실습보고서 4

교과목명	시스템프로그래밍		분반	001
제출일자	2021-12-13		교수자명	장희숙
팀명	이름	학번	이름	학번
초코파이썬칩	김기태	20173152		
	안대현	20173217		

▣ 실습 주제

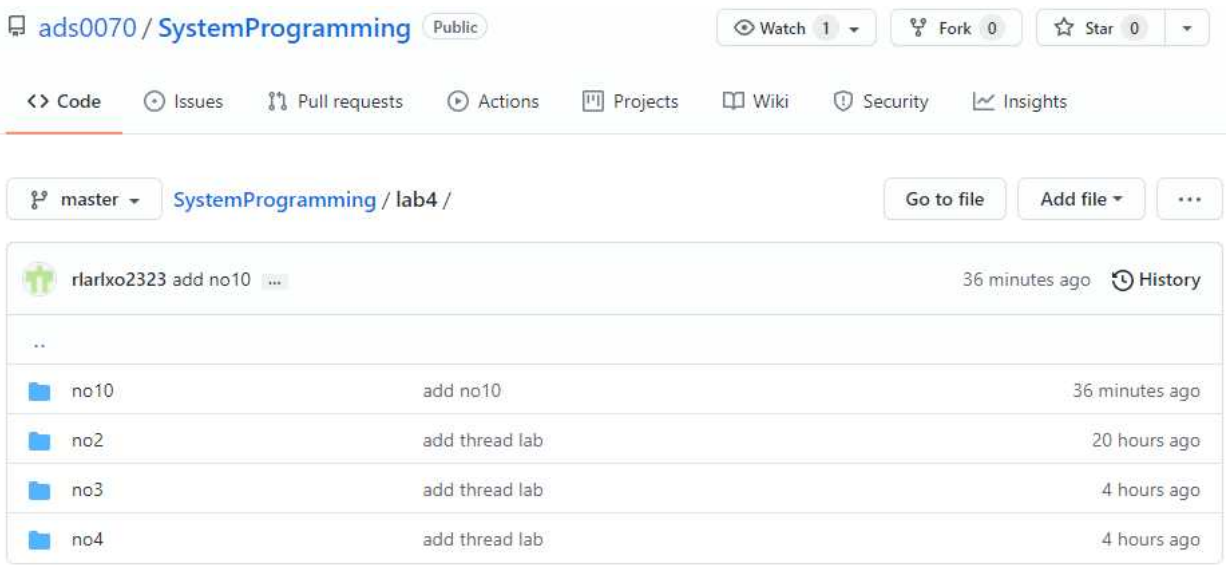
실습 4. 스레드, 소켓 및 GUI 프로그래밍

▣ 실습 결과

실습 순서

1. 자신의 github 저장소에 lab4 프로젝트를 생성하고 아래의 모든 과제 프로그램을 업로드한다.

git 주소 : https://github.com/ads0070/SystemProgramming/tree/master/lab4



2. 스레드 관련 함수들을 사용하여 프로그램을 작성하고 실행하고, 익숙해지도록 사

용해본다.

1) 스레드 생성 및 종료

hellothread.c

```
#include <pthread.h>
#include <stdio.h>
void *hello_thread (void *arg)
{
        printf ("Thread: Hello, World!₩n");
        return arg;
}
int main()
{
        pthread_t tid;
        int status;
        /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL, hello_thread, NULL);
        if (status !=0)
                perror ("Create thread");
        pthread_exit (NULL);
}
```

실행 결과

```
adh@adh:~/test/SystemProgramming/lab4/no2$ gedit hellothread.c
adh@adh:~/test/SystemProgramming/lab4/no2$ gcc -pthread -o hellothread hellothread.c
adh@adh:~/test/SystemProgramming/lab4/no2$ ./hellothread
Thread: Hello, World!
```

