



دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

گزارش کار آزمایشگاه آزمایشگاه سیستم‌های عامل

گزارش آزمایش شماره ۷
(آشنایی با ریشه‌ها)

۲۰

ارشیا یوسف‌نیا (۴۰۱۱۱۰۴۱۵)

محمدعارف زارع زاده (۴۰۱۱۰۶۰۱۷)

دکتر بیگی

تابستان ۱۴۰۴

شماره‌ی گروه:

گروه:

استاد درس:

تاریخ:

فهرست مطالب

۱	آشنایی اولیه	۱
۱	۱.۱
۱	۲.۱
۲	۳.۱
۴	۴.۱
۵	۵.۱
۷	ریسه‌های چندتایی	۲
۸	تفاوت بین پردازها و ریشه‌ها	۳
۸	۱.۳
۸	۲.۳
۹	۳.۳
۱۱	پاس دادن متغیر به ریشه‌ها	۴

لیست تصاویر

۱	ایجاد یک ریشه جدید و چاپ یک پیام در آن	۱
۲	گام‌های کامپایل و اجرای برنامه شکل ۱	۲
۳	چاپ شماره پردازه در ریشه اصلی و ریشه جدید	۳
۳	گام‌های کامپایل و اجرای برنامه شکل ۳	۴
۴	دسترسی به متغیر عمومی از دو ریشه متفاوت	۵
۵	گام‌های کامپایل و اجرای برنامه شکل ۵	۶
۶	اعداد ۲ تا عدد ورودی در ریشه جدید جمع زده می‌شوند	۷
۶	گام‌های کامپایل و اجرای برنامه شکل ۷	۸
۷	ایجاد چند ریشه	۹
۷	گام‌های کامپایل و اجرای برنامه شکل ۹	۱۰
۸	دسترسی به متغیر عمومی در ریشه اصلی و فرعی	۱۱
۹	گام‌های کامپایل و اجرای برنامه شکل ۱۱	۱۲
۹	دسترسی به متغیر عمومی از پردازه والد و فرزند	۱۳
۱۰	گام‌های کامپایل و اجرای برنامه شکل ۱۳	۱۴
۱۱	پاس دادن داده در غالب ساختار به ریشه و دسترسی به آن	۱۵
۱۱	گام‌های کامپایل و اجرای برنامه شکل ۱۵	۱۶

۱ آشنایی اولیه

۱.۱

وارد سیستم لینوکس می‌شویم.

۲.۱

در ادامه با کد شکل ۱ یک ریسه می‌سازیم که صرفاً داخل خود یک پیام چاپ میکند. داخل ریسۀ اصلی با توقف برای یک زمان، زمانی ریسۀ اصلی را خاتمه می‌دهیم که ریسۀ دوم هم تمام شده باشد. این روش تضمینی نیست ولی برای این حالت ساده کار می‌کند. شکل ۲ گام‌های کامپایل و اجرا و را نشان می‌دهد.



```
GNU nano 7.2 thread_1.c *
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *child(void *arg) {
    puts("Hello from child thread!");
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, child, NULL);
    sleep(1);
    return 0;
}
```

The screenshot shows a terminal window with the nano text editor. The editor is editing a file named 'thread_1.c'. The code is a C program that creates a new thread. The 'child' function prints 'Hello from child thread!'. The 'main' function creates the thread, sleeps for 1 second, and then returns 0. The nano editor's status bar at the bottom shows various keyboard shortcuts for editing and file operations.

شکل ۱: ایجاد یک ریسۀ جدید و چاپ یک پیام در آن

```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_1.c
> gcc thread_1.c -o thread_1 -lpthread
> ./thread_1
Hello from child thread!
```

شکل ۲: گام‌های کامپایل و اجرای برنامه شکل ۱

۳.۱

در این قسمت با `pthread_create` و `pthread_join` یک برنامه می‌نویسیم که یک ریسۀ جدید درست کند و هم در آن و هم در ریسۀ اصلی شمارهٔ پردازش را چاپ کند. این اعداد باید یکسان باشند. شکل ۳ این برنامه را نشان می‌دهد و شکل ۴ مراحل کامپایل و اجرای آن را نشان می‌دهد.



