Week 7– Assignments

1. Explain in detail and Write a C program for Page Buffering Algorithm (A/P)

Buffering in Operating System

The ***buffer*** is an area in the **main memory** used to store or hold the data **temporarily**. In other words, buffer temporarily stores data transmitted from one place to another, either between two devices or an application. The act of storing data temporarily in the buffer is called ***buffering***.

A buffer may be used when moving data between processes within a computer. Buffers can be implemented in a fixed memory location in hardware or by using a virtual data buffer in software, pointing at a location in the physical memory. In all cases, the data in a data buffer are stored on a physical storage medium.

Most buffers are implemented in software, which typically uses the faster RAM to store temporary data due to the much faster access time than hard disk drives. Buffers are typically used when there is a difference between the rate of received data and the rate of processed data, for example, in a printer spooler or online video streaming.

A buffer often adjusts timing by implementing a queue or FIFO algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

Purpose of Buffering

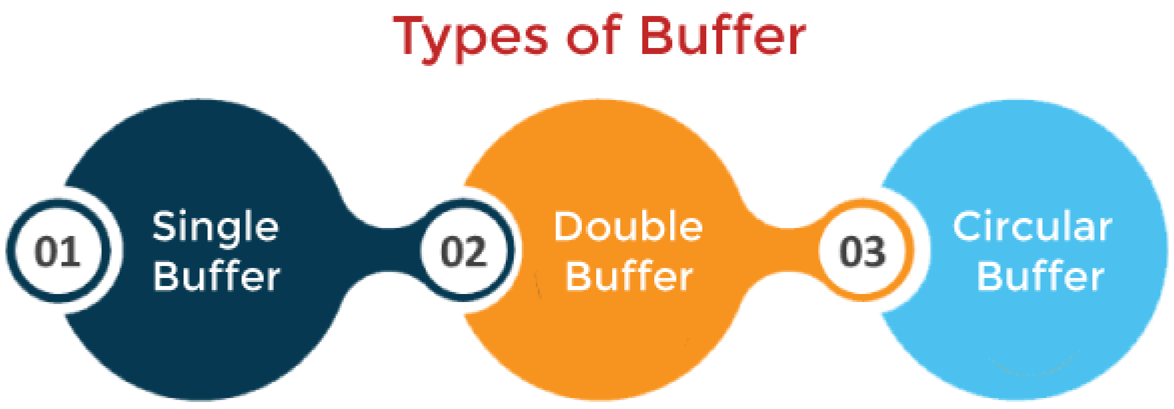
You face buffer during watching videos on YouTube or live streams. In a video stream, a buffer represents the amount of data required to be downloaded before the video can play to the viewer in real-time. A buffer in a computer environment means that a set amount of data will be stored to preload the required data before it gets used by the CPU.

Computers have many different devices that operate at varying speeds, and a buffer is needed to act as a temporary placeholder for everything interacting. This is done to keep everything running efficiently and without issues between all the devices, programs, and processes running at that time. There are three reasons behind buffering of data,

1. It helps in ***matching speed*** between two devices in which the data is transmitted. For example, a hard disk has to store the file received from the modem. As we know, the transmission speed of a modem is slow compared to the hard disk. So bytes coming from the modem is accumulated in the buffer space, and when all the bytes of a file has arrived at the buffer, the entire data is written to the hard disk in a single operation.
2. It helps the devices with different ***sizes of data transfer*** to get adapted to each other. It helps devices to manipulate data before sending or receiving it. In computer networking, the large message is fragmented into small fragments and sent over the network. The fragments are accumulated in the buffer at the receiving end and reassembled to form a complete large message.
3. It also supports ***copy semantics***. With copy semantics, the version of data in the buffer is guaranteed to be the version of data at the time of system call, irrespective of any subsequent change to data in the buffer. Buffering increases the performance of the device. It overlaps the I/O of one job with the computation of the same job.

