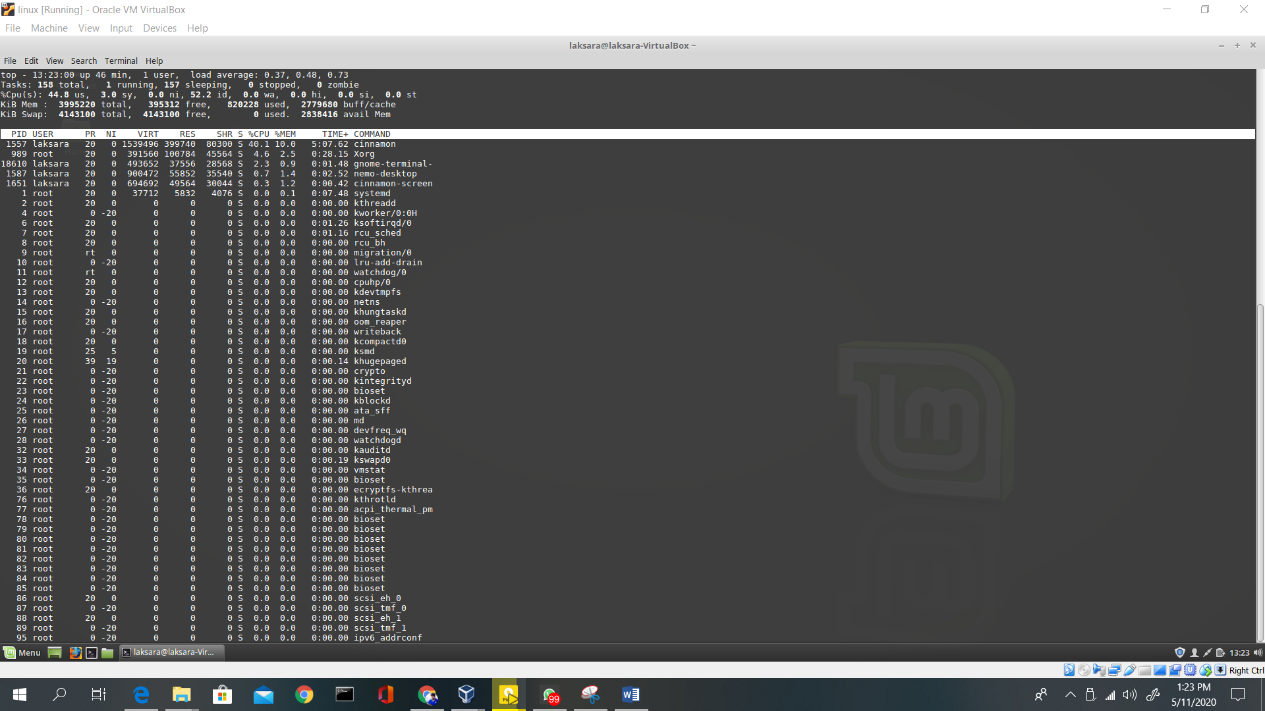
**Multiprocessing**

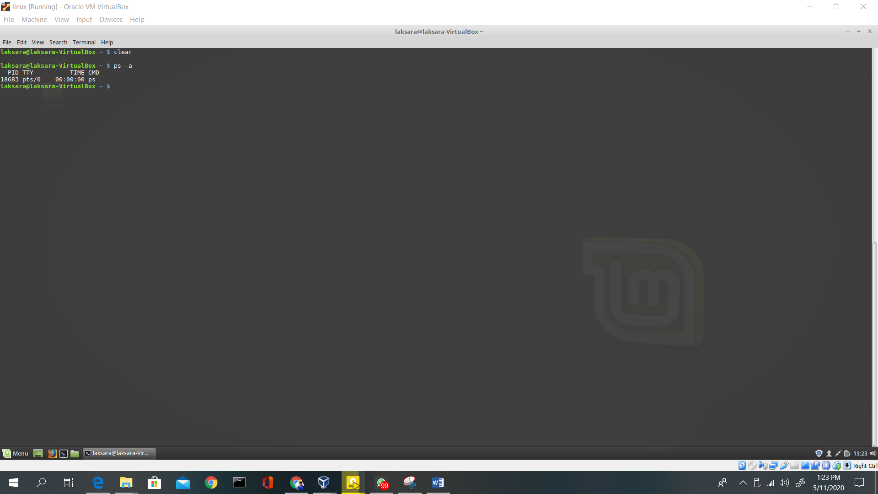
**Exercise 1**

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

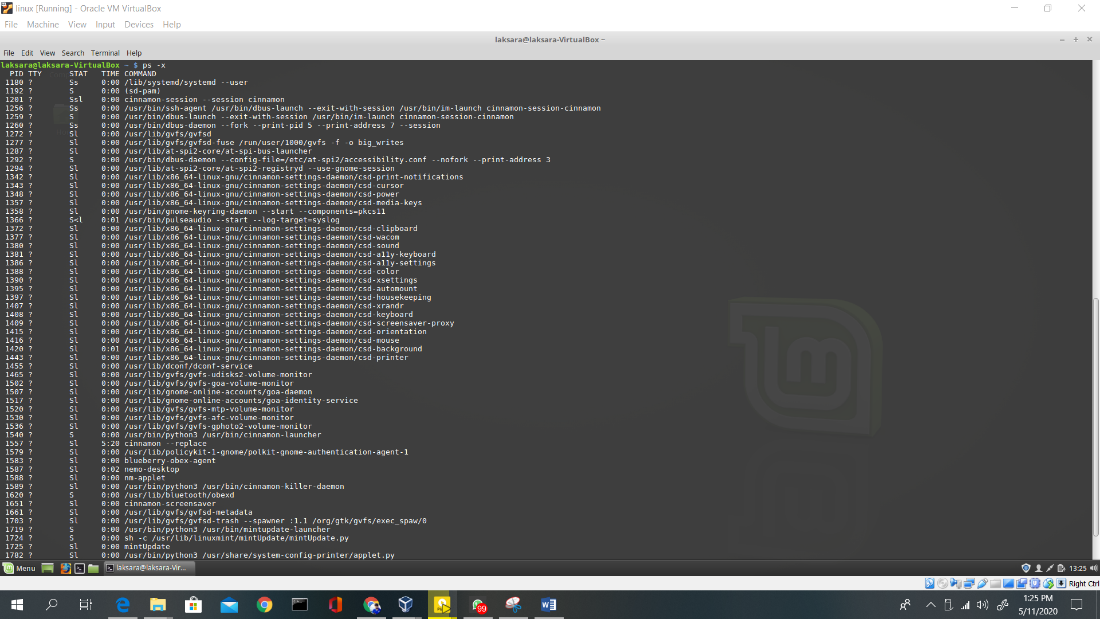


1. Run ps with the following options: -a, -x, -u, -w. What is the name of the process with PID 1?

