# Source Code:

### Main.go

package main

import (

"bufio"

"consumer"

"encoding/csv"

"fmt"

"io"

"log"

"math"

"os"

"producer"

"stats"

"strconv"

"strings"

"time"

)

// Input Variables

var (

M float64 // number of times to repeat the tests (avg)

TICKS float64 // length of the test

TICK\_time float64 // 1 TICK = X milliseconds

lambda float64 // Average number of packets generated /arrived (packets per second)

L float64 // Length of a packet in bits

C float64 // The service time received by a packet. (Example: The transmission rate of the output link in bits per second.)

K float64 // The buffer size, 0 for infinite size

//Averages over M

Avg\_Avg\_packets stats.Avg

Avg\_Avg\_sojourn stats.Avg

Avg\_Proportion\_idle stats.Avg

Avg\_Probability\_loss stats.Avg

Avg\_total\_packets stats.Avg

Avg\_elapsed\_time stats.Avg

csv\_cols int = 15

)

var logger \*log.Logger

var get\_int\_debug = false

/\*

func get\_int\_csv(r string, b \*int) {

\_, errI := fmt.Sscan(r, b)

if errI != nil {

logger.Fatalf("converted = %d\n%v\n", b, errI)

}

}\*/

func get\_float64\_csv(r string, b \*float64) {

\_, errI := fmt.Sscan(r, b)

if errI != nil {

logger.Fatalf("converted = %f\n%v\n", b, errI)

}

}

func get\_val(r \*bufio.Reader) string {

var val, err = r.ReadString('\n')

if err != nil {

logger.Fatalln("| err:", err)

}

var trimval = strings.TrimRight(val, "\n")

if get\_int\_debug {

logger.Println("Val =", val)

logger.Println("trimmed =", trimval)

}

return trimval

}

/\*func get\_int(r \*bufio.Reader, b \*int) {

var trimval = get\_val(r)

// valI, errI := strconv.ParseInt(trimval, 10, 32)

\_, errI := fmt.Sscan(trimval, b)

if errI != nil {

logger.Fatalf("converted = %d\n%v\n", b, errI)

}

}\*/

func get\_float64(r \*bufio.Reader, b \*float64) {

var trimval = get\_val(r)

// valI, errI := strconv.ParseInt(trimval, 10, 32)

\_, errI := fmt.Sscan(trimval, b)

if errI != nil {

logger.Fatalf("converted = %f\n%v\n", b, errI)

}

}

func main() {

//Header

fmt.Println("ECE 358 Project 1 - Written in GO (golang.org)")

fmt.Println("Submitted by:")

fmt.Println("\tJon Shahen \t(20334465)")

fmt.Println("\tKevin Carlton \t(20337152)")

fmt.Println("---\n")

// End of Header

// set up the logger to point to stdout

logger = log.New(os.Stdout, "[ECE358 P1] ", log.LstdFlags)

// Get Variables

if len(os.Args) == 2 {

get\_args\_params()

} else {

get\_user\_params()

}

// End of Get Variables

// Display Variables

fmt.Println("\nVariables being used:")

fmt.Println("\t M ", M)

fmt.Println("\t TICKS ", TICKS)

fmt.Println("\t TICK\_time ", TICK\_time, "milliseconds")

fmt.Println("\t lambda ", lambda)

fmt.Println("\t L ", L, "bits")

fmt.Println("\t C ", C, "bits/sec")

if K == 0 {

fmt.Println("\t K ", "Infinity")

} else {

fmt.Println("\t K ", K)

}

//End of display Variables

// Loop for average statistics

var test\_t time.Time

var test\_b time.Time = time.Now()

var tsince time.Duration

for m := 1.0; m <= M; m++ {

test\_t = time.Now()

fmt.Println("-------\n\n")

wait\_tick := get\_tick\_wait()

qm := consumer.Init(logger, wait\_tick)

qm.MaxSize = K

producer.Init(logger, qm, lambda, TICK\_time)

stats.Init(logger)

logger.Println("---")

for t := 1.0; t < TICKS; t++ {

producer.Tick(t)

consumer.Tick(t)

