



دانشگاه صنعتی شریف  
دانشکده مهندسی کامپیوتر

عنوان:

## پیاده سازی وب سرور چندریسه ای

اعضای گروه

محمد مهدی ابوترابی  
محمد علی خدا بنده لو  
علی ونکی فراهانی

نام درس

سیستم های عامل

نیم سال اول ۱۴۰۱-۱۴۰۲

نام استاد درس

حسین اسدی

**چکیده:** هدف اصلی در این پروژه، ایجاد یک معماری سرور-کلاینت برای مدیریت درخواست‌های ورودی است. قسمت سرور این معماری، بر پایه‌ی چند ریسمانی ساخته شده است. نکته‌ی قابل توجه دیگر درباره‌ی معماری سمت سرور این است که از یک حوضچه‌ی ریسمان برای مدیریت انجام درخواست‌ها انجام می‌شود. علت این امر این است که ساخت ریسمان‌ها فقط یک بار انجام شود و از دفعات بعدی، از ریسمان‌هایی که قبلاً ساخته شده‌اند استفاده کنیم. همچنین برای مدیریت درخواست‌ها، از چند سیاست مختلف مانند Priority Scheduling, EDF, FCFS و ... استفاده شده است. در نهایت، بر اساس داده‌های تولید شده در حین اجرای برنامه، تحلیل‌های مختلف روی عملکرد و کارایی برنامه انجام شده است.

**واژه‌های کلیدی:** حوضچه ریسمان، چندریسمانی، سرور، کلاینت

## ۱ مقدمه

پروژه‌ی انجام شده، به اختصار، پیاده‌سازی یک وب‌سرور بر اساس معماری حوضچه‌ی ریسمان برای مدیریت درخواست‌های ورودی می‌باشد. این مسئله دارای اهمیت زیادی می‌باشد، زیرا همواره یکی از دغدغه‌های اصلی در بحث طراحی وب‌سرورها، پردازش درخواست‌ها به صورت موازی برای بالا بردن کارایی و استفاده‌ی حداکثری از منابع بوده است. در این پروژه قصد پیاده‌سازی یک وب‌سرور با چندین ریسمان را خواهیم داشت. در واقع، وب‌سرور موردنظر بایستی این قابلیت را داشته باشد که بتواند به چندین وظیفه به صورت همزمان با استفاده از ریسمان‌هایی که به صورت هم‌روند اجرا می‌شوند، پاسخ دهد.

برای پیاده‌سازی این پروژه، ابتدا حالت ساده شده آن یعنی یک وب‌سرور تک ریسمانه را که در هر زمان تنها یک وظیفه را انجام می‌دهد، پیاده‌سازی می‌کنیم. در این مرحله سعی می‌کنیم به دلیل سادگی کار ابتدا در نظر بگیریم که وب‌سرور مورد نظر به صورت non-primitive دستورات را اجرا می‌کند. یعنی تا زمانی که اجرای یک وظیفه به اتمام نرسیده باشد، اجرای وظیفه دیگر را آغاز نمی‌کند. برای پیاده‌سازی این قسمت نیاز داریم که ابتدا لیست درخواست‌ها را درون صفی قرار دهیم و با استفاده از یک الگوریتم مانند FCFS درخواست‌ها را به ترتیب اجرا کنیم. با انجام دادن این بخش، در نهایت وب‌سروری خواهیم داشت که با استفاده از یک ریسمان درخواست‌های دریافتی خود را که درون یک صف ذخیره کرده است، انجام می‌دهد. هم‌چنین سعی می‌شود که در این قسمت از استراتژی و الگوریتمی برای انتخاب وظایف استفاده کنیم که در نهایت مشکل گرسنگی برای وظایف به وجود نیاید. در نهایت نیز سعی می‌کنیم که وب‌سرور را به شکل preemptive

دریابوریم تا در صورتی که در اجرای یک وظیفه مشکلی به وجود آمد، تک ریسمان موجود در سرور برای همیشه درگیر آن نباشد.

در قسمت دوم پروژه قصد داریم که وب سرور تک ریسمان طراحی شده را به شکل چندین ریسمان دریابوریم. در این حالت نیازمند یک استخر ریسمان هستیم که با استفاده از آن تعدادی ریسمان را ذخیره کرده و در هر لحظه برای اجرای یک وظیفه، ریسمانی که در حال اجرای وظیفه دیگری نمی باشد را انتخاب کنیم. مهم ترین ویژگی این قسمت این است که نبایست اجازه دهیم ریسمانی برای مدت طولانی بیکار بماند. در حقیقت، بایستی سعی کنیم به گونه ای این قسمت پیاده سازی شود که اگر ریسمانی بیکار باشد و وظیفه ای برای اجرا باقی مانده باشد، آن وظیفه به ریسمان مربوطه داده شود. برای پیاده سازی این قسمت بهتر است که یک ریسمان تحت عنوان ریسمان مستر داشته باشیم که وظایف را به ریسمان های دیگر محول می کند.

## ۱-۱ تعریف مسئله

در این مسئله، به طور کلی دو ماژول اصلی وجود دارد. ماژول اول که تحت عنوان کلاینت شناخته می شود، وظیفه دارد تا درخواست هایی را بر اساس نیازهای مسئله و تحلیل هایی که نیاز است تا انجام شوند تولید کرده و به سرور تحویل دهد. نکته ای قابل توجه درباره ی درخواست های ارسالی این است که ماهیت آنها مورد بحث نیستند و می توانند صرفاً یک درخواست ساده برای جمع دود عدد باشند. در ادامه این درخواست باید در سمت سرور وارد یک صف مربوط به درخواست ها شود. در سمت سرور نیز چند تابع مختلف وجود دارد. یکی از توابع مسئول برداشتن درخواست ها از صف ورودی و تحویل دادن آن به یک ریسمان است. نکته ای قابل توجه این است که در سمت سرور، یک حوضچه ی ریسمان در ابتدا ساخته می شود و در تمام طول برنامه از این ریسمان های ساخته شده استفاده می شود. در نهایت، برنامه ی نوشته شده توسط ابزارهای موجود برای استفاده ی درست از منابع بررسی می شود. همچنین در طول برنامه، داده هایی مانند زمان رسیدن هر درخواست، زمان تحویل آن به یک ریسمان و زمان پایان پردازش آن درخواست توسط سرور تولید می شود. در نهایت از این داده های تولید شده برای تحلیل و بررسی عملکرد سرور استفاده می شود.

## ۲-۱ اهمیت موضوع

پیاده سازی و تحلیل چنین برنامه دارای اهمیت بسیاری می باشد. از جمله ی موارد حائز اهمیت، می توان به نکات زیر اشاره کرد:

- رسیدگی همزمان به درخواست‌ها و افزایش کارایی برنامه
- انتخاب بهترین الگوریتم برای رسیدگی به درخواست‌ها با توجه به آمارهای تولید شده
- بررسی کد نوشته شده توسط نرم‌افزارهای مختلف و آگاهی نسبت به مشکلات و اشتباهات متداول در روند توسعه‌ی کد

## ۲ پیاده‌سازی

در این قسمت، به بررسی پیاده‌سازی کد کلاینت و سرور می‌پردازیم.

### ۱-۲ پیاده‌سازی کلاینت

در این قسمت به شرح و بررسی پیاده‌سازی کد سمت کلاینت می‌پردازیم. همانطور که در شکل ۱ مشاهده می‌کنید، مقادیر ip و port به گونه‌ای تعریف شده‌اند که کلاینت، درخواست‌هایش را به پورت 8080 و روی localhost ارسال کند تا توسط سرور دریافت شوند. در این قسمت توجه کنید که آدرس آپی ۱۲۷.۰.۰.۱ آدرس مربوط به host local می‌باشد که درخواست‌های ارسال شده به این آدرس در واقع بر روی همان ای‌host که درخواست را ارسال کرده است، دریافت می‌شوند.

```

8
9  #define PORT 8080
10 #define SERVER_ADDRESS "127.0.0.1"
11

```

شکل ۱

در ادامه، به شرح تابع اصلی کد سمت کلاینت می‌پردازیم. همانطور که در شکل ۳ مشاهده می‌کنید، این تابع ابتدا مقادیر را از کاربر دریافت می‌کند و سپس توسط تابعی که در ادامه توضیح خواهیم داد، آنها را برای سرور ارسال می‌کند. هر کدام از ورودی‌ها را به صورت مختصر توضیح می‌دهیم:

- **Enter two numbers:** در این قسمت، دو ورودی‌ای که قرار است عملیات ریاضی (در اینجا جمع) روی آنها انجام شود را به عنوان ورودی از کاربر دریافت می‌کنیم.