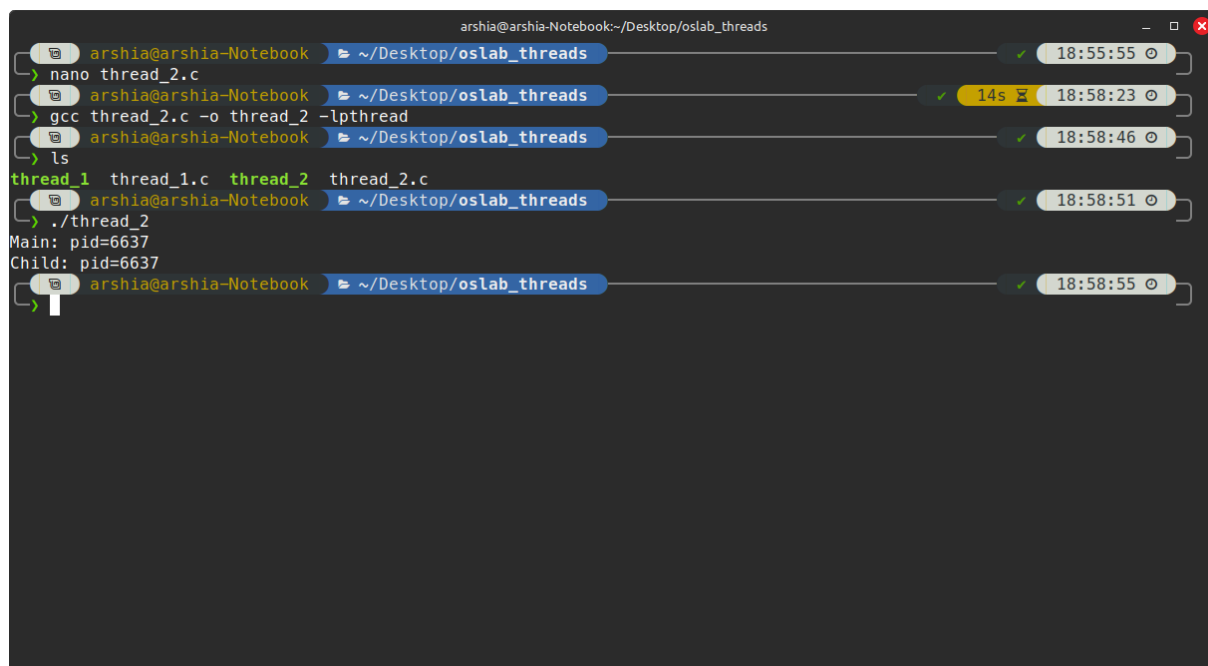
```
GNU nano 7.2 thread_2.c *
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *child(void *arg) {
    printf("Child: pid=%d\n", getpid());
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, child, NULL);
    printf("Main: pid=%d\n", getpid());
    pthread_join(thread, NULL);
    return 0;
}

Save modified buffer?
Y Yes
N No ^C Cancel
```

شکل ۳: چاپ شماره پردازش در ریسۀ اصلی و ریسۀ جدید



```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_2.c
> gcc thread_2.c -o thread_2 -lpthread
> ls
thread_1 thread_1.c thread_2 thread_2.c
> ./thread_2
Main: pid=6637
Child: pid=6637
```

شکل ۴: گام‌های کامپایل و اجرای برنامه شکل ۳

در این قسمت یک متغیر عمومی به برنامه اضافه می‌کنیم و در هر دو ریشه برنامه قبل مقدار آن را تغییر می‌دهیم و بررسی می‌کنیم. این آزمایش مشترک بودن حافظه ریشه‌ها را نشان می‌دهد. در شکل ۵ متن برنامه و در شکل ۶ هم کامپایل و اجرای آن آمده است.

```

GNU nano 7.2                                thread_3.c *
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int oslab = 1;

void *child(void *arg) {
    oslab = 2;

    printf("Child: pid=%d\n", getpid());

    printf("In child oslab is now changed to: %d\n", oslab);
}

int main() {
    printf("Original oslab vlaue: %d\n", oslab);

    pthread_t thread;
    pthread_create(&thread, NULL, child, NULL);
    printf("Main: pid=%d\n", getpid());
    pthread_join(thread, NULL);

    printf("In main thread oslab is %d\n", oslab);
    return 0;
}

