با نام خدا

گزارش کار ازمایشگاه سیستم عامل

گزارش شماره ۴

MessagePassing

&&

Shared Memory

امیر حسین متقیان ۴۰۱۳۱۰۴۳

کیان پور اذر ۴۰۱۳۱۴۰۳

سوال اول : پیاده سازی Consumer, Producer با استفاده از Shared memory

برای این سوال از دو راه حل استفاده کردم در راه حل اول از Fork استفاده کردم و برای Producer و Conuser هر یک یک process ایجاد کرده ام و در راه حل دوم دو file جدا که در یکی Consumer و در دیگری Producer است قرار داده ام و نتیجه همان است و برای هر یک باز هم process ساخته میشود

نکته قابل توجه ازمایش این است که باید از اینکه ابتدا Producer تولید کند و منتظر بماند که Conuser مصرف کند و دوباره تولید کند که برای ان از یک flag استفاده کرده ام و با استفاده از loop انرا بررسی میکنیم . پس از اینکه producer تولید کرد flag مربوطه 0 میشود و تا زمانی که ۱ نشود producer دیگر نمی تواند تولید کند از طرفی در این وضعیت consumer میتوان مصرف کند زیرا flag صفر است و پس از مصرف flag را یک میکند و حال producer تولید میکند و به همین ترتیب از race condition جلوگیری میکنیم که از معایب share memory بود

شاید به نظر برسد که loop ها باعث busy waiting میشود که با استفاده از یک sleep کوتاه در loop ها تا حد زیادی از busy waiting جلوگیری میکنیم برای از بین بردن کامل ان باید از semaphore ها استفاده کنیم که از بحث این گزارش خارج است

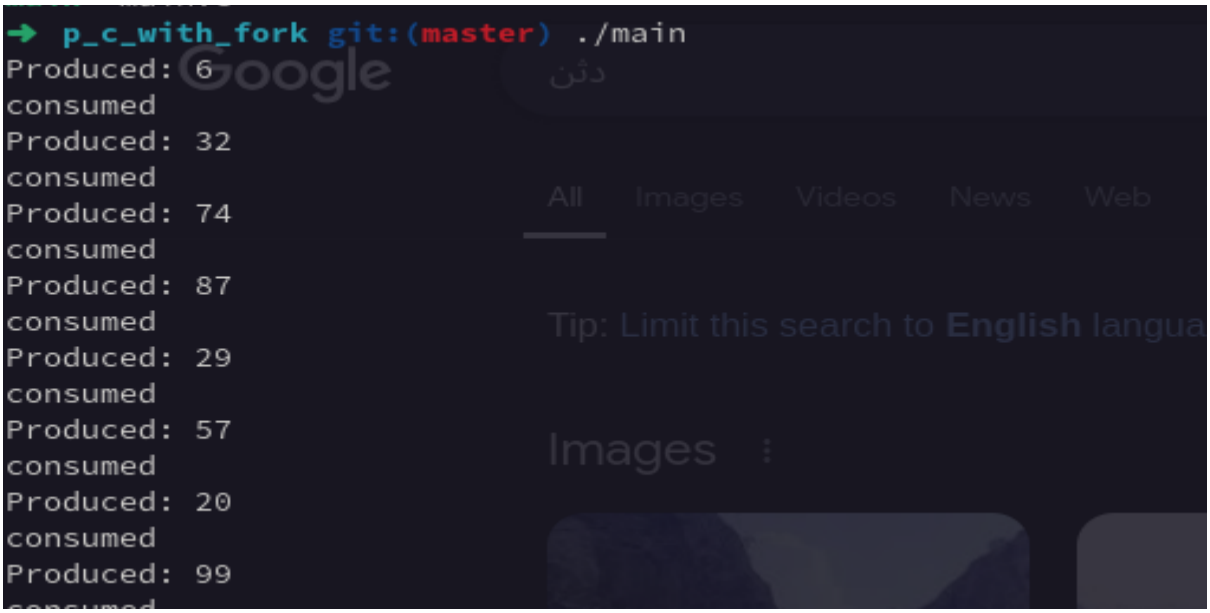
حالت اول (استفاده از fork)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <time.h>