- **Enter execution time**: این پارامتر، زمان انجام تسک را مشخص می‌کند. البته لازم به ذکر است که این زمان انجام یک مقدار فرضی و رندوم است و صرفاً برای تست کردن و اطمینان از عملکرد سرور از آن استفاده می‌کنیم.
- **Enter deadline of job**: همانطور که از اسم آن نیز مشخص است، در این قسمت، deadline تسک ورودی را از کاربر دریافت می‌کنیم و از آن برای پیاده‌سازی الگوریتم‌هایی که نیاز به deadline دارند در سمت سرور استفاده می‌کنیم.
- **Enter Priority (lower number means higher priority)**: پارامتر نهایی‌ای که از کاربر دریافت می‌کنیم، اولویت تسک ارسالی است که از آن برای مشخص کردن اولویت‌ها در زمان Priority Scheduling در سمت سرور استفاده می‌کنیم.

```

63
64 int main(int argc, char *argv[])
65 {
66     int nums[5];
67     pthread_t thread;
68
69     printf("Enter two numbers:\n");
70     scanf("%d %d", &nums[0], &nums[1]);
71
72     printf("Enter execution time:\n");
73     scanf("%d", &nums[2]);
74
75     printf("Enter deadline of job:\n");
76     scanf("%d", &nums[3]);
77
78     printf("Enter Priority (lower number means higher priority):\n");
79     scanf("%d", &nums[4]);
80
81     if (pthread_create(&thread, NULL, send_data, (void*)nums) < 0)
82     {
83         printf("Can not create a new thread.\n");
84         return 1;
85     }
86
87     pthread_join(thread, NULL);
88
89     return 0;
90 }

```

شکل ۲

در شکل ۲، می‌توانید یک نمونه از اجرای کد سمت کلاینت و نحوه‌ی گرفتن ورودی‌ها از کاربر را مشاهده کنید. در این نمونه، ابتدا دو عدد از کاربر برای انجام عملیات ریاضی دریافت شده است. پس از دریافت دو عدد، به ترتیب زمان اجرای تسک، ددلاین تسک و اولویت آن از کاربر دریافت شده‌اند.

```
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$ ./client
Enter two numbers:
67
98
Enter execution time:
25
Enter deadline of job:
5
Enter Priority (lower number means higher priority):
3
Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 6
Received message from server: Done: 67 + 98 = 165
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$
```

شکل ۳

```
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$ ./client
Enter two numbers:
67
768
Enter execution time:
55
Enter deadline of job:
231
Enter Priority (lower number means higher priority):
4
Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 5
Received message from server: Done: 67 + 768 = 835
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$
```

شکل ۴

```
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$ ./client
Enter two numbers:
34
56
Enter execution time:
35
Enter deadline of job:
23
Enter Priority (lower number means higher priority):
5
Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 4
Received message from server: Done: 34 + 56 = 90
aboots@DESKTOP-QUS76G0:~/code/aboots/OS-Project$
```

شکل ۵

بعد از دریافت ورودی‌ها از کاربر، سراغ ارسال آنها برای سرور می‌رویم.

وظیفه‌ی ارتباط با سرور بر عهده‌ی یک تابع جداگانه تحت عنوان `send_data` می‌باشد که در یک `thread` جداگانه اجرا می‌شود. در این تابع، به ترتیب، مراحل مورد نیاز برای اتصال به سرور انجام می‌شوند و در صورت بروز خطا در هر یک از مراحل اتصال، خطای مربوطه چاپ شده و تابع به پایان می‌رسد. بعد از اطمینان از اتصال به سرور، اعداد دریافت شده از کاربر در تابع `main`، در یک `buffer` قرار گرفته و برای سرور ارسال می‌شوند. در شکل ۴، همانطور که توضیح داده شد، پیاده‌سازی مربوط به اتصال به سرور و اطمینان از اتصال در مراحل مختلف و همچنین چاپ خطا را مشاهده می‌کنید.

```
11
12 void* send_data(void* data)
13 {
14     int* numbers = (int*)data;
15     int socket_desc, c;
16     struct sockaddr_in server;
17
18     socket_desc = socket(AF_INET, SOCK_STREAM, 0);
19     if (socket_desc == -1)
20     {
21         printf("Can not create socket.\n");
22         return 0;
23     }
24
25     server.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);
26     server.sin_family = AF_INET;
27     server.sin_port = htons(PORT);
28
29     if (connect(socket_desc, (struct sockaddr *)&server, sizeof(server)) < 0)
30     {
31         printf("Connection failed.\n");
32         return 0;
33     }
34
35     printf("Client Connected to server.\n");
36
```

شکل ۶: اتصال به سرور و چاپ کردن خطا در صورت بروز مشکل

در ادامه، تابع منتظر رسیدن جواب از سرور می‌ماند. در ابتدا، کلاینت منتظر می‌ماند تا مطمئن شود که پیام ارسالی، با موفقیت توسط سرور دریافت شده است. بعد از اطمینان از دریافت شدن اطلاعات توسط سرور، کلاینت دوباره منتظر می‌ماند تا جواب مربوط به عملیات ریاضی از سرور برگردد. بعد از دریافت جواب از سرور، جواب دریافت شده چاپ شده و سپس تابع به پایان می‌رسد. در تصویر ۵ نیز، پیاده‌سازی مربوط به آماده‌سازی داده‌های دریافت شده از کاربر و ارسال آنها به سرور و در ادامه، دریافت جواب از سرور قابل مشاهده است.

```
37
38     char buffer[sizeof(int) * 5];
39     for (int i = 0; i < 5; i++)
40     {
41         int num = htonl(numbers[i]);
42         memcpy(buffer + (i * sizeof(int)), &num, sizeof(int));
43     }
44
45     send(socket_desc, buffer, sizeof(buffer), 0);
46     printf("Numbers sent to server.\n");
47
48     char message[100];
49     memset(message, 0, sizeof(message));
50     recv(socket_desc, message, sizeof(message), 0);
51
52     printf("Received message from server: %s\n", message);
53
54     // received answer from server
55
56     memset(message, 0, sizeof(message));
57     recv(socket_desc, message, sizeof(message), 0);
58
59     printf("Received message from server: %s\n", message);
60
61     return 0;
62 }
63
```

شکل ۷



## ۲-۲ پیاده‌سازی سرور

در این قسمت، توضیحات مربوط به پیاده‌سازی کد سرور ارائه خواهد شد. همانطور که قبلاً نیز اشاره کردیم، در پیاده‌سازی سمت سرور، یک بخش کلی وجود دارد که مربوط به دریافت اطلاعات و مدیریت کردن ریسمان‌هاست. همچنین هنگام اجرای کد سرور، در ابتدای کار، می‌توان الگوریتم مورد نظر که به اساس آن زمانبندی ریسمان‌ها انجام می‌شود را انتخاب کرد.

### ۱-۲-۲ متغیرهای گلوبال و داده‌ساختارها

در این بخش، به صورت مختصر، متغیرهای گلوبال و داده‌ساختارهای استفاده شده در پیاده‌سازی سرور را مورد بررسی قرار می‌دهیم. همانطور که در تصویر ۸ مشاهده می‌کنید، مقادیری به صورت گلوبال برای استفاده در طور برنامه تعریف شده‌اند. این مقادیر عبارتند از:

- **Port**: این مقدار نشان‌دهنده‌ی این است که سرور روی چه پورتی منتظر درخواست‌های ارسالی از سمت کلاینت می‌ماند.
- **Thread Pool Size**: همانطور که از نام این مقدار مشخص است، تعداد ریسمان‌های موجود در حوضچه‌ی ریسمان، با توجه به این مقدار مشخص می‌شود.
- **Max Clients**: این مقدار مشخص می‌کند که حداکثر چند کلاینت می‌توانند به صورت همزمان به سرور متصل شوند.
- **Max Clients**: این مقدار مشخص می‌کند که حداکثر چند کلاینت می‌توانند به صورت همزمان به سرور متصل شوند.
- **Mem Size**: مشخص‌کننده‌ی حداکثر تعداد درخواست‌هایی که می‌توانند به صورت همزمان در انتظار پردازش باشند یا به طور معادل، طول صف.
- **Base Log File Name**: در این مقدار، پیشوند نام مورد نظر برای فایل log که قرار است در طول اجرای برنامه تولید شود مشخص می‌شود.