```

Help Write Out Where Is Cut Execute Location Undo Set Mark
Exit Read File Replace Paste Justify Go To Line Redo Redo Copy

شکل ۵: دسترسی به متغیر عمومی از دو ریشه متفاوت

```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> ls
thread_1 thread_1.c thread_2 thread_2.c thread_3.c
> gcc thread_3.c -o thread_3 -lpthread
> ./thread_3
Original oslab vlaue: 1
Main: pid=7074
Child: pid=7074
In child oslab is now changed to: 2
In main thread oslab is 2
```

شکل ۶: گام‌های کامپایل و اجرای برنامه شکل ۵

۵.۱

اکنون یک برنامه می‌نویسیم که یک ریسۀ جدید درست کند که در آن عددی که ورودی گرفته‌ایم داده می‌شود و برنامه از عدد ۲ تا آن عدد را جمع می‌کند و حاصل را نمایش می‌دهد. شکل ۷ این برنامه را نشان می‌دهد و شکل ۸ کامپایل و اجرای آن را نشان می‌دهد.



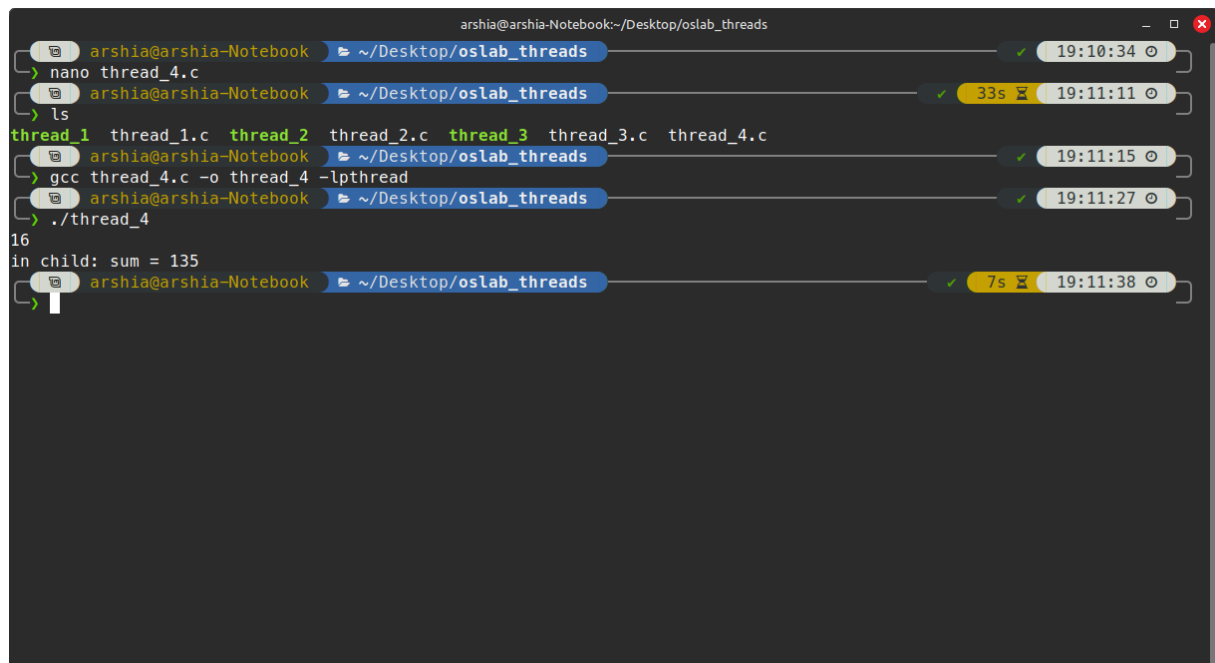
```
GNU nano 7.2 thread_4.c *
#include <pthread.h>
#include <stdio.h>

void *child(void *arg) {
    int n = *(int*)arg;
    int sum = 0;
    for (int i = 2; i <= n; i++)
        sum += i;
    printf("in child: sum = %d\n", sum);
}

int main() {
    int n;
    scanf("%d", &n);
    pthread_t thread;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create(&thread, &attr, child, &n);
    pthread_join(thread, NULL);
    return 0;
}

Save modified buffer?
Y Yes
N No ^C Cancel
```

