teehee

1:

/\* CS 415 Project 3- dtsv1.c

\* Name: Carter Young;

\* Duck ID: cartery;

\* UO: 951690164;

\* Completed: 05/28/23;

\*

\* This is my own work. Conversed sparingly re: the project with Freddy Lopez,

\* Mason Kline, and Sydney Whiting.

\*/

#include "BXP/bxp.h"

#include <assert.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#define UNUSED \_\_attribute\_\_((unused))

#define SERVICE "DTS"

#define HOST "localhost"

#define PORT "19999"

void \*svcFxn(void \*args) {

BXPService bxps = (BXPService)args;

BXPEndpoint ep;

char query[1024], response[1025];

unsigned qlen;

while((qlen = bxp\_query(bxps, &ep, query, 1024)) > 0) {

sprintf(response,"1%s", query);

bxp\_response(bxps, &ep, response, qlen+1);

}

return NULL;

}

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

BXPService bxps;

pthread\_t svcThread;

assert(bxp\_init(19999, 1)); // Bind to port 19999

bxps = bxp\_offer("DTS"); // Offer service named "DTS"

assert(! pthread\_create(&svcThread, NULL, svcFxn, (void \*)bxps));

pthread\_join(svcThread, NULL); // Wait for the service thread to finish

return 0;

}

2:

/\* CS 415 Project 3- dtsv2.c

\* Name: Carter Young;

\* Duck ID: cartery;

\* UO: 951690164;

\* Completed: 05/28/23;

\*

\* This is my own work. Conversed sparingly re: the project with Freddy Lopez,

\* Mason Kline, and Sydney Whiting.

\*/

#include "BXP/bxp.h"

#include <pthread.h>

#include <assert.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define UNUSED \_\_attribute\_\_((unused))

#define SERVICE "DTS"

#define HOST "localhost"

#define PORT "19999"

BXPService \*bxps;

int checkCommand(char \*command, int numArgs) {

if (strcmp(command, "OneShot") == 0 && numArgs == 7) {

return 1;

} else if (strcmp(command, "Repeat") == 0 && numArgs == 8) {

return 1;

} else if (strcmp(command, "Cancel") == 0 && numArgs == 2) {

return 1;

}

return 0;

}

void \*svcFxn(void \*args) {

BXPService bxps = (BXPService)args;

BXPEndpoint ep;

char query[1024], response[1025], original\_query[1024];

unsigned qlen;

while ((qlen = bxp\_query(bxps, &ep, query, 1024)) > 0) {

char\* command;

int numArgs = 1;

// Save the original query before tokenizing

strncpy(original\_query, query, 1024);

command = strtok(query, "|");

while (strtok(NULL, "|") != NULL) {

numArgs++;

}

if(checkCommand(command, numArgs)){

// Prepend "1" to the response when the command is valid

sprintf(response, "1%s", original\_query);

} else {

// Prepend "0" to the response when the command is invalid

sprintf(response, "0%s", original\_query);

}

unsigned rlen = strlen(response) + 1;

assert(bxp\_response(bxps, &ep, response, rlen));

}

return NULL;

}

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

BXPService bxps;

pthread\_t svcThread;

assert(bxp\_init(19999, 1)); // Bind to port 19999

bxps = bxp\_offer("DTS"); // Offer service named "DTS"

assert(! pthread\_create(&svcThread, NULL, svcFxn, (void \*)bxps));

pthread\_join(svcThread, NULL); // Wait for the service thread to finish

return 0;

}

3:

/\* CS 415 Project 3- dtsv3.c

\* Name: Carter Young;

\* Duck ID: cartery;

\* UO: 951690164;

\* Completed: 05/28/23;

\*

\* This is my own work. Conversed sparingly re: the project with Freddy Lopez,

\* Mason Kline, and Sydney Whiting. Some code inspired by psuedocode shared in Discord.

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <pthread.h>

#include <assert.h>

#include <unistd.h>

#include <sys/time.h>

#include "BXP/bxp.h"

#include "ADTs/ADTdefs.h"

#include "ADTs/iterator.h"

#include "ADTs/heapprioqueue.h"

#include "ADTs/hashmap.h"

static unsigned long next\_sid = 0;

#define UNUSED \_\_attribute\_\_ ((unused))

#define MAX\_CMD\_LENGTH 256

#define SERVICE "DTS"

#define PORT 19999

#define HOST "localhost"

typedef struct event {

unsigned long event\_time;

unsigned long sid;

char \*host;

char \*service;

unsigned int port;

int cancelled;

int hashmap\_lookup;

} Event; // event information

// create an event

Event\* event\_create() {

Event\* event = (Event\*)malloc(sizeof(Event));

event->sid = next\_sid++;

event->hashmap\_lookup = 0;

return event;

}

// return event info

char\* event\_str(Event\* event) {

static char buf[BUFSIZ];

sprintf(buf, "EVENT- server\_id: %lu, hashmap\_lookup: %d", event->sid, event->hashmap\_lookup);

return buf;

}

// compare function

int event\_compare(void\* sid1, void\* sid2) {

// cast to an unsigned long pointer, then dereference that pointer with \*

return (\*(unsigned long\*)sid1) - (\*(unsigned long\*)sid2);

}

// free function

void event\_delete(void\* event) {

free((Event\*)event);

}

// hash function

long event\_hash(void\* sid, long N) {

return (\*(unsigned long\*)sid) % N;

}

void handle\_oneshot(const PrioQueue \*pqueue, const Map \*hmap, char\* command) {

Event\* event = event\_create();

BXPConnection \*conn;

conn = bxp\_connect(event->host, event->port, event->service, 1, 1);