در ادامه، متغیرهای گلوبال استفاده شده در برنامه شرح داده خواهند شد:

- **workers data**: این آرایه که سائز اصلی آن برابر با تعداد ریسمان‌های موجود در حوضچه‌ی ریسمان است، داده‌های مربوط به هر ریسمان کارگر را در خودش ذخیره می‌کند. مقدار اول

نشان‌دهنده‌ی مشغول یا در حال کار بودن ریسمان کارگر است. مقدار دوم، مقدار سوکت مربوط به کلاینتی که ریسمان کارگر در حال انجام درخواست مربوط به آن می‌باشد را مشخص می‌کند. مقادیر سوم و چهارم دو عدد ارسال شده از سمت کلاینت را در خود نگه می‌دارند و در نهایت مقدار پنجم، مدت زمان اجرای جاب را نشان می‌دهد.

- **workers time data**: از این آرایه، برای انتقال داده‌های مربوط به زمان اجرا و انتظار تسک‌ها به ریسمان استفاده می‌شود.

- **data array**: داده‌های ورودی که از سمت کلاینت ارسال شده‌اند، برای پردازش اولیه در این آرایه قرار می‌گیرند.

- **arriving times**: زمان رسیدن همه‌ی تسک‌ها در این آرایه ذخیره می‌شود.

- **semaphore**: یک آرایه از سمافورها برای قفل کردن تردهای بدون تسک در طول اجرای برنامه.

- **main index, master index**: دو عدد یکتا و متفاوت که به ریسمان‌های main و master تعلق می‌گیرند تا از یکدیگر قابل تفکیک باشند.

- **main index, master index**: دو عدد یکتا و متفاوت که به ریسمان‌های main و master تعلق می‌گیرند تا از یکدیگر قابل تفکیک باشند.

- **strategy**: الگوریتمی که تسک‌ها بر اساس آن زمان‌بندی می‌شوند. این مقدار در ابتدای اجرای برنامه توسط کاربر انتخاب می‌شود.

- **start**: زمان شروع اجرای برنامه.

- **log file lock**: یک mutex برای جلوگیری از مشکلات دسترسی برای نوشتن لاگ‌ها در داخل فایل.

در نهایت، به شرح داده‌ساختاری که برای پیاده‌سازی درخواست‌ها از آن استفاده کرده‌ایم می‌پردازیم. این داده ساختار که node نام دارد، به صورت کلی به این شکل عمل می‌کند که تمام اطلاعات مربوط به یک تسک را در خودش ذخیره می‌کند. علاوه بر این، می‌تواند به node بعدی خودش نیز اشاره کند. به این ترتیب، با استفاده از این داده ساختار، ابتدا مقادیر مربوط به تسک‌ها که ذخیره شده‌اند را دریافت می‌کنیم و سپس آنها را درون صف قرار می‌دهیم.

در ادامه، می‌توانیم با توجه به الگوریتم‌هایی که برای زمان‌بندی درخواست‌های ورودی داشتیم، آنها را به صف منتقل کنیم و سپس با استفاده از ریسمان‌های کارگر آنها را اجرا کنیم. متغیرها، آرایه‌ها و داده‌ساختارهای تشریح شده در این بخش را می‌توانید در شکل ۸ مشاهده کنید.

```
13
14 #define PORT 8080
15 #define THREAD_POOL_SIZE 2
16 #define MAX_CLIENTS 50
17 #define MEM_SIZE 10000
18 #define BASE_LOG_FILE_NAME "server_log"
19
20 int timeout = 5; // timeout in seconds
21 fd_set readset;
22 struct timeval tv;
23
24 int workers_data[THREAD_POOL_SIZE][5]; // 0: is_free, 1: client_socket, 2: first_num, 3: second_num, 4: execution_time
25 double workers_time_data[THREAD_POOL_SIZE][2]; // 0: arriving time, 1: waiting_time
26 int data_array[MEM_SIZE][6]; // 0: client_socket, 1: first_num, 2: second_num 3: execution time, 4: deadline, 5: priority
27 double arriving_times[MEM_SIZE];
28 sem_t semaphore[THREAD_POOL_SIZE];
29 int main_index;
30 int master_index;
31 int strategy;
32 clock_t start;
33 pthread_mutex_t log_file_lock;
34
35 typedef struct node
36 {
37     int first_num;
38     int second_num;
39     int client_socket;
40     int execution_time;
41     int deadline;
42     int priority;
43     double arrived_time;
44     struct node *next;
45 } node;
46
```

شکل ۸

این تابع در ابتدا یک ورودی برای انتخاب الگوریتم زمان‌بندی از کاربر دریافت می‌کند. در ادامه، عملیات‌های مورد نیاز برای initialize کردن لاک‌ها و متغیرها انجام می‌شود و سپس، عملیات‌های مورد نیاز برای پابلیش کردن سرور روی پورت مورد نظر انجام می‌شود. مانند پیاده‌سازی سمت کلاینت، در صورت بروز خطا در هر کدام از مراحل اتصال یا پابلیش، با چاپ خطای مربوطه اجرای کد متوقف می‌شود.

```

311
312 int main(int argc, char const *argv[])
313 {
314     printf("Select your scheduling algorithm:\n1- FCFS\n2- Priority\n3- SJF\n4- EDF\n");
315     scanf("%d", &strategy);
316
317     if (pthread_mutex_init(&log_file_lock, NULL) != 0)
318     {
319         printf("\n mutex init for file has failed\n");
320         return 1;
321     }
322
323     int server_socket = socket(AF_INET, SOCK_STREAM, 0);
324     if (server_socket < 0)
325     {
326         perror("Error creating socket");
327         return 1;
328     }
329
330     struct sockaddr_in server_address;
331     memset(&server_address, 0, sizeof(server_address));
332     server_address.sin_family = AF_INET;
333     server_address.sin_port = htons(PORT);
334     server_address.sin_addr.s_addr = htonl(INADDR_ANY);
335     if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
336     {
337         perror("Error binding socket");
338         return 1;
339     }
340     if (listen(server_socket, MAX_CLIENTS) < 0)
341     {
342         perror("Error listening on socket");
343         return 1;
344     }

```

شکل ۹

بعد از اجرای موارد گفته شده، وارد یک حلقه‌ی while که همیشه در حال اجراست می‌شود. بعد از ورود به حلقه، منتظر کلاینت می‌مانیم تا متصل شود. بعد از اتصال کلاینت، با در نظر گرفتن یک مقدار به عنوان تایم‌اوت، اجازه می‌دهیم تا کلاینت، داده‌های خودش را ارسال کند. بعد از دریافت اطلاعات از کلاینت، آنها را split می‌کنیم و سپس، یک پیام برای کلاینت ارسال می‌کنیم که اطلاعات با موفقیت دریافت شده‌اند. بعد از دریافت این اطلاعات، آنها را وارد data array می‌کنیم که قبلاً توضیحات مربوط به آن ارائه شده است.

```

358
359     while (1)
360     {
361         struct sockaddr_in client_address;
362         socklen_t client_address_len = sizeof(client_address);
363         int client_socket = accept(server_socket, (struct sockaddr *)&client_address, &client_address_len);
364         if (client_socket < 0)
365         {
366             perror("Error accepting connection");
367             continue;
368         }
369
370         printf("client %d is accepted\n", client_socket);
371         FD_ZERO(&readset);
372         FD_SET(client_socket, &readset);
373         tv.tv_sec = timeout;
374         tv.tv_usec = 0;
375
376         int retval = select(client_socket + 1, &readset, NULL, NULL, &tv);
377         if (retval == -1)
378         {
379             printf("Error receiving data from client %d:\n", client_socket);
380         }