// Shared memory to store 1 integer and 1 flaG
#define SHM_SIZE (2*sizeof(int))

int consumer(int *number, int *flag) {
   int sum = 0;
   for (int i = 0; i < 100; i++) {

      // Wait for the producer to write a number with a Short sleep to avoid busy waiting
      while (*flag == 0) {
         usleep(100);
      }

      // Mark the number as consumed
      *flag = 0;
      printf("consumed\n");
      sum += *number;
   }

   return sum;
}

void produce(int *number, int *flag) {
   srand(time(NULL));
   for (int i = 0; i < 100; i++) {

      // Wait until the consumer has consumed the number with a  Short sleep to avoid busy waiting
      while (*flag == 1) {
         usleep(100);
      }

      // Generate a random number between 0 and 99 and Mark the number as ready for consumption
      *number = rand() % 100;
      *flag = 1;
      printf("Produced: %d\n", *number);
```

```
    }

}

int main() {
    // Create shared memory segment
    int shared_memory_id = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
    if (shared_memory_id == -1) {
        perror("Failed to allocate shared memory");
        exit(1);
    }

    // Attach shared memory
    int *shared_memory = (int *) shmat(shared_memory_id, NULL, 0);
    if (shared_memory == (int *) -1) {
        perror("Failed to attach shared memory to process");
        exit(1);
    }

    // Pointer to shared memory for the number and flags (1: ready, 0: not ready)
    int *number = &shared_memory[0];
    int *flag = &shared_memory[1];

    // Initially, no number is written
    *flag = 0;

    // Fork a new process
    pid_t pid = fork();
    if (pid < 0) {
        perror("Failed to fork new process");
        exit(1);
    }

    if (pid == 0) {

        // Child process: Consumer
        int sum = consumer(number, flag);

        // Detach shared memory
        shmdt(shared_memory);

        // print the result
        printf("Total Sum: %d\n", sum);

        // finish the child process
        exit(0);
    } else {
```

```
    // Parent process: Producer
    produce(number, flag);

    // Wait for the consumer to finish
    wait(NULL);

    // Detach shared memory
    shmdt(shared_memory);

    // Remove shared memory segment
    shmctl(shared_memory_id, IPC_RMID, NULL);
  }

  return 0;
}
```

خروجی



و در نهایت sumation تمام ۱۰۰ را میتواند مشاهده کرد

حالت دوم (فایل های جداگانه)

Producer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <time.h>


// Shared memory size for 1 integer and 1 flag
#define SHM_SIZE (2 * sizeof(int))
// Shared memory key
#define SHM_KEY 1234


void produce(int *number, int *flag) {
  srand(time(NULL));
  for (int i = 0; i < 100; i++) {


    // Wait until the consumer has consumed the previous number
    while (*flag == 1) {
      usleep(100);
    }
```

```
        // Generate a random number between 0 and 99
        *number = rand() % 100;
        *flag = 1;
        printf("Produced: %d\n", *number);
        usleep(1000000);
    }
}

int main() {
    // Create or attach to the shared memory segment using a fixed key
    int shm_id = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("Failed to allocate shared memory");
        exit(1);
    }

    // Attach shared memory
    int *shared_memory = (int *) shmat(shm_id, NULL, 0);
    if (shared_memory == (int *) -1) {
        perror("Failed to attach shared memory");
        exit(1);
    }

    // Pointer to shared memory for the number and flags (1: ready, 0: not ready)
    int *number = &shared_memory[0];
    int *flag = &shared_memory[1];

    // Initially, no number is written
    *flag = 0;

    // Produce 100 numbers
    produce(number, flag);

    // Detach shared memory
    shmdt(shared_memory);

    return 0;
}
```