- pthread_create : 스레드 생성

* pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg);
* thread: 생성될 스레드의 ID 가 저장될 위치의 주소
* attr: 스레드의 속성을 나타내는 pthread_attr_t 형의 속성 개체
* start: 스레드의 시작 루틴
* arg: 스레드 시작 루틴이 넘겨받는 void * 형의 인자

- pthread_exit : 호출하는 스레드를 종료

* pthread_exit(void *retval);
* retval : 스레드가 종료할 때 반환 값

2) 스레드 인자 전달

hellotheads.c

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#define NUM_THREADS 3
void *hello_thread (void *arg)
{
        printf("Thread %p: Hello, World!₩n", arg);
        return arg;
}
int main()
{
        pthread_t tid[NUM_THREADS];
        int i, status;
        /* 쓰레드 생성 */
```

```
        for (i =0; i < NUM_THREADS; i++) {
                status = pthread_create (&tid[i], NULL, hello_thread, (void *) i);
                if (status !=0) {
                        fprintf (stderr, "Create thread %d: %d", i, status);
                        exit (1);
                }
        }
        pthread_exit (NULL);
}
```

실행 결과

```
adh@adh:~/test/SystemProgramming/lab4/no2$ gedit hellothreads.c
adh@adh:~/test/SystemProgramming/lab4/no2$ gcc -pthread -o hellothreads hellothreads.c
```

```
adh@adh:~/test/SystemProgramming/lab4/no2$ ./hellothreads
Thread (nil): Hello, World!
Thread 0x1: Hello, World!
Thread 0x2: Hello, World!
```

스레드 생성 시 인자값을 넘겨주어 스레드 옆에 인자값이 출력됨을 확인할 수 있다.

3) 스레드 취소

cancelthread.c

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
void *cancel_thread(void *arg)
{
        int i, state;
        for (i=0;;i++) {
                /* disables cancelability */
                pthread_setcancelstate(
                                PTHREAD_CANCEL_DISABLE, &state);
                printf("Thread: cancel state disabled₩n");
                sleep (1);
                /* restores cancelability */
                pthread_setcancelstate(state, &state);
                printf("Thread: cancel state restored₩n");
                if (i % 5 ==0)
                        pthread_testcancel();
        }
        return arg;
}
int main(int argc, char *argv[])
{
        pthread_t tid;
        int arg, status;
        void *result;
        if (argc <2) {
                fprintf (stderr, "Usage: cancelthreadtime(sec)₩n");
                exit(1);
        }
        /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL,
                        cancel_thread, NULL);
        if (status !=0) {
                fprintf (stderr, "Create thread: %d", status);
```

```
                exit (1);
        }
        arg = atoi (argv[1]);
        sleep (arg);
        status = pthread_cancel (tid);
        if (status !=0) {
                fprintf (stderr, "Cancel thread: %d", status);
                exit (1);
        }
        status = pthread_join (tid, &result);
        if (status !=0) {
                fprintf (stderr, "Join thread: %d", status);
                exit (1);
        }
        return (int)result;
}
```

실행 결과

```
adh@adh:~/test/SystemProgramming/lab4/no2$ ./cancelthread 3
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
```

* pthread_cancel – 스레드를 취소
* pthread_setcancelstate – 스레드 취소 상태 변경
* pthread_setcanceltype – 스레드 취소 유형 변경
* pthread_testcancel – 스레드 취소 요청 점검

인자 값의 크기만큼 취소상태를 바꿔 가는 것을 확인할 수 있다.

3. 스레드를 사용하여 생산자, 소비자 문제를 해결하는 제한 버퍼(Bounded Buffer)를 생성하고 활용하는 프로그램을 구현하시오, 단, 생산자 소비자 스레드는 각각 둘 이상 가능해야 한다.