```

شکل ۱۰

```

380     }
381     else if (retval)
382     {
383         char buffer[sizeof(int) * 5];
384         recv(client_socket, buffer, sizeof(buffer), 0);
385
386         clock_t request_received = clock();
387         double elapsed_time = (double)(request_received - start) / CLOCKS_PER_SEC;
388
389         int first_num, second_num, priority, deadline, execution_time;
390         memcpy(&first_num, buffer, sizeof(int));
391         memcpy(&second_num, buffer + sizeof(int), sizeof(int));
392         memcpy(&execution_time, buffer + (sizeof(int) * 2), sizeof(int));
393         memcpy(&deadline, buffer + (sizeof(int) * 3), sizeof(int));
394         memcpy(&priority, buffer + (sizeof(int) * 4), sizeof(int));
395
396         first_num = ntohl(first_num);
397         second_num = ntohl(second_num);
398         execution_time = ntohl(execution_time);
399         deadline = ntohl(deadline);
400         priority = ntohl(priority);
401         printf("Received numbers from client %d.\n", client_socket);

```

شکل ۱۱

```

402
403     char message[100];
404     sprintf(message, "Numbers Received Successfully, your client id is %d\n", client_socket);
405
406     send(client_socket, message, strlen(message), 0);
407
408     arriving_times[main_index % MEM_SIZE] = elapsed_time;
409     data_array[main_index % MEM_SIZE][0] = client_socket;
410     data_array[main_index % MEM_SIZE][1] = first_num;
411     data_array[main_index % MEM_SIZE][2] = second_num;
412     data_array[main_index % MEM_SIZE][3] = execution_time;
413     data_array[main_index % MEM_SIZE][4] = deadline;
414     data_array[main_index % MEM_SIZE][5] = priority;
415     main_index++;
416 }
417 else
418 {
419     printf("Timeout happened for client %d:\n", client_socket);
420 }
421 }
422
423 pthread_detach(master_thread_id);
424 pthread_mutex_destroy(&log_file_lock);
425 return 0;
426 }

```

شکل ۱۲

این تابع، در ابتدا با توجه به مقدار Thread Pool، ریسمان ایجاد می‌کند و همچنین سمافور مربوط به آنها را initialize می‌کند. در این تابع نیز مانند تابع main یک حلقه‌ی while با شرط همواره درست داریم. بعد از ورود به این حلقه، همواره چک می‌کند که آیا ریسمان کارگری وجود دارد که مشغول انجام کار نباشد یا خیر. در صورت پیدا کردن چنین ریسمانی، یک تسک را با توجه به الگوریتم از صف خارج کرده و در اختیار آن ریسمان قرار می‌دهد و به سمافور مربوط به آن ریسمان نیز سیگنال می‌دهد تا مشغول به انجام کار شود.

```

162 void *master_thread(void *arg)
163 {
164     int fcfs_counter = 0;
165     pthread_t worker_thread_id[THREAD_POOL_SIZE];
166     memset(worker_thread_id, 0, sizeof(worker_thread_id));
167
168     int i;
169     for (i = 0; i < THREAD_POOL_SIZE; i++)
170     {
171         int *id = (int *)malloc(sizeof(int));
172         *id = i;
173         if (pthread_create(&worker_thread_id[i], NULL, worker_thread, id) != 0)
174         {
175             printf("Error creating worker thread with id %d\n", i);
176             i--;
177         }
178
179         if (sem_init(&semaphore[i], 0, 0) != 0)
180         {
181             printf("\n semaphore init has failed for worker thread with id %d\n", i);
182             i--;
183         }
184     }
185

```

شکل ۱۳

```

186     node *pq = NULL;
187     while (1)
188     {
189         for (int i = 0; i < THREAD_POOL_SIZE; i++)
190         {
191             if (main_index != master_index)
192             {
193                 int client_socket = data_array[master_index % MEM_SIZE][0];
194                 int num1 = data_array[master_index % MEM_SIZE][1];
195                 int num2 = data_array[master_index % MEM_SIZE][2];
196                 int execution_time = data_array[master_index % MEM_SIZE][3];
197                 int deadline = data_array[master_index % MEM_SIZE][4];
198                 int priority = data_array[master_index % MEM_SIZE][5];
199                 double arrived_time = arriving_times[master_index % MEM_SIZE];
200                 master_index++;
201
202                 int final_priority;
203                 if (strategy == 1)
204                 {
205                     final_priority = fcfs_counter;
206                     fcfs_counter++;
207                 }
208                 else if (strategy == 2)
209                 {
210                     final_priority = priority;
211                 }
212                 else if (strategy == 3)
213                 {
214                     final_priority = execution_time;
215                 }
216                 else if (strategy == 4)
217                 {
218                     final_priority = deadline;
219                 }
220

```

شکل ۱۴



این تابع از دو تابع قبلی کارکرد ساده‌تری دارد. در این تابع نیز یک حلقه‌ی همواره درست داریم که وارد آن می‌شویم و ابتدای آن، تابع توسط سمافور متوسط می‌شود تا زمانی تابع master آن را بیدار کند. بعد از بیدار شدن، بررسی می‌کنند که آیا واقعا master آنها را به درستی بیدار کرده است یا نه و در صورت درست بودن، داده‌های مربوط به خودشان را دریافت می‌کنند. بعد از دریافت اطلاعات، عملیات ریاضی را انجام می‌دهد و آنها را برای کاربر ارسال می‌کند و بعد از ارسال اطلاعات نیز داده‌های زمانی تولید شده را داخل فایل اضافه می‌کند. برای نوشتن در فایل نیز یک لاک وجود دارد که هر ریسمان موقع نوشتن در فایل آن را می‌گیرد و بعد از انجام عملیاتش، لاک را آزاد می‌کند. در نهایت نیز به تابع master اطلاع می‌دهد که مشغول کار نیست و دوباره به ابتدای حلقه برگشته و منتظر رسیدن تسک جدید می‌ماند.

```

109
110 void *worker_thread(void *args)
111 {
112     int id = *((int *)args);
113     workers_data[id][0] = 1;
114
115     while (1)
116     {
117         sem_wait(&semaphore[id]);
118         if (workers_data[id][0] == 0)
119         {
120             int num1 = workers_data[id][1];
121             int num2 = workers_data[id][2];
122             int client_socket = workers_data[id][3];
123             int execution_time = workers_data[id][4];
124             int res = num1 + num2;
125             printf("Client with id %d is assigned to worker with id %d\n", client_socket, id + 1);
126             sleep(execution_time);
127
128             char message[100];
129             sprintf(message, "Done: %d + %d = %d\n", num1, num2, res);
130
131             send(client_socket, message, strlen(message), 0);
132
133             clock_t end = clock();
134             double total_time = (double)(end - start) / CLOCKS_PER_SEC;
135             double total_time_in_system = total_time - workers_time_data[id][0];
136
137             sprintf(message, "Done for client_socket %d : %d + %d = %d", client_socket, num1, num2, res);
138             printf("%s\n", message);
139
140             time_t now = time(NULL);
141             struct tm *timeinfo = localtime(&now);
142             char buffer[80];
143             char buffer2[100];

