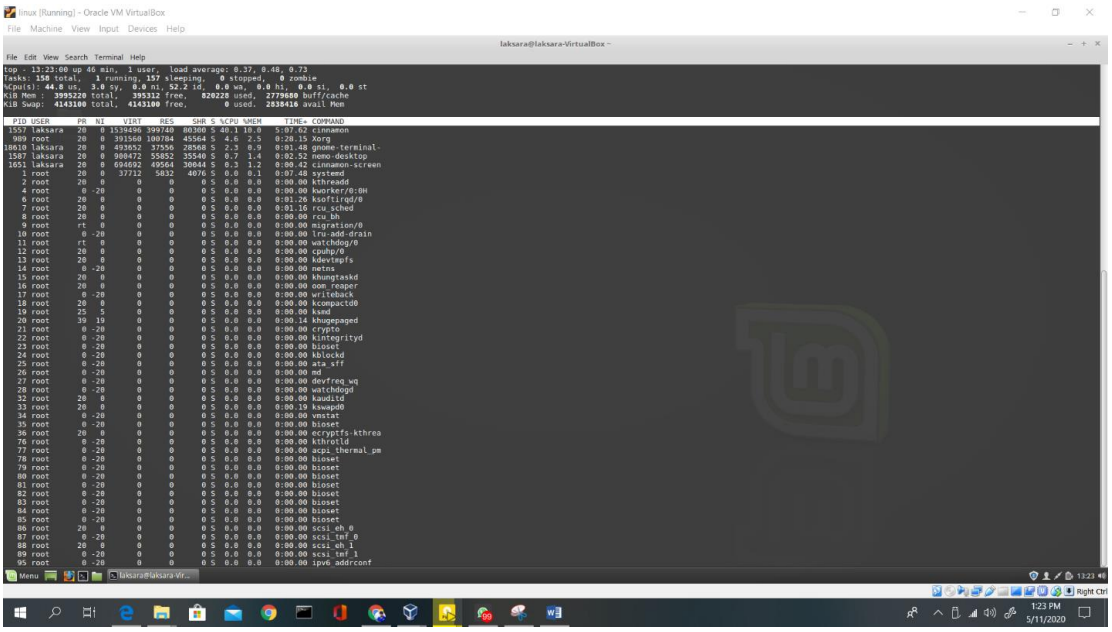
Multiprocessing

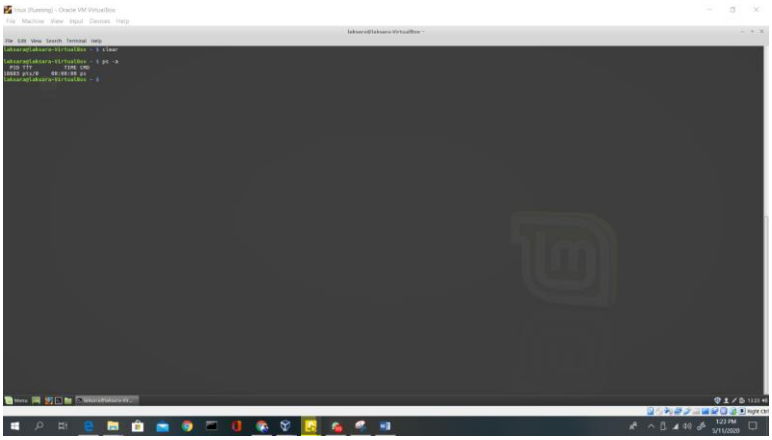
Exercise 1

1. Top shows you details of active processes. The processes are sorted by CPU usage by default. Sort them by memory usage

