c

- Sharing between threads is just allocating things in the heap or globals.

- Compile your server_thread.c file with the *–pthread* flag
  - DEMO

CLIENT

Test from the client side

- `nc localhost 80`
- `wget http://127.0.0.1/page.html`
- `links http://127.0.0.1/page.html`

Client

- You can run multiple programs from the client side to send requests to the server in parallel
  - To run two programs simultaneously on the command line, put an & between them
    - Here, a program can be wget or curl or any program that can download from a web server. Or you can even write you own test program as HTTP is not a complicated protocol.