|  |  |
| --- | --- |
| 년도-학기 | 2021년 1학기 |
| 과목명 | 임베디드시스템설계 |

|  |  |
| --- | --- |
| **LAB번호** | **제목** |
| 2 | Multi Threads, Semaphore, Mutex |

|  |  |
| --- | --- |
| 실험 일자 | 2021년 3월 12일 |
| 제출자 이름 | 강\*\* |
| 제출자 학번 | 201803\*\*\*\* |
| 팀원 이름 |  |
|  |  |

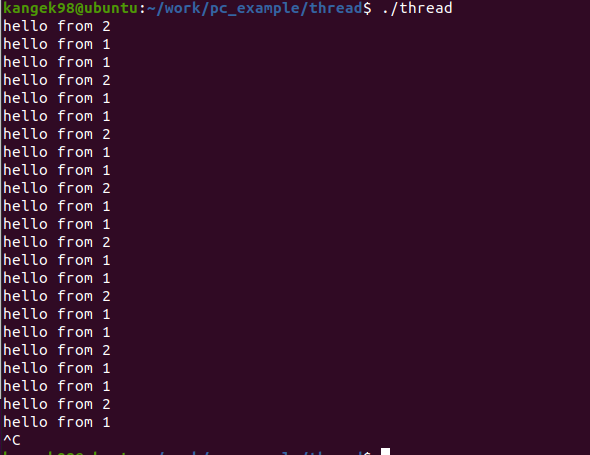
**Chapter 1. 프로그램의 동작 방식 설명 - 생략**

**Chapter 2. 결과**

**< Exercise 1 >**

- 예제 프로그램 thread.c를 수정하여 다음을 만족하는 프로그램을 작성한다.

- 2개의 thread를 생성하고 thread 1은 1초 간격으로, thread 2는 2초 간격으로 출력 메시지를 프린트한다.



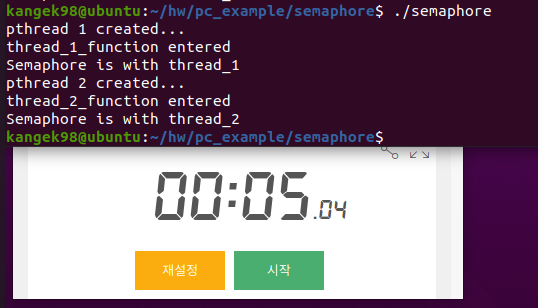
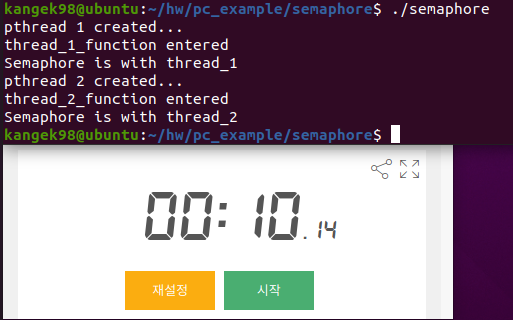
**< Exercise 2 >**

- 예제 프로그램 semaphore.c를 수정하여 다음을 만족하는 프로그램을 작성한다.

- 각 thread function에서 while(1)을 삭제한다.

- thread\_1 이 semaphore를 획득한 후, 5초 후에 thread\_2가 semaphore를 획득하도록 하고, 시간을 측정해서 정상 동작하는지 확인한다.

- 10초 후로 바꾸어서 동일하게 확인한다.



**< Exercise 3 >**

- 앞의 예제에서 삭제했던 while(1)을 다시 넣는다.

- thread\_1의 while loop를 아래와 같이 수정한다.

while(1)

{

sem\_wait(&binaySemphore); /\* down semaphore \*/

printf("Semaphore is with %s\n",ucThreadBuff);

sleep(5);

sem\_post(&binaySemphore); /\* up semaphore \*/

sleep(1);

}

- thread\_2의 while loop를 아래와 같이 수정한다.

while(1)

{

sem\_wait(&binaySemphore); /\* down semaphore \*/

printf("Semaphore is with %s\n",ucThreadBuff);

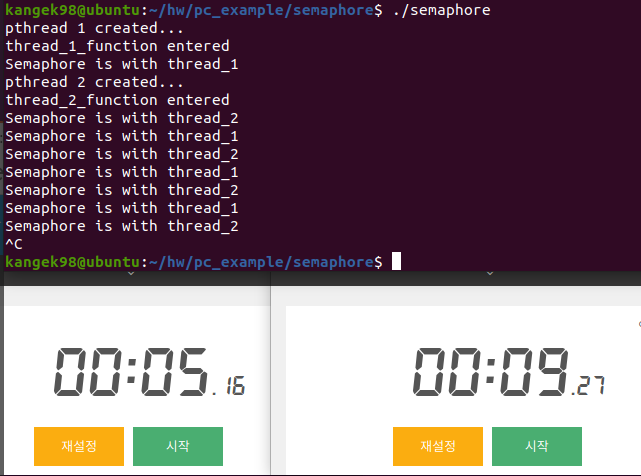
sleep(10);

sem\_post(&binaySemphore); /\* up semaphore \*/

sleep(1);

}

- 이 프로그램을 실행해서 각 thread가 semaphore를 가지고 있는 시간을 측정해서 확인해 본다.