```

شکل ۱۵

## ۳-۲ بررسی توسط valgrind

بعد از به اتمام رسیدن پیاده‌سازی کد سرور، این برنامه توسط نرم‌افزار valgrind تحلیل شد. این نرم‌افزار کمک می‌کند تا در صورت وجود memory leak در برنامه، آنرا پیدا و برطرف کنیم. طبق گزارش valgrind، برنامه ایراد جدی‌ای از نظر نظر memory leak نداشت. در ادامه، تصاویر مربوط به تحلیل توسط valgrind را مشاهده می‌کنید.

```
==8784== Process terminating with default action of signal 2 (SIGINT)
==8784==   at 0x48664FF: accept (accept.c:26)
==8784==   by 0x10A6C3: main (in /home/aboots/code/aboots/OS-Project/server)
==8784==
==8784== HEAP SUMMARY:
==8784==   in use at exit: 824 bytes in 5 blocks
==8784== total heap usage: 20 allocs, 15 frees, 14,124 bytes allocated
==8784==
==8784== 272 bytes in 1 blocks are possibly lost in loss record 2 of 3
==8784==   at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==8784==   by 0x40149DA: allocate_dtv (dl-tls.c:286)
==8784==   by 0x40149DA: _dl_allocate_tls (dl-tls.c:532)
==8784==   by 0x485C322: allocate_stack (allocatestack.c:622)
==8784==   by 0x485C322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==8784==   by 0x10A64B: main (in /home/aboots/code/aboots/OS-Project/server)
==8784==
==8784== 544 bytes in 2 blocks are possibly lost in loss record 3 of 3
==8784==   at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==8784==   by 0x40149DA: allocate_dtv (dl-tls.c:286)
==8784==   by 0x40149DA: _dl_allocate_tls (dl-tls.c:532)
==8784==   by 0x485C322: allocate_stack (allocatestack.c:622)
==8784==   by 0x485C322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==8784==   by 0x109CDE: master_thread (in /home/aboots/code/aboots/OS-Project/server)
==8784==   by 0x485B608: start_thread (pthread_create.c:477)
==8784==   by 0x4995132: clone (clone.S:95)
==8784==
==8784== LEAK SUMMARY:
==8784==   definitely lost: 0 bytes in 0 blocks
==8784==   indirectly lost: 0 bytes in 0 blocks
==8784==   possibly lost: 816 bytes in 3 blocks
==8784==   still reachable: 8 bytes in 2 blocks
==8784==   suppressed: 0 bytes in 0 blocks
==8784== Reachable blocks (those to which a pointer was found) are not shown.
==8784== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==8784==
==8784== For lists of detected and suppressed errors, rerun with: -s
==8784== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

شکل ۱۶

## ۴-۲ تحلیل و بررسی پیاده‌سازی

برای بررسی کد پیاده‌سازی شده به این صورت عمل شد که یک سرور ایجاد شده و سپس ۸ کلاینت به آن متصل شدند. بعد از اتصال کلاینت‌ها به سرور، هر کدام یک تسک تولید کرده و برای سرور ارسال می‌کنند. در ادامه، نتایج مربوط به هر کدام از الگوریتم‌ها را مشاهده می‌کنید.

### ۱-۴-۲ الگوریتم FCFS

همانطور که از اسم این الگوریتم نیز مشخص است، اولویت را به تسکی می‌دهد که زودتر از بقیه وارد شده است. در ادامه، تصاویر مربوط به اتصال ۸ کلاینت را مشاهده می‌کنید.

```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
1
2
Enter execution time:
25
Enter deadline of job:
20
Enter Priority (lower number means higher priority):
2
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 240
Received message from server: Done: 1 + 2 = 3

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
23
25
Enter execution time:
22
Enter deadline of job:
66
Enter Priority (lower number means higher priority):
8
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 244
Received message from server: Done: 23 + 25 = 48
```

شكل ١٧

```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
12
9
Enter execution time:
35
Enter deadline of job:
99
Enter Priority (lower number means higher priority):
8
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 248
Received message from server: Done: 12 + 9 = 21

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
89
Enter execution time:
32
Enter deadline of job:
125
Enter Priority (lower number means higher priority):
1
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 252
Received message from server: Done: 14 + 89 = 103
```

شكل ١٨

```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
12
52
Enter execution time:
45
Enter deadline of job:
87
Enter Priority (lower number means higher priority):
1
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 256
Received message from server: Done: 12 + 52 = 64

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
3
9
Enter execution time:
14
Enter deadline of job:
98
Enter Priority (lower number means higher priority):
6
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 260
Received message from server: Done: 3 + 9 = 12
```

شكل ١٩

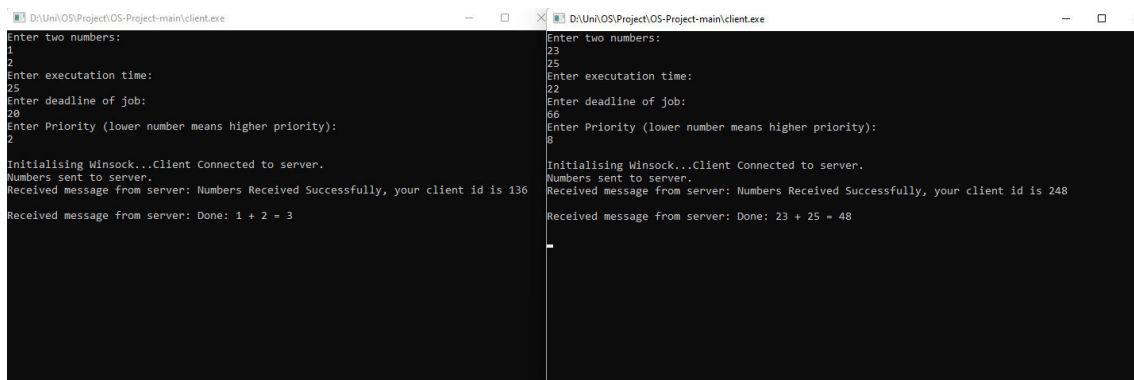
شکل ۲۰

```
Select your scheduling algorithm:
1- FCFS
2- Priority
3- SJF
4- EDF
1
Server Successfully binded to port 8080 and listen on it
client 240 is accepted
Received numbers from client 240.
Client with id 240 is assigned to worker with id 1
client 244 is accepted
Received numbers from client 244.
Client with id 244 is assigned to worker with id 2
client 248 is accepted
Received numbers from client 248.
client 252 is accepted
Received numbers from client 252.
client 256 is accepted
Received numbers from client 256.
client 260 is accepted
Received numbers from client 260.
client 264 is accepted
Received numbers from client 264.
client 268 is accepted
Received numbers from client 268.
Done for client_socket 244 : 23 + 25 = 48
Client with id 248 is assigned to worker with id 2
Done for client_socket 240 : 1 + 2 = 3
Client with id 252 is assigned to worker with id 1
Done for client_socket 252 : 14 + 89 = 103
Client with id 256 is assigned to worker with id 1
Done for client_socket 248 : 12 + 9 = 21
Client with id 260 is assigned to worker with id 2
Done for client_socket 260 : 3 + 9 = 12
Client with id 264 is assigned to worker with id 2
Done for client_socket 256 : 12 + 52 = 64
Client with id 268 is assigned to worker with id 1
Done for client_socket 264 : 5 + 15 = 20
Done for client_socket 268 : 14 + 32 = 46
```

شکل ۲۱

## ۲-۴-۲ الگوریتم Priority

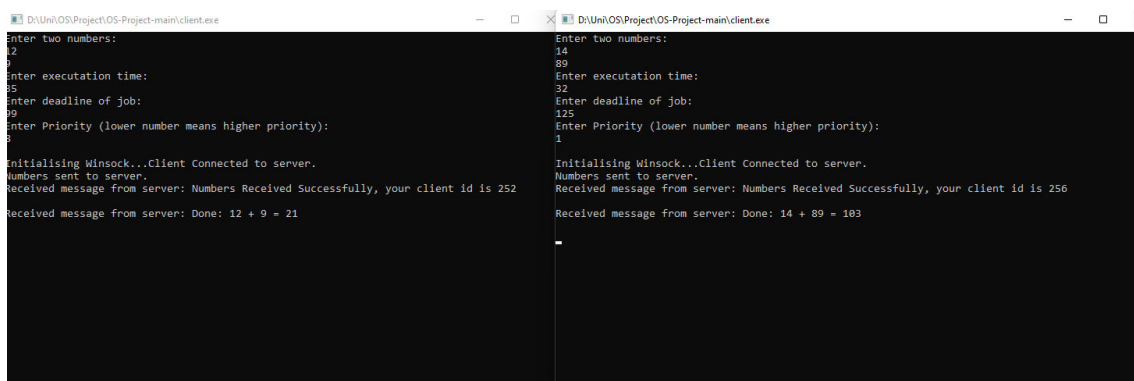
در این الگوریتم، به هر تسک یک اولویت یا Priority نسبت داده می‌شود و تسک با بیشترین اولویت زودتر انجام می‌شود.