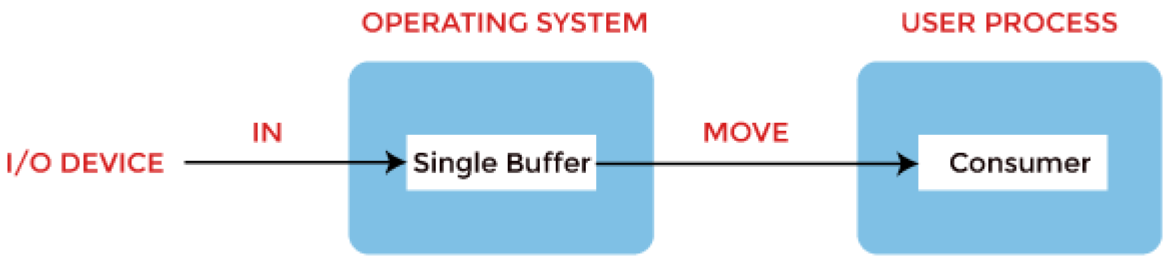
Types of Buffering

There are three main types of buffering in the operating system, such as:



1. Single Buffer

In Single Buffering, only one buffer is used to transfer the data between two devices. The producer produces one block of data into the buffer. After that, the consumer consumes the buffer. Only when the buffer is empty, the processor again produces the data.



SINGLE I/O BUFFERING

Block oriented device: The following operations are performed in the block-oriented device,

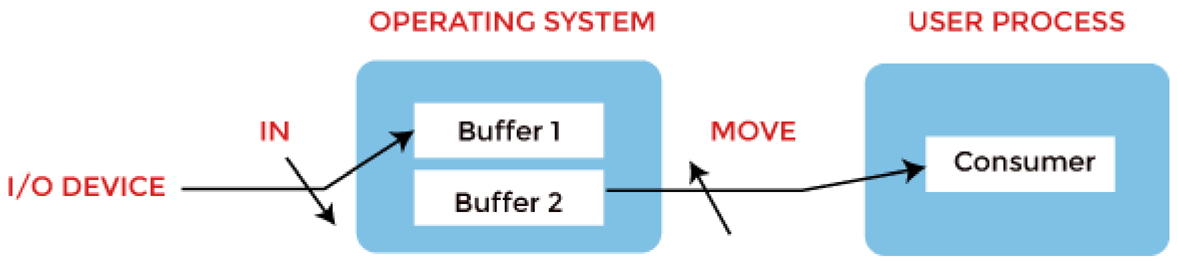
- System buffer takes the input.
- After taking the input, the block gets transferred to the user space and then requests another block.
- Two blocks work simultaneously. When the user processes one block of data, the next block is being read in.
- OS can swap the processes.
- OS can record the data of the system buffer to user processes.

Stream oriented device: It performed the following operations, such as:

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

2. Double Buffer

In **Double Buffering**, two schemes or two buffers are used in the place of one. In this buffering, the producer produces one buffer while the consumer consumes another buffer simultaneously. So, the producer not needs to wait for filling the buffer. Double buffering is also known as buffer swapping.



DOUBLE I/O BUFFERING

Block oriented: This is how a double buffer works. There are two buffers in the system.