// Parse the command

char\* tokens[7]; // "OneShot", <id>, <time>, <host>, <service>, <port>

char\* token = strtok(command, "|");

int i = 0;

while (token != NULL && i < 7) {

tokens[i] = token;

token = strtok(NULL, "|");

i++;

}

// Tokenize

//Event \*event = event\_create();

event->sid = atoi(tokens[1]);

event->event\_time = atol(tokens[2]);

event->host = strdup(tokens[3]);

event->service = strdup(tokens[4]);

event->port = atoi(tokens[5]);

event->cancelled = 0;

// Add the event to the pQ and hM

pqueue->insert(pqueue, (void \*)(&event->sid), (void \*)(event));

hmap->put(hmap, (void \*)(&event->sid), (void \*)(event));

// send response to client

char response[20];

sprintf(response, "1%08lu", event->sid);

bxp\_call(conn, response, strlen(response) + 1, NULL, 0, NULL);

bxp\_disconnect(conn);

}

void handle\_cancel(const Map \*hmap, char\* command) {

Event \*event;

BXPConnection \*conn;

conn = bxp\_connect(event->host, event->port, event->service, 1, 1);

char\* tokens[2]; // "Cancel", <sid>

char\* token = strtok(command, "|");

int i = 0;

while (token != NULL && i < 2) {

tokens[i] = token;

token = strtok(NULL, "|");

i++;

}

// Get event from hM

//Event \*event;

unsigned long sid = atol(tokens[1]);

hmap->get(hmap, (void \*)(&sid), (void \*\*)(&event));

// Set event as cancelled

event->cancelled = 1;

hmap->remove(hmap, (void \*)(&sid));

// send response to client

char response[20];

sprintf(response, "1%08lu", sid);

bxp\_call(conn, response, strlen(response) + 1, NULL, 0, NULL);

bxp\_disconnect(conn);

}

void notify\_client(Event\* event) {

char req[1024];

char resp[1024];

unsigned reqlen, resplen;

BXPConnection\* bxpc = bxp\_connect(event->host, event->port, event->service, 1, 1);

reqlen = (unsigned) (sprintf(req, "%lu", event->sid) + 1);

bxp\_call(bxpc, req, reqlen, resp, sizeof resp, &resplen);

bxp\_disconnect(bxpc);

}

void handle\_event\_firing(Event\* event) {

printf("Event fired: %lu|%s|%s|%u\n", event->sid, event->host, event->service, event->port);

notify\_client(event);

event\_delete(event);

// Connect to client and send notification

BXPConnection \*conn;

conn = bxp\_connect(event->host, event->port, event->service, 1, 1);

char req[BUFSIZ];

char resp[BUFSIZ];

unsigned reqlen, resplen;

reqlen = (unsigned) (sprintf(req, "%lu", event->sid) + 1);

bxp\_call(conn, req, reqlen, resp, sizeof resp, &resplen);

bxp\_disconnect(conn);

}

int main(UNUSED int argc, UNUSED char\*\* argv) {

// initialize pQ

const PrioQueue\* pqueue = PrioQueue\_create(event\_compare, doNothing, event\_delete);

// initialize hM

const Map\* hmap = HashMap(0, 0.0, event\_hash, event\_compare, doNothing, doNothing);

// connect to client

BXPConnection \*conn;

BXPEndpoint ep;

int ifEncrypted = 1;

assert(bxp\_init(PORT, ifEncrypted)); // Initialize BXP

conn = bxp\_offer(SERVICE);

if(!conn){

printf("Failed to connect to server.\n");

return 1;

}

char command[MAX\_CMD\_LENGTH];

char \*response;

unsigned rlen;

unsigned rsiz = 1024;

// Main loop

while (1) {

// Get command

fgets(command, MAX\_CMD\_LENGTH, stdin);

// Remove trailing newline if present

size\_t len = strlen(command);

if (len > 0 && command[len-1] == '\n') command[len-1] = '\0';

// Send command to server

unsigned clen = strlen(command) + 1;

assert(bxp\_call(conn, command, clen, response, rsiz, &rlen));

// Receive response from server

rlen = bxp\_response(conn, &ep, response, 1025);

if (rlen > 0) {

printf("Received from server: %s\n", response);

// Parse command

if (strncmp(command, "OneShot", 7) == 0) {

handle\_oneshot(pqueue, hmap, command);

} else if (strncmp(command, "Cancel", 6) == 0) {

handle\_cancel(hmap, command);

} else if (strncmp(command, "Repeat", 6) == 0) {

char response[MAX\_CMD\_LENGTH + 2]; // "+2" for the "0" and the null terminator

sprintf(response, "0%s", command);

bxp\_call(conn, response, strlen(response) + 1, NULL, 0, NULL);

} else {

printf("Unknown command: %s\n", command);

}

}

// Check for events to fire

unsigned long current\_time = time(NULL); // get the current time

// Peek at the next event in the priority queue

//Event\* next\_event = (Event\*) pqueue->peek(pqueue);

// Check if the next event is due to fire

if (pqueue != NULL) {

// Remove the event from the priority queue

Event\* next\_event = (Event\*) pqueue->removeMin(pqueue, (void\*)(&next\_event->event\_time), (void\*)(&next\_event->sid));

// check if the event is due to be fired

if (next\_event->event\_time <= current\_time){

// If the event is not cancelled, handle it

if (!next\_event->cancelled) {

handle\_event\_firing(next\_event);

} else {

// Free the event

event\_delete(next\_event);

}

// Peek at the next event in the priority queue

//next\_event = (Event\*) pqueue->peek(pqueue);

}

// Sleep for a while to avoid using 100% CPU

usleep(100000); // sleep for 100 ms

}

}

bxp\_disconnect(conn); // close the connection

bxp\_shutdown(); // and shutdown

// Cleanup

hmap->destroy(hmap);

pqueue->destroy(pqueue);

return 0;

}