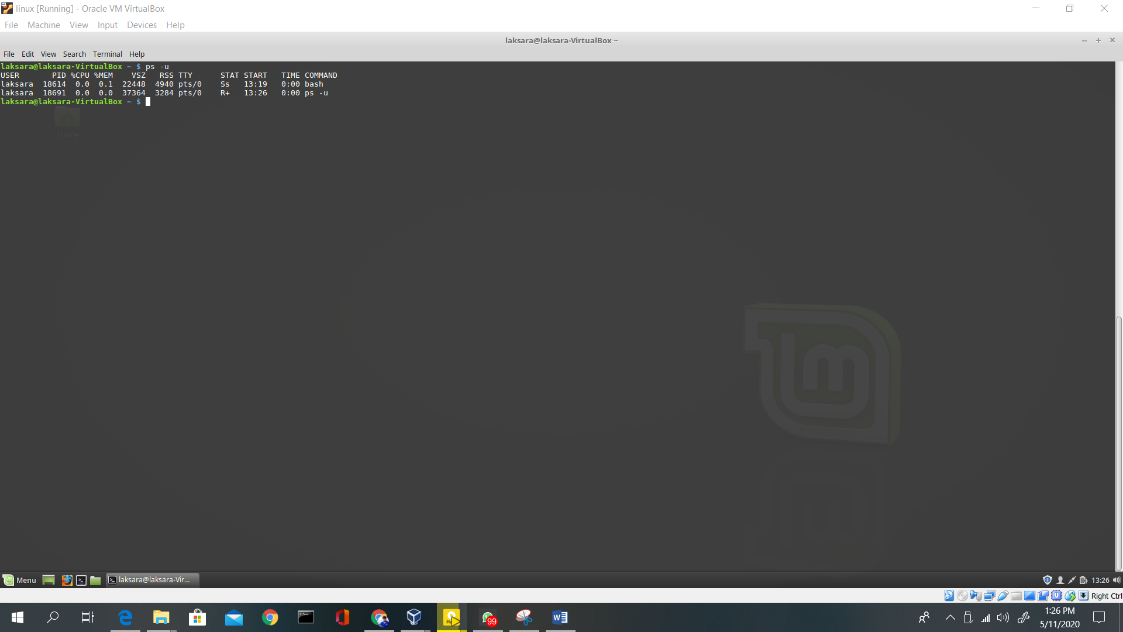
* ps –a



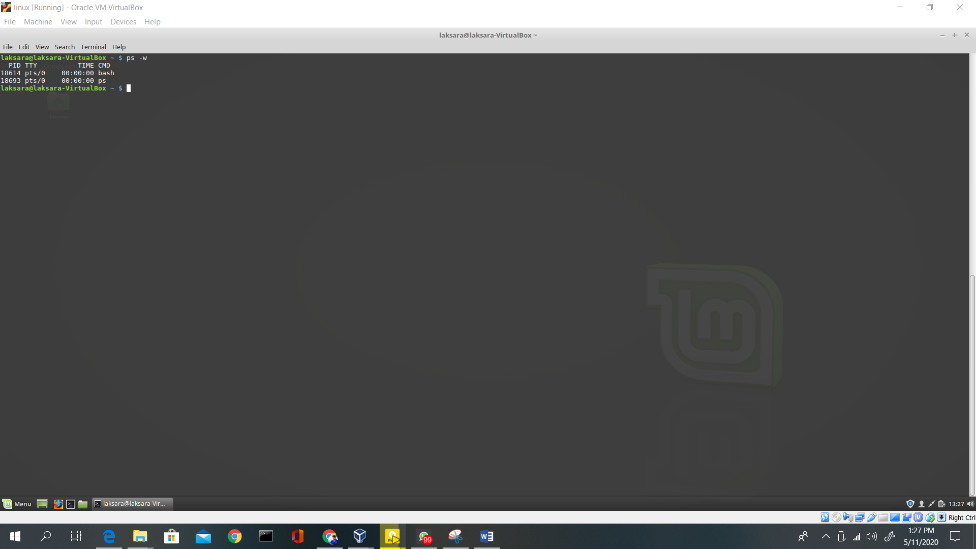
* ps –x



* ps –u



* 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

1. 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

19773

19774

19780

19778

19776

19779

19777

19775