Consumer.c

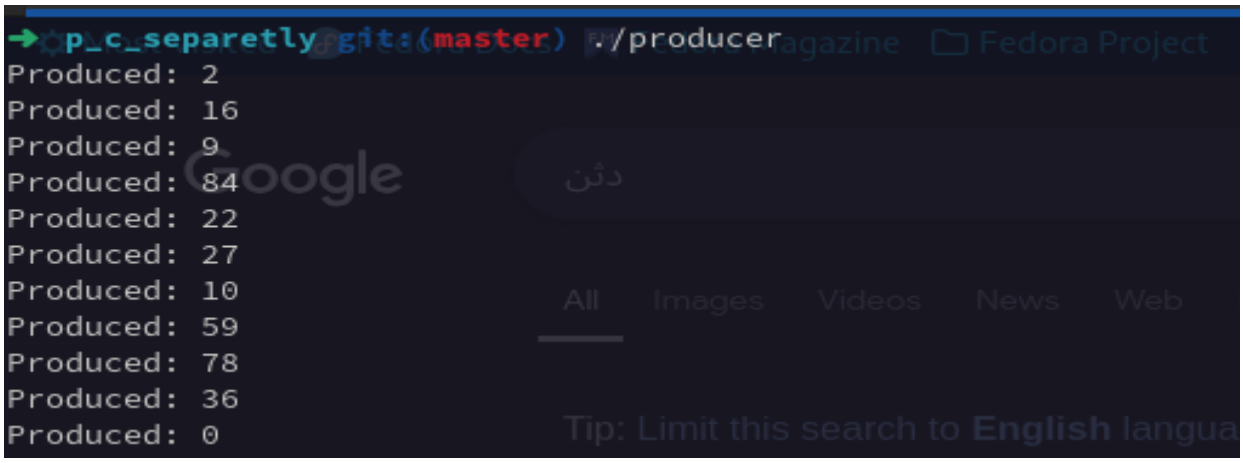
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
```

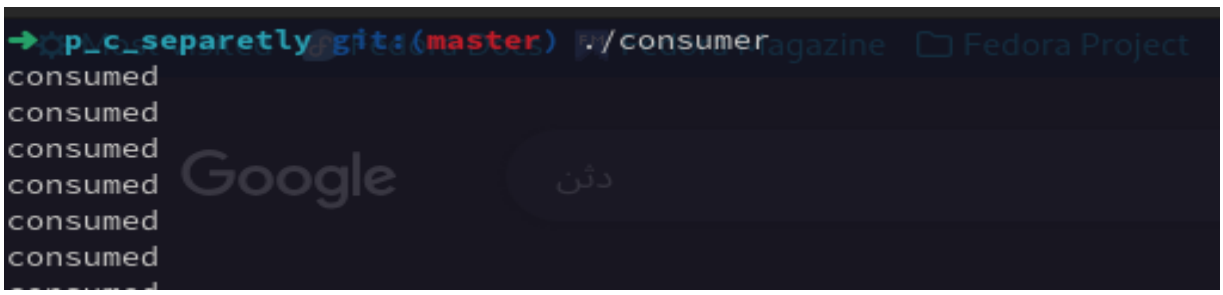
```c
// Shared memory size for 1 integer and 1 flag
#define SHM_SIZE (2 * sizeof(int))
// Shared memory key
#define SHM_KEY 1234

int consumer(int *number, int *flag) {
    int sum = 0;
    for (int i = 0; i < 100; i++) {

        // Wait for the producer to write a number
        while (*flag == 0) {
            usleep(100);
        }

        // Consume the number
        sum += *number;
        printf("consumed\n");
        *flag = 0;

        usleep(1000000);

    }

    return sum;
}

int main() {
    // Attach to the existing shared memory segment using the fixed key
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id == -1) {
        perror("Failed to attach to shared memory");
        exit(1);
    }

    // Attach shared memory
    int *shared_memory = (int *) shmat(shm_id, NULL, 0);
    if (shared_memory == (int *) -1) {
        perror("Failed to attach shared memory");
        exit(1);
    }