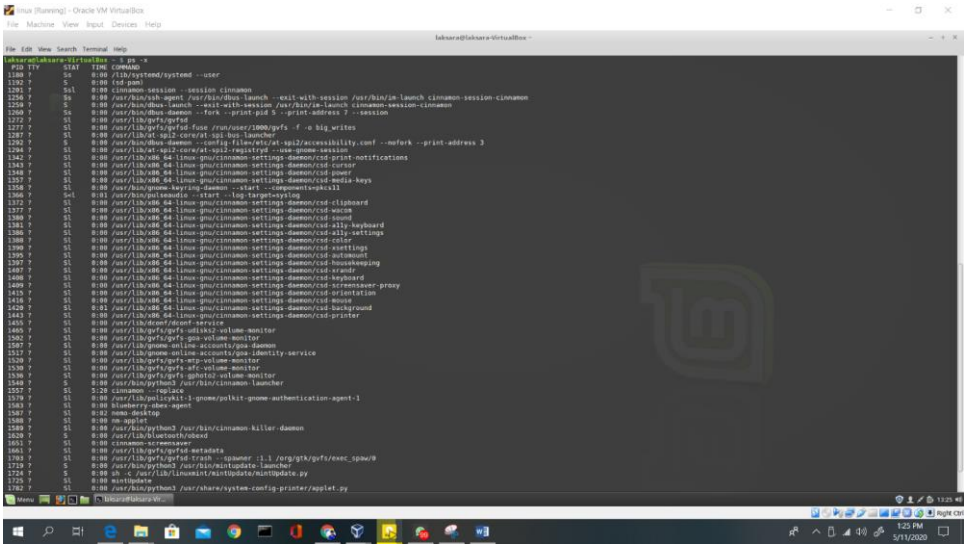


2. Run ps with the following options: -a, -x, -u, -w. What is the name of the process with PID 1?

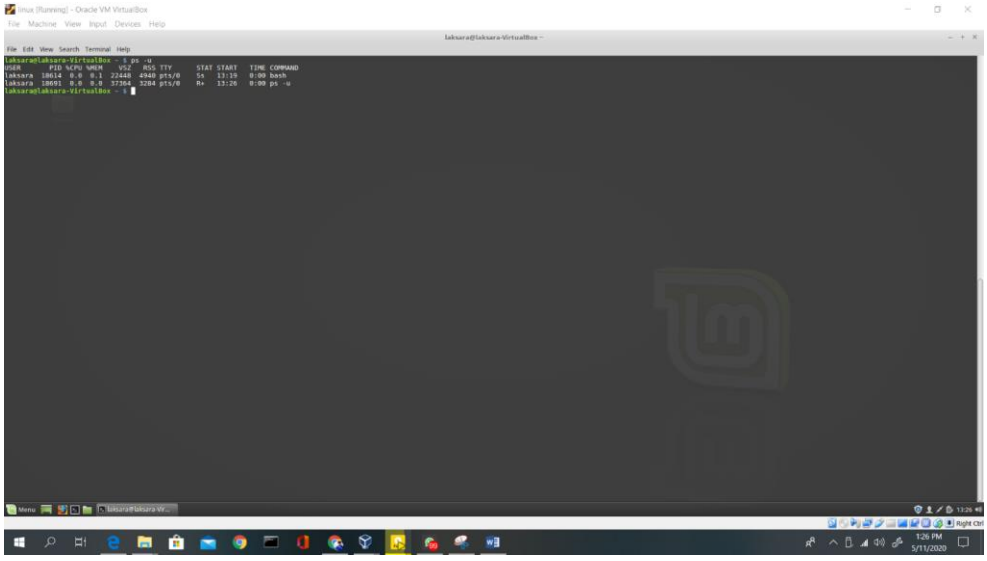
    - ps –a



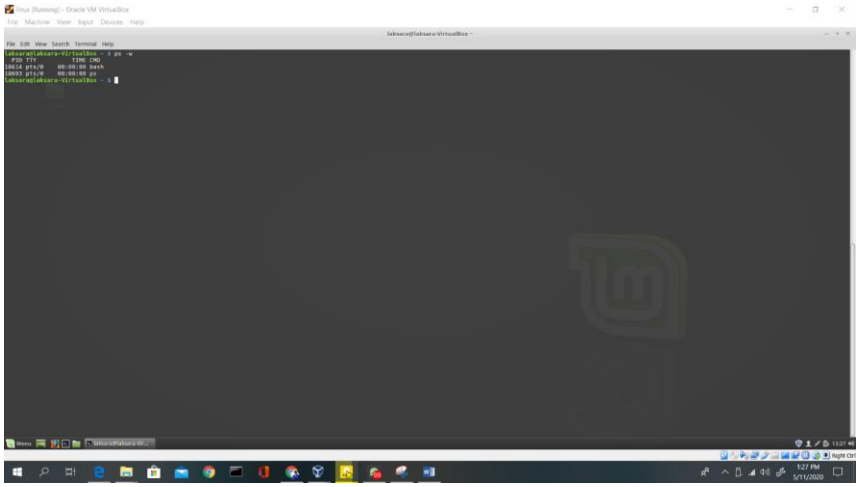
- ps –x


laksara@laksara-VirtualBox ~
laksara@laksara-VirtualBox:~$ ps -x
PID TTY STAT TIME COMMAND
1 ? Ss 0:00 /lib/systemd/systemd --user
2 ? S 0:00 (sd-pam)
1201 ? Ssl 0:00 cinnamon-session --session cinnamon
1256 ? Ss 0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --exit-with-session /usr/bin/im-launch cinnamon-session-cinnamon
1259 ? S 0:00 /usr/bin/dbus-launch --exit-with-session /usr/bin/im-launch cinnamon-session-cinnamon
1260 ? Ss 0:00 /usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7 --session
1272 ? Sl 0:00 /usr/lib/gvfs/gvfsd
1277 ? Sl 0:00 /usr/lib/gvfs/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
1287 ? Sl 0:00 /usr/lib/at-spi2-core/at-spi-bus-launcher
1292 ? S 0:00 /usr/bin/dbus-daemon --config-file=/etc/at-spi2/accessibility.conf --nofork --print-address 3
1294 ? Sl 0:00 /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-session
1342 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-print-notifications
1343 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-cursor
1348 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-power
1357 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-media-keys
1358 ? Sl 0:00 /usr/bin/gnome-keyring-daemon --start --components=pkcs11
1366 ? S<l 0:01 /usr/bin/pulseaudio --start --log-target=syslog
1372 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-clipboard
1377 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-wacom
1380 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-sound
1381 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-a11y-keyboard
1386 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-a11y-settings
1388 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-color
