Thread Programming

과목	운영체제
제출 일자	2020년 09월 27일
담당 교수	박호현 교수님
학 과	컴퓨터공학과
학 번	20172129
이 름	박예솔

1. 코드 설명

* 전역 변수 설명

```
int gNumber;
int save = 0;
```

- 과제 조건②에서 요구한 global 변수 gNumber
- save가 0이면 producer에서 새로운 랜덤 숫자가 생성되지 않은 상태다.
  즉, consumer에서도 producer가 생성한 랜덤 숫자를 화면에 출력했다.
- save가 1이면 producer에서 새로운 랜덤 숫자가 생성된 상태이다.
  즉, consumer에서 아직 producer가 생성한 랜덤 숫자를 출력하지 않았다.

① 두 개의 thread를 만들어서 한 개는 'producer()'로, 다른 한 개는 'consumer()'라고 명명

- pthread_t 와 pthread_create( )를 사용해서 producer과 consumer 스레드를 만든다.
- prod_func()는 producer 스레드가 실행하는 함수이고, cons_func()는 consumer가 실행하는 함수이다.

② producer thread는 0~99 사이의 정수 중 하나를 random하게 생성하여 gNumber라는 이름의 global 변수에 저장하고 자신이 만든 random number를 화면(stdout)에 출력

- producer thread가 실행하는 prod_func() 함수에서 rand()를 이용해서 0~99사이의 정수를 랜덤하게 생성한다. 생성한 변수는 앞선 언급한 전역변수 gNumber에 저장한다.
- 새로 랜덤 숫자가 생성되어 gNumber에 저장되었으므로 save를 1로 변경한다.

③ consumer thread는 producer가 random number를 생성하여 gNumber라는 이름의 global 변수에 저장하면 즉시 이를 화면(stdout)에 출력

- gNumber에 새로 랜덤 숫자가 저장되면 save가 1이 될 것이므로 save == 1이면 cons_func()에서 바로 숫자 출력을 실행한다.

④ producer와 consumer는 위의 2, 3 단계의 동작을 각각 100번 반복한 후에 자신이 그동안 화면에 출력한 gNumber의 합을 main thread에 보내고 종료

- 두 스레드의 실행 순서는 프로세서가 결정하므로 무엇이 먼저 실행될 지 알 수 없다. 새로운 랜덤 숫자가 생성되고 난 후, consumer가 화면에 숫자를 출력하는 순서를 지키기 위해서 if-else 조건문 속에 while문을 사용한다. 자신의 스레드의 차례가 아니면 자신의 차례가 될 때까지 while문에서 wait()로 기다릴 수 있도록 한다. 각각 스레드가 2, 3단계 동작을 100번 반복할 수 있도록 한다.

⑤ main thread는 producer와 consumer가 종료될 때까지 기다렸다가 producer와 consumer가 자신에게 보내온 gNumber의 합이 일치하면 'success'를 화면에 출력하고, 아니면 'fail'을 화면에 출력하고 종료

- pthread_join()을 이용해서 각 스레드가 실행한 함수가 종료할 때 반환하는 pthread_exit()의 return 값을 받아온다.
- 이때 return하는 값의 자료형은 정수인데 전역변수가 아니므로 (void *)으로 결과값을 받아온다. 하지만 화면에 출력할 때는 정수로 출력하므로 (int *)로 형변환을 진행해준다.

