```
func main() {

	s := time.Now()
	args := os.Args[1:]
	if len(args) != 6 { // for format  LogExtractor.exe -f "From Time" -t "To Time" -i "Log file directory location"
		fmt.Println("Please give proper command line arguments")
		return
	}
	startTimeArg := args[1]
	finishTimeArg := args[3]
	fileName := args[5]

	file, err := os.Open(fileName)

	if err != nil {
		fmt.Println("cannot able to read the file", err)
		return
	}

	defer file.Close() //close after checking err

	queryStartTime, err := time.Parse("2006-01-02T15:04:05.0000Z", startTimeArg)
	if err != nil {
		fmt.Println("Could not able to parse the start time", startTimeArg)
		return
	}

	queryFinishTime, err := time.Parse("2006-01-02T15:04:05.0000Z", finishTimeArg)
	if err != nil {
		fmt.Println("Could not able to parse the finish time", finishTimeArg)
		return
	}

	filestat, err := file.Stat()
	if err != nil {
		fmt.Println("Could not able to get the file stat")
		return
	}

	fileSize := filestat.Size()
	offset := fileSize - 1
	lastLineSize := 0

	for {
		b := make([]byte, 1)
		n, err := file.ReadAt(b, offset)
```

```
                if err != nil {
                        fmt.Println("Error reading file ", err)
                        break
                }
                char := string(b[0])
                if char == "\n" {
                        break
                }
                offset--
                lastLineSize += n
        }

        lastLine := make([]byte, lastLineSize)
        _, err = file.ReadAt(lastLine, offset+1)

        if err != nil {
                fmt.Println("Could not able to read last line with offset", offset, "and lastline size", lastLineSize)
                return
        }

        logSlice := strings.SplitN(string(lastLine), ",", 2)
        logCreationTimeString := logSlice[0]

        lastLogCreationTime, err := time.Parse("2006-01-02T15:04:05.0000Z", logCreationTimeString)
        if err != nil {
                fmt.Println("can not able to parse time : ", err)
        }

        if lastLogCreationTime.After(queryStartTime) && lastLogCreationTime.Before(queryFinishTime) {
                Process(file, queryStartTime, queryFinishTime)
        }

        fmt.Println("\nTime taken - ", time.Since(s))
}

func Process(f *os.File, start time.Time, end time.Time) error {

        linesPool := sync.Pool{New: func() interface{} {
                lines := make([]byte, 250*1024)
                return lines
        }}

        stringPool := sync.Pool{New: func() interface{} {
                lines := ""
                return lines
        }}
```

```
        r := bufio.NewReader(f)

        var wg sync.WaitGroup

        for {
                buf := linesPool.Get().([]byte)

                n, err := r.Read(buf)
                buf = buf[:n]

                if n == 0 {
                        if err != nil {
                                fmt.Println(err)
                                break
                        }
                        if err == io.EOF {
                                break
                        }
                        return err
                }

                nextUntillNewline, err := r.ReadBytes('\n')

                if err != io.EOF {
                        buf = append(buf, nextUntillNewline...)
                }

                wg.Add(1)
                go func() {
                        ProcessChunk(buf, &linesPool, &stringPool, start, end)
                        wg.Done()
                }()

        }

        wg.Wait()
        return nil
}

func ProcessChunk(chunk []byte, linesPool *sync.Pool, stringPool *sync.Pool, start time.Time, end time.Time) {

        var wg2 sync.WaitGroup

        logs := stringPool.Get().(string)
        logs = string(chunk)

        linesPool.Put(chunk)
```

```
	logsSlice := strings.Split(logs, "\n")

	stringPool.Put(logs)

	chunkSize := 300
	n := len(logsSlice)
	noOfThread := n / chunkSize

	if n%chunkSize != 0 {
		noOfThread++
	}

	for i := 0; i < (noOfThread); i++ {

		wg2.Add(1)
		go func(s int, e int) {
			defer wg2.Done() //to avaoid deadlocks
			for i := s; i < e; i++ {
				text := logsSlice[i]
				if len(text) == 0 {
					continue
				}
				logSlice := strings.SplitN(text, ",", 2)
				logCreationTimeString := logSlice[0]

				logCreationTime, err := time.Parse("2006-01-02T15:04:05.0000Z", logCreationTimeString)
				if err != nil {
					fmt.Printf("\n Could not able to parse the time :%s for log : %v", logCreationTimeString, text)
					return
				}

				if logCreationTime.After(start) && logCreationTime.Before(end) {
					//fmt.Println(text)
				}
			}


		}(i*chunkSize, int(math.Min(float64((i+1)*chunkSize), float64(len(logsSlice)))))
	}

	wg2.Wait()
	logsSlice = nil
}
```