شکل ۷: اعداد ۲ تا عدد ورودی در ریسۀ جدید جمع زده می شوند



```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_4.c
> ls
thread_1 thread_1.c thread_2 thread_2.c thread_3 thread_3.c thread_4.c
> gcc thread_4.c -o thread_4 -lpthread
> ./thread_4
16
in child: sum = 135
>
```

شکل ۸: گام های کامپایل و اجرای برنامه شکل ۷

۲ ریشه‌های چندتایی

در این قسمت ۵ ریشه می‌سازیم و در هر کدام یک پیام چاپ می‌کنیم. در نهایت زمانی ریشه اصلی را تمام می‌کنیم که همه ریشه‌ها تمام شده باشند. شکل ۹ برنامه را نشان می‌دهد. اجرا و خروجی در شکل ۱۰ آمده است.



```
GNU nano 7.2                                thread_5.c *
#include <pthread.h>
#include <stdio.h>

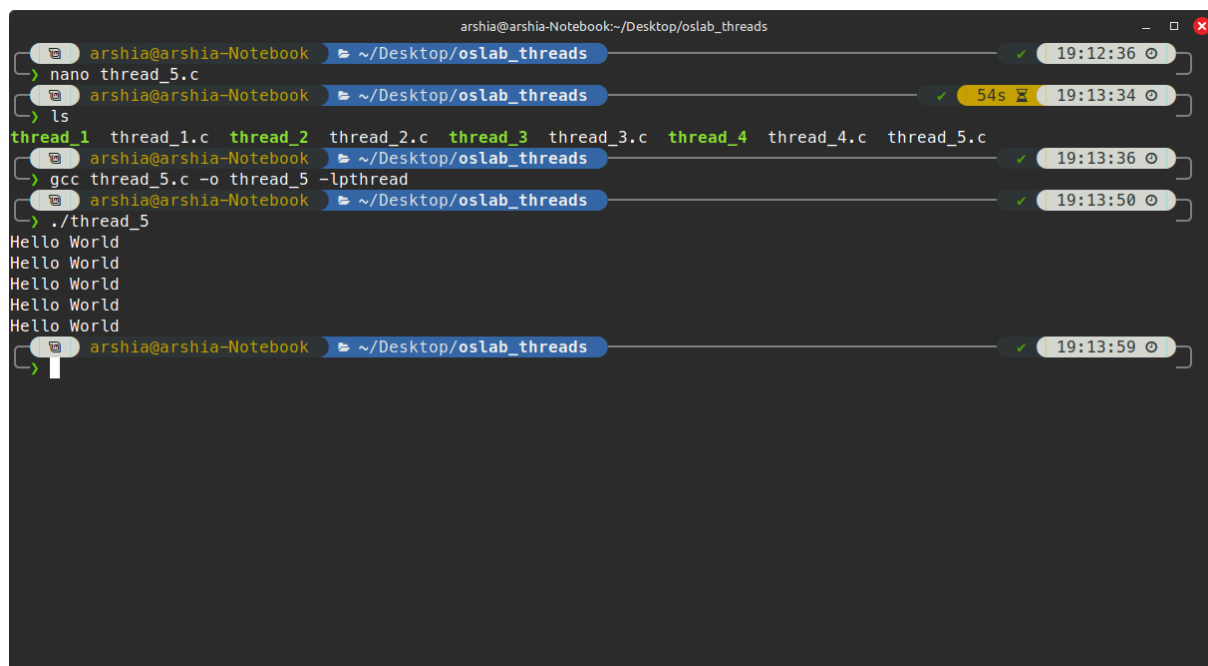
void *child(void *arg) {
    printf("Hello World\n");
    pthread_exit(0);
}

int main() {
    int thread_count = 5;

    pthread_t threads[thread_count];
    for (int i = 0; i < thread_count; i++)
        pthread_create(&threads[i], NULL, child, NULL);
    for (int i = 0; i < thread_count; i++)
        pthread_join(threads[i], NULL);
    return 0;
}
```

