**MWENDA KELVIN MUTHOMI**

**CIT-223-008/2016**

**Bsc COMPUTER SCIENCE**

**CCS 2312 SYSTEMS PROGRAMMING**

**a)**

**Question b**

**IO buffering-**The process of temporarily storing data that is passing between a processor and a peripheral. The usual purpose is to smooth out the difference in rates at which the two devices can handle data.

**There are different IO buffering approaches:**

**1. Single buffering:**

* When a user process issues an I/O request, the operating system assigns a buffer in the system portion of main memory to the operation.
* In the block oriented devices, the techniques can be used as follows: Input transfers are made to the system buffer. When the transfer is complete, the process moves the block into user space and request another block. This is called reading ahead, it is done in the expectation that the block will be needed sometimes in future.
* This approach will generally provide a speed up compared to the lack of system buffering. The O.S must keep track of the assignment of system buffers to user processes.
* Operating system assigns a buffer in Kernel’s memory foran I/O request .•
* Similar considerations apply to block oriented output. When data are being transmitted to a device, they are first copied from user space into the system buffer, from which they will ultimately be written. The requesting process is now free to continue.
* Suppose T is the time required to input one block and C is the computation time required for input request.
  + Without buffering: Execution time is T+C.
  + Without buffering: Execution time is max [C,T]+M, where M is time requied to move the data from system buffer to user memory.

In stream oriented I/O, it can be used in two ways,

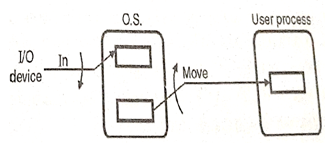
* Line-at a time fashion. Line- at a time operation is used for scroll made terminals. User inputs one line at a time, with a carriage return signaling at the end of a line.
* Byte-at a time fashion. Byte-at a time operation is used on forms mode, terminals when each keystroke is significant.

**2. Double buffering**

* An improvement over single buffering is by assigning two system buffers to the operations.
* A process transfers data to one buffer while operating system empties the other as shown in fig.
* For block oriented transfer execution time is Max[C,T]. It is possible to keep the block oriented device going at full speed.
  + If C<=T, i.e. computation time is less than the time required to input one block.
  + If C>T, i.e. computation time is greater than the time required to input one block, then double buffering ensures that the process will not have to wait on I/O.

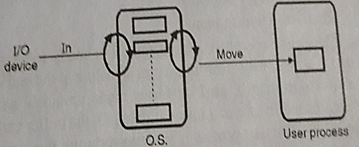
For Stream oriented input again two types.

* For line- at a time I/O, the user process need not be suspended for input or output, unless process runs ahead of double buffer.
* For byte- at a time operations, double buffer offers no advantage over a single buffer of twice the length.



**3. Circular buffer**

* Double buffering may be inadequate, if the process performs rapid burst of I/O. When two or more buffers are used.
* The collection of buffers is called as a circular buffer, with each buffer being one unit in the circular buffer.



**Question c**

**Connection oriented and connectionless communication**

**Connection-oriented**Requires a session connection (analogous to a phone call) be established before any data can be sent. This method is often called a "reliable" network service. It can guarantee that data will arrive in the same order. Connection-oriented services set up virtual links between end systems through a network

**Connectionless**Does not require a session connection between sender and receiver. The sender simply starts sending packets (called datagrams) to the destination. This service does not have the reliability of the connection-oriented method, but it is useful for periodic burst transfers. Neither system must maintain state information for the systems that they send transmission to or receive transmission from. A connectionless network provides minimal services.

Difference between connection-oriented and connectionless service

* A prior connection setup is needed in connection-oriented service but not in connectionless service.
* Connection-oriented service guarantees reliability but not connectionless service.
* Congestion is very unlikely in connection-oriented service but not in connectionless.
* Lost data retransmission is possible in connection-oriented service but not in connectionless service.
* Connection-oriented is suitable for long connection while connectionless is suitable for a bursty connection
* Packets reach the destination following the same route in connection-oriented service, but for connectionless, the packets can take different paths.
* Resource allocation is needed in the connection-oriented but not in the case of connectionless service.
* The transfer is slower in the connection-oriented due to connection setup time and ACK but is faster in connectionless service due to missing initial setup and ACK.

Circumstances for connection oriented communication

* [Connection-oriented Ethernet](https://en.wikipedia.org/wiki/Connection-oriented_Ethernet)
* [DCCP](https://en.wikipedia.org/wiki/Datagram_Congestion_Control_Protocol)
* [Frame Relay](https://en.wikipedia.org/wiki/Frame_Relay)
* [GPRS](https://en.wikipedia.org/wiki/GPRS)
* [IPX/SPX](https://en.wikipedia.org/wiki/IPX/SPX)
* [Multiprotocol Label Switching](https://en.wikipedia.org/wiki/Multiprotocol_Label_Switching)
* [SCTP](https://en.wikipedia.org/wiki/SCTP)
* [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)

Circumstances for connectionless communication

* Internet **Protocol** (IP)
* User Datagram **Protocol** (UDP)
* Internet Control Message **Protocol** (ICMP)
* Internetwork Packet Exchange (IPX)
* Transparent Inter-process **Communication**.
* NetBEUI.
* Fast and Secure **Protocol** (FASP)

**Question d**

**i) Normal termination**

There are some operating systems that make use of the return value from a program. If the program runs correctly and returns zero, they go on the do the next step (normal termination.)

.model small

.stack

.data

a db 10, 13, "Do you want to Try again? Y/N $"

.code

org 0100h

main:

mov ax, @data

mov ds, ax

dsply:

mov ah, 9

lea dx, a

int 21h

mov ah, 1

int 21h

mov cl, al

cmp cl, 'Y'

je dsply

cmp cl, 'N'

je endmain

endmain:

mov ah, 4ch

int 21h

end main

**ii) Abnormal termination**

If the program detects a problem (e.g. missing database, no Internet connection, etc.) and returns a non-zero value, they abort the next step (abnormal termination.)

**iii) Disposition**

The signal disposition is a per-process attribute: in a multithreaded application, the disposition of a particular signal is the same for all threads. The disposition of a signal is how it is handled.