Maltepe University

**FACULTY OF ENGINEERING**

**COMPUTER & SOFTWARE ENGINEERING DEPARTMENT**

**SE 364 Linux Scripts and Tools**

**PROJECT – 2023/24 Fall**

**PROJECT REPORT**

**DELIVERY DATE**

08/01/2024

**PREPARED BY**

**200706027**   **EMRECAN KAHRAMAN**

**200706037 MUNİP KATAYIFÇI**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <string.h>

#include <errno.h>

#define BUFFER\_SIZE 512

#define BUFFER\_SIZE\_SMALL 128

#define DEFAULT\_FILE\_NAME "inventory.txt"

#define OPEN\_MODE\_READ\_PLUS "r+"

#define OPEN\_MODE\_WRITE\_PLUS "w+"

void\* link\_thread\_open();

void\* link\_thread\_read();

void\* link\_thread\_write();

void\* link\_thread\_update();

void\* link\_thread\_close();

void\* link\_thread\_client();

enum KernelFunction {

OP\_READ = 'read',

OP\_WRITE = 'write',

OP\_UPDATE = 'update',

OP\_OPEN = 'open',

OP\_CLOSE = 'close',

OP\_EXIT = 'exit'

};

enum UpdateType {

UP\_UPDATE = 0,

UP\_ADD,

UP\_DELETE

};

typedef struct {

enum KernelFunction type;

char\* write\_content;

enum UpdateType update\_type;

char\* updating\_field;

char\* updating\_item\_name;

char\* updating\_field\_new\_value;

char\* add\_item\_name;

char\* add\_item\_price;

char\* add\_item\_quantity;

char\* deleting\_item\_name;

} Command;

char command[BUFFER\_SIZE];

Command command\_common;

sem\_t serv\_sem, cli\_sem, read\_sem, update\_sem, open\_sem, write\_sem, close\_sem;

FILE\* file\_p;

char \*sliceString(char \*str, int start, int end);

char\* get\_new\_item\_entry\_content();

void parse\_command();

int main() {

sem\_init(&serv\_sem, 0, 0);

sem\_init(&cli\_sem, 0, 0);

sem\_init(&read\_sem, 0, 0);

sem\_init(&write\_sem, 0, 0);

sem\_init(&update\_sem, 0, 0);

sem\_init(&open\_sem, 0, 0);

sem\_init(&close\_sem, 0, 0);

pthread\_t thread\_read, thread\_write, thread\_update, thread\_open, thread\_close;

pthread\_create(&thread\_read, NULL, link\_thread\_read, NULL);

pthread\_create(&thread\_write, NULL, link\_thread\_write, NULL);

pthread\_create(&thread\_update, NULL, link\_thread\_update, NULL);

pthread\_create(&thread\_open, NULL, link\_thread\_open, NULL);

pthread\_create(&thread\_close, NULL, link\_thread\_close, NULL);

pthread\_t thread\_client;

pthread\_create(&thread\_client, NULL, link\_thread\_client, NULL);

sem\_post(&cli\_sem);

while (1) {

int r = sem\_wait(&serv\_sem);

parse\_command();

switch (command\_common.type) {

case OP\_READ:

sem\_post(&read\_sem);

break;

case OP\_WRITE:

sem\_post(&write\_sem);

break;

case OP\_UPDATE:

sem\_post(&update\_sem);

break;

case OP\_OPEN:

sem\_post(&open\_sem);

break;

case OP\_CLOSE:

sem\_post(&close\_sem);

break;

case OP\_EXIT:

pthread\_cancel(thread\_client);

pthread\_cancel(thread\_read);

pthread\_cancel(thread\_write);

pthread\_cancel(thread\_update);

pthread\_cancel(thread\_open);

pthread\_cancel(thread\_close);

sem\_destroy(&serv\_sem);

sem\_destroy(&cli\_sem);

sem\_destroy(&read\_sem);

sem\_destroy(&write\_sem);

sem\_destroy(&update\_sem);

sem\_destroy(&open\_sem);

sem\_destroy(&close\_sem);

exit(0);

default:

printf("Invalid command: %s", command\_common.type);

exit(2);

