**עבודת גמר-שפת סי מתקדמים 15/07/20**

**מגישים:**

**דוד מוסייב 319455119**

**שקד ספקטור 308132281**

**Main.c**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*switch-emulator\*\*Exam-Project\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*author1:David Musaev

/\*Version 1.0 date:15/07/2020\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*The program simulate a switch that recive a packet file to each port ,while in this

/\*version there are 4 in ports, after the switch emulator will read the 4 input ports

/\*it will build a routing table and will deside in which port each packet from input

/\*ports will route to. each packet have a QOS element /priority element that will

/\*decide in which queue to put the packet in the output port.

/\*In addition the the emulator will check each packet if the packet is not corputed

, it will happend threw XOR on all packet element except the checkSum frame.

after that the emulator will send all the packets by the Time Line (TIME FIFO) and by

the QOS/priority elemt when the first to out will be these with

priority 0 and X time after that priority 1 with X time, the product will be exported to files Port i.out .

the emulator can be run from command line by the command EXAMPLE:C:\Users\David\source\repos\switch\Debug\switch.exe route.txt port1.in port2.in port3.in port4.in

while

switch.exe - name of the EXE file

route.txt - logic of the routing table - 1st column:a= add ,d =delete , 2nd column:da elemnt/frame ,3rd column(exist only while cloumn 1==a) output port

porti.in- i=1 or 2 or 3 or 4 the input file that simulate packet in the in ports

each packet has frame of [Time][Source Adress][Destination Adress][Priority][Data Length][Data(size of Data Length)][checksum]

after each use the emulator will free all the memory that was heap located for the emulator

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#define TEST\_TIME 3000

#ifndef STRUCTERS\_H

#include "stucters.h"

#endif // !STRUCTERS\_H

#ifndef FUNCTIONS\_H

#include "functions.h"

#endif // !FUNCTIONS\_H

#ifndef STDIO\_H

#include <stdio.h>

#endif // !STDIO\_H

#ifndef STDLIB\_H

#include <stdlib.h>

#endif // !STDLIB\_H

#ifndef STRING\_H

#include <string.h>

#endif // !STRING\_H

/\*data types, structs and enums\*/

/\*functions decelerations\*/

void packet\_read(FILE \*fp,packet \*pkt);//read a line from a file portX.in

void packet\_write(FILE \*fp, packet \*pkt);//write a line from a file portX.in

Bool checksum\_check(const packet \*pkt);//checking xor on all particals of the packet and compare it to checksum field if even so the value will true else false

S\_node \* add\_route(S\_node \*root, char da, char output\_port);//add a node to a tree for route table

S\_node \*delete\_route(S\_node \*root, char da);//delete a node from a tree

void freeNode(S\_node \*node);//Null arguments of the node and free the heap memory

S\_node \*search\_route(const S\_node \*root, char da);//search a node in binary tree by DA verfication

void print\_routing\_table(const S\_node \*root);//print a binary tree by in order

S\_node \* build\_route\_table(FILE \*fp, S\_node \*root);//build route table matched to binary tree from route file

void freePacket(packet \*pkt);

void freePacket\_withoutmem(packet \* pkt);

//free memory of packet data type

void enque\_pkt(S\_Out\_Qs\_mgr \*QM\_ptr, packet \*pkt);//put a packet by priority in the last place in the queue

packet \*deque\_pkt(S\_Out\_Qs\_mgr \*QM\_ptr, char priority);

void copyPacket(const packet \* pkt, packet \* dstPkt);//copy packet structer

/\*main function\*/

int main(int argc, char \*argv[])

{

/\*file handle input argv[0]=filename , argv[1]= file argumnet 1 ,,,,\*/

printf("\n%s", argv[1]);

printf("\n%s", argv[2]);

printf("\n%s", argv[3]);

printf("\n%s", argv[4]);

printf("\n%s", argv[5]);

FILE \*route = fopen(argv[1], "r+");//open route.txt

FILE \*port1In = fopen(argv[2], "r+");//open port1.in

FILE \*port2In = fopen(argv[3], "r+");//open port2.in

FILE \*port3In = fopen(argv[4], "r+");//open port3.in

FILE \*port4In = fopen(argv[5], "r+");//open port4.in

FILE \*port1Out = fopen("port1.out", "w+");//open port1.in

FILE \*port2Out = fopen("port2.out", "w+");//open port2.in

FILE \*port3Out = fopen("port3.out", "w+");//open port3.in

FILE \*port4Out = fopen("port4.out", "w+");//open port4.in

/\*building routing table\*/

S\_node \*rootTree=NULL;

rootTree=build\_route\_table(route, rootTree);

printf("\nbuilding the tree");

printf("\nprinted the tree");

print\_routing\_table(rootTree);