**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.

1. 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{

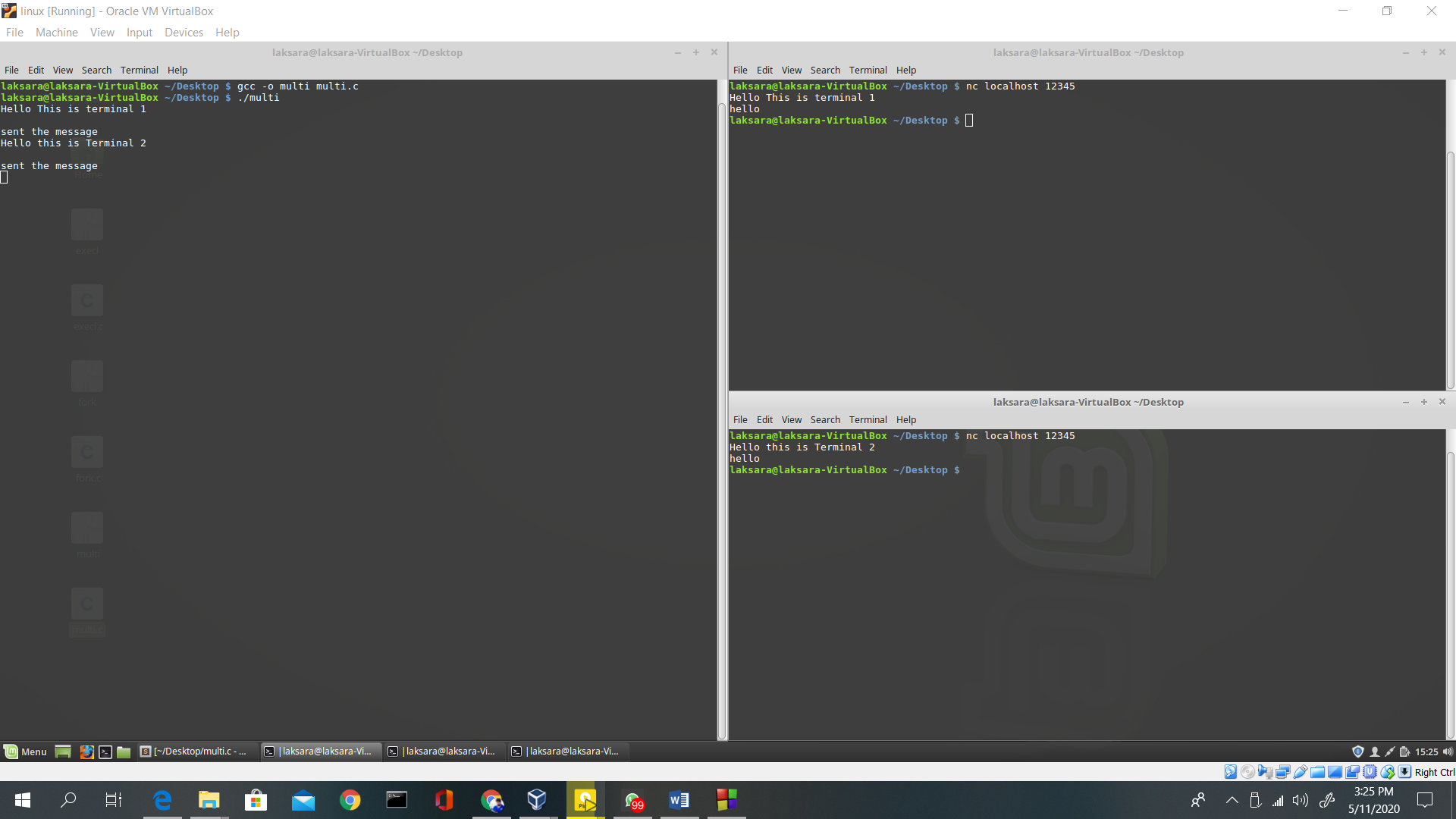
close(newsockfd);

}

}

return 0;

}



1. 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.

1. 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

1. 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