}

sem\_wait(&serv\_sem);

sem\_post(&cli\_sem);

}

return 0;

}

void parse\_command() {

if (strncmp(command, "open", 4) == 0) {

command\_common.type = OP\_OPEN;

} else if (strncmp(command, "close", 4) == 0) {

command\_common.type = OP\_CLOSE;

} else if (strncmp(command, "write", 5) == 0) {

command\_common.type = OP\_WRITE;

command\_common.write\_content = sliceString(command, 5, strlen(command));

} else if (strncmp(command, "read", 4) == 0) {

command\_common.type = OP\_READ;

} else if(strncmp(command, "add", 3) == 0) {

char\* splitted\_command\_name = strtok(command, " ");

command\_common.type = OP\_UPDATE;

command\_common.update\_type = UP\_ADD;

command\_common.add\_item\_name = strtok(0, " ");

command\_common.add\_item\_price = strtok(0, " ");

command\_common.add\_item\_quantity = strtok(0, " ");

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

char\* splitted\_command\_name = strtok(command, " ");

command\_common.type = OP\_UPDATE;

command\_common.update\_type = UP\_DELETE;

command\_common.deleting\_item\_name = strtok(0, " ");

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

char\* splitted\_command\_name = strtok(command, " ");

command\_common.type = OP\_UPDATE;

command\_common.update\_type = UP\_UPDATE;

command\_common.updating\_item\_name = strtok(0, " ");

command\_common.updating\_field = strtok(0, " ");

command\_common.updating\_field\_new\_value = strtok(0, " ");

} else if (strncmp(command, "exit", 4) == 0) {

command\_common.type = OP\_EXIT;

} else {

printf("Invalid command");

exit(1);

}

}

char \*sliceString(char \*str, int start, int end)

{

int i;

int size = (end - start) + 2;

char \*output = (char \*)malloc(size \* sizeof(char));

for (i = 0; start <= end; start++, i++)

{

output[i] = str[start];

}

output[size] = '\0';

return output;

}

void\* link\_thread\_client() {

while (1) {

sem\_wait(&cli\_sem);

printf("\nExample commands:\n");

printf("Open the file: open\n");

printf("Close the file: close\n");

printf("Read the file: read\n");

printf("Add new entry: add item\_name item\_price item\_quantity\n");

printf("Delete an entry: delete deleting\_item\_name\n");

printf("Update an entry: update tem\_name updating\_field: price or quantity new\_value\n\n");

printf("Please enter the command you want to execute: ");

gets(&command);

sem\_post(&serv\_sem);

}

}

void\* link\_thread\_write() {

while (1) {

sem\_wait(&write\_sem);

file\_p = fopen(DEFAULT\_FILE\_NAME, OPEN\_MODE\_WRITE\_PLUS);

if (file\_p == NULL) {

printf("Unable to open file.");

} else {

fprintf(file\_p, "%s\n", command\_common.write\_content);

fclose(file\_p);

}

sem\_post(&serv\_sem);

}

}

void\* link\_thread\_open() {

while (1) {

sem\_wait(&open\_sem);

file\_p = fopen(DEFAULT\_FILE\_NAME, OPEN\_MODE\_READ\_PLUS);

if (file\_p == NULL) {

printf("Unable to open file. Code: %d\n", errno);

} else {

printf("File opened.\n");

fclose(file\_p);

}

sem\_post(&serv\_sem);

}

}

void\* link\_thread\_read() {

while (1) {

sem\_wait(&read\_sem);

FILE\* fp = fopen(DEFAULT\_FILE\_NAME, OPEN\_MODE\_READ\_PLUS);

if (fp == NULL) {

printf("Couldn't open file!");

} else {

char buffer[BUFFER\_SIZE];

while (fgets(buffer, BUFFER\_SIZE, fp) != NULL) {

printf("%s", buffer);

}

}

sem\_post(&serv\_sem);