/\*input and logic\*/

/\*array of packets that holding the information\*/

packet \*infoIn[1000];

/\*inputs for the ports that reciving info from port1.in port2.in etc..\*/

infoIn[0]= (packet\*)calloc(1, sizeof(packet));

infoIn[1] = (packet\*)calloc(1, sizeof(packet));

infoIn[2] = (packet\*)calloc(1, sizeof(packet));

infoIn[3] = (packet\*)calloc(1, sizeof(packet));

S\_node \*Route\_P\_IN[4];

S\_Out\_Qs\_mgr \*QueuePo0=(Out\_Qs\_mgr\*)calloc(1, sizeof(S\_Out\_Qs\_mgr)), \*QueuePo1 = (Out\_Qs\_mgr\*)calloc(1, sizeof(S\_Out\_Qs\_mgr)), \*QueuePo2 = (Out\_Qs\_mgr\*)calloc(1, sizeof(S\_Out\_Qs\_mgr)), \*QueuePo3 = (Out\_Qs\_mgr\*)calloc(1, sizeof(S\_Out\_Qs\_mgr));

int i = 0, k = -1,j=4;

while (port1In || port2In || port3In || port4In)

{

/\* initilization for time check\*/

/\*the check is a competition between 4 in ports the one who was sent earlier will get in the queue first,

only the one port that sent the packet is refreshing and now have a new packet to put in the time competition\*/

infoIn[j] = (packet\*)calloc(1, sizeof(packet));

infoIn[j]->time = 10000000000;//max value

//read 4 ports k=-1 is initial state and k=0 ,1,2,3 is the matched number of port -1

//if the file of ports is at the end of file put max time(10000000000) for elimnate these ports from check

if (port1In && (k==0 || k==-1))

{

if (feof(port1In))

{

port1In = NULL;

infoIn[0]->time = 10000000000;

}

else

packet\_read(port1In, infoIn[0]);

}

if (port2In && (k == 1 || k == -1))

{

if (feof(port2In))

{

port2In = NULL;

infoIn[1]->time = 10000000000;

}

else

packet\_read(port2In, infoIn[1]);

}

if (port3In && (k == 2 || k == -1))

{

if (feof(port3In))

{

port3In = NULL;

infoIn[2]->time = 10000000000;

}

else

packet\_read(port3In, infoIn[2]);

}

if (port4In && (k == 3 || k == -1))

{

if (feof(port4In))

{

port4In = NULL;

infoIn[3]->time = 10000000000;

}

else

packet\_read(port4In, infoIn[3]);

}

if (k == -1)//initial state for creating first obj to compare it

{

copyPacket(infoIn[0], infoIn[j]);//first time only

k = 0;

}

/\*find in which time the packet was sent\*/

//compare time and save the index with minimal time bettween the 4 cells and keep it on the 4th index

for (i = 0; i < 4; i++)

{

if (infoIn[j]->time > infoIn[i]->time)

{

copyPacket(infoIn[i], infoIn[j]);//copy save the index that matched to infoin array

k = i;

}

}

Route\_P\_IN[0] =search\_route(rootTree, infoIn[j]->da) ;//find its route to out port number

/\*PUT IN QUEUE 1,2,3,4 and priority 0,1\*/

if(Route\_P\_IN[0]!=NULL)

{

switch (Route\_P\_IN[0]->output\_port)

{

case 1://port out 1

enque\_pkt(QueuePo0, infoIn[j]);//enque to the matched priority queue

if (infoIn[j]->da != infoIn[k]->da )

{

j--;

}

j++;

break;

case 2://port out 2

enque\_pkt(QueuePo1, infoIn[j]);

if (infoIn[j]->da != infoIn[k]->da)

{

j--;

}

j++;

break;

case 3://port out 3

enque\_pkt(QueuePo2, infoIn[j]);

if (infoIn[j]->da != infoIn[k]->da)

{

j--;

}

j++;

break;

case 4://port out 4

enque\_pkt(QueuePo3, infoIn[j]);

if (infoIn[j]->da != infoIn[k]->da)