Save modified buffer? [Y] Yes [N] No [^C] Cancel

شکل ۹: ایجاد چند ریشه



```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_5.c
> ls
thread_1 thread_1.c thread_2 thread_2.c thread_3 thread_3.c thread_4 thread_4.c thread_5.c
> gcc thread_5.c -o thread_5 -lpthread
> ./thread_5
Hello World
Hello World
Hello World
Hello World
Hello World
> arshia@arshia-Notebook:~/Desktop/oslab_threads
```

شکل ۱۰: گام‌های کامپایل و اجرای برنامه شکل ۹

۳ تفاوت بین پردازنده‌ها و ریشه‌ها

۱.۳

برای سهولت گزارش تابع این قسمت را در کد قسمت بعد آورده‌ایم.

۲.۳

در این قسمت یک متغیر عمومی را در ریشه تغییر می‌دهیم و اثر آن بر ریشه اصلی را می‌بینیم. شکل ۱۱ برنامه را نشان می‌دهد و شکل ۱۲ نتیجه آن را نشان می‌دهد.



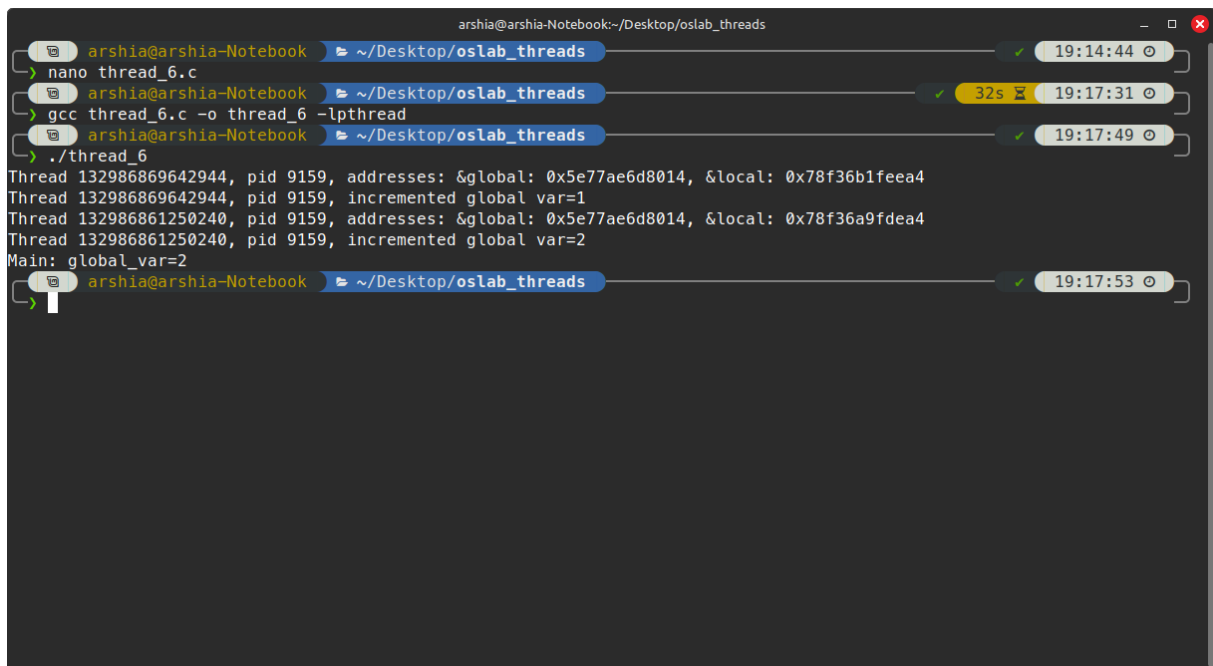
```
GNU nano 7.2                                nano thread_6.c
thread_6.c *
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int global_var;

void *child(void *arg) {
    int local_var;
    printf("Thread %ld, pid %d, addresses: &global: %p, &local: %p \n",
        pthread_self(), getpid(), &global_var, &local_var);
    global_var++;
    printf("Thread %ld, pid %d, incremented global var=%d\n",
        pthread_self(), getpid(), global_var);
    pthread_exit(0);
}

int main() {
    global_var = 0;
    pthread_t t1, t2;
    pthread_create(&t1, NULL, child, NULL);
    pthread_create(&t2, NULL, child, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Main: global_var=%d\n", global_var);
    return 0;
}
Save modified buffer? [Y] Yes
N No ^C Cancel
```