1390 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-xsettings
1395 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-automount
1397 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-housekeeping
1407 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-xrandr
1408 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-keyboard
1409 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-screensaver-proxy
1415 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-orientation
1416 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-mouse
1420 ? Sl 0:01 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-background
1443 ? Sl 0:00 /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-printer
1455 ? Sl 0:00 /usr/lib/dconf/dconf-service
1465 ? Sl 0:00 /usr/lib/gvfs/gvfs-udisks2-volume-monitor
1502 ? Sl 0:00 /usr/lib/gvfs/gvfs-goa-volume-monitor
1507 ? Sl 0:00 /usr/lib/gnome-online-accounts/goa-daemon
1517 ? Sl 0:00 /usr/lib/gnome-online-accounts/goa-identity-service
1520 ? Sl 0:00 /usr/lib/gvfs/gvfs-mtp-volume-monitor
1530 ? Sl 0:00 /usr/lib/gvfs/gvfs-afc-volume-monitor
1536 ? Sl 0:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
1548 ? S 0:00 /usr/bin/python3 /usr/bin/cinnamon-launcher
1557 ? Sl 5:28 cinnamon --replace
1579 ? Sl 0:00 /usr/lib/policykit-1-gnome/polkit-gnome-authentication-agent-1
1583 ? Sl 0:00 blueberry-obex-agent
1587 ? Sl 0:02 nemo-desktop
1588 ? Sl 0:00 nm-applet
1589 ? Sl 0:00 /usr/bin/python3 /usr/bin/cinnamon-killer-daemon
1620 ? S 0:00 /usr/lib/bluetooth/obexd
1651 ? Sl 0:00 cinnamon-screensaver
1661 ? Sl 0:00 /usr/lib/gvfs/gvfsd-metadata
1703 ? Sl 0:00 /usr/lib/gvfs/gvfsd-trash --spawner :1.1 /org/gtk/gvfs/exec_spaw/0
1719 ? S 0:00 /usr/bin/python3 /usr/bin/mintupdate-launcher
1724 ? S 0:00 sh -c /usr/lib/linuxmint/mintUpdate/mintUpdate.py
1725 ? Sl 0:00 mintUpdate
1782 ? Sl 0:00 /usr/bin/python3 /usr/share/system-config-printer/applet.py
laksara@laksara-Vir...
1:25 PM
5/11/2020

