آزمایشگاه سیستم عامل

دکتر بیگی

آزمایش ۷

الینا هژبری – ۴۰۱۱۷۰۶۶۱

ملیکا علیزاده – ۴۰۱۱۰۶۲۵۵

آزمایش ۷

۷-۴-۱- آشنایی اولیه

۱. وارد سیستم‌عامل شده و کد داده شده را با lpthread- اجرا می‌کنیم. در این کد پردازه والد یک ریسه‌ی فرزند می‌سازد و برای ۱ ثانیه می‌خوابد. در این فاصله ریسه‌ی فرزند اجرا شده و پیام Hello from child thread را چاپ می‌کند.

```
elina@elina-vm:~/Desktop$ nano thread.c
elina@elina-vm:~/Desktop$ cat thread.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;
}
elina@elina-vm:~/Desktop$ gcc thread.c -o threads -lpthread
elina@elina-vm:~/Desktop$ ./threads
Hello from child thread!
```

۲. در این بخش به جای استفاده از sleep()، از pthread_join() استفاده می‌کنیم. شماره پردازه‌ها همانطور که مشاهده می‌شود، یکسان خواهند بود زیرا ریسه‌ها در یک پردازه اجرا می‌شوند و PID مشترک دارند.

```
elina@elina-vm:~/Desktop$ nano thread2.c
elina@elina-vm:~/Desktop$ cat thread2.c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

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

int main() {
    pthread_t thread;
    printf("Main PID: %d\n", getpid());
    pthread_create(&thread, NULL, child, NULL);
    pthread_join(thread, NULL);
    puts("Main: child thread finished.\n");
    return 0;
}
elina@elina-vm:~/Desktop$ gcc thread2.c -o threads2 -lpthread
elina@elina-vm:~/Desktop$ ./threads2
Main PID: 4739
Child thread PID: 4739
Main: child thread finished.
```

۳. در این بخش متغیر oslab را در ریسه main، ۲- کرده و در ریسه فرزند ۵+ می‌کنیم. همانطور که مشخص است آدرس پوینتر oslab در هر دو ریسه یکی است و تغییرات هر دو ریسه بر متغیر oslab اعمال می‌شود. ریسه‌ها کپی جداگانه از متغیر ندارند و فضای آدرس یکسانی را به اشتراک می‌گذارند.

```
elina@elina-vm:~/Desktop$ nano thread3.c
elina@elina-vm:~/Desktop$ cat thread3.c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int oslab = 0;

void *child(void *arg) {
    printf("child (before): oslab= %d, addreass= %p\n", oslab, &oslab);
    oslab += 5;
    printf("child (after): oslab= %d, addreass= %p\n", oslab, &oslab);
}

int main() {
    pthread_t thread;
    printf("main (before): oslab= %d, addreass= %p\n", oslab, &oslab);
    oslab -= 2;
    printf("main (after): oslab= %d, addreass= %p\n", oslab, &oslab);
    pthread_create(&thread, NULL, child, NULL);
    pthread_join(thread, NULL);
    printf("OSlab after join: oslab= %d, addreass= %p\n", oslab, &oslab);
    return 0;
}
elina@elina-vm:~/Desktop$ gcc thread3.c -o threads3 -lpthread
elina@elina-vm:~/Desktop$ ./threads3
main (before): oslab= 0, addreass= 0x589717d2a014
main (after): oslab= -2, addreass= 0x589717d2a014
child (before): oslab= -2, addreass= 0x589717d2a014
child (after): oslab= 3, addreass= 0x589717d2a014
OSlab after join: oslab= 3, addreass= 0x589717d2a014
```

۴. در این بخش، عدد n را از کاربر می‌گیریم و با pthread_attr_init ویژگی‌های پیش‌فرض ریسه را آماده کردیم و سپس در ریسه فرزند از ۲ تا n را جمع زدیم و نتیجه را چاپ کردیم.

```
elina@elina-vm:~/Desktop$ cat thread4.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int n;

void *child(void *arg) {
    long s = 0;
    if (n<2) {
        printf("n is less than 2!\n");
        return NULL;
    }
    for (int i = 2; i <= n; i++) {
        s += i;
    }
    printf("sum = %ld\n", s);
    return NULL;
}

int main(void) {
    printf("Enter a number: ");
    scanf("%d", &n);
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_t th;
    pthread_create(&th, &attr, child,NULL);
    pthread_attr_destroy(&attr);
    pthread_join(th, NULL);
    return 0;
}
```

```
elina@elina-vm:~/Desktop$ gcc thread4.c -o threads4 -lpthread
elina@elina-vm:~/Desktop$ ./threads4
Enter a number: 6
sum = 20
```