شکل ۱۱: دسترسی به متغیر عمومی در ریشه اصلی و فرعی



```
arshia@arshia-Notebook: ~/Desktop/oslab_threads
> nano thread_6.c
> gcc thread_6.c -o thread_6 -lpthread
> ./thread_6
Thread 132986869642944, pid 9159, addresses: &global: 0x5e77ae6d8014, &local: 0x78f36b1feea4
Thread 132986869642944, pid 9159, incremented global var=1
Thread 132986861250240, pid 9159, addresses: &global: 0x5e77ae6d8014, &local: 0x78f36a9fdea4
Thread 132986861250240, pid 9159, incremented global var=2
Main: global_var=2
```

شکل ۱۲: گام‌های کامپایل و اجرای برنامه شکل ۱۱

۳.۳

در این قسمت همان کار قسمت قبل را با پروژه‌ها انجام می‌دهیم. شکل ۱۳ برنامه و شکل ۱۴ اجرای آن را نشان می‌دهد.



```
GNU nano 7.2 thread_7.c *
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int global_var;

int main() {
    global_var = 5;
    int local_var = 10;
    pid_t pid = fork();
    if (pid == 0) {
        global_var++;
        local_var++;
        printf("Child: global=%d, local=%d\n", global_var, local_var);
    } else {
        wait(NULL);
        printf("Parent: global=%d, local=%d\n", global_var, local_var);
    }
    return 0;
}
```

شکل ۱۳: دسترسی به متغیر عمومی از پروژه والد و فرزند

```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_7.c
> gcc thread_7.c -o thread_7 -lpthread
> ./thread_7
Child: global=6, local=11
Parent: global=5, local=10
>
```

The screenshot shows a terminal window titled 'arshia@arshia-Notebook:~/Desktop/oslab_threads'. It contains four command-line entries, each with a timestamp and a status icon. The first command is 'nano thread_7.c' at 19:18:51. The second is 'gcc thread_7.c -o thread_7 -lpthread' at 19:19:18, which took 22 seconds to execute. The third is './thread_7' at 19:19:28, which executed successfully. The output of the program is 'Child: global=6, local=11' and 'Parent: global=5, local=10'. The fourth command is a prompt '>' at 19:19:33.

شکل ۱۴: گام‌های کامپایل و اجرای برنامه شکل ۱۳

۴ پاس دادن متغیر به ریشه‌ها

در این قسمت پاس دادن ساختارها به ریشه‌ها را بررسی می‌کنیم که محدودیت یک پارامتر ورودی به ریشه را حل می‌کند. شکل ۱۵ برنامه و شکل ۱۶ اجرا را نشان می‌دهد.