```
Enter two numbers:
1
2
Enter execution time:
25
Enter deadline of job:
28
Enter Priority (lower number means higher priority):
2
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 136
Received message from server: Done: 1 + 2 = 3

Enter two numbers:
23
25
Enter execution time:
22
Enter deadline of job:
66
Enter Priority (lower number means higher priority):
8
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 248
Received message from server: Done: 23 + 25 = 48
```

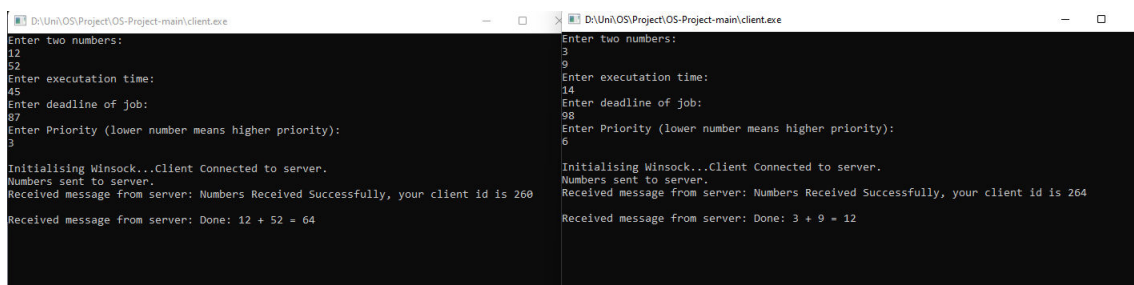
شکل ۲۲



```
Enter two numbers:
12
9
Enter execution time:
85
Enter deadline of job:
99
Enter Priority (lower number means higher priority):
3
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 252
Received message from server: Done: 12 + 9 = 21

Enter two numbers:
14
89
Enter execution time:
32
Enter deadline of job:
125
Enter Priority (lower number means higher priority):
1
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 256
Received message from server: Done: 14 + 89 = 103
```

شکل ۲۳



```
Enter two numbers:
12
52
Enter execution time:
45
Enter deadline of job:
87
Enter Priority (lower number means higher priority):
3
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 260
Received message from server: Done: 12 + 52 = 64

Enter two numbers:
3
9
Enter execution time:
14
Enter deadline of job:
98
Enter Priority (lower number means higher priority):
6
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 264
Received message from server: Done: 3 + 9 = 12
```

شکل ۲۴

```

D:\Unit\OS\Project\OS-Project-main\client.exe
Enter two numbers:
5
15
Enter execution time:
61
Enter deadline of job:
128
Enter Priority (lower number means higher priority):
12
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 268
Received message from server: Done: 5 + 15 = 20

D:\Unit\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
32
Enter execution time:
78
Enter deadline of job:
75
Enter Priority (lower number means higher priority):
5
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 272
Received message from server: Done: 14 + 32 = 46

```

شکل ۲۵