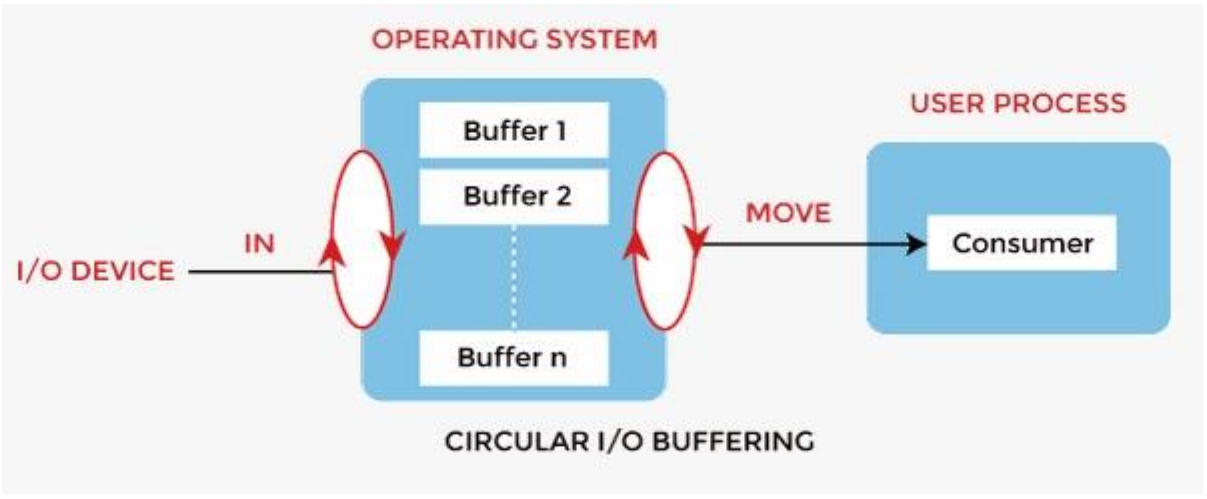
- The driver or controller uses one buffer to store data while waiting for it to be taken by a higher hierarchy level.
- Another buffer is used to store data from the lower-level module.
- A major disadvantage of double buffering is that the complexity of the process gets increased.
- If the process performs rapid bursts of I/O, then using double buffering may be deficient.

Stream oriented: It performs these operations, such as:

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

3. Circular Buffer

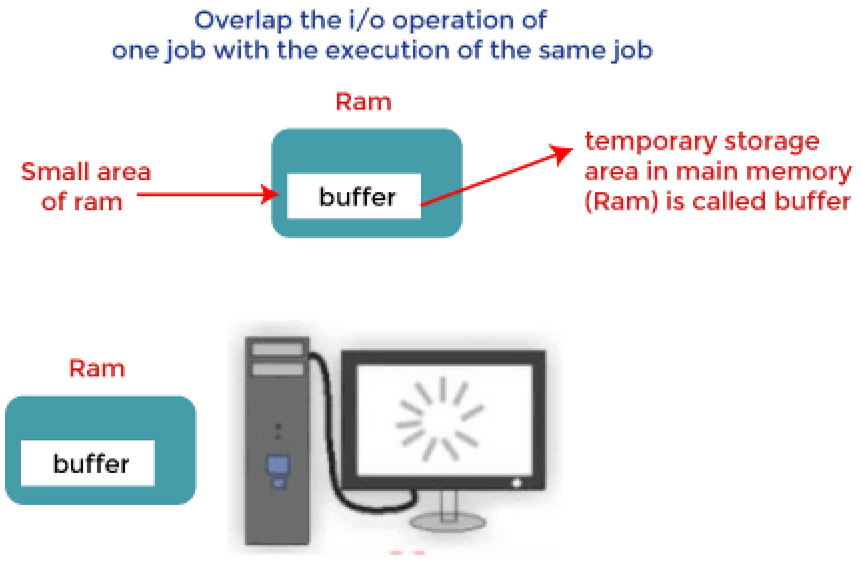
When more than two buffers are used, the buffers' collection is called a **circular buffer**. Each buffer is being one unit in the circular buffer. The data transfer rate will increase using the circular buffer rather than the double buffering.



- In this, the data do not directly pass from the producer to the consumer because the data would change due to overwriting of buffers before consumed.
- The producer can only fill up to buffer x-1 while data in buffer x is waiting to be consumed.

How Buffering Works

In an operating system, buffer works in the following way:



Buffering

- Buffering is done to deal effectively with a speed mismatch between the producer and consumer of the data stream.
- A buffer is produced in the main memory to heap up the bytes received from the modem.
- After receiving the data in the buffer, the data get transferred to a disk from the buffer in a single operation.
- This process of data transfer is not instantaneous. Therefore the modem needs another buffer to store additional incoming data.
- When the first buffer got filled, then it is requested to transfer the data to disk.
- The modem then fills the additional incoming data in the second buffer while the data in the first buffer gets transferred to the disk.
- When both the buffers completed their tasks, the modem switches back to the first buffer while the data from the second buffer gets transferred to the disk.
- Two buffers disintegrate the producer and the data consumer, thus minimising the time requirements between them.
- Buffering also provides variations for devices that have different data transfer sizes.