    // Pointer to shared memory for the number and flags (1: ready, 0: not ready)
    int *number = &shared_memory[0];
    int *flag = &shared_memory[1];

    // Consume 100 numbers
    int sum = consumer(number, flag);
```

```
    // Print the result
    printf("Total Sum: %d\n", sum);

    // Detach shared memory
    shmdt(shared_memory);

    // Remove the shared memory segment (cleanup happens here)
    shmctl(shm_id, IPC_RMID, NULL);

    return 0;
}
```

هر فایل را به صورت جدا گانه اجرا میکنیم

خروجی





و در نهایت

سوال دوم

اجزای مهم کد

۱ـ ساختن کلاینت و سرور و باز کردن و اتصال به سوکت

در این سوال بعد از ساختن کلاینت و سرور و اتصال به سوکت از کلاینت میتوانیم message ارسال کنیم و پاسخ بگیریم .

۲ـ ارسال پیام با فرمت مشخص و پارس کردن پیام

برای ارسال پیام از پروتکل خود ساخته استفاده کردم که اجزای پیام را با صورت زیر ارسال میکند

Command|arg|arg|arg|....

و ریکوست ها را در سمت سرور پارس کرده و بر اساس پیام دستور را انجام میدهیم

۳ـذخیره محصولات و ارسال انها

با استفاده از link list که از لایبری Glib استفاده کردم امکان ساخت و اپدیت و حذف را داریم

۴ـدر حالت پیشرفته هم نیاز است اجازه اتصال تعداد نا محدودی کاربر را بدهیم.

برای این چالیش میتوانیم از fork , thread استفاده کنیم که با توجه به بحث ازمایشگاه از fork استفاده میکنیم

۵ـمحصولات هر کاربر باید منحصر به خودش باشد

از انجا که از fork استفاده میکنیم هر کاربر در ابتدا برای خود یک کپی خالی از لیست دارد و از لیست خودش استفاده میکند

برای کامپایل کد server از داشتن لایبری glib مطمعن شوید و یا از cmake استفاده کنید که فایلشو قرار میدم یا از دستور زیر استفاده کنید

o main $(pkg-config --cflags --libs glib-2.0)- gcc main.c

کد حالت پیشرفته رو میزارم که کد حالت ساده رو هم در بر میگیره (داخل فایل ها کد حالت ساده هم وجود داره)

Client.c

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>



#define PORT 8080

int socket_connect() {
    int sock = 0;
    struct sockaddr_in serv_addr;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\nSocket creation error\n");
        return -1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported\n");
        return -1;
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed\n");
        return -1;
    }
```

```
    return sock;
}

int main() {
    int valread;
    char buffer[2048] = {0};
    char response[100];
    char request[1000];

    int sock = socket_connect();

    printf("For showing the list of products, enter: List \n");
    printf("For creating a product, enter: Create \n");
    printf("For Increase Amount of a product, enter: Add \n");
    printf("For Reduce Amount of a product, enter: Reduce \n");
    printf("For Remove a product, enter: Remove \n");
    printf("For exit, enter: Exit \n");

    while (1) {
        scanf("%s", response);

        // List Command
        if (strcmp(response, "List") == 0) {
            snprintf(request, sizeof(request), "%s", response);
        }
            // Create Command
        else if (strcmp(response, "Create") == 0) {
            char product_name[100];
            int amount;

            printf("Enter product name: ");
            scanf("%99s", product_name);

            getchar();

            printf("Enter product amount (press Enter for default 0): ");
            char amount_input[10];
            fgets(amount_input, sizeof(amount_input), stdin);
            if (strlen(amount_input) == 0 || amount_input[0] == '\n') {
                amount = 0; // Default to 0 if no input is given
            } else {
                sscanf(amount_input, "%d", &amount);
            }

            snprintf(request, sizeof(request), "%s|%s|%d", response, product_name, amount);
        }
            // Increase Amount Command
        else if (strcmp(response, "Add") == 0) {
            char product_name[100];
```

```
        int amount;

        printf("Enter product name: ");
        scanf("%99s", product_name);

        getchar();

        printf("Enter the amount to increase: ");
        scanf("%d", &amount);

        snprintf(request, sizeof(request), "%s|%s|%d", response, product_name, amount);
    }
        // Reduce Amount Command
    else if (strcmp(response, "Reduce") == 0) {
        char product_name[100];
        int amount;

        printf("Enter product name: ");
        scanf("%99s", product_name);

        getchar();

        printf("Enter the amount to increase: ");
        scanf("%d", &amount);

        snprintf(request, sizeof(request), "%s|%s|%d", response, product_name, amount);
    }
        // Remove Product Command
    else if (strcmp(response, "Remove") == 0) {
        char product_name[100];

        printf("Enter product name: ");
        scanf("%99s", product_name);

        snprintf(request, sizeof(request), "%s|%s", response, product_name);
    }
        // Exit Command
    else if (strcmp(response, "Exit") == 0) {
        send(sock, "Exit", strlen("Exit"), 0);
        break;
    }
        // Unknown Command
    else {
        printf("Unknown command. Please enter List, Create, or Exit.\n");
        continue; // Skip sending unknown commands
    }

    // Send message
```

```
        send(sock, request, strlen(request), 0);

        // Get response
        valread = read(sock, buffer, sizeof(buffer) - 1);
        if (valread < 0) {
            break;
        }
        buffer[valread] = '\0';

        printf("******* Response from the server *******\n");
        char *token = strtok(buffer, "|");
        while (token != NULL) {
            printf("Token: %s\n", token);
            token = strtok(NULL, "|");
        }
        printf("*******     End of Response     *******\n");

        memset(buffer, 0, sizeof(buffer));
    }

    close(sock);
    return 0;
}
```

