Project#3

Name: Ashritha Puradamane Balachandra

Compile and run: csim64.gcc project#3.c -o project#3

./ project#3

Prequierements of the program implementaion:

number of clients = 100

number of server = 1

Transmission delay = 0.8 = (data size/bandwidth)

Query delay (used to calculate simulation time) = 25

Cache size = 100

IR interval, L = 20

In sim function:

- created a process called "sim" and it will run till simulation time ends.
- Simulation Time is calculated as : CACHE_SIZE * Query_dealy * 10)
- Inputs: Mean update value and Mean query generate time
- init function, client function, server function and results function are called in sim function.

Init function:

- mailboxes are created for clients and server.
- created an array 'a' of size 1000 which represents the database of the server.

At server side:

server function:

- Created a new process called "serverprocess" and it will run till the simulation time ends.
- This function will call Update function, IR function and Receive functions.

update function:

- At server node update function creates a new process called "updateproc" and it will run till the simulation time ends.
- Data update probability is obtained using uniform distribution.
- If data update probability is less than 0.33, then

choose the hot data item between 0 to 49 from the database using uniform distribution for the update.

- If data update probability is greater than 0.33, then

choose the cold data item between 50 to 999 from the database using uniform distribution for the update.

- Now, update the hot or cold data item as obtained by using uniform distribution:

updated data item id is assigned to array index of 'a' and current time(data updated time at the server) is assigned to array value. Here array represent the database of the server.

output:

```
 Enter the mean update value(50 to 300): 15

 Enter the mean query generate time(25 to 300): 100

 data update probability at server is 0.513870

 Sever updated the cold data item with id 216 at time: 15.000

 Generated IR contains: the following id(s) sending at 20.000 seconds :

 216,

 Broadcasted IR from server node to all clients at 20.000 seconds

 Broadcasted IR is received by clients at 20.800 seconds

 data update probability at server is 0.308652

 Sever updated the hot data item with id 26 at time: 30.000

 Generated IR contains: the following id(s) sending at 40.000 seconds :
```

```
26, 216,

Broadcasted IR from server node to all clients at 40.000 seconds

Broadcasted IR is received by clients at 40.800 seconds

data update probability at server is 0.947628

Sever updated the cold data item with id 212 at time: 45.000

Generated IR contains: the following id(s) sending at 60.000 seconds :

26, 212, 216,

Broadcasted IR from server node to all clients at 60.000 seconds

data update probability at server is 0.702231

Sever updated the cold data item with id 264 at time: 60.000

Broadcasted IR is received by clients at 60.800 seconds

data update probability at server is 0.494773

Sever updated the cold data item with id 168 at time: 75.000

Generated IR contains: the following id(s) sending at 80.000 seconds :

26, 168, 212, 216, 264,

Broadcasted IR from server node to all clients at 80.000 seconds

Broadcasted IR is received by clients at 80.800 seconds

data update probability at server is 0.083899

Sever updated the hot data item with id 19 at time: 90.000
```

IR function:

- Create a new process called "irproc" and it will run till the simulation time ends.
- hold(L); //L=20 seconds
- generate IR at server(at server node is 100):

    if ir_queue is NIL then allocate memory for iReport using malloc.

IR contains the updated data item id and its updated time at the server.(updated data item id is fetched from the array 'a')

- Broadcast the generated IR to all clients.

output:

```
data update probability at server is 0.513870
 Sever updated the cold data item with id 216 at time: 15.000
 Generated IR contains: the following id(s) sending at 20.000 seconds :
 216,
 Broadcasted IR from server node to all clients at 20.000 seconds
 Broadcasted IR is received by clients at 20.800 seconds
```

Recv function:- Requested data item by one of the client is received by the server and it will broadcast the requested data item to all clients.