}

}

void\* link\_thread\_update() {

while (1) {

sem\_wait(&update\_sem);

char commandStr[1024];

if (command\_common.update\_type == UP\_UPDATE) {

sprintf(

commandStr,

"awk -v OPERATION\_TYPE='update' -v ITEM\_NAME='%s' -v UPDATE\_TYPE='%s' -v NEW\_VALUE='%s' -f ./update.awk %s",

command\_common.updating\_item\_name,

command\_common.updating\_field,

command\_common.updating\_field\_new\_value,

DEFAULT\_FILE\_NAME

);

} else if (command\_common.update\_type == UP\_ADD) {

sprintf(

commandStr,

"echo \"%s\t%s\t%s\" >> %s",

command\_common.add\_item\_name,

command\_common.add\_item\_price,

command\_common.add\_item\_quantity,

DEFAULT\_FILE\_NAME

);

} else if (command\_common.update\_type == UP\_DELETE) {

sprintf(

commandStr,

"awk -v OPERATION\_TYPE='delete' -v ITEM\_NAME='%s' -f ./update.awk %s",

command\_common.deleting\_item\_name,

DEFAULT\_FILE\_NAME

);

}

int ret = system(commandStr);

if(ret != 0) {

fprintf(stderr, "AWK script execution failed\n");

}

sem\_post(&serv\_sem);

}

}

char\* get\_new\_item\_entry\_content() {

char\* item\_row = (char\*)malloc(

strlen(command\_common.add\_item\_name) +

strlen(command\_common.add\_item\_price) +

strlen(command\_common.add\_item\_quantity) +

2 + // for 2 \t characters

1 // null terminator

);

strcat(item\_row, command\_common.add\_item\_name);

strcat(item\_row, "\t");

strcat(item\_row, command\_common.add\_item\_price);

strcat(item\_row, "\t");

strcat(item\_row, command\_common.add\_item\_quantity);

return item\_row;

}

void\* link\_thread\_close() {

while (1) {

sem\_wait(&close\_sem);

if (file\_p == NULL) {

} else {

fclose(file\_p);

}

sem\_post(&serv\_sem);

}

}

update.awk

BEGIN {

FS = OFS = "\t"

}

function update\_inventory(old\_name, update\_type, new\_value) {

for(i = 0; i <= NR; i++) {

for(j = 0; j <= NF; j++) {

if ($j == old\_name) {

if (update\_type == "price") {

$(j+1) = new\_value

}

if (update\_type == "quantity") {

$(j+2) = new\_value

}

}

}

}

}

function delete\_inventory(name) {

for(i=1; i<=NR; i++) {

for(j = 0; j <= NF; j++) {

if ($j == name) {

$0 = "" # Set the whole line to empty

return

}

}

}

}

{

if (OPERATION\_TYPE == "update") {

update\_inventory(ITEM\_NAME, UPDATE\_TYPE, NEW\_VALUE)

print > FILENAME

}

if (OPERATION\_TYPE == "delete") {

delete\_inventory(ITEM\_NAME)

if (NF) {

print > FILENAME

}

}

}

opening the file metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Reading the filemetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Adding the file:

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Deleting the filemetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Pseudocodes:

Main Program:

Initialize semaphores and variables.

Create threads for client interaction, reading, writing, updating, opening, and closing file operations.

Start a loop:

Wait for a signal from the client thread.

Parse the received command.

Based on the command type:

Signal the appropriate worker thread to execute the operation.

Wait for a signal back from the worker thread.

Signal the client thread to continue.

If the command is "exit", clean up resources and exit.

Client Thread:

Start a loop:

Display available commands.

Prompt the user for a command.

Send the command to the main program.

Wait for a signal from the main program to continue.

Read, Write, Open, Close Threads

Start a loop:

Wait for a signal from the main program.

Perform the corresponding file operation (open, read, write, close).

Signal the main program that the operation is complete.

Update Thread:

Start a loop:

Wait for a signal from the main program.

Construct an AWK command based on the update type (update, add, delete).

Execute the AWK command using system().

Signal the main program that the update is complete