```

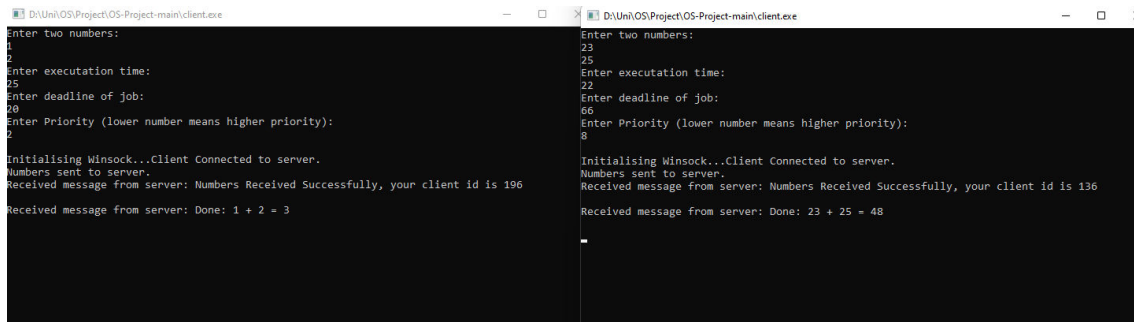
Select your scheduling algorithm:
1- FCFS
2- Priority
3- SJF
4- EDF
2
Server Successfully binded to port 8080 and listen on it
client 136 is accepted
Received numbers from client 136.
Client with id 136 is assigned to worker with id 1
client 248 is accepted
Received numbers from client 248.
Client with id 248 is assigned to worker with id 2
client 252 is accepted
Received numbers from client 252.
client 256 is accepted
Received numbers from client 256.
client 260 is accepted
Received numbers from client 260.
client 264 is accepted
Received numbers from client 264.
client 268 is accepted
Received numbers from client 268.
client 272 is accepted
Received numbers from client 272.
Done for client_socket 248 : 23 + 25 = 48
Client with id 256 is assigned to worker with id 2
Done for client_socket 136 : 1 + 2 = 3
Client with id 260 is assigned to worker with id 1
Done for client_socket 256 : 14 + 89 = 103
Client with id 272 is assigned to worker with id 2
Done for client_socket 260 : 12 + 52 = 64
Client with id 264 is assigned to worker with id 1
Done for client_socket 264 : 3 + 9 = 12
Client with id 252 is assigned to worker with id 1
Done for client_socket 252 : 12 + 9 = 21
Client with id 268 is assigned to worker with id 1
Done for client_socket 272 : 14 + 32 = 46
Done for client_socket 268 : 5 + 15 = 20

```

شکل ۲۶

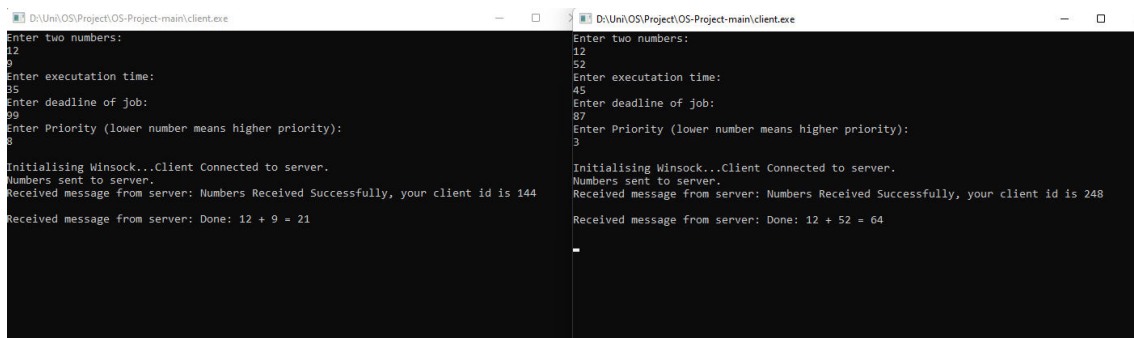
## ۳-۴-۲ الگوریتم SJF

در این روش، کار با کمترین طول زمان اجرا، بیشترین اولویت را خواهد داشت. در واقع این الگوریتم نوعی Priority Scheduling به حساب می‌آید که در آن، اولویت معادل با معکوس طول زمان اجراست.



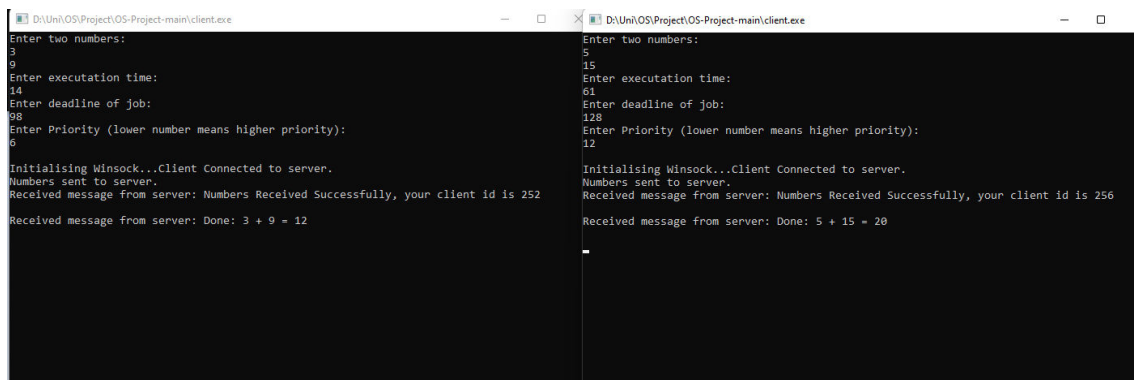
The screenshot shows two terminal windows side-by-side. The left window shows a client program running with inputs: two numbers (1, 2), execution time (25), deadline (20), and priority (2). It receives a message from the server: "Numbers Received Successfully, your client id is 196" and then "Done: 1 + 2 = 3". The right window shows a client program running with inputs: two numbers (23, 25), execution time (22), deadline (66), and priority (8). It receives a message from the server: "Numbers Received Successfully, your client id is 136" and then "Done: 23 + 25 = 48".

شکل ۲۷



The screenshot shows two terminal windows side-by-side. The left window shows a client program running with inputs: two numbers (12, 9), execution time (15), deadline (15), and priority (8). It receives a message from the server: "Numbers Received Successfully, your client id is 144" and then "Done: 12 + 9 = 21". The right window shows a client program running with inputs: two numbers (12, 52), execution time (45), deadline (87), and priority (3). It receives a message from the server: "Numbers Received Successfully, your client id is 248" and then "Done: 12 + 52 = 64".

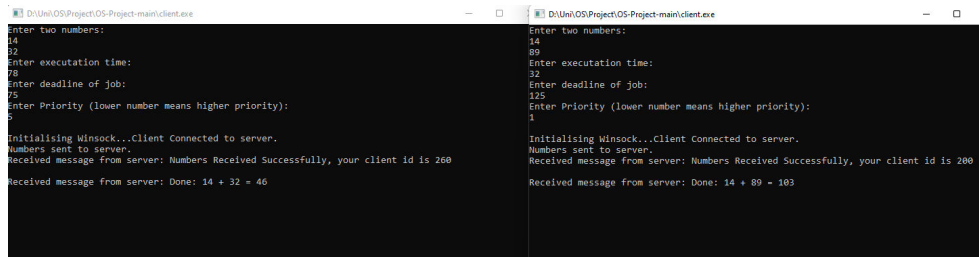
شکل ۲۸



The screenshot shows two terminal windows side-by-side. The left window shows a client program running with inputs: two numbers (3, 9), execution time (14), deadline (98), and priority (6). It receives a message from the server: "Numbers Received Successfully, your client id is 252" and then "Done: 3 + 9 = 12". The right window shows a client program running with inputs: two numbers (5, 15), execution time (61), deadline (128), and priority (12). It receives a message from the server: "Numbers Received Successfully, your client id is 256" and then "Done: 5 + 15 = 20".

شکل ۲۹

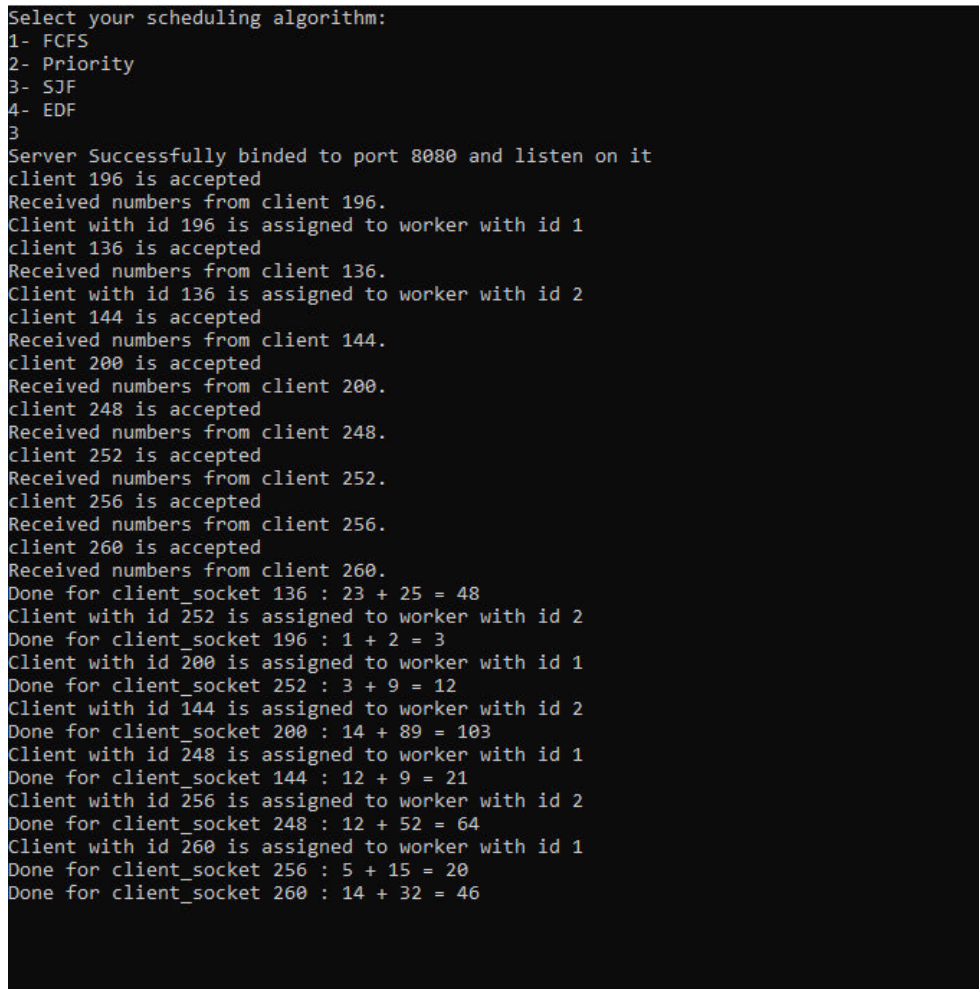




```
D:\Unl\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
32
Enter execution time:
78
Enter deadline of job:
75
Enter Priority (lower number means higher priority):
5
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 260
Received message from server: Done: 14 + 32 = 46