- created a new process called "serverrecvproc" and it will run till the simulation time ends.
- receive the requested data item requested by a client.

  ```
  receive(node[SERVER_NODE].input, (long *)&m);
  ```

- wait for the next IR event to occur.

  ```
  wait(ir_event);
  ```

- then form a reply message from the server to broadcast and send requested data item id it to all the clients.

output:

```
Query generated at client side for the id:49 at 100.000 seconds
 data item is not cached and hence query request is sending to server at 100.000 seconds
 Server received the requested query id:49 at 100.000seconds
 Generated IR contains: the following id(s) sending at 100.000 seconds :
 19, 26, 168, 212, 216, 264,
 Broadcasted IR from server node to all clients at 100.000 seconds
```

```
Broadcasted IR is received by clients at 100.800 seconds

Server broadcasted the requested data item to clients

Requested data item id:49 is received by the all clients at 100.800 seconds
```

At Client side:

client function:

- create a new process called "clientprocess" and it will run till the simulation time ends.
- RecvIR function, query function and RecvMsg functions are called in this function.

RecvIR function:

- create a new process called "clientreceiveirproc" and it will run till the simulation time ends.
- hold(L) ; //L=20
- All clients mailbox will receive the broadcasted IR

```
receive(node[l].input, (long *)&new_ir);
```

- Now check if data id present in the IR is also present in the client cache

if present: then compare if data item updated time of the server which is present in IR is greater than updated time present in client cache. If yes then invalidate that particular data item of client cache.

output:

```
data update probability at server is 0.513870

Sever updated the cold data item with id 216 at time: 15.000

Generated IR contains: the following id(s) sending at 20.000 seconds :

216,

Broadcasted IR from server node to all clients at 20.000 seconds

Broadcasted IR is received by clients at 20.800 seconds
```

---

query function:

- create a new process called "clientqueryproc" and it will run till run till the simulation time ends.
- hold(mean_query_generate_time)
- choose a client for generating query(client :0 to 99)
- choosing the data item for request: hot data access probability is obtained using uniform distribution. If hot data access probability is less than 0.8 then client will generate a query for the datat item 0 to 49(hot data item) using uniform distribution otherwise, generate a query for data item chosen between 50 to 999(cold data item) using uniform distribution.
- check whether requested data item is present in the client cache.
- create a new message from the client requesting for this data item. Increment the query_generated_count by 1.
- If requested data item is not present in cache then immediately send request to server. Otherwise, wait for the next IR event to occur.
- If requested data item is present in cache increment the cache hit count by 1, then after receiving the IR, check cache valid bit of the data item.

valid bit of requested data item is 0 then send request to server.

otherwise, data item is valid and hence client access this data item. Current time(data accessing time at client) is stored in client cache last access time. Calculate query delay (query_delay = query_delay + cur_time - m->start_time)

output:

```
Query generated at client side for the id:49 at 100.000 seconds

data item is not cached and hence query request is sending to server at 100.000 seconds

Server received the requested query id:49 at 100.000seconds

Generated IR contains: the following id(s) sending at 100.000 seconds :

19, 26, 168, 212, 216, 264,

Broadcasted IR from server node to all clients at 100.000 seconds

Broadcasted IR is received by clients at 100.800 seconds

Server broadcasted the requested data item to clients

Requested data item id:49 is received by the all clients at 100.800 seconds
```