Main.c

```
#include <glib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>

GList *list = NULL;
int PORT = 8080;
int SERVER_FD;
int CLIENT_SOCKET;
volatile sig_atomic_t RUNNING = 1;

// Define a struct to hold data
typedef struct {
    unsigned int id;
    char name[50];
    int amount;
} Product;
```

```c
// List and Product Functions
int addProduct(const char *, int);
int increaseProductAmount(char *, int );
int reduceProductAmount(char *, int );
int removeProduct(const char *);
char *sendProductsList();
void freeList();

// Server Functions
void handle_shutdown(int);
void setup_server();
void handle_client(struct sockaddr_in);
int process_command(char *);



// AddProduct to User Product: 0 for success,-1 for failed allocate,-2 for duplicated name
int addProduct(const char *name, int amount) {
    unsigned int currentLength = g_list_length(list);

    for (GList *l = list; l != NULL; l = l->next) {
        Product *p = (Product *) l->data;

        if (strcmp(p->name, name) == 0) {
            return -2;
        }
    }

    Product *p = malloc(sizeof(Product));
    if (p == NULL) {
        return -1;
    }

    p->id = currentLength + 1;
    snprintf(p->name, sizeof(p->name), "%s", name);
    p->amount = amount;

    list = g_list_append(list, p);
    return 0;
}

// Increase the Amount of a Product: the result is -1 product not found,0 for success
int increaseProductAmount(char *name, int change) {
    for (GList *l = list; l != NULL; l = l->next) {
        Product *p = (Product *) l->data;

        if (strcmp(p->name, name) == 0) {
            p->amount += change;
            return 0;
        }
```

```
    }
    return -1;
}

// Reduce the Amount of a Product: if the result is -1 product not found,0 for success
int reduceProductAmount(char *name, int change) {
    for (GList *l = list; l != NULL; l = l->next) {
        Product *p = (Product *) l->data;

        if (strcmp(p->name, name) == 0) {
            p->amount -= change;
            return 0;
        }
    }
    return -1;
}

// Remove a Product: if the result is -1 product not found,0 for success
int removeProduct(const char *name) {
    GList *current = list;
    GList *prev = NULL;

    while (current != NULL) {
        Product *p = (Product *) current->data;

        if (strcmp(p->name, name) == 0) {
            if (prev == NULL) {
                list = g_list_remove_link(list, current);
            } else {
                prev->next = current->next;
                g_list_free_1(current);
            }

            free(p);
            return 0;
        }

        prev = current;
        current = current->next;
    }

    return -1;
}

// Prepare a response for product list
char *sendProductsList() {
    char product[100];
    char *response = malloc(10000 * sizeof(char));
    if (response == NULL) {
```

```
        perror("Failed to allocate memory");
        exit(EXIT_FAILURE);
    }
    response[0] = '\0';

    if (list == NULL) {
        response = "You have no Product :)";
        return response;
    }

    for (GList *l = list; l != NULL; l = l->next) {
        Product *p = (Product *) l->data;
        snprintf(product, sizeof(product), "ID: %d, Name: %s, Amount: %d|", p->id, p->name, p->amount);
        strcat(response, product);
    }

    return response;
}

// free up the list
void freeList() {
    for (GList *l = list; l != NULL;) {
        GList *next = l->next;
        Product *p = (Product *) l->data;
        free(p);
        g_list_free_1(l);
        l = next;
    }
}

// Handler for shutdown when a signal come
void handle_shutdown(int signum) {
    static volatile sig_atomic_t shutting_down = 0;

    if (shutting_down) {
        return;
    }

    shutting_down = 1; // Mark as shutting down
    RUNNING = 0; // Update the running flag
    printf("Server shutting down...\n");

    close(SERVER_FD); // Close the server socket