- ps –u


laksara@laksara-VirtualBox ~
laksara@laksara-VirtualBox:~$ ps -u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
laksara 18614 0.0 0.1 22448 4940 pts/0 Ss 13:19 0:00 bash
laksara 18691 0.0 0.0 37364 3284 pts/0 R+ 13:26 0:00 ps -u
laksara@laksara-VirtualBox:~$
laksara@laksara-Vir...
1:26 PM
5/11/2020

- ps –w



Exercise 2

1. In what order are the messages from parent and child printed? Is the order always the same?

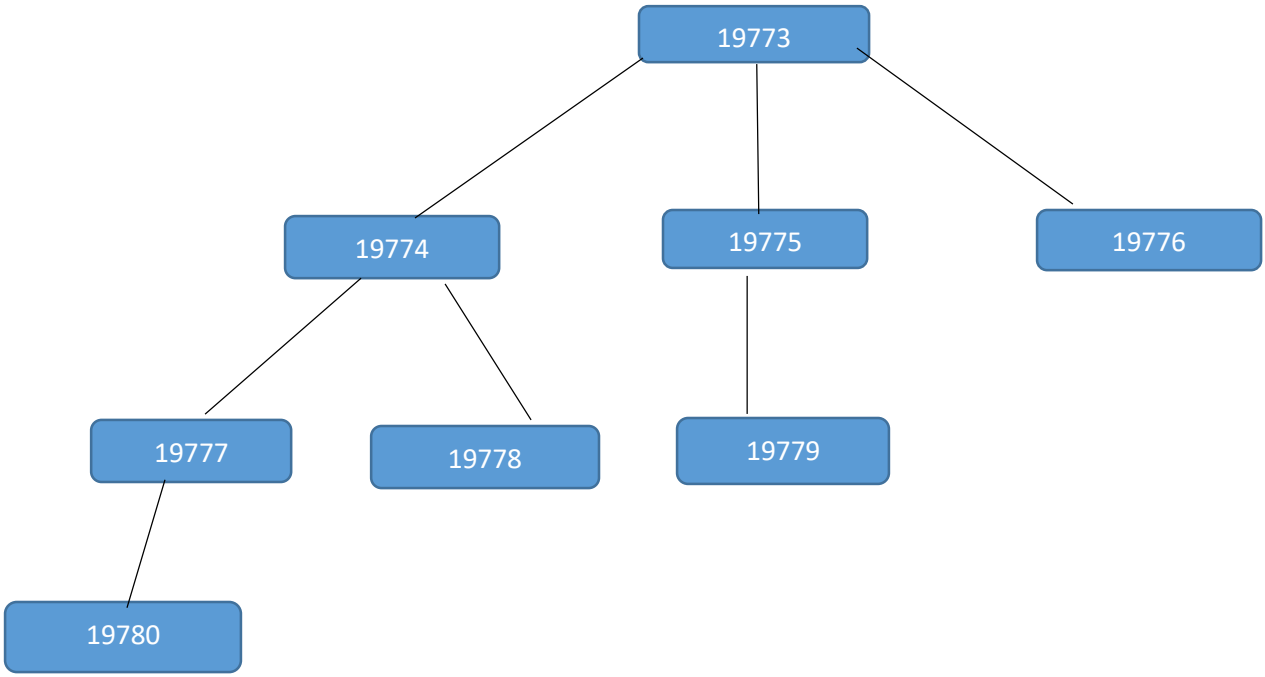
   This is parent process
   This is child process