{

j--;

}

j++;

default:

break;

}

}

}

/\*OUT PUT FILES\*/

i = 0;

while (i<TEST\_TIME)//test time can be modify in head of the code

{

i++;

//export priority 0 queue in time=i

if (QueuePo0->head\_p0 && i == QueuePo0->head\_p0->pkt->time)

{

packet\_write(port1Out, deque\_pkt(QueuePo0, 0));

}

if (QueuePo1->head\_p0 && i == QueuePo1->head\_p0->pkt->time)

{

packet\_write(port2Out, deque\_pkt(QueuePo1, 0));

}

if (QueuePo2->head\_p0 && i == QueuePo2->head\_p0->pkt->time)

{

packet\_write(port3Out, deque\_pkt(QueuePo2, 0));

}

if (QueuePo3->head\_p0 && i == QueuePo3->head\_p0->pkt->time)

{

packet\_write(port4Out, deque\_pkt(QueuePo3, 0));

}

////export priority 1 queue in time=i

if (QueuePo0->head\_p1 && i == QueuePo0->head\_p1->pkt->time)

{

packet\_write(port1Out, deque\_pkt(QueuePo0, 1));

}

if (QueuePo1->head\_p1 && i == QueuePo1->head\_p1->pkt->time)

{

packet\_write(port2Out, deque\_pkt(QueuePo1, 1));

}

if (QueuePo2->head\_p1 && i == QueuePo2->head\_p1->pkt->time)

{

packet\_write(port3Out, deque\_pkt(QueuePo2, 1));

}

if (QueuePo3->head\_p1 && i == QueuePo3->head\_p1->pkt->time)

{

packet\_write(port4Out, deque\_pkt(QueuePo3, 1));

}

}

//save the files

fclose(port1Out);

fclose(port2Out);

fclose(port3Out);

fclose(port4Out);

/\*memory free\*/

for (i = 0; i < 1000; i++)

{

if ((infoIn[i]->time > TEST\_TIME) && (i > 5))

break;

freePacket(infoIn[i]);

}

freePacket(infoIn[i]);

free(QueuePo0);

free(QueuePo1);

free(QueuePo2);

free(QueuePo3);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Function.h**

#pragma once

#ifndef STDIO\_H

#include <stdio.h>

#endif // !STDIO\_H

#ifndef STDLIB\_H

#include <stdlib.h>

#endif // !STDLIB\_H

#ifndef STRING\_H

#include <string.h>

#endif // !STRING\_H

#ifndef STRUCTERS\_H

#include "stucters.h"

#endif // !STRUCTERS\_H

/\*function definitions\*/

void freeNode(S\_node \*node);//Null arguments of the node and free the heap memory

void packet\_read(FILE \*fp, packet \*pkt)

{

FILE \*thisFile = fp;//file to read

packet \*thisPacket = pkt;//packet to write in

char tmp\_DataHandle[100];

fscanf(thisFile, "%u %d %d %d %d %[^\n]%\*c", &thisPacket->time, &thisPacket->da, &thisPacket->sa, &thisPacket->prio, &thisPacket->data\_length, tmp\_DataHandle);

thisPacket->data = (char\*)calloc(thisPacket->data\_length, sizeof(char));//exactly memory allocation by the number of the object in the data

if (feof(fp))

{

return;

}

/\*split the tmp\_dataHandle by spaces to data and checksum\*/

/\*some variables for simplicty\*/

char \*thisData = thisPacket->data;

/\*define token for strtok to split the tmp\_dataHandle\*/

char \*token = strtok(tmp\_DataHandle, " ");

// loop through the string to extract all data and checsums from tokens

int i = 0;

for (i; i < thisPacket->data\_length + 1; i++)

{

char tmp[20];//tmpory array for handling the token

strcpy(tmp, token);//copy the token into tmp

thisData[i] = atoi(tmp);//convert tmp into int and store as a charcter

if (i == thisPacket->data\_length)//in the last i

{

char checksum[20];//define another tmp array for handling token for the checksum

strcpy(checksum, token);//copy the token into cheksum

thisPacket->checksum = atoi(checksum);//convert tmp into int and store as a charcter

thisData[i] = '\0';//close the string of thisData

}

token = strtok(NULL, (" "));//move foward to next space the token

}

}

void packet\_write(FILE \* fp, packet \* pkt)

