ADVANCED UNIX PROGRAMMING FINAL EXAM REPORT

TEAM 9 — 林禾堃、馬毓昇、陳曦

Dec 2023

1. Question 1

1.1 How to run the code

`./q1 &`

`kill -USR1 %1`

`kill -INT %1`

`kill -TERM %1`

Just as the how the sample output was shown.

1.2 Code implementation

Thread 1 sets `SIGINT`'s handler as below to print the desired message indicating the signal is being handled:

```c
void thread1_handler(int signo)
{
    printf("T1 handing SIGINT\n");
}

void* thread1(void* arg)
{
    signal(SIGINT, thread1_handler);
    return NULL;
}
```

Similarly, thread 2 sets `SIGTERM`'s handler as below to print the desired message indicating the signal is being handled:

```c
void thread2_handler(int signo)
{
    printf("T2 handing SIGTERM\n");
}

void* thread2(void* arg)
{
    signal(SIGTERM, thread2_handler);
    return NULL;
}
```

Also, for thread 3, `SIGUSR1`'s handler is set as below to print the desired message indicating the signal is being handled:

```c
void thread3_handler(int signo)
{
    printf("T3 handing SIGUSR1\n");
}

void* thread3(void* arg)
{
    signal(SIGUSR1, thread3_handler);
    return NULL;
}
```

Finally, in main function, we create the threads.

```c
int main()
{
    pthread_t t1, t2, t3;

    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);
    pthread_create(&t3, NULL, thread3, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    while(1);

    return 0;
}
```

2. Question 2

2.1 How to run the code

`./q2 <usec>`

Just as the how the sample output was shown.

2.2 Code implementation

The `sleep_us` function was built upon `select`, which is done by setting the `tval` according to the input `nusecs`.

```c
void
sleep_us(unsigned int nusecs)
{
    struct timeval tval;
    tval.tv_sec = nusecs / 1000000;
    tval.tv_usec = nusecs % 1000000;
    select(0, NULL, NULL, NULL, &tval);
}
```

In main function, we call `sleep_us` with the command line argument, and record the time before and after the call, then calculate and output the elapsed time in microseconds by them.

```c
int main(int argc, char *argv[])
{
    unsigned int t = atoi(argv[1]);

    struct timeval start, end;
    gettimeofday(&start, NULL);
    sleep_us(t);
    gettimeofday(&end, NULL);

    long tt = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
    printf("Sleep time: %ld us\n", tt);

    return 0;
}
```

3. Question 3

3.1 How to run the code

`./q3`

3.2 Code implementation

The three functions are implemented as follows, using `alarm`:

```c
void alarm_handler(int signo)
{
    printf("Alarm!\n");
}

void setAlarm(int sec)
{
    signal(SIGALRM, alarm_handler);
    alarm(sec);
}

void clearAlarm()
{
    alarm(0);
}
```

And we tested it with the given main function:

```c
int main()
{
    setAlarm(2); //set 2 sec alarm at 0s, will finish at 2s after execution
    sleep(1);
    setAlarm(6); //set 6 sec alarm at 1s, will finish at 7s after execution
    sleep(1);
    setAlarm(3); //set 3 sec alarm at 2s, will finish at 5s after execution
    sleep(4);
    clearAlarm();

    return 0;
}
```

3.3 Result

The output was as follows:

```
root@genet0:~/yusheng # ./q3
Alarm!
```

Since we only utilized single `alarm` to implement this, the alarm newly created by `setAlarm` will overwrite the old ones.

Following diagram illustrates the state changes of the alarms created by the main function from 0 second to 7 second:

As the chart shows, in this case, the “Alarm!” is output by the alarm created by the 3rd call of `setAlarm`.