۷-۴-۲- ریسه‌های چندتایی

در این بخش با pthread_create، ۶ ریشه می‌سازیم و پس از چاپ پیام Hello World، ریسه‌ها را با pthread_exit خاتمه می‌دهیم. در آخر ریسه main نیز از pthread_exit استفاده کردیم تا منتظر اتمام تمام ریسه‌ها بماند و کاری شبیه به pthread_join انجام می‌دهد.

```
elina@elina-vm:~/Desktop$ nano multi_thread.c
elina@elina-vm:~/Desktop$ cat multi_thread.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int thread_num = 6;

void *hello_world(void *arg) {
    int id = *(int *)arg;
    printf("Hello World thread %d\n", id);
    free(arg);
    pthread_exit(0);
}

int main(void) {
    pthread_t threads[thread_num];
    for (int i = 0; i < thread_num; i++) {
        int *id = malloc(sizeof(int));
        *id = i + 1;
        pthread_create(&threads[i], NULL, hello_world, id);
    }
    pthread_exit(NULL);
    return 0;
}
```

```
elina@elina-vm:~/Desktop$ gcc multi_thread.c -o multi_thread -lpthread
elina@elina-vm:~/Desktop$ ./multi_thread
Hello World thread 1
Hello World thread 2
Hello World thread 5
Hello World thread 3
Hello World thread 4
Hello World thread 6
```

۷-۴-۳- تفاوت بین پردازه‌ها و ریسه‌ها

۱. ابتدا متغیر global را در main مقدار ۱۰ می‌دهیم. سپس در هر ریسه آن را ۱+ می‌کنیم. از آنجایی که متغیر عمومی است و ریسه‌ها از یک آدرس مشترک برای آن استفاده می‌کنند، مقدار global_var پس از دو ریسه ۲+ می‌شود.

```
elina@elina-vm:~/Desktop$ nano processORthread.c
elina@elina-vm:~/Desktop$ cat processORthread.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.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() {
    pthread_t th1, th2;
    global_var = 10;
    printf("Main: global_var=%d\n", global_var);
    pthread_create(&th1, NULL, child, NULL);
    pthread_create(&th2, NULL, child, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);
    printf("Main after threads: global_var=%d\n", global_var);
    return 0;
}
```

```
elina@elina-vm:~/Desktop$ gcc processORthread.c -o pth -lpthread
elina@elina-vm:~/Desktop$ ./pth
Main: global_var=10
Thread 126625024833216, pid 8025, addresses: &global: 0x60af107aa014, &local: 0x732a2f9feea4
Thread 126625024833216, pid 8025, incremented global var=11
Thread 126625016440512, pid 8025, addresses: &global: 0x60af107aa014, &local: 0x732a2f1fdea4
Thread 126625016440512, pid 8025, incremented global var=12
Main after threads: global_var=12
```

۲. سپس در کد دیگری متغیر global و local را تعریف می‌کنیم به گونه‌ای که مقادیر ۱۰ و ۵ را دارند. سپس با استفاده از fork یک پردازه فرزند ایجاد می‌کنیم و در آن مقدار هر رو ۱۰+ می‌کنیم. همانطور که مشاهده می‌شود، آدرس‌ها در پردازه‌های والد و فرزند متفاوت است به همین دلیل مقادیر نیز متفاوت است.

```
elina@elina-vm:~/Desktop$ nano forkANDthread.c
elina@elina-vm:~/Desktop$ cat forkANDthread.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int global_var;

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

```
elina@elina-vm:~/Desktop$ gcc forkANDthread.c -o fth -lpthread
elina@elina-vm:~/Desktop$ ./fth
Main before fork: global_var=10, local_var=5
Child: global_var=20, local_var=15
Parent: global_var=10, local_var=5
```

نتیجه: در ریسه، تمام ریسه‌ها از یک آدرس مشترک استفاده می‌کنند اما در پردازه، آدرس پردازه والد و فرزند متفاوت است. به همین دلیل در ریسه وقتی متغیر عمومی تعریف می‌کنیم، تغییرات روی تمامی ریسه‌ها اعمال می‌شود اما در پردازه این‌گونه نیست.

۷-۴-۴- پاس دادن متغیر ها به ریسه‌ها

ابتدا ساختار داده شده را به کد اضافه می‌کنیم. سپس تابع child را داریم که در آن عدد و پیام ریسه را چاپ می‌کنیم. در main هم یک بار برای ریسه ۱ داده‌ها را ایجاد کرده و ریسه را با ساختار خواسته شده می‌سازیم و یک بار برای ریسه ۲.

```
elina@elina-vm:~/Desktop$ cat struct.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>

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

void* child(void* arg) {
    stdata* data = (stdata*) arg;
    printf("Thread number: %d\n", data->thread_no);
    printf("Message: %s\n", data->message);
    return NULL;
}

int main() {
    pthread_t th1, th2;
    stdata data1, data2;
    data1.thread_no = 1;
    strcpy(data1.message, "Message from thread 1!");
    pthread_create(&th1, NULL, child, (void*)&data1);
    pthread_join(th1, NULL);

    data2.thread_no = 2;
    strcpy(data2.message, "Message from thread 2!");
    pthread_create(&th2, NULL, child, (void*)&data2);
    pthread_join(th2, NULL);

    return 0;
}
```

```
elina@elina-vm:~/Desktop$ gcc struct.c -o struct
elina@elina-vm:~/Desktop$ ./struct
Thread number: 1
Message: Message from thread 1!
Thread number: 2
Message: Message from thread 2!
```