// Getting the average packets in the queue

stats.Avg\_packets.AddAvg(float64(qm.Size))

}

logger.Println("Test Finished\n")

tsince = time.Since(test\_t)

fmt.Println("[Stats] Elapsed Time =\t", tsince)

fmt.Println("[Stats] End Queue Size =\t", qm.Size)

fmt.Println("[Stats] Average Packets in Queue (E[N]) =\t", stats.Avg\_packets.GetAvg())

fmt.Println("[Stats] Average Sojourn Time (E[T]) (TICKS) =\t", stats.Avg\_sojourn.GetAvg())

fmt.Println("[Stats] Probability Packet Loss (P\_LOSS) =\t", stats.Probability\_loss.GetProportion())

fmt.Println("[Stats] Proportion Server Idle (P\_IDLE) =\t", stats.Proportion\_idle.GetProportion())

fmt.Println("[Stats] Total Packets Produced =\t", stats.Probability\_loss.Total)

fmt.Println("[Stats] Total Packets Consumed =\t", stats.Avg\_sojourn.Num)

fmt.Println("[Stats] Time Simulated (s) =\t", TICKS\*TICK\_time/1000)

// Averages over M

Avg\_Avg\_packets.AddAvg(stats.Avg\_packets.GetAvg())

Avg\_Avg\_sojourn.AddAvg(stats.Avg\_sojourn.GetAvg())

Avg\_Probability\_loss.AddAvg(stats.Probability\_loss.GetProportion())

Avg\_Proportion\_idle.AddAvg(stats.Proportion\_idle.GetProportion())

Avg\_total\_packets.AddAvg(stats.Probability\_loss.Total)

Avg\_elapsed\_time.AddAvg(tsince.Seconds()) //in seconds

}

tsince = time.Since(test\_b)

fmt.Println("\n---\n")

fmt.Println("[Stats] Elapsed Time =\t", tsince)

fmt.Printf("\n\tAverages over M (%d)\n", int(M))

fmt.Println("[Stats] Avg Average Packets in Queue (E[N]) =\t", Avg\_Avg\_packets.GetAvg())

fmt.Println("[Stats] Avg Average Sojourn Time (E[T]) (TICKS) =\t", Avg\_Avg\_sojourn.GetAvg())

fmt.Println("[Stats] Avg Probability Packet Loss (P\_LOSS) =\t", Avg\_Probability\_loss.GetAvg())

fmt.Println("[Stats] Avg Proportion Server Idle (P\_IDLE) =\t", Avg\_Proportion\_idle.GetAvg())

fmt.Println("[Stats] Avg Total Packets Produced =\t", Avg\_total\_packets.GetAvg())

fmt.Println("[Stats] Avg Total Packets Consumed =\t", Avg\_Avg\_sojourn.GetAvg())

fmt.Println("[Stats] Avg Time Elapsed (s) =\t", Avg\_elapsed\_time.GetAvg())

write\_csv\_header()

write\_csv\_output(tsince.Seconds())

}

func get\_tick\_wait() float64 {

return float64(math.Ceil(((float64(L) / C) \* 1000) / float64(TICK\_time)))

}

func get\_args\_params() {

file, err := os.Open(os.Args[1])

if err != nil {

logger.Fatalf("Error:", err)

}

defer file.Close()

reader := csv.NewReader(file)

rec, err := reader.Read()

if err == io.EOF {

logger.Fatalf("Error: No Headers")

} else if err != nil {

logger.Fatalf("Error:", err)

}

// throwaway header

rec, err = reader.Read()

if err == io.EOF {

logger.Fatalf("Error: No Data")

} else if err != nil {

logger.Fatalf("Error:", err)

}

get\_float64\_csv(rec[0], &M)

get\_float64\_csv(rec[1], &TICKS)

get\_float64\_csv(rec[2], &TICK\_time)

get\_float64\_csv(rec[3], &lambda)

get\_float64\_csv(rec[4], &L)

get\_float64\_csv(rec[5], &C)

get\_float64\_csv(rec[6], &K)