    printf("Free the memory...\n");
    freeList(); // Free the product list
}

// Setup server socket
void setup_server() {
```

```
    struct sockaddr_in address;

    // Create server socket
    if ((SERVER_FD = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Initialize address struct
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind socket to the address
    if (bind(SERVER_FD, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Start listening on the server socket
    if (listen(SERVER_FD, 10) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Listening on %s:%d\n", inet_ntoa(address.sin_addr), ntohs(address.sin_port));
}

// Handle a client connection
void handle_client(struct sockaddr_in address) {
    char buffer[1024] = {0};
    char client_info[100];

    snprintf(client_info, sizeof(client_info), "%s:%d", inet_ntoa(address.sin_addr), ntohs(address.sin_port));
    while (1) {
        int valread = read(CLIENT_SOCKET, buffer, 1024);
        if (valread <= 0) {
            printf("Client %s disconnected.\n", client_info);
            break;
        }

        int res = process_command(buffer);
        if (res == -1) {
            printf("Client %s disconnected.\n", client_info);            break;
        }
        memset(buffer, 0, sizeof(buffer));
    }

    close(CLIENT_SOCKET);
```

```
}

// Process a command from the client
int process_command(char *buffer) {
    char *tokens[10] = {0};
    char *response = "";

    // Tokenize the input command
    char *token = strtok(buffer, "|");
    for (int i = 0; token != NULL && i < 10; i++) {
        tokens[i] = token;
        token = strtok(NULL, "|");
    }

    // List Command
    if (strcmp(tokens[0], "List") == 0) {
        response = sendProductsList();
    }
        // Create Command
    else if (strcmp(tokens[0], "Create") == 0) {
        int res = addProduct(tokens[1], atoi(tokens[2]));
        response = (res == -1) ? "Fail to allocate memory" :
                   (res == -2) ? "The name must be unique" : "Product created successfully";
    }
        // Increase Amount Command
    else if (strcmp(tokens[0], "Add") == 0) {
        int res = increaseProductAmount(tokens[1], atoi(tokens[2]));
        response = (res == -1) ? "There is no item with given name" : "Amount of product increased successfully";
    }
        // Reduce Amount Command
    else if (strcmp(tokens[0], "Reduce") == 0) {
        int res = reduceProductAmount(tokens[1], atoi(tokens[2]));
        response = (res == -1) ? "There is no item with given name" : "Amount of product reduced successfully";
    }
        // Remove Product Command
    else if (strcmp(tokens[0], "Remove") == 0) {
        int res = removeProduct(tokens[1]);
        response = (res == -1) ? "There is no item with given name" : "Product removed successfully";
    }
    else if (strcmp(tokens[0], "Exit") == 0){
        return -1;
    }

    // Send response
    send(CLIENT_SOCKET, response, strlen(response), 0);
    return 0;
}

// Start server
void start_server() {
    struct sockaddr_in address;
```

```
    const int address_len = sizeof(address);

    while (RUNNING) {
        if ((CLIENT_SOCKET = accept(SERVER_FD, (struct sockaddr *)&address, (socklen_t *)&address_len)) < 0) {
            if (!RUNNING) break;
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        printf("Client connected: %s:%d\n", inet_ntoa(address.sin_addr), ntohs(address.sin_port));

        if (fork() == 0) {
            close(SERVER_FD);
            handle_client(address);
            exit(0);
        }

        close(CLIENT_SOCKET);
    }
}

// Main function
int main() {
    // handle kill signal for shutdown
    signal(SIGINT, handle_shutdown);
    signal(SIGTERM, handle_shutdown);

    // setup server
    setup_server();

    // start server
    start_server();

    return 0;
}
```

اجرا و خروجی

سرور

```
/home/amir/Desktop/CPP/WarehouseManagement/cmake-build-debug/WarehouseManagement
Listening on 0.0.0.0:8080
```

کلاینت ۱

```
sh-5.2$ ./client
For showing the list of products, enter: List
For creating a product, enter: Create
For Increase Amount of a product, enter: Add
For Reduce Amount of a product, enter: Reduce
For Remove a product, enter: Remove
For exit, enter: Exit
```

کلاینت ۲

```
sh-5.2$ ./client
For showing the list of products, enter: List
For creating a product, enter: Create
For Increase Amount of a product, enter: Add
For Reduce Amount of a product, enter: Reduce
For Remove a product, enter: Remove
For exit, enter: Exit
```

کلاینت ۳

```
sh-5.2$ ./client
For showing the list of products, enter: List
For creating a product, enter: Create
For Increase Amount of a product, enter: Add
For Reduce Amount of a product, enter: Reduce
For Remove a product, enter: Remove
For exit, enter: Exit
```

وضعیت سرور

```
/home/amir/Desktop/CPP/WarehouseManagement/cmake-build-debug/WarehouseManagement
Listening on 0.0.0.0:8080
Client connected: 127.0.0.1:48536
Client connected: 127.0.0.1:51768
Client connected: 127.0.0.1:60720
```

List

```
For exit, enter: Exit
List
******* Response from the server *******
Token: You have no Product :)
*******      End of Response     *******
```

Create

```
Create
Enter product name: Meet
Enter product amount (press Enter for default 0): 20
******* Response from the server *******
Token: Product created successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: ID: 1, Name: Meet, Amount: 20
*******      End of Response     *******
```

Add

```
Add
Enter product name: Meet
Enter the amount to increase: 20
******* Response from the server *******
Token: Amount of product increased successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: ID: 1, Name: Meet, Amount: 40
*******      End of Response     *******
```

Reduce

```
Reduce
Enter product name: Meet
Enter the amount to increase: 10
******* Response from the server *******
Token: Amount of product reduced successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: ID: 1, Name: Meet, Amount: 30
*******      End of Response     *******
```

Remove

```
Remove
Enter product name: Meet
******* Response from the server *******
Token: Product removed successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: You have no Product :)
*******      End of Response     *******
```

Create in Two Client

```
Create
Enter product name: Meet
Enter product amount (press Enter for default 0): 20
******* Response from the server *******
Token: Product created successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: ID: 1, Name: Meet, Amount: 20
*******      End of Response     *******
```

```
Create
Enter product name: Test
Enter product amount (press Enter for default 0): 20
******* Response from the server *******
Token: Product created successfully
*******      End of Response     *******
List
******* Response from the server *******
Token: ID: 1, Name: Test, Amount: 20
*******      End of Response     *******
```