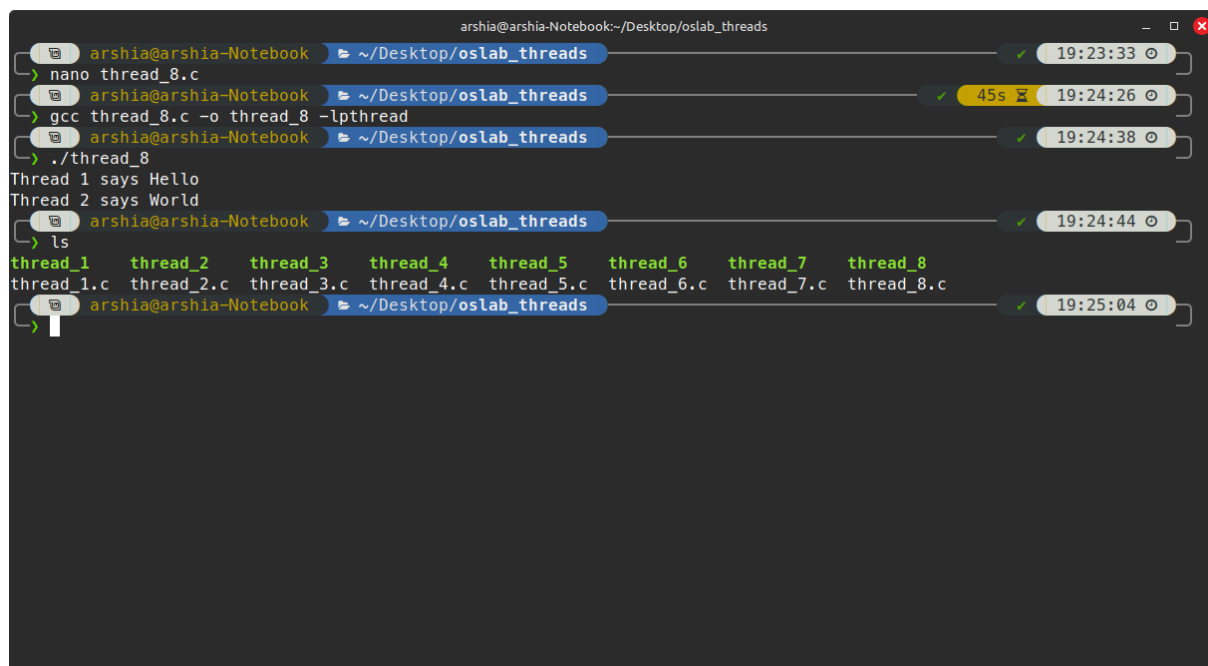
```
GNU nano 7.2                                thread_8.c *
#include <pthread.h>
#include <stdio.h>

typedef struct thdata {
    int thread_no;
    char message[100];
} stdata;

void *printmsg(void *ptr) {
    stdata *data;
    data = (stdata *) ptr;
    printf("Thread %d says %s\n", data->thread_no, data->message);
}

int main() {
    pthread_t t1, t2;
    stdata data1, data2;
    data1.thread_no = 1;
    sprintf(data1.message, "Hello");
    data2.thread_no = 2;
    sprintf(data2.message, "World");
    pthread_create(&t1, NULL, printmsg, &data1);
    pthread_create(&t2, NULL, printmsg, &data2);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
Save modified buffer? [Y] Yes
N No ^C Cancel
```

شکل ۱۵: پاس دادن داده در غالب ساختار به ریشه و دسترسی به آن



```
arshia@arshia-Notebook:~/Desktop/oslab_threads
> nano thread_8.c
arshia@arshia-Notebook:~/Desktop/oslab_threads 45s
> gcc thread_8.c -o thread_8 -lpthread
arshia@arshia-Notebook:~/Desktop/oslab_threads
> ./thread_8
Thread 1 says Hello
Thread 2 says World
arshia@arshia-Notebook:~/Desktop/oslab_threads
> ls
thread_1  thread_2  thread_3  thread_4  thread_5  thread_6  thread_7  thread_8
thread_1.c thread_2.c thread_3.c thread_4.c thread_5.c thread_6.c thread_7.c thread_8.c
arshia@arshia-Notebook:~/Desktop/oslab_threads
```

شکل ۱۶: گام‌های کامپایل و اجرای برنامه شکل ۱۵