{

FILE \*thisFile = fp;//file to out

packet \*thisPacket = pkt;//packet to write

/\*print into file until the data\*/

fprintf(thisFile, "%u %d %d %d %d", thisPacket->time, thisPacket->da, thisPacket->sa, thisPacket->prio, thisPacket->data\_length);

/\*print the data as integers\*/

int i = 0;

for (i; i < thisPacket->data\_length; i++)

{

fprintf(thisFile, " %d", thisPacket->data[i]);

}

/\*print the check sum\*/

fprintf(thisFile, " %d\n", thisPacket->checksum);

}

Bool checksum\_check(const packet \* pkt)

{

const Packet \*thisPacket = pkt;//simplicity

int checkSum\_tmp = thisPacket->da ^ thisPacket->sa ^ thisPacket->prio ^thisPacket->data\_length;//xor between the number

int i = 0;

for (i; i < thisPacket->data\_length; i++)

{

checkSum\_tmp ^= thisPacket->data[i];//keep xor

}

if (checkSum\_tmp == (int)(thisPacket->checksum))//if true

{

return True;

}

else

{

return False;

}

}

void freePacket(packet \*pkt)

{

/\*Null value and free\*/

pkt->checksum = NULL;

pkt->da = NULL;

pkt->sa = NULL;

pkt->prio = NULL;

int i = 0;

for (i; i < pkt->data\_length; i++)

{

pkt->data[i] = NULL;

}

pkt->data\_length = NULL;

pkt->time = NULL;

free(pkt);

}

void freePacket\_withoutmem(packet \*pkt)

{

/\*Null value and free\*/

pkt->checksum = NULL;

pkt->da = NULL;

pkt->sa = NULL;

pkt->prio = NULL;

int i = 0;

for (i; i < pkt->data\_length; i++)

{

pkt->data[i] = NULL;

}

pkt->data\_length = NULL;

pkt->time = NULL;

//free(pkt);

}

S\_node \* add\_route(S\_node \* root, char da, char output\_port)

{

/\*simplicty\*/

S\_node \*thisRoot = root;

unsigned char thisDa = (unsigned char)da;

unsigned char thisOutPort = (unsigned char)output\_port;

/\*if this tree root is empty\*/

if (thisRoot == NULL)

{

thisRoot = (S\_node\*)calloc(1, sizeof(S\_node));

thisRoot->da = (unsigned char)thisDa;

thisRoot->output\_port = (unsigned char)thisOutPort;

return thisRoot;

}

/\*if the tree isnt empty\*/

/\*if its the same as da in the root\*/

if (thisRoot->da == thisDa)

{

return thisRoot;

}

/\*if the da is smaller then the root \*/

if (thisRoot->da > thisDa)

{

thisRoot->left = add\_route(thisRoot->left, thisDa, thisOutPort);//recourse to left

return thisRoot;

}

/\*if the da is larger then the root \*/

if (thisRoot->da < thisDa)

{

thisRoot->right = add\_route(thisRoot->right, thisDa, thisOutPort);//recourse to right

return thisRoot;

}

}

S\_node \* delete\_route(S\_node \* root, char da)

{

/\*simplicty\*/

S\_node \*thisRoot = root;

unsigned char thisDa = (unsigned char)da;

/\*if the tree is empty\*/

if (thisRoot == NULL)

return NULL;

/\*if the tree isnt empty\*/

/\*if its the same as da in the root\*/

if (thisRoot->da == thisDa)

{

S\_node \*thisLeft = thisRoot->left;//holding the point of left

S\_node \*thisRight = thisRoot->right;//holding the pont of right

if (thisLeft != NULL) //only if the left child exist

{

/\*if left dont have a right child \*/

if (thisLeft->right == NULL)

thisLeft->right = thisRight;

/\*if left have right child\*/

else

{

while (thisLeft->right != NULL)

{

thisLeft = thisLeft->right;//go until the last child in the tree of the right brench

}

thisLeft->right = thisRight;//put the right child in the end of the right brench

thisLeft = thisRoot->left;

}

/\*free memory\*/

freeNode(root);

return thisLeft;

}

else

{

if (thisRight != NULL)//only if the right child exist

{

/\*free memory\*/

freeNode(root);

return thisRight;

}

root->da = NULL;

root->output\_port = NULL;

free(root);

return NULL;//if there are no childrens

}

}

/\*if the da is smaller then the root \*/

if (thisRoot->da > thisDa)