2. Explain in detail and Write a C program for LRU Approximation Algorithm

```
// C++ implementation of the approach

#include <bits/stdc++.h>

using namespace std;


// Function to find an element in the queue as

// std::find does not work for a queue

bool findQueue(queue<int> q, int x)

{

        while (!q.empty()) {

                if (x == q.front())
                return true;

            q.pop();

        }


        // Element not found

        return false;

}


// Function to implement LRU Approximation

void LRU_Approximation(vector<int> t, int capacity)

{

        int n = t.size();

        queue<int> q;


        // Capacity is the size of the queue

        // hits is number of times page was

        // found in cache and faults is the number

        // of times the page was not found in the cache

        int hits = 0, faults = 0;


        // Array to keep track of bits set when a

        // certain value is already in the queue

        // Set bit --> 1, if its a hit

        // find the index and set bitref[index] = 1

        // Set bit --> 0, if its a fault, and the front

        // of the queue has bitref[front] = 1, send front
        // to back and set bitref[front] = 0
        bool bitref[capacity] = { false };

        // To find the first element that does not
        // have the bitref set to true
        int ptr = 0;

        // To check if the queue is filled up or not
        int count = 0;
        for (int i = 0; i < t.size(); i++) {
            if (!findQueue(q, t[i])) {

                // Queue is not filled up to capacity
                if (count < capacity) {
                        q.push(t[i]);
                        count++;
                }

                // Queue is filled up to capacity
                else {
                        ptr = 0;

                        // Find the first value that has its
                        // bit set to 0
                        while (!q.empty()) {
```

```
        // If the value has bit set to 1
        // Set it to 0
        if (bitref[ptr % capacity])
            bitref[ptr % capacity] = !bitref[ptr % capacity];

        // Found the bit value 0
        else
            break;
        ptr++;
    }

    // If the queue was empty
    if (q.empty()) {
        q.pop();
        q.push(t[i]);
    }

    // If queue was not empty
    else {
        int j = 0;

        // Rotate the queue and set the front's
        // bit value to 0 until the value where
        // the bitref = 0
        while (j < (ptr % capacity)) {
            int t1 = q.front();
```

```
                                        q.pop();

                                        q.push(t1);

                                        bool temp = bitref[0];


                                        // Rotate the bitref array

                                        for (int counter = 0; counter < capacity - 1;
counter++)

                                             bitref[counter] = bitref[counter + 1];

                                        bitref[capacity - 1] = temp;

                                        j++;

                                   }


                                   // Remove front element

                                   // (the element with the bitref = 0)

                                   q.pop();


                                   // Push the element from the

                                   // page array (next input)

                                   q.push(t[i]);

                              }

                         }

                         faults++;

                    }


                    // If the input for the iteration was a hit

                    else {
```

```
                    queue<int> temp = q;
                    int counter = 0;
                    while (!q.empty()) {
                        if (q.front() == t[i])
                            bitref[counter] = true;
                        counter++;
                        q.pop();
                    }
                    q = temp;
                    hits++;
                }
        }
        cout << "Hits: " << hits << "\nFaults: " << faults << '\n';
}

// Driver code
int main()
{
        vector<int> t = { 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 };
        int capacity = 4;
        LRU_Approximation(t, capacity);

        return 0;
}
```

3. Explain in detail and Write a C program to illustrate clock page replacement algorithm.

Related Question

In some cases individuals may care about the date at which the uncertainty they face is resolved. Suppose, for example, that an individual knows that his or her consumption will be 10 units today (C1)(C1) but that tomorrow's consumption (C2)(C2) will be either 10 or 2.5,2.5, depending on whether a coin comes up heads or tails. Suppose also that the individual's utility function has the simple Cobb-Douglas form

U(C1,C2)=C1C2−−−−−√.U(C1,C2)=C1C2.

