Final report

Q1

- Run

```
make q1
./q1 &
kill -USR1 %1
kill -INT %1
kill -TERM %1
```

```
group17@generic:~/Advanced-UNIX-Programming_Student/mid3 $ kill -INT %1
T1 handling SIGINT
group17@generic:~/Advanced-UNIX-Programming_Student/mid3 $ kill -TERM %1
T2 handling SIGTERM
group17@generic:~/Advanced-UNIX-Programming_Student/mid3 $ kill -USR1 %1
T3 handling SIGUSR1
group17@generic:~/Advanced-UNIX-Programming_Student/mid3 $
```

- Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <pthread.h>
#include <signal.h>
const int N = 3;

static void sig_usr(int signo) {
        if(signo == SIGINT) printf("T1 handling SIGINT\n");
        else if(signo == SIGTERM) printf("T2 handling SIGTERM\n");
        else if(signo == SIGUSR1) printf("T3 handling SIGUSR1\n");
}

void *F1() {
        if(signal(SIGINT, sig_usr) == SIG_ERR)
                perror("can't catch SIGINT");
}

void *F2() {
        if(signal(SIGTERM, sig_usr) == SIG_ERR)
                perror("can't catch SIGTERM");
}

void *F3() {
        if(signal(SIGUSR1, sig_usr) == SIG_ERR)
                perror("can't catch SIGUSR1");
}

int main(void) {
        pthread_t threads[N];
        if(pthread_create(&threads[0], NULL, F1, NULL) != 0) {
                printf("T1 create error\n");
        }
        if(pthread_create(&threads[1], NULL, F2, NULL) != 0) {
                printf("T2 create error\n");
        }
        if(pthread_create(&threads[2], NULL, F3, NULL) != 0) {
                printf("T3 create error\n");
        }
        while(1);
}
```

- Line 32-41
  Create 3 threads(T1, T2, T3) to handle SIGINT, SIGTERM, and SIGUSR1.

- Line 16-29
  Handle signals.

Q2

- Code

```
1  #include <stdio.h>
2  #include <sys/select.h>
3  #include <sys/time.h>
4
5  void sleep_us(long val) {
6      struct timeval tval;
7      tval.tv_sec = val / 1000000;
8      tval.tv_usec = val % 1000000;
9      int ret = select(0, NULL, NULL, NULL, &tval);
10     // if (ret == -1) {
11     //     fprintf(stderr, "select error");
12     // } else if (ret == 0) {
13     //     fprintf(stderr, "select timeout\n");
14     // } else {
15     //     fprintf(stdout, "select success\n");
16     // }
17 }
18
19 int main(int argc, char *argv[]) {
20     long val = 0;
21     for (int i = 0; argv[1][i]; ++i)
22         val = val * 10 + (argv[1][i] - '0');
23
24     struct timeval start, end;
25     gettimeofday(&start, NULL);
26     sleep_us(val);
27     gettimeofday(&end, NULL);
28
29     printf("Sleep time: %ld us\n", (end.tv_sec - start.tv_sec)
30            * 1000000 + end.tv_usec - start.tv_usec);
31     return 0;
32 }
```

- Line 5 to Line 17
  將 `tval` (timeout) 設定為執行時輸入的微秒數（也就是要sleep的時間）
- Line 20 to Line 22
  將 argv 要讀進來的微秒數從 char[] 轉成 long

- Line 25 and Line 27
  用 `gettimeofday` 記錄進入 `sleep_us` 和執行完畢的時間
- Line 29 and Line 30
  將 `end` 和 `start` 的秒數和微秒相減即為執行的微秒數

Q3

- Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/resource.h>
4  #include <unistd.h>
5  #include <signal.h>
6
7  void printAlarm() {
8      puts("Alarm!");
9  }
10
11 void setAlarm(int x) {
12     alarm(x);
13 }
14
15 void clearAlarm() {
16     alarm(0);
17 }
18
19 int main() {
20     signal(SIGALRM, printAlarm);
21     setAlarm(2);
22     //set 2 sec alarm at 0s, will finish at 2s after execution
23     sleep(1);
24     setAlarm(6);
25     //set 6 sec alarm at 1s, will finish at 7s after execution
26     sleep(1);
27     setAlarm(3);
28     //set 3 sec alarm at 2s, will finish at 5s after execution
29     sleep(4);
30     clearAlarm(); //clear all alarms at 6s after execution
31     return 0;
32 }
```

- Line 8~10: 接收到 SIGALRM 後執行的 function
- Line 12~14: 把 alarm 設定為 x 秒後

- Line 16~18: 清除 alarm (傳入 0 為刪除)

在執行第五秒的時候會印出一個 “Alarm!”，其他兩個 alarm 因為被後面的覆蓋掉因此不會執行。