Student number:	2467273S
Course title:	COMPSCI2024, Networks and Operating Systems Essentials 2
Questions answered:	ALL

Start exam here

1. Networked Systems

(a)

- Layer – Data Link Layer
- Protocol – Internet Protocol (IP)

(b)

Connection is set up via the 3-way handshake:

1. Host A sends SYN (synchronise) bit with a random sequence number (x)
2. Host B receives these and responds with the same SYN bit, a randomly chosen sequence number (y) and acknowledges (ACK) the initial sequence number (x)
3. Host A sends an acknowledgment with the other sequence number (y)

The two sequence numbers ensure robustness by ensuring that data does not get corrupted while being sent in either direction and that delayed packets can be relocated (the sequence number can act as an identifier).

(c)

Server 1 runs on UDP as it does not establish a connection, messages are directed to specific addresses:
"message, clientAddress = serverSocket.**recvfrom**(2048)"
"serverSocket.**sendto**(modifiedMessage.encode(),**clientAddress**)",
the socket is created using datagrams to specify that this will be the socket type:
"serverSocket = socket(AF_INET, SOCK_**DGRAM**)".

Server 2 runs on TCP as it accepts a connection request (establishes connection via the 3-way handshake):
"connectionSocket, addr = serverSocket.**accept**()",
sends and receives data with send() and recv(), respectively:
"phrase = connectionSocket.**recv**(1024).decode()"
"connectionSocket.**send**(capsPhrase.encode())",
and it also closes the connection at the end:
"connectionSocket.**close**()".

(d)

1. clientSocket.connect((server_address, 13000))

2. It creates a queue where new connection requests will be added by the OS kernel. The length of this queue in the code is 1. If one connection is already established and processing, another can wait in the queue to be processed.

3. It is the client socket that the server communicates with and which can be specified when that is the case.

4. 13000

5. Up to 1024

(e)

Because server1.py is run over UDP, there is no connection established (UDP is connectionless), which is why server1 does not need to keep track of two sockets (end-points/hosts), just its own server socket, unlike server2.py, which is run over TCP; hence, server2.py needs to keep track of both sockets for the established connection to continue (and finally end with connectionSocket.close()).

(f)

This is possible with a Network Address Translation (NAT) box. The NAT box is given the IPv4 address, each home network host is given a private address, and the NAT performs address translation by rewriting packet headers to match its external IP address, while also possibly rewriting the TCP/UDP port number.

2. Security

(a)

TLS (Transport Layer Security) – the standard for Internet security that is based on the Secure Socket Layer protocol (SSL). It enables privacy and data integrity between two communicating applications.

With TCP, the client sends a SYN bit and the server responds with SYN ACK. Then the client sends over a “hello” message, and the server responds with a server hello and sends over a certificate. The client checks the certificate, decides on a cipher and session key to use with the server, then sends those over. The server receives the cipher, changes its own cipher to the one offered by the client. By the end, the connection is established.

(b)

If the attacker is in close proximity, they can do a man-in-the-middle attack by impersonating a router or by sniffing for incoming connection requests from Alice to the public router and intercepting them midway. Then the attacker would impersonate Alice to Gmail by sending an HTTP connection request, being redirected to HTTPS, then going through the TLS handshake with Gmail

servers (client hello, server hello & certificate, client session key, server “secure connection” message), and then send their HTTPS connection to Alice and see all incoming and outgoing information in this exchange.

(c)

Because teleconferencing requires fast delivery of data, I would use the Hypertext Transfer Protocol Secure (HTTPS), i.e., HTTP over TLS, which uses the Advanced Encryption Standard (AES) for the main data encryption, which is fast as it is a type of symmetric cryptography, and the Diffie-Hellman algorithm to distribute the session key, which is slower but more secure as it is a type of public-key cryptography. Furthermore, the slowness of Diffie-Hellman is not as significant as usual as it is only the session key (a relatively short sequence of characters) that is encrypted with this algorithm.

Ensuring that the entire packet is encrypted is possible if the entire payload is encrypted with Encapsulating Security Payload (EPS), and the session key is encrypted with the Internet Key Exchange protocol (IKE). Additional security can be ensured with the Authentication Header, also under IPSec.

(d)

1. The cryptography categories to consider would be symmetric vs. antisymmetric cryptography. Symmetric would be fast, but may need base64 encoding, but a concern is how to distribute the cipher key between communicating parties.
2. Integrity can be ensured with a combination of cryptographic hash and public key cryptography, meaning that a unique hash code can be made with the message as its input, then sent over with public key cryptography for the receiving party to compare their own, locally made hash code of the message with the one received from the original party.

3. Operating Systems

(a)

The register file is an array of available registers for the CPU to use. In a 64-bit architecture, the file contains 64 x 64-bit registers, which means that a word is of size 64 bits, and the page sizes are 4kB big. Some of the registers are special, e.g. the program counter and the stack pointer. The Base Register is used to store the first address in a process’ address space, and the Limit Register is used for the last address.

(b)

Execution order: top to bottom:

Running process	**Execution interval (*ms*)**

P0	0-5
P1	5-10
P2	10-15
P3	15-20
P4	20-25
P1	25-29
P2	29-34
P3	34-39
P4	39-44
P2	44-49
P4	49-50

Each process (times in *ms*):

- P0:
  - Turnaround time: 5-0 = 0
  - Execution interval: 0-5
- P1:
  - Turnaround time: 29-3 = 26
  - Execution intervals: 5-10, 25-29
- P2:
  - Turnaround time: 49-4 = 45
  - Execution intervals: 10-15, 29-34, 44-49
- P3:
  - Turnaround time: 39-6 = 33
  - Execution intervals: 15-20, 34-39
- P4:
  - Turnaround time: 50-8 = 42
  - Execution intervals: 20-25, 39-44, 49-50

(c)

Execution order: top to bottom:

Running process	**Execution interval (*ms*)**
P0	0-5
P1	5-14
P3	14-24
P2	24-39
P4	39-50

Waiting times (*ms*):

- P0: 0-0 = 0
- P1: 5-3 = 2
- P2: 24-4 = 20
- P3: 14-6 = 8
- P4: 39-8 = 31
- Avg: 12.2