a. If an individual cares only about the expected value of utility, will it matter whether the coin is flipped just before day 1 or just before day 2 ? Explain. b. More generally, suppose that the individual's expected utility depends on the timing of the coin flip. Specifically, assume that

expected utility =E1[(E2{U(C1,C2)})α] expected utility =E1[(E2{U(C1,C2)})α]

where E1E1 represents expectations taken at the start of day 1,E21,E2 represents expectations at the start of day 2,2, and αα represents a parameter that indicates timing preferences. Show that if α=1,α=1, the individual is indifferent about when the coin is flipped. c. Show that if α=2α=2, the individual will prefer early resolution of the uncertainty-that is, flipping the coin at the start of day 1 d. Show that if α=.5,α=.5, the individual will prefer later resolution of the uncertainty (flipping at the start of day 2). e. Explain your results intuitively and indicate their relevance for information theory. (Note: This problem is an illustration of "resolution seeking" and "resolution-averse" behavior. See D. M. Kreps and E. L. Porteus, "Temporal Resolution of Uncertainty and Dynamic Choice Theory," Econometrica

4. Explain in detail and Write a C program for Enhanced Second Chance Algorithm

```
// Java program to find largest in an array
// without conditional/bitwise/ternary/ operators
// and without library functions.
import java.util.*;
import java.io.*;
class secondChance
{
        public static void main(String args[])throws IOException
        {
                String reference_string = "";
                int frames = 0;

                //Test 1:
                reference_string = "0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4";
                frames = 3;
                //Output is 9
                printHitsAndFaults(reference_string,frames);

                //Test 2:
                reference_string = "2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1";
                frames = 4;

                //Output is 11
                printHitsAndFaults(reference_string,frames);

        }

        //If page found, updates the second chance bit to true
        static boolean findAndUpdate(int x,int arr[],
                                        boolean second_chance[],int frames)

        {
                int i;

                for(i = 0; i < frames; i++)
                {

                        if(arr[i] == x)
                        {
                                //Mark that the page deserves a second chance
                                second_chance[i] = true;

                                //Return 'true', that is there was a hit
                                //and so there's no need to replace any page
                                return true;
                        }
                }

                //Return 'false' so that a page for replacement is selected
                //as he reuested page doesn't exist in memory
                return false;

        }


        //Updates the page in memory and returns the pointer
        static int replaceAndUpdate(int x,int arr[],
                                boolean second_chance[],int frames,int pointer)
        {
                while(true)
                {

                        //We found the page to replace
                        if(!second_chance[pointer])
                        {
                                //Replace with new page
                                arr[pointer] = x;

                                //Return updated pointer
                                return (pointer+1)%frames;
                        }

                        //Mark it 'false' as it got one chance
                        // and will be replaced next time unless accessed again
                        second_chance[pointer] = false;

                        //Pointer is updated in round robin manner
                        pointer = (pointer+1)%frames;
                }
        }

        static void printHitsAndFaults(String reference_string,

        int frames)
        {
                int pointer,i,l,x,pf;

                //initially we consider frame 0 is to be replaced
                pointer = 0;

                //number of page faults
                pf = 0;

                //Create a array to hold page numbers
                int arr[] = new int[frames];

                //No pages initially in frame,
                //which is indicated by -1
                Arrays.fill(arr,-1);

                //Create second chance array.
                //Can also be a byte array for optimizing memory
                boolean second_chance[] = new boolean[frames];

                //Split the string into tokens,
                //that is page numbers, based on space
                String str[] = reference_string.split(" ");

                //get the length of array
                l = str.length;

                for(i = 0; i<l; i++)
                {

                        x = Integer.parseInt(str[i]);

                        //Finds if there exists a need to replace
                        //any page at all
                        if(!findAndUpdate(x,arr,second_chance,frames))
                        {
                                //Selects and updates a victim page
                                pointer = replaceAndUpdate(x,arr,
                                                second_chance,frames,pointer);

                                //Update page faults
                                pf++;
                        }
                }

                System.out.println("Total page faults were "+pf);
        }
}
```