buffer_test.c

```
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#define BUFFER_SIZE 20
#define NUMITEMS 30
typedef struct {
    int item[BUFFER_SIZE];
    int totalitems;
    int in, out;
    pthread_mutex_t mutex;
    pthread_cond_t full;
    pthread_cond_t empty;
} buffer_t;
buffer_t bb = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, 0, 0, 0,
               PTHREAD_MUTEX_INITIALIZER,
               PTHREAD_COND_INITIALIZER,
               PTHREAD_COND_INITIALIZER};
```

```
int produce_item() {
    int item = (int) (100.0 * rand() / (RAND_MAX +1.0));
    sleep((unsigned long) 1);
    printf("------------------------------------------------------------
------------------------------------------------------------₩n");
    printf("현재 투자자는 %d 명입니다.₩n", item);
    return item;
}
int insert_item(int item) {
    int status;
    status = pthread_mutex_lock(&bb.mutex);
    if (status !=0)
        return status;
    while (bb.totalitems >= BUFFER_SIZE && status ==NULL)
        status = pthread_cond_wait(&bb.empty,
                                   &bb.mutex);
    if (status !=0) {
        pthread_mutex_unlock(&bb.mutex);
        return status;
    }
    bb.item[bb.in] = item;
    bb.in = (bb.in +1) % BUFFER_SIZE;
    bb.totalitems++;
    if (status = pthread_cond_signal(&bb.full)) {
        pthread_mutex_unlock(&bb.mutex);
        return status;
    }
    return pthread_mutex_unlock(&bb.mutex);
}
int consume_item(int item) {
    sleep((unsigned long) 1);
    printf("₩n1 초 전에 투자한 %d명의 투자자에게 배당금을 지급합니다₩n", item);
    printf("------------------------------------------------------------
------------------------------------------------------------₩n");
}
int remove_item(int *temp) {
    int status;
    status = pthread_mutex_lock(&bb.mutex);
    if (status !=0)
        return status;
    while (bb.totalitems <=0 && status ==NULL)
        status = pthread_cond_wait(&bb.full, &bb.mutex);
    if (status !=0) {
        pthread_mutex_unlock(&bb.mutex);
        return status;
    }
    *temp = bb.item[bb.out];
    bb.out = (bb.out +1) % BUFFER_SIZE;
    bb.totalitems--;
    if (status = pthread_cond_signal(&bb.empty)) {
        pthread_mutex_unlock(&bb.mutex);
        return status;
    }
    return pthread_mutex_unlock(&bb.mutex);
}
void *producer(void *arg) {
    int item;
    while (1) {
        item = produce_item();
```

```
            insert_item(item);
        }
}
void *consumer(void *arg) {
        int item;
        while (1) {
            remove_item(&item);
            consume_item(item);
        }
}
void main() {
        int status;
        void *result;
        pthread_t producer_tid, consumer_tid;
        /* 쓰레드 생성 */
        status = pthread_create(&producer_tid, NULL,
                                producer, NULL);
        if (status !=0)
            perror("Create producer thread");
        status = pthread_create(&consumer_tid, NULL,
                                consumer, NULL);
        if (status !=0)
            perror("Create consumer thread");
        status = pthread_join(producer_tid, NULL);
        if (status !=0)
            perror("Join producer thread");
        status = pthread_join(consumer_tid, NULL);
        if (status !=0)
            perror("Join consumer thread");
}
```

실행 결과

```
adh@adh:~/test/SystemProgramming/lab4/no3$ gedit buffer_test.c
adh@adh:~/test/SystemProgramming/lab4/no3$ gcc -o buffer_test buffer_test.c -lpthread
```

```
adh@adh:~/test/SystemProgramming/lab4/no3$ ./buffer_test
현재 투자자는 84 명입니다.
현재 투자자는 39 명입니다.
1 초 전에 투자한 84명의 투자자에게 배당금을 지급합니다

현재 투자자는 78 명입니다.
1 초 전에 투자한 39명의 투자자에게 배당금을 지급합니다

현재 투자자는 79 명입니다.
1 초 전에 투자한 78명의 투자자에게 배당금을 지급합니다

현재 투자자는 91 명입니다.
1 초 전에 투자한 79명의 투자자에게 배당금을 지급합니다

현재 투자자는 19 명입니다.
1 초 전에 투자한 91명의 투자자에게 배당금을 지급합니다

현재 투자자는 33 명입니다.
1 초 전에 투자한 19명의 투자자에게 배당금을 지급합니다

현재 투자자는 76 명입니다.
1 초 전에 투자한 33명의 투자자에게 배당금을 지급합니다
```

조건 변수를 통해 공유 자원에 접근 시 해당 조건에 만족하는지에 따라 대기와 접근을 반복한다. 따라서 배당금에 접근하는 현재 투자자와 1초전 투자자의 수가 번갈아가며 접근하게 된다.