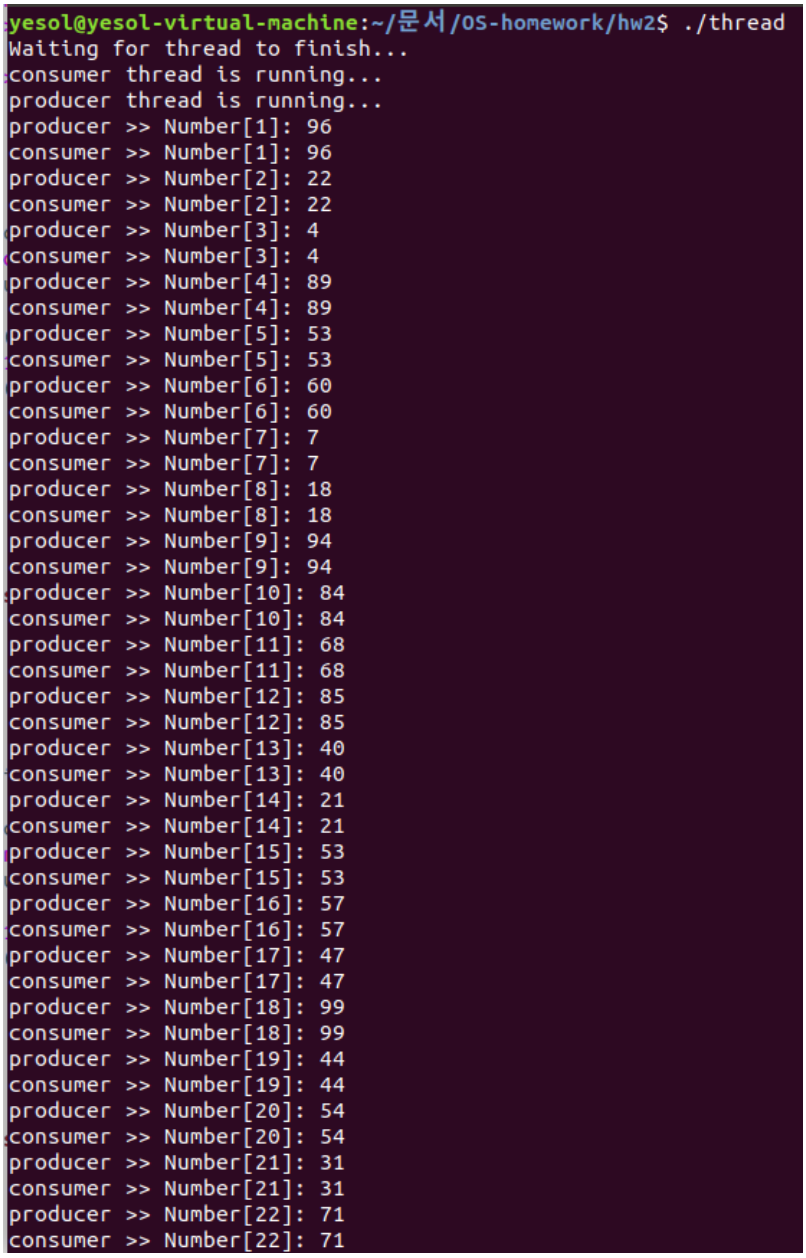
* 스레드가 실행하는 함수 코드

```
void *producer_func(void *arg){
        printf("producer thread is running...\n");
        int prod_sum = 0;
        int i = 0;
        srand(time(NULL));
        while(i < 100){
                if(save == 0){
                        gNumber = rand() % 100; // make random number
                        printf("producer >> Number[%d]: %d\n", i+1, gNumber);
                        fflush(stdout);
                        prod_sum += gNumber;
                        save = 1;
                        i++;
                }
                else{
                        while(save != 0){
                                wait();
                        }
                }
        }
        sleep(1);
        pthread_exit((void *)prod_sum);
}
void *consumer_func(void *arg){
        printf("consumer thread is running...\n");
        int cons_sum = 0;
        int i = 0;
        while(i < 100){
                if(save == 1){
                        printf("consumer >> Number[%d]: %d\n", i+1, gNumber);
                        fflush(stdout);
                        cons_sum += gNumber;
                        save = 0;
                        i++;
                }
                else{
                        while(save != 1){
                                wait();
                        }
                }
        }
        sleep(1);
        pthread_exit((void *)cons_sum);
}
```