{

thisRoot->left = delete\_route(thisRoot->left, thisDa);

return thisRoot;

}

/\*if the da is larger then the root \*/

if (thisRoot->da < thisDa)

{

thisRoot->right = delete\_route(thisRoot->right, thisDa);

return thisRoot;

}

}

void freeNode(S\_node \* node)

{

node->da = NULL;

node->output\_port = NULL;

node->right = NULL;

node->left = NULL;

free(node);

}

S\_node \* search\_route(const S\_node \* root, char da)

{

/\*simplicty\*/

S\_node \*thisRoot = (S\_node\*)root;

unsigned char thisDa = (unsigned char)da;

/\*if this tree root is empty\*/

if (thisRoot == NULL)

{

return NULL;

}

/\*if its the same as da in the root\*/

if (thisRoot->da == thisDa)

{

return (S\_node\*)thisRoot;

}

/\*if the da is smaller then the root \*/

if (thisRoot->da > thisDa)

{

return search\_route(thisRoot->left, thisDa);

}

/\*if the da is larger then the root \*/

if (thisRoot->da < thisDa)

{

return search\_route(thisRoot->right, thisDa);

}

}

void print\_routing\_table(const S\_node \* root)

{

if (root == NULL)

{

printf("\n\t");

return;

}

/\*in order print Binary tree\*/

print\_routing\_table(root->left);

printf("DA: %d ,Port\_out:%d", root->da, root->output\_port);

print\_routing\_table(root->right);

}

S\_node \* build\_route\_table(FILE \* fp, S\_node \* root)

{

/\*simplicty\*/

FILE \*thisFile = fp;

S\_node \* thisRoot = root;

thisRoot = (S\_node\*)calloc(1, sizeof(S\_node));

unsigned char lineHandle[6];

int i = 0, num\_lines = 0;

/\*add objects in the tree\*/

while (!feof(thisFile))

{

fscanf(thisFile, " %c %u %u", &lineHandle[0], &lineHandle[1], &lineHandle[2]);

num\_lines++;

if (lineHandle[0] == 'a')

{

thisRoot = add\_route(thisRoot, lineHandle[1], lineHandle[2]);

}

if (lineHandle[0] == 'd')

{

thisRoot = delete\_route(thisRoot, lineHandle[1]);

}

}

/\*relocate and check the root of the tree\*/

for (i; i < num\_lines; i++)

{

lineHandle[0] = thisRoot->da;//keep the information of the root

lineHandle[1] = thisRoot->output\_port;

thisRoot = delete\_route(thisRoot, thisRoot->da);//remove the root

thisRoot = add\_route(thisRoot, lineHandle[0], lineHandle[1]);//return the root to its matched position in the tree

}

return thisRoot;

}

void enque\_pkt(S\_Out\_Qs\_mgr \* QM\_ptr, packet \* pkt)

{

/\*simplicty\*/

S\_Out\_Qs\_mgr \*thisQueuMgr = QM\_ptr;

packet \*thisPacket = pkt;

/\*build the queue\*/

if (thisPacket->prio == 0)//classify by the priority 0

{

if (checksum\_check(thisPacket) == True)//if the checksum is TRUE

{

//check if the queue is mngr exist but emptyof nodes

if (!thisQueuMgr->head\_p0)

{

pkt\_node \*thisQ0 = (pkt\_node\*)calloc(1, sizeof(pkt\_node));//create the first node

thisQ0->pkt = thisPacket;

thisQ0->next = NULL;

thisQueuMgr->head\_p0 = thisQ0;//set head this q0

thisQueuMgr->tail\_p0 = thisQ0;//set tail this q0

thisQueuMgr->tail\_p0->next = NULL;

}

else//not empty of nodes

{

pkt\_node \*thisQ0 = (pkt\_node\*)calloc(1, sizeof(pkt\_node));//create the new node

//add to end of the queue

thisQ0->pkt = thisPacket;//put this packet value in the queue1

thisQ0->next = NULL;

thisQueuMgr->tail\_p0->next = thisQ0;//connect thisQ0 to the tail

thisQueuMgr->tail\_p0 = thisQ0;//set tail

}

}

else//if the CHECKSUM IS FALSE

{

freePacket(thisPacket);//free memory function

return;

}

}

if (thisPacket->prio == 1)//classify by the priority 1

{

if (checksum\_check(thisPacket) == True)//if the checksum is TRUE

{

//check if the queue is mngr exist but emptyof nodes

if (!thisQueuMgr->head\_p1)