   Order always the same

2. How many children will the following program spawn? Draw a diagram illustrating the parent-child relationships between processes.

```
int main(void)
 {
     for (int i=0; i<3; i++)
              fork ();
 }
```

   7 children



Exercise 3

```
int main(void){
   int  pid;
   pid  =  fork();
   if (pid  <  0){
     perror("fork");
     exit(1);
   }
if (pid  ==  0){
      puts("This  is the  child  process");
}else{
       wait(0);
       puts("This  is the  parent  process");
}
```

```
return  0;
}
```

Exercise 4

1. Compile and run the above code giving it a path as an argument. How many times is the message “Program ls has terminated” printed?

   The printed message is never printed because after a execl() is called current one is replaces with the new process context.

2. Write a very simple shell that repeatedly prompts the user for a command and runs it with any arguments given. Make sure your shell waits until the command has completed before prompting the user for the next command.

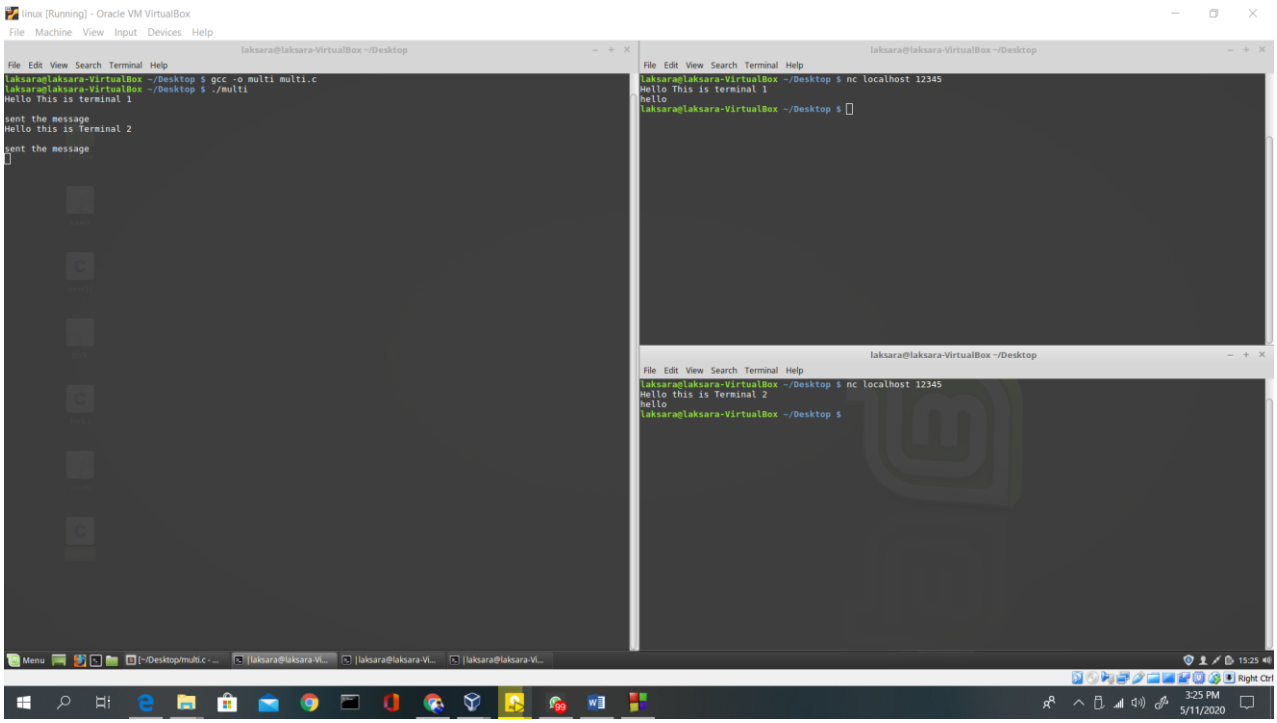
```
#include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>
 #include<sys/wait.h>
int  main(char  argc,  char **argv){
            char path [20];
             char arg[20];
             while(1){
                        scanf("%s %s", path,arg);
                        int pid= fork();
                        if(pid == 0){
                           execl(path, arg, argv[1],  NULL);
                        }else{
                           wait(0);
                        }
            }
            return 0; }
```