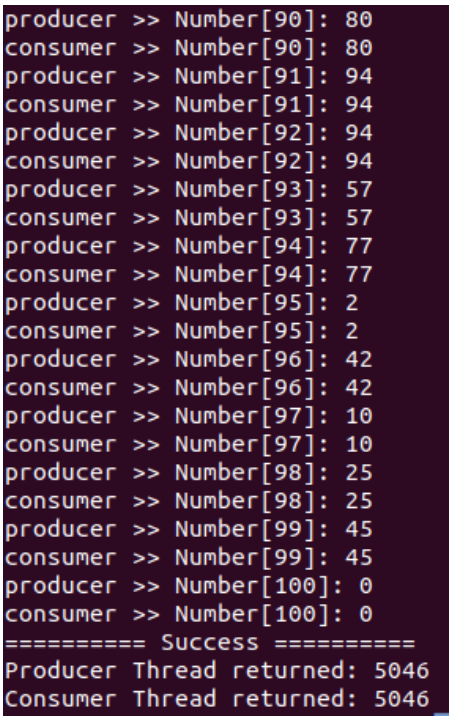
* 메인 함수 코드

```
int main(){
        int prod_res;
        int cons_res;
        void *prod_result;
        void *cons_result;

        pthread_t producer;
        pthread_t consumer;

        prod_res = pthread_create(&producer, NULL, producer_func, NULL);
        cons_res = pthread_create(&consumer, NULL, consumer_func, NULL);

        if(prod_res != 0 || cons_res != 0){
                printf("Thread create return error code\n");
                exit(-1);
        }

        printf("Waiting for thread to finish...\n");
        prod_res = pthread_join(producer, &prod_result);
        cons_res = pthread_join(consumer, &cons_result);

        if(prod_res != 0 || cons_res != 0){
                printf("Thread join return error code\n");
                exit(-1);
        }

        if((int*)prod_result == (int*)cons_result){
                printf("========== Success ==========\n");
                printf("Producer Thread returned: %d\n", (int*)prod_result);
                printf("Consumer Thread returned: %d\n", (int*)cons_result);
        }
        else{
                printf("========== Fail ==========\n");
                printf("Producer Thread returned: %d\n", (int*)prod_result);
                printf("Consumer Thread returned: %d\n", (int*)cons_result);
        }

        exit(NULL);
}
```

2. 실행 결과 화면



...



(23~89까지 중간과정은 다음 장에 첨부하겠다.)

⑥ 위의 5단계에서 'fail'이 출력되면 왜 그런지 이유를 분석

과제를 수행하는 과정에서 처음에는 간혹 fail이 출력되는 오류가 있었다. 계속 오류의 원인을 찾기 위해 고민했는데, 그 이유가 thread는 동시(concurrent)에 진행된다는 점이 문제였던 것 같다. 내가 따로 스레드의 동작을 제어해주지 않으면, producer가 진행되는 동안에도 consumer가 진행되는 문제가 발생한다. 따라서 producer → consumer 순서로 깔끔하게 진행시키기 위해 save 전역 변수, wait(), sleep() 등의 조건을 적절하게 사용하였더니 내가 오류가 해결되는 것을 확인할 수 있었다.

