**CS2106 Operating Systems**

**Lab 2 – Fork and Pipe Answer Sheet**

|  |  |
| --- | --- |
| Student ID 1:  A0161868R | Name 1:  Joshua Lee Kai Sheng |
| Student ID 2: | Name 2: |
| Student ID 3: | Name 3 : |

**Question 1.** (5 marks)

Parent’s process ID: 24202

Child’s process ID: 24203

Child’s parent’s process ID (as reported by child): 1312

Parent’s parent’s process ID (as reported by parent): 2448

Parent of the parent: bash

**Question 2.** (3 marks)

The web browser cannot connect to the web server because …

When deliverHTTP is called, it will try to read in values from the connfd. However, there is no input to the connection established using telnet. Hence, the execution does not proceed past the read method. As such the deliverHTTP for the first connection is not fully executed when the Mozilla browser tries to establish another connection. And deliverHTTP for this second connection is not executed. (only support one connection at a time)

**Question 3a.** (5 marks)

The modifications I made, and my explanations for them are:

while(1)

{

connfd = accept(listenfd, (struct sockaddr \*) NULL, NULL);

if(fork() != 0){

writeLog("Connection received.");

deliverHTTP(connfd);

}

}

Create a fork after the accept function, which establishes a connection with a client.

The parent process will then handle the deliverHTTP() function.

The child process can proceed on to the next loop in order to connect with the next client, regardless of whether deliverHTTP() function in parent is blocked.

**Question 3b.** (7 marks)

The maximum number of connections I can make is:

**Question 4a.** (3 marks)

We must close the ends of the pipe we are not using because..

We want the pipe to provide single directional communication.

We first close the read descriptor in the parent process as we only want the parent to write to the pipe in order not to waste file descriptors and to detect possible dying reader in the child process.

We first close the write descriptor in the child process so that the child can detect the EOF condition and read from the pipe.

**Question 4b.** (5 marks)

This statement is false because…

The child and parent processes do not share the same data (memory space, arguments, environment variables) as they are in different processes and hence will have different contexts.

Hence, the two processes do not have access to the same buffer array. They have, in fact, different copies of buffer arrays.

**Question 5.** (7 marks)

The modifications I made, with explanation, are:

int fd[2];

int main(int ac, char \*\*av)

{

pipe(fd);

if (fork() != 0) {

int status;

close(fd[0]);

startServer(PORTNUM);

close(fd[1]);

wait(&status);

} else {

char buffer[LOG\_BUFFER\_LEN];

FILE\* fptr;

fptr = fopen("log.txt", "wb");

close(fd[1]);

int n;

while (1) {

if ((n = read(fd[0], buffer, LOG\_BUFFER\_LEN)) > 0) {

FILE\* fptr;

fptr = fopen("log.txt", "a");

//fprintf(fptr, "%s\n", buffer);

fwrite(buffer, 1, n, fptr);

fclose(fptr);

}

}

close(fd[0]);

exit(1);

}

}

void writeLog(const char \*format, ...)

{

char logBuffer[LOG\_BUFFER\_LEN];

va\_list args;

sprintf(logBuffer, "%s: ", getCurrentTime());

va\_start(args, format);

vsprintf(logBuffer + strlen(logBuffer), format, args);

va\_end(args);

sprintf(logBuffer + strlen(logBuffer), "\n");

write(fd[1], logBuffer, strlen(logBuffer));

}

Created a fork in the main method where the parent will be responsible calling writeLog which will write the logs to the pipe.

The child process, whereas, will be responsible for reading from the pipe and write the log read to the log.txt. The while loop in the child process ensures that the writing to log file process is executed continuously.

The file descriptors are closed accordingly at the appropriate places.