|  |  |  |  |
| --- | --- | --- | --- |
| رقم المقعد | الفصل | رقم الجلوس | اسم الطالب |
| **35** | **1** | **03533** | **أدهم محمد عيد عبدالسلام** |

Ordered Packet Delivery

# Overview

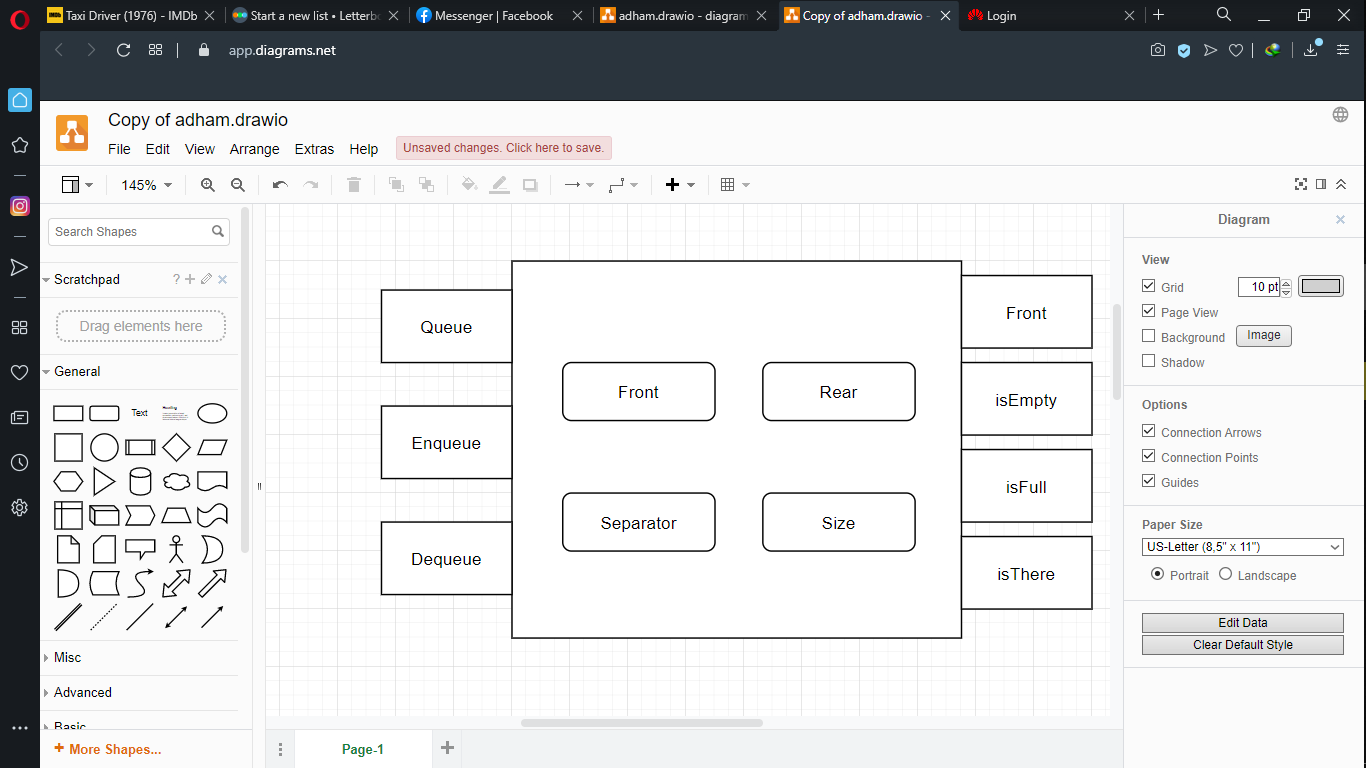


Figure 1 Queue

The only difference between this implementation and the usual circular queue is the enqueue function which we will discuss now. Enqueue function should be passed the value to be enqueued along with the next expected packet (initSeq). It uses this expected value to sort the wait queue depending on the priority of the coming sequence. This is done by sorting all values larger than the initSeq in ascending order at the top of the queue and beneath them the values less than the initSeq in ascending order as well. So now it’s obvious that we need some variable to hold the index of the first value that’s less than initSeq to be used to manipulate different cases of enqueuing which is the job of the separator and to indicate that the queue doesn’t contain any values less than initSeq separator should equal -1.

# Main Flows

## psedu-code of the main program

START;

check for error in arguments;

if (there is error in arguments)

{

print error messages;

END program;

}

else

{

save initSeq,winSize,seqNumBits from input arguments;

create W queue;

create R , D strings;

for(data=first\_packet;packets not finished;data=next\_packet)

{

if(data=initSeq)

{

Add data in R string; // RECIEVED

initSeq = (initSeq+1)%MAXSEQNU;

wait.expected = initSeq;

while(initSeq=wait.front())

{

dequeue first value from wait and add it in R string; // RECIEVED

initSeq = (initSeq+1)%MAXSEQNU;

wait.expected = initSeq;

}

continue;

}

determine window range of values;

if(data not in window range or repeated) // A packet is repeated if it's already in the wait queue

{

Add data in D string; // DROPPED

}

else

{

Enqueue data in Wait Queue;

}

}

Print R,E,W,D;

END program;

}

## Code Snippets

This part of the code is handling the packets after checking for errors and making sure there are none.

Queue wait(winSize+1); // creating the wait queue (winSize+1 because there is an empty element)

string R = "R ";

string D = "D ";

int minWin; // first value in window range

int maxWin; // last value in window range

int j; // iterator to be used later

bool in\_win\_range; // flag to tell if the packet is in the window range or not

for (int i = 4; i < argc; i++)

{

if (atoi(argv[i])==initSeq)

{

R = R + argv[i] +" ";

initSeq = (initSeq+1)%MAXSEQNU;

while ( (!(wait.isEmpty())) && (wait.front() == initSeq) )

{

R = R + to\_string(wait.dequeue()) + " ";

initSeq = (initSeq+1)%MAXSEQNU;

}

continue;

}

minWin = initSeq;

maxWin = ((minWin+winSize)%MAXSEQNU)-1;

if (maxWin == -1)

{

maxWin = MAXSEQNU-1; // to apply circularity

}

j = minWin;

in\_win\_range = false;

while(j!=((maxWin+1)%MAXSEQNU))

{

if (atoi(argv[i]) == j)

{

in\_win\_range = true;

break;

}

j = (j+1)%MAXSEQNU;

}

if ( (!in\_win\_range) || ((!(wait.isEmpty())) && (wait.isThere(atoi(argv[i])))) )

{

D = D + argv[i] +" "; // packet should be dropped if it’s out of window range or repeated

}

else

{

wait.enqueue(atoi(argv[i]),initSeq);

}

}