4. 클라이언트(자식) 스레드들로부터 메시지 전송 요청을 받으면 서버(부모) 스레드는 모든 클라이언트 스레드에게 메시지를 방송하는 프로그램을 구현하시오.

send_message.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#define NUM_THREADS 3
pthread_mutex_t mutex;
int status =0;
int caller =0;
int sum;
pthread_t tid[5];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // 뮤텍스
pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // 조건 변수
void *child_thread(void *arg) {
    while (1) {
        sleep(1);
        pthread_mutex_lock(&mutex);
        int random = rand() % 100;

        if (random <10){
            printf("₩n%d번 째 자식 쓰레드 : 메시지 전송 요청₩n", *((int *)arg));
            status =1;
            caller =*((int *)arg);
        }

        pthread_cond_wait(&cond, &mutex);
```

```
            printf("%d번 째 자식 쓰레드 : 부모로부터 메시지 수신₩n", *((int *) arg));
            pthread_mutex_unlock(&mutex);
        }
}
void *parent_thread(void *arg){
    while(1){
            sleep(1);
            if (status ==1) {
                pthread_mutex_lock(&mutex);
                pthread_cond_broadcast(&cond);
                printf("₩n부모 쓰레드 : 메세지 전송 요청이 호출되었음!₩n₩n");
                status =0;
                pthread_mutex_unlock(&mutex);
            }
    }
}
void main() {
    srand(time(NULL));
    int i;
    int id[5];
    pthread_create(&tid[0], NULL, parent_thread, (void*)&id[0]);

    for(i =1; i <5; i++){
        id[i] = i;
        pthread_create(&tid[i],NULL,child_thread,(void*)&id[i]);
    }
    for(i =1; i <5 ; i++){
        pthread_join(tid[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);
}
```

실행 결과

```
adh@adh:~/test/SystemProgramming/lab4/no4$ gedit send_message.c
adh@adh:~/test/SystemProgramming/lab4/no4$ gcc -o send_message send_message.c -lrt -lpthread
adh@adh:~/test/SystemProgramming/lab4/no4$ ./send_message

3번 째 자식 쓰레드 : 메시지 전송 요청

부모 쓰레드 : 메세지 전송 요청이 호출되었음!

3번 째 자식 쓰레드 : 부모로부터 메시지 수신
4번 째 자식 쓰레드 : 부모로부터 메시지 수신
1번 째 자식 쓰레드 : 부모로부터 메시지 수신
2번 째 자식 쓰레드 : 부모로부터 메시지 수신

2번 째 자식 쓰레드 : 메시지 전송 요청

부모 쓰레드 : 메세지 전송 요청이 호출되었음!

2번 째 자식 쓰레드 : 부모로부터 메시지 수신
4번 째 자식 쓰레드 : 부모로부터 메시지 수신
3번 째 자식 쓰레드 : 부모로부터 메시지 수신
1번 째 자식 쓰레드 : 부모로부터 메시지 수신
```

시그널 핸들러를 통해 특정 스레드에 시그널을 보낼 수 있다.
- SIG_SETMASK(시그널 설정)
- SIG_BLOCK(시그널 추가)
- SIG_UNBLOCK(시그널 제거)

시그널 동작과 시그널 핸들러는 프로세스 단위로 존재하기 때문에
자식 스레드가 부모 스레드에게 요청하면 부모스레드는 속해있는 모든 자식 스레드에게 메시지를

수신하는 방식으로 구현하였다.