- 23 ~ 89까지 출력 화면

```
producer >> Number[23]: 35
consumer >> Number[23]: 35
producer >> Number[24]: 38
consumer >> Number[24]: 38
producer >> Number[25]: 89
consumer >> Number[25]: 89
producer >> Number[26]: 45
consumer >> Number[26]: 45
producer >> Number[27]: 34
consumer >> Number[27]: 34
producer >> Number[28]: 23
consumer >> Number[28]: 23
producer >> Number[29]: 61
consumer >> Number[29]: 61
producer >> Number[30]: 36
consumer >> Number[30]: 36
producer >> Number[31]: 52
consumer >> Number[31]: 52
producer >> Number[32]: 9
consumer >> Number[32]: 9
producer >> Number[33]: 10
consumer >> Number[33]: 10
producer >> Number[34]: 56
consumer >> Number[34]: 56
producer >> Number[35]: 98
consumer >> Number[35]: 98
producer >> Number[36]: 63
consumer >> Number[36]: 63
producer >> Number[37]: 69
consumer >> Number[37]: 69
producer >> Number[38]: 57
consumer >> Number[38]: 57
producer >> Number[39]: 34
consumer >> Number[39]: 34
producer >> Number[40]: 63
consumer >> Number[40]: 63
producer >> Number[41]: 42
consumer >> Number[41]: 42
producer >> Number[42]: 54
consumer >> Number[42]: 54
producer >> Number[43]: 48
consumer >> Number[43]: 48
producer >> Number[44]: 34
consumer >> Number[44]: 34
producer >> Number[45]: 75
consumer >> Number[45]: 75
producer >> Number[46]: 2
consumer >> Number[46]: 2
producer >> Number[47]: 92
consumer >> Number[47]: 92
producer >> Number[48]: 75
consumer >> Number[48]: 75
producer >> Number[49]: 53
consumer >> Number[49]: 53
producer >> Number[50]: 36
consumer >> Number[50]: 36
producer >> Number[51]: 29
consumer >> Number[51]: 29
producer >> Number[52]: 36
consumer >> Number[52]: 36
producer >> Number[53]: 7
consumer >> Number[53]: 7
producer >> Number[54]: 64
consumer >> Number[54]: 64
producer >> Number[55]: 74
consumer >> Number[55]: 74
producer >> Number[56]: 49
consumer >> Number[56]: 49
producer >> Number[57]: 61
consumer >> Number[57]: 61
producer >> Number[58]: 9
consumer >> Number[58]: 9
producer >> Number[59]: 72
consumer >> Number[59]: 72
producer >> Number[60]: 74
consumer >> Number[60]: 74
producer >> Number[61]: 97
consumer >> Number[61]: 97
producer >> Number[62]: 25
consumer >> Number[62]: 25
producer >> Number[63]: 83
consumer >> Number[63]: 83
producer >> Number[64]: 7
consumer >> Number[64]: 7
producer >> Number[65]: 33
consumer >> Number[65]: 33
producer >> Number[66]: 81
consumer >> Number[66]: 81
producer >> Number[67]: 71
consumer >> Number[67]: 71
producer >> Number[68]: 2
consumer >> Number[68]: 2
producer >> Number[69]: 90
consumer >> Number[69]: 90
producer >> Number[70]: 57
consumer >> Number[70]: 57
producer >> Number[71]: 65
consumer >> Number[71]: 65
producer >> Number[72]: 32
consumer >> Number[72]: 32
producer >> Number[73]: 11
consumer >> Number[73]: 11
producer >> Number[74]: 66
consumer >> Number[74]: 66
producer >> Number[75]: 19
consumer >> Number[75]: 19
producer >> Number[76]: 87
consumer >> Number[76]: 87
producer >> Number[77]: 20
consumer >> Number[77]: 20
producer >> Number[78]: 11
consumer >> Number[78]: 11
producer >> Number[79]: 62
consumer >> Number[79]: 62
producer >> Number[80]: 73
consumer >> Number[80]: 73
producer >> Number[81]: 99
consumer >> Number[81]: 99
producer >> Number[82]: 91
consumer >> Number[82]: 91
producer >> Number[83]: 10
consumer >> Number[83]: 10
producer >> Number[84]: 58
consumer >> Number[84]: 58
producer >> Number[85]: 7
consumer >> Number[85]: 7
producer >> Number[86]: 36
consumer >> Number[86]: 36
producer >> Number[87]: 7
consumer >> Number[87]: 7
producer >> Number[88]: 68
consumer >> Number[88]: 68
producer >> Number[89]: 97
consumer >> Number[89]: 97
```