To demonstrate the working of LRU in cache, I have taken 2 clients and cache size = 5 case: When cache is full and client 0 generate a query a query for the data item id 15 present in cache. Client access this data item after next IR event from cache since it is valid. When next query is generated, client 0 recently accessed data item 15 and hence replace the data item id 11 whereas in client node 1, it will replace the data item 15 since it is not recently accessed and it is replaced based on FCFS order.

```
Requested data item id:38 is received by the all clients at 260.800 seconds

 index value is 4

 in cache 0: data id:15 and valid bit: 1,

 in cache 1: data id:11 and valid bit: 1,

 in cache 2: data id:4 and valid bit: 1,

 in cache 3: data id:558 and valid bit: 1,

 in cache 4: data id:38 and valid bit: 1,


 index value is 4

 in cache 0: data id:15 and valid bit: 1,

 in cache 1: data id:11 and valid bit: 1,

 in cache 2: data id:4 and valid bit: 1,

 in cache 3: data id:558 and valid bit: 1,

 in cache 4: data id:38 and valid bit: 1,

 Generated IR contains: the following id(s) sending at 280.000 seconds :

 313, 949,

 Broadcasted IR from server node to all clients at 280.000 seconds

 Broadcasted IR is received by clients at 280.800 seconds

 data update probability at server is 0.780237

 Sever updated the cold data item with id 830 at time: 300.000

 Query generated at client side for the id:15 at 300.000 seconds

 Generated IR contains: the following id(s) sending at 300.000 seconds :
```

```
313, 830,

Broadcasted IR from server node to all clients at 300.000 seconds

Broadcasted IR is received by clients at 300.800 seconds

data item id:15 is valid in cache and hence it is accessed by the client node 0 at 300.800 seconds

Generated IR contains: the following id(s) sending at 320.000 seconds :

313, 830,

Broadcasted IR from server node to all clients at 320.000 seconds

Broadcasted IR is received by clients at 320.800 seconds

Generated IR contains: the following id(s) sending at 340.000 seconds :

313, 830,

Broadcasted IR from server node to all clients at 340.000 seconds

Broadcasted IR is received by clients at 340.800 seconds

Query generated at client side for the id:543 at 350.000 seconds

data item is not cached and hence query request is sending to server at 350.000 seconds

Server recieved the requested query id:543 at 350.000seconds

Generated IR contains: the following id(s) sending at 360.000 seconds :

313, 830,

Broadcasted IR from server node to all clients at 360.000 seconds

Broadcasted IR is received by clients at 360.800 seconds

Server broadcasted the requested data item to clients

Requested data item id:543 is received by the all clients at 360.800 seconds

index value is 1

in cache 0: data id:15 and valid bit: 1,

in cache 1: data id:543 and valid bit: 1,

in cache 2: data id:4 and valid bit: 1,
in cache 3: data id:558 and valid bit: 1,

in cache 4: data id:38 and valid bit: 1,


index value is 0

in cache 0: data id:543 and valid bit: 1,

in cache 1: data id:11 and valid bit: 1,

in cache 2: data id:4 and valid bit: 1,

in cache 3: data id:558 and valid bit: 1,

in cache 4: data id:38 and valid bit: 1,
```

RecvMsg function:

- Create a new process called "clientRecvMsgProc" and it will run till the simulation time ends.
- All clients mailboxes will receive the requested data item(id) sent by the server.
- Calculate the query delay. (query_delay = query_delay + cur_time - m->start_time)
- If client cache has the data item i.e particular data item id, then update client cache update time and set valid bit to 1.
- Otherwise, check if there is any empty block in the cache. If empty block is present, then insert the received data item in the cache and update cache update time and also update the valid bit to 1 .
- If cache is full then implement LRU as cache replacement policy.

Check for the least recently accessed data item i.e loop through the cache to find the cache block corresponding to the data item with the smallest value for cache access time and replace the data item having smallest access time with the received data item.

output:

```
Query generated at client side for the id:49 at 100.000 seconds

 data item is not cached and hence query request is sending to server at 100.000 seconds

 Server received the requested query id:49 at 100.000seconds
```

Generated IR contains: the following id(s) sending at 100.000 seconds :

19, 26, 168, 212, 216, 264,

Broadcasted IR from server node to all clients at 100.000 seconds

Broadcasted IR is received by clients at 100.800 seconds

Server broadcasted the requested data item to clients

Requested data item id:49 is received by the all clients at 100.800 seconds

---

Results:

Once cache becomes full, all performance parameters are reset to 0 and then taken the values.

For calculations, the formulae:

```
Cache hit ratio = cache_hit_count/query_generated_count;

avg_query_delay = query_delay/(query_generated_count * NUM_CLIENTS));

avg_number_of_queries_served = number_of_queries_served/(total_intervals)
```