}

func get\_user\_params() {

var stdinR = bufio.NewReader(os.Stdin)

fmt.Printf("M: ")

get\_float64(stdinR, &M)

fmt.Printf("TICKS: ")

get\_float64(stdinR, &TICKS)

fmt.Printf("TICK Time (1 TICK = X milliseconds): ")

get\_float64(stdinR, &TICK\_time)

fmt.Printf("lambda: ")

get\_float64(stdinR, &lambda)

fmt.Printf("L (bits): ")

get\_float64(stdinR, &L)

fmt.Printf("C (bits per sec): ")

get\_float64(stdinR, &C)

fmt.Printf("K (zero = infinity): ")

get\_float64(stdinR, &K)

}

func write\_csv\_output(t\_since float64) {

file, err := os.OpenFile("test\_out.csv", os.O\_RDWR|os.O\_APPEND, 0660)

if err != nil {

logger.Fatalf("Error in opening write file:", err)

}

defer file.Close()

writter := csv.NewWriter(file)

rec := make([]string, csv\_cols)

var i = 0

rec[i] = strconv.FormatFloat(M, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(TICKS, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(TICK\_time, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(lambda, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(L, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(C, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(K, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(get\_tick\_wait(), 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_total\_packets.GetAvg(), 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_Avg\_sojourn.GetAvg(), 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_Probability\_loss.GetAvg()\*TICK\_time/1000, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_Proportion\_idle.GetAvg()\*TICK\_time/1000, 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_Avg\_packets.GetAvg(), 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(Avg\_elapsed\_time.GetAvg(), 'f', -1, 64)

i++

rec[i] = strconv.FormatFloat(t\_since, 'f', -1, 64)

i++

writter.Write(rec)

writter.Flush()

}

func write\_csv\_header() {

\_, err := os.Open("test\_out.csv")

if os.IsNotExist(err) {

file, \_ := os.Create("test\_out.csv")

writter := csv.NewWriter(file)

var i = 0

headers := make([]string, csv\_cols)

headers[i] = "M"

i++

headers[i] = "TICKS"

i++

headers[i] = "TICK Time (sec)"

i++

headers[i] = "lambda"

i++

headers[i] = "L : Packet length (bits)"

i++

headers[i] = "C = bits per secound"

i++

headers[i] = "K: Buffer capacity (zero = inf)"

i++

headers[i] = "Service Time (Ticks)"

i++

headers[i] = "E[N] Avg num of packets in queue"

i++

headers[i] = "E[T] Avg sojourn time (sec)"

i++

headers[i] = "P\_Loss - packet loss"

i++

headers[i] = "P\_Idle - server idle (sec)"

i++

headers[i] = "Total Packets"

i++

headers[i] = "Average Time (sec)"

i++

headers[i] = "Total Time (sec)"

i++

writter.Write(headers)

writter.Flush()

file.Close()

}

}

### Queue.go

package stats

import (

// "log"

"fmt"

)

type QueueMgr struct {

Size float64 // the current size of the queue

MaxSize float64 // the maximum size of the queue, 0 for infinite

Head \*QueueItem // the head of the queue

Tail \*QueueItem // the tail of the queue

}

type QueueItem struct {

Next \*QueueItem

Data Packet

}

type Packet struct {

Generated float64

Finished float64

}

func (qm \*QueueMgr) Push(item Packet) error {

var qItem = QueueItem{nil, item}

if qm.Size == 0 {

qm.Head = &qItem

qm.Tail = &qItem

qm.Size++

} else if qm.MaxSize == 0 || qm.Size < qm.MaxSize {

qm.Tail.Next = &qItem

qm.Tail = &qItem

qm.Size++

} else {

//add one to the packet loss

Probability\_loss.AddOne()

return fmt.Errorf("Queue is full, max size %d packets", qm.MaxSize)

}

//successfully put packet in queue

Probability\_loss.Total++

return nil

}

func (qm \*QueueMgr) Pop() (Packet, error) {

var item Packet

if qm.Size != 0 {

item = qm.Head.Data

qm.Head = qm.Head.Next

qm.Size--

return item, nil

}

return item, fmt.Errorf("Queue is full, max size %d packets", qm.MaxSize)