**< Exercise 4 >**

- 예제 프로그램 mutex.c를 실행하여 출력을 관찰한다.

- 예제 프로그램 mutex.c를 다음과 같이 수정하여 출력을 관찰한다. (mutex lock과 unlock을 삭제)

for(i=0;i<10;i++) {

//pthread\_mutex\_lock(&lock);

printf("\rShared integer's value = %d before\n", shared\_data);

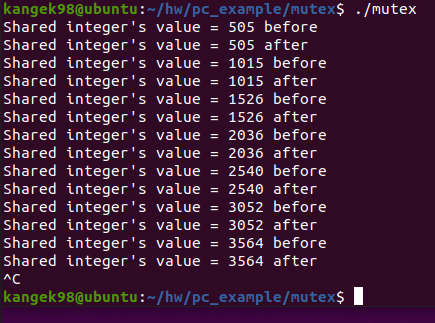
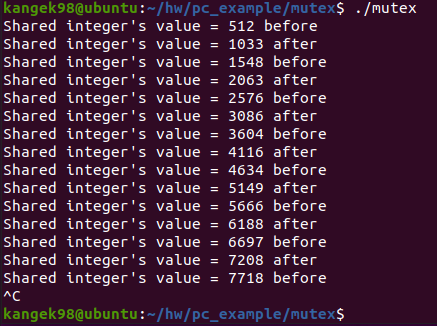
sleep(2);

printf("\rShared integer's value = %d after\n", shared\_data);

//pthread\_mutex\_unlock(&lock);

sleep(1);

}

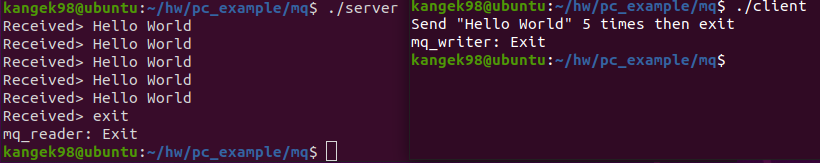
- 위의 두 경우의 출력이 다르게 나오는 이유에 대해서 설명하시오

수정 전 수정 후

수정 전 코드를 보면 pthread\_mutex\_lock(&lock)이 실행되고 sleep(1)에 의해 1초동안 lock 상태가 된다. 그래서 이 1초 동안에는 shared\_data를 증가시킬 수 없다. 그 후 pthread\_mutex\_unlock(&lock)을 통해 unlock 상태가 되면 shared\_data를 증가시킬 수 있다. 반면 수정 후의 코드는 mutex의 lock과 unlock을 없앴기 때문에 sleep(1) 전과 후 어디든 상관없이 shared\_data가 증가될 수 있는 것이다.

**< Exercise 5 >**

-“Hello World”라는 메시지를 1초에 한번씩 총 5회 보낸 후 “exit을 보내는 client 프로 그램을 작성해서 실행한다



**Chapter 3. 결론 및 Discussion - 생략**

**부록**

**Exercise 1 코드**

#include <unistd.h>

#include<stdio.h>

#include<pthread.h>

void\* say\_hello(void\* data)

{

char \*str;

str = (char\*)data;

while(1)

{

printf("%s\n",str);

sleep(1);

}

}

void\* say\_hello2(void\* data)

{

char \*str;

str = (char\*)data;

while(1)

{

printf("%s\n",str);

sleep(2);

}

}

void main()

{

pthread\_t t1,t2;

pthread\_create(&t1,NULL,say\_hello,"hello from 1");

pthread\_create(&t2,NULL,say\_hello2,"hello from 2");

pthread\_join(t1,NULL);

}

**Exercise 2 코드 (10초인 경우는 빨간색 표시한 줄에서 숫자를 10으로 바꾼다.)**

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <semaphore.h>

sem\_t binaySemphore;

void\* thread\_1\_function(void \*ptr);

void\* thread\_2\_function(void \*ptr);

int main()

{

int iRet;

pthread\_t thread\_1;

pthread\_t thread\_2;

unsigned char ucBuff[10];

/\* second param = 0 - semaphore is local() \*/

/\* second param = 1 - semaphore is global\*/

/\* third param = n - semaphore count value\*/

sem\_init(&binaySemphore, 0, 1);

strcpy(ucBuff,"thread\_1");

iRet=pthread\_create(&thread\_1, NULL,thread\_1\_function,(void \*)ucBuff);

if(iRet == 0)

{

printf("pthread 1 created...\n");

}

sleep(1);

strcpy(ucBuff,"thread\_2");

iