دستور کار آزمایشگاه سیستم های عامل

آزمایشگاه سیستم های عامل

آرین محسنی

آبان 1403

دستور کار آزمایشگاه سیستم های عامل

بخش 1

در این بخش باید مانند دستور کار یک کد IPC بنویسیم که میان پردازه ها تبادل اطلاعات وجود دارد.

OS_Lab > Lab4 > HW > Hw_part1 > C part1.c > main()

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

#define SHM_SIZE sizeof(int) * 2  // Shared memory size to hold two integers
#define COUNT 10                  // Number of random numbers to produce and consume

int main() {
    int shm_id;
    int *shared_memory;

    // Step 1: Create shared memory segment
    shm_id = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id < 0) {
        perror("shmget failed");
        exit(1);
    }

    // Step 2: Fork a new process
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {  // Child process (Consumer)
        // Attach to shared memory
        shared_memory = (int *)shmat(shm_id, NULL, 0);
        if (shared_memory == (int *)-1) {
            perror("shmat failed in consumer");
            exit(1);
        }
```

دستور کار آزمایشگاه سیستم های عامل

```c
        int sum = 0;

        for (int i = 0; i < COUNT; i++) {
            // Wait for producer to write a new number
            while (shared_memory[1] == 0) {
                usleep(100);  // Small delay to prevent busy waiting
            }

            // Read number from shared memory
            int num = shared_memory[0];
            printf("Consumer read: %d\n", num);
            sum += num;

            // Reset shared memory flag
            shared_memory[1] = 0;
        }

        printf("Total Sum: %d\n", sum);

        // Detach shared memory
        shmdt(shared_memory);
    } else {  // Parent process (Producer)
        // Attach to shared memory
        shared_memory = (int *)shmat(shm_id, NULL, 0);
        if (shared_memory == (int *)-1) {
            perror("shmat failed in producer");
            exit(1);
        }

        for (int i = 0; i < COUNT; i++) {
            int num = rand() % 100;  // Generate random number

            // Wait until consumer has read the previous number
            while (shared_memory[1] == 1) {
                usleep(100);  // Small delay to prevent busy waiting
            }

            // Write number to shared memory
            shared_memory[0] = num;
            shared_memory[1] = 1;
            printf("Producer wrote: %d\n", num);
        }

        // Wait for child process to finish
        wait(NULL);

        // Detach and remove shared memory
        shmdt(shared_memory);
        shmctl(shm_id, IPC_RMID, NULL);
    }

    return 0;
}
```

دستور کار آزمایشگاه سیستم های عامل

در این کد ابتدا یک حافظه مشترک که قرار است در آن بنویسیم و از آن بخوانیم ساخته می‌شود.

سپس با دستور فورک پردازه های متفاوت ایجاد می‌کنیم که پردازه فرزند به خواندن از آن حافظه و جمع کردن عدد داخل آن و پردازه والد به تولید عدد تصادفی در آن حافظه می‌پردازد.

خروجی :

```
arian@arian-VirtualBox:~/Desktop/University/OS/OS_Lab/Lab4/HW/Hw_part1$ ./part1
Producer wrote: 83
Consumer read: 83
Producer wrote: 86
Consumer read: 86
Producer wrote: 77
Consumer read: 77
Producer wrote: 15
Consumer read: 15
Producer wrote: 93
Consumer read: 93
Producer wrote: 35
Consumer read: 35
Producer wrote: 86
Consumer read: 86
Producer wrote: 92
Consumer read: 92
Producer wrote: 49
Consumer read: 49
Producer wrote: 21
Consumer read: 21
Total Sum: 637
```

دستور کار آزمایشگاه سیستم های عامل

بخش 2

در این آزمایش باید یک برنامه کاربردی مدیریت انبارگردانی (Warehouse management) را به وسیله‌ی زبان C پیاده‌سازی کنید. این برنامه دو بخش دارد: سرور و کاربر. سرور اطلاعات مربوط به اجناس انبار را نگه‌داری می‌کند و وقتی از سمت انبارگردان ،به عنوان کاربر، پیامی‌دریافت می‌کند بسته به نوع پیام عملیات مدنظر کاربر را انجام می‌دهد. اطلاعات اجناس یک انبار شامل یک لیست از اسامی‌اجناس موجود و مقدار موجودی هر یک می‌باشد.

برای این بخش کد کلاینت و سرور را به نحوی پیاده سازی میکنیم که تنها یک کاربر اجازه اتصال داشته باشد.

کد بخش کلاینت:

OS_Lab > Lab4 > HW > Hw_part2 > C client.c > main(int, char * [])

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s [server-hostname] [server-port] [client-name]\n", argv[0]);
        exit(1);
    }

    char *hostname = argv[1];
    int port = atoi(argv[2]);

    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    inet_pton(AF_INET, hostname, &server_addr.sin_addr);

    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(1);
    }

```

دستور کار آزمایشگاه سیستم های عامل

```
30      char buffer[BUFFER_SIZE];
31      while (1) {
32          printf("Enter command: ");
33          fgets(buffer, BUFFER_SIZE, stdin);
34          buffer[strcspn(buffer, "\n")] = 0;  // Remove newline character
35
36          send(client_socket, buffer, strlen(buffer), 0);
37          memset(buffer, 0, BUFFER_SIZE);
38
39          int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
40          if (bytes_received > 0) {
41              printf("Server: %s\n", buffer);
42          }
43
44          if (strcmp(buffer, "quit") == 0) {
45              break;
46          }
47      }
48
49      close(client_socket);
50      return 0;
51  }
52
```

:

دستور کار آزمایشگاه سیستم های عامل

کد بخش سرور :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define BUFFER_SIZE 1024
#define MAX_PRODUCTS 100

typedef struct {
    char name[50];
    int quantity;
} Product;

Product inventory[MAX_PRODUCTS];
int product_count = 0;

int find_product(const char *name) {
    for (int i = 0; i < product_count; i++) {
        if (strcmp(inventory[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}

void list_products(int client_socket) {
    char buffer[BUFFER_SIZE] = "Inventory List:\n";
    for (int i = 0; i < product_count; i++) {
        char product_info[100];
        snprintf(product_info, sizeof(product_info), "%s: %d\n", inventory[i].name, inventory[i].quantity);
        strcat(buffer, product_info);
    }
    send(client_socket, buffer, strlen(buffer), 0);
}

void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE];
    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) break;

        char command[50], name[50];
        int amount;

        sscanf(buffer, "%s %s %d", command, name, &amount);

        if (strcmp(command, "list") == 0) {
            list_products(client_socket);
        } else if (strcmp(command, "create") == 0) {
            if (find_product(name) != -1) {
                send(client_socket, "Error: Product already exists.\n", 30, 0);
            } else {
                inventory[product_count++] = (Product){.name = "", .quantity = amount};
                strcpy(inventory[product_count - 1].name, name);
                send(client_socket, "Product created successfully.\n", 30, 0);
            }
        } else if (strcmp(command, "add") == 0) {
            int index = find_product(name);
            if (index == -1) {
                send(client_socket, "Error: Product not found.\n", 27, 0);
            } else {
                inventory[index].quantity += amount;
                send(client_socket, "Quantity added successfully.\n", 29, 0);
            }
```

دستور کار آزمایشگاه سیستم های عامل

```
67          } else if (strcmp(command, "reduce") == 0) {
68              int index = find_product(name);
69              if (index == -1) {
70                  send(client_socket, "Error: Product not found.\n", 27, 0);
71              } else if (inventory[index].quantity < amount) {
72                  send(client_socket, "Error: Insufficient stock.\n", 27, 0);
73              } else {
74                  inventory[index].quantity -= amount;
75                  send(client_socket, "Quantity reduced successfully.\n", 31, 0);
76              }
77          } else if (strcmp(command, "remove") == 0) {
78              int index = find_product(name);
79              if (index == -1 || inventory[index].quantity != 0) {
80                  send(client_socket, "Error: Product not removable.\n", 30, 0);
81              } else {
82                  for (int i = index; i < product_count - 1; i++) {
83                      inventory[i] = inventory[i + 1];
84                  }
85                  product_count--;
86                  send(client_socket, "Product removed successfully.\n", 30, 0);
87              }
88          } else if (strcmp(command, "quit") == 0) {
89              break;
90          } else {
91              send(client_socket, "Error: Unknown command.\n", 24, 0);
92          }
```

```
93      }
94      close(client_socket);
95  }
96
97  int main(int argc, char *argv[]) {
98      if (argc != 2) {
99          fprintf(stderr, "Usage: %s [port]\n", argv[0]);
100         exit(1);
101     }
102
103     int port = atoi(argv[1]);
104     int server_socket = socket(AF_INET, SOCK_STREAM, 0);
105     struct sockaddr_in server_addr;
106
107     server_addr.sin_family = AF_INET;
108     server_addr.sin_port = htons(port);
109     server_addr.sin_addr.s_addr = INADDR_ANY;
110
111     bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
112     listen(server_socket, 1);
113
114     printf("Server listening on port %d...\n", port);
115     while (1) {
116         int client_socket = accept(server_socket, NULL, NULL);
117         if (client_socket >= 0) {
118             printf("Client connected.\n");
119             handle_client(client_socket);
120             printf("Client disconnected.\n");
121         }
122     }
123
124     close(server_socket);
125     return 0;
126 }
```

دستور کار آزمایشگاه سیستم های عامل

خروجی:

روی پورت 8080 سرور را اجرا می کنیم.

```
arian@arian-VirtualBox:~/Desktop/University/OS/OS_Lab/Lab4/HW/Hw_part2$ ./server 8080
Server listening on port 8080...
```

برای کلاینت

```
arian@arian-VirtualBox:~/Desktop/University/OS/OS_Lab/Lab4/HW/Hw_part2$ ./client server 8080 client_1
Enter command:
```

مثال از اضافه کردن و مشاهده و سپس حذف کالا:

```
arian@arian-VirtualBox:~/Desktop/University/OS/OS_Lab/Lab4/HW/Hw_part2$ ./client server 8080 client_1
Enter command: list
Server: Inventory List:

Enter command: create test 0
Server: Product created successfully.

Enter command: list
Server: Inventory List:
test: 0

Enter command: create product#20 20
Server: Product created successfully.

Enter command: list
Server: Inventory List:
test: 0
product#20: 20

Enter command: remove test
Server: Product removed successfully.

Enter command: list
Server: Inventory List:
```