}

func (qm \*QueueMgr) Clear() {

qm.Size = 0

qm.Head = nil

qm.Tail = nil

}

### Stats.go

package stats

import (

"log"

)

type Avg struct {

Total float64

Num float64

}

type Pro struct {

Less float64

Total float64

}

var (

logger \*log.Logger

// Stats

Avg\_packets Avg // E[N]

Avg\_sojourn Avg // E[T]

Proportion\_idle Pro // P\_IDLE

Probability\_loss Pro // P\_LOSS

)

func Init(l \*log.Logger) {

logger = l

logger.Println("[Stats] Started")

//clear values

Avg\_packets.Clear()

Avg\_sojourn.Clear()

Proportion\_idle.Clear()

Probability\_loss.Clear()

}

// Helper functions

func (a Avg) GetAvg() float64 {

if a.Num == 0 {

return 0

}

return a.Total / a.Num

}

func (a \*Avg) AddAvg(add float64) {

a.Total += add

a.Num++

}

func (a \*Avg) Clear() {

a.Total = 0

a.Num = 0

}

func (p Pro) GetProportion() float64 {

if p.Total == 0 {

return 0

}

return p.Less / p.Total

}

func (p \*Pro) AddOne() {

p.Total++

p.Less++

}

func (p \*Pro) Clear() {

p.Total = 0

p.Less = 0

}

func (p Packet) SojournTime() float64 {

return p.Finished - p.Generated

}

### Producer.go

package producer

import (

"log"

"math"

"math/rand"

"stats"

"time"

)

var (

logger \*log.Logger

nextTick float64

lambda float64

qm \*stats.QueueMgr

time\_2\_tick float64 //1 tick = X milliseconds

)

func Init(l \*log.Logger, \_qm \*stats.QueueMgr, \_lambda float64, TICK\_time float64) {

logger = l

lambda = \_lambda

qm = \_qm

time\_2\_tick = TICK\_time

nextTick = 1

logger.Println("[Producer] Started")

rand.Seed(time.Now().UnixNano())

}

func Tick(t float64) {

if t == nextTick {

producePacket(t)

nextTick = t + getExpRandNum(lambda)

//logger.Println("[Producer] Next Tick", nextTick)

}

}

var debug\_getExpRandNum = false

func getExpRandNum(l float64) float64 {

a := 1 - rand.Float64()

b := math.Log(a)

c := (-1 / l)

d := b \* c \* 1000 // in milliseconds

ans := d / float64(time\_2\_tick)

if debug\_getExpRandNum {

logger.Println("b", b, "\nc", c, "\nd", d, "\nans", ans, "\n")

}

return math.Ceil(ans)

}

func producePacket(i float64) {

var p = stats.Packet{i, 0}

if err := qm.Push(p); err != nil {

//logger.Printf("[Producer] Received Error %v\n", err)

}

//logger.Println("[Producer] Packet produced")

}

### Consumer.go

package consumer

import (

"log"

"stats"

)

var logger \*log.Logger

var (

next\_tick float64

wait\_time\_tick float64 // in milliseconds

queue stats.QueueMgr

)

func Init(l \*log.Logger, wait\_tick float64) \*stats.QueueMgr {

logger = l

logger.Println("[Consumer] Started")

wait\_time\_tick = wait\_tick

next\_tick = 1

logger.Println("[Consumer] Service time is", wait\_time\_tick, "ticks")

queue.Clear()

return &queue

}

func Tick(t float64) {

if next\_tick <= t {

consume\_packet(t)

}

}

func consume\_packet(t float64) {

//logger.Println("[Consumer] Consuming Packet for", wait\_time\_tick, "ticks")

if queue.Size > 0 {

var p stats.Packet

p, err := queue.Pop()

if err != nil {

logger.Fatalf("Recieved Error From Pop: %v", err)

}

p.Finished = t

stats.Avg\_sojourn.AddAvg(p.SojournTime())

next\_tick = t + wait\_time\_tick

stats.Proportion\_idle.Total++

} else {

stats.Proportion\_idle.AddOne()

}

}