Exercise 5

1. Open three terminals and run the server in one. Use nc() to connect as two clients concurrently on port 12345. Type some text in both clients and examine the client and server outputs.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<string.h>

#define PORT 12345

void handle_client(int new_socket){


    char buffer[1024];

    int valread = read(new_socket, buffer, 1024);
    printf("%s\n",buffer );

    send(new_socket, "hello", strlen("hello"), 0);
    printf("sent");
}

int main(char argc, char **argv){


    struct sockaddr_in cli_addr;

    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = INADDR_ANY;
    cli_addr.sin_port = htons(PORT);

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);

    bind(sockfd, (struct sockaddr*)&cli_addr, sizeof(cli_addr));
```

```
    listen(sockfd, 5);

    int clilen = sizeof(cli_addr);


    while(1){
        int newsockfd = accept(sockfd, (struct sockaddr*)&cli_addr, &clilen);

        if(newsockfd < 0){
            perror("ERROR on accept");
            exit(1);
        }
        int pid = fork();

        if(pid<0){
            perror("ERROR on fork");
            exit(1);
        }

        if(pid == 0){
            close(sockfd);
            handle_client(newsockfd);
            exit(0);
        }else{
            close(newsockfd);
        }
    }

    return 0;
}
```



2. Suppose we modify the server parent process to call wait() on the last line above (highlited) to wait until the child serving a client terminates. What would happen?

   Then the main while loop will be blocked by wait (). So there wont be any concurrent clients.

3. What happens if you terminate the server while a client is connected, and then try to restart it? (Resolving this issue requires a signal handler.)

   All the connected clients will be terminated. And even the server restarted again connection will be refused

4. Modify this server to do the following: The client sends the path to a file whose contents the server will send back to the client (if the file exists.) Verify that your new server can handle multiple concurrent connections by using nc(). Can two concurrent clients request the same file?

   ```
   #include<stdio.h>
   #include<stdlib.h>
   #include<sys/wait.h>
   #include<sys/socket.h>
   #include<netinet/in.h>
   ```

```
#include<unistd.h>
#include<string.h>

#define PORT 12345

void handle_client(int new_socket){

	while(1){

		char buffer[1024];
		char *sendBuffer;
		size_t size = 0;

		int valread = read(new_socket, buffer, 1024);
		buffer[valread - 1] = '\0';
		printf("%s\n",buffer);

		FILE *fp = fopen(buffer, "r");

		fseek(fp, 0, SEEK_END);
		size = ftell(fp);
		rewind(fp);

		sendBuffer = malloc((size + 1) * sizeof(*buffer));

		fread(sendBuffer, size, 1, fp);

		sendBuffer[size] = '\0';

		printf("%s\n",sendBuffer );
		send(new_socket, sendBuffer, size, 0);
		printf("\nData sent\n");
	}
}

int main(char argc, char **argv){


	struct sockaddr_in cli_addr;

	cli_addr.sin_family = AF_INET;
	cli_addr.sin_addr.s_addr = INADDR_ANY;
	cli_addr.sin_port = htons(PORT);
```

```
	int sockfd = socket(AF_INET, SOCK_STREAM, 0);

	bind(sockfd, (struct sockaddr*)&cli_addr, sizeof(cli_addr));
	listen(sockfd, 5);

	int clilen = sizeof(cli_addr);


	while(1){
		int newsockfd = accept(sockfd, (struct sockaddr*)&cli_addr, &clilen);

		if(newsockfd < 0){
			perror("ERROR on accept\n");
			exit(1);
		}
		int pid = fork();

		if(pid<0){
			perror("ERROR on fork\n");
			exit(1);
		}

		if(pid == 0){
			close(sockfd);
			handle_client(newsockfd);
			exit(0);
		}else{
			close(newsockfd);
		}
	}

	return 0;
}
```

Yes, two concurrent client request the same file