D:\Unl\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
89
Enter execution time:
32
Enter deadline of job:
125
Enter Priority (lower number means higher priority):
1
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 280
Received message from server: Done: 14 + 89 = 103
```

شکل ۳۰



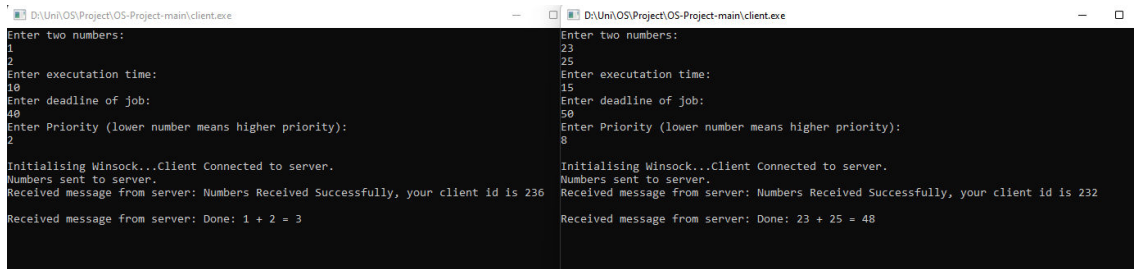
```
Select your scheduling algorithm:
1- FCFS
2- Priority
3- SJF
4- EDF
3
Server Successfully binded to port 8080 and listen on it
client 196 is accepted
Received numbers from client 196.
Client with id 196 is assigned to worker with id 1
client 136 is accepted
Received numbers from client 136.
Client with id 136 is assigned to worker with id 2
client 144 is accepted
Received numbers from client 144.
client 200 is accepted
Received numbers from client 200.
client 248 is accepted
Received numbers from client 248.
client 252 is accepted
Received numbers from client 252.
client 256 is accepted
Received numbers from client 256.
client 260 is accepted
Received numbers from client 260.
Done for client_socket 136 : 23 + 25 = 48
Client with id 252 is assigned to worker with id 2
Done for client_socket 196 : 1 + 2 = 3
Client with id 200 is assigned to worker with id 1
Done for client_socket 252 : 3 + 9 = 12
Client with id 144 is assigned to worker with id 2
Done for client_socket 200 : 14 + 89 = 103
Client with id 248 is assigned to worker with id 1
Done for client_socket 144 : 12 + 9 = 21
Client with id 256 is assigned to worker with id 2
Done for client_socket 248 : 12 + 52 = 64
Client with id 260 is assigned to worker with id 1
Done for client_socket 256 : 5 + 15 = 20
Done for client_socket 260 : 14 + 32 = 46
```

شکل ۳۱



## ۴-۴-۲ الگوریتم EDF

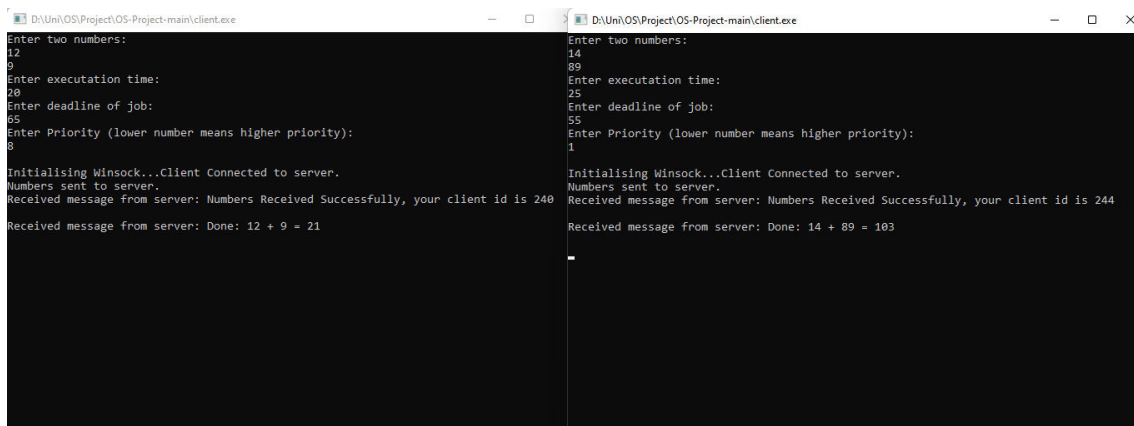
در این الگوریتم، بر اساس سیاست Earliest Deadline first عمل می‌شود. یعنی تسک با نزدیک‌ترین ددلاین، زودتر از بقیه انجام می‌شود. نکته‌ی قابل توجه درباره‌ی این الگوریتم این است که در صورت گذشتن ددلاین، تسک تایم‌اوت می‌شود و این تایم‌اوت شدن به کلاینت اطلاع داده می‌شود.



```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
1
2
Enter execution time:
10
Enter deadline of job:
40
Enter Priority (lower number means higher priority):
2
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 236
Received message from server: Done: 1 + 2 = 3

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
23
25
Enter execution time:
15
Enter deadline of job:
50
Enter Priority (lower number means higher priority):
8
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 232
Received message from server: Done: 23 + 25 = 48
```

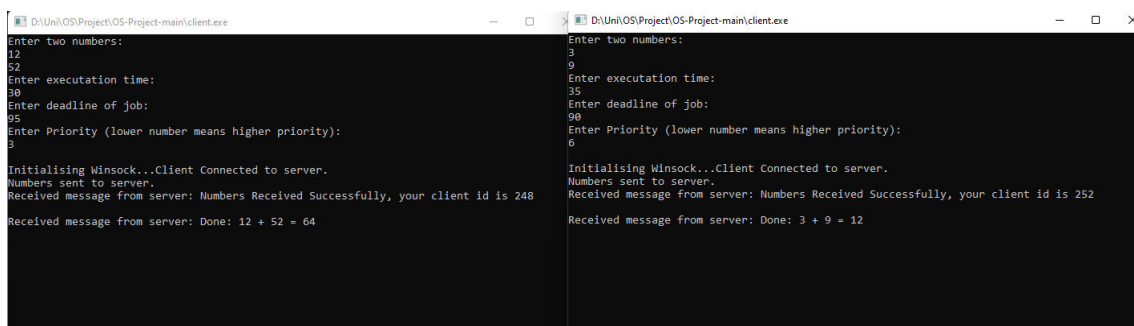
شکل ۳۲



```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
12
9
Enter execution time:
20
Enter deadline of job:
65
Enter Priority (lower number means higher priority):
8
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 240
Received message from server: Done: 12 + 9 = 21

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
89
Enter execution time:
25
Enter deadline of job:
55
Enter Priority (lower number means higher priority):
1
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 244
Received message from server: Done: 14 + 89 = 103
```

شکل ۳۳



```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
12
52
Enter execution time:
30
Enter deadline of job:
85
Enter Priority (lower number means higher priority):
3
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 248
Received message from server: Done: 12 + 52 = 64

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
3
9
Enter execution time:
35
Enter deadline of job:
90
Enter Priority (lower number means higher priority):
6
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 252
Received message from server: Done: 3 + 9 = 12
```

شکل ۳۴

```
D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
5
15
Enter execution time:
40
Enter deadline of job:
150
Enter Priority (lower number means higher priority):
12
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 256
Received message from server: Done: 5 + 15 = 20

D:\Uni\OS\Project\OS-Project-main\client.exe
Enter two numbers:
14
32
Enter execution time:
45
Enter deadline of job:
140
Enter Priority (lower number means higher priority):
7
Initialising Winsock...Client Connected to server.
Numbers sent to server.
Received message from server: Numbers Received Successfully, your client id is 260
Received message from server: Done: 14 + 32 = 46
```

شکل ۳۵

```
Select your scheduling algorithm:
1- FCFS
2- Priority
3- SJF
4- EDF
4
Server Successfully binded to port 8080 and listen on it
client 232 is accepted
Received numbers from client 232.
Client with id 232 is assigned to worker with id 1
client 236 is accepted
Received numbers from client 236.
Client with id 236 is assigned to worker with id 2
client 240 is accepted
Received numbers from client 240.
client 244 is accepted
Received numbers from client 244.
client 248 is accepted
Received numbers from client 248.
client 252 is accepted
Received numbers from client 252.
client 256 is accepted
Received numbers from client 256.
client 260 is accepted
Received numbers from client 260.
Done for client_socket 236 : 1 + 2 = 3
Client with id 244 is assigned to worker with id 2
Done for client_socket 232 : 23 + 25 = 48
Client with id 240 is assigned to worker with id 1
Done for client_socket 240 : 12 + 9 = 21
Client with id 252 is assigned to worker with id 1
Done for client_socket 244 : 14 + 89 = 103
Client with id 248 is assigned to worker with id 2
Done for client_socket 248 : 12 + 52 = 64
Client with id 260 is assigned to worker with id 2
Done for client_socket 252 : 3 + 9 = 12
Client with id 256 is assigned to worker with id 1
Done for client_socket 256 : 5 + 15 = 20
Done for client_socket 260 : 14 + 32 = 46
```

شکل ۳۶

## References

- [geeksforgeeks Server Client Example Code in C](#)
- [nachtinwald](#)
- [geeksforgeeks POSIX Semaphores in C](#)

## مطالب تکمیلی

یکی از قابلیت‌های سرور که در قسمت‌های قبلی راجع به آن صحبت نکردیم سیستم لاگینگ آن است که به ازای هر روز یک فایل لاگ با نام همان روز می‌سازد و اطلاعات مربوط به کلاینت‌ها و آماره‌ها و زمان دریافت ریکوئست‌ها و تسک‌های انجام شده و ... را در آن ذخیره می‌کند. که این فایل لاگ همراه پروژه موجود است که می‌توانید آن را ببینید. همچنین از آن‌جا که ورکرها و تردهای مختلف در آن می‌نویسند با استفاده از مفاهیم قفل و سمافور این کار را مدیریت کردیم که وقتی ورکرهای مختلف می‌خواهند در یک فایل بنویسند مشکلی یا نامغایرتی پیش نیاید.

برای مشاهده‌ی کدها می‌توانید به **گیت‌هاب پروژه** مراجعه کنید.