{

pkt\_node \*thisQ1 = (pkt\_node\*)calloc(1, sizeof(pkt\_node));//create the first node

thisQ1->pkt = thisPacket;

thisQ1->next = NULL;

thisQueuMgr->head\_p1 = thisQ1;//set head this q1

thisQueuMgr->tail\_p1 = thisQ1;//set tail this q1

thisQueuMgr->tail\_p1->next = NULL;

}

else//not empty of nodes

{

pkt\_node \*thisQ1 = (pkt\_node\*)calloc(1, sizeof(pkt\_node));//create the new node

//add to end of the queue

thisQ1->pkt = thisPacket;//put this packet value in the queue1

thisQ1->next = NULL;

thisQueuMgr->tail\_p1->next = thisQ1;//connect thisQ1 to the tail

thisQueuMgr->tail\_p1 = thisQ1;//set tail

}

}

else//if the CHECKSUM IS FALSE

{

freePacket(thisPacket);//free memory function

return;

}

}

}

packet \* deque\_pkt(S\_Out\_Qs\_mgr \* QM\_ptr, char priority)

{

/\*simplicty\*/

S\_Out\_Qs\_mgr \*thisQueuMgr = QM\_ptr;

char thisPriority = priority;

packet \*tmpPacket;

if (thisPriority == 0)//which priority queue to deque, prio=0

{

if (thisQueuMgr->head\_p0 == thisQueuMgr->tail\_p0->next)//if the head is next of the tail its as the same the queue is empty

return NULL;

tmpPacket = thisQueuMgr->head\_p0->pkt;//release the first to return

thisQueuMgr->head\_p0 = thisQueuMgr->head\_p0->next;//set head as the next one in the list

return tmpPacket;

}

if (thisPriority == 1)//which priority queue to deque, prio=1

{

if (thisQueuMgr->head\_p1 == thisQueuMgr->tail\_p1->next)//if the head is next of the tail its as the same the queue is empty

return NULL;

tmpPacket = thisQueuMgr->head\_p1->pkt;//release the first to return

thisQueuMgr->head\_p1 = thisQueuMgr->head\_p1->next;//set head as the next one in the list

return tmpPacket;

/\*free() of these returned packets data types will be in the main function\*/

}

}

void copyPacket(const packet \* pkt, packet \*dstPkt)

{

dstPkt->time = pkt->time;

dstPkt->da = pkt->da;

dstPkt->sa = pkt->sa;

dstPkt->prio = pkt->prio;

dstPkt->data\_length = pkt->data\_length;

dstPkt->checksum = pkt->checksum;

dstPkt->data = (char\*)calloc(dstPkt->data\_length, sizeof(char));

char \*desData = dstPkt->data;

char \*souData = pkt->data;

int i = 0;

for (i; i < dstPkt->data\_length; i++)

{

desData[i] = souData[i];

}

desData[i] = '\0';

}

**Stucters.h**

#pragma once

enum Bool\_t { True, False };//boolean type true/false

typedef enum Bool\_t Bool;

typedef struct packet

{

unsigned int time;//time the packet in portX.in

unsigned char da;//Distation address, must converted from integer while recived from the portX.in

unsigned char sa;//Source address, must converted from integer while recived from the portX.in

unsigned char prio;//priority of the packet 0 is High 1 is Low

unsigned char data\_length;//length (posetive intger) of data frame in the packet

char \*data;//data array in the length of data\_length size, need mem allocation by the size of data\_length

unsigned char checksum;//validation tool of the packet , also xor between da^sa^prio^data\_length^{data0^data1^...^datan}

}Packet;//packets struct

typedef struct route\_node

{

unsigned char da;//Distation address,

unsigned char output\_port;//Distation address,

struct route\_node \*left;//left brench of the root

struct route\_node \*right;//right brench of the root

}S\_node;//node struct in binary tree

typedef struct pkt\_node //queue of packets

{

packet \*pkt;

struct pkt\_node \*next;

} S\_pkt;

typedef struct Out\_Qs\_mgr //queue of packets by priority field of packets struct

{

struct pkt\_node \*head\_p1;

struct pkt\_node \*tail\_p1;

struct pkt\_node \*head\_p0;

struct pkt\_node \*tail\_p0;

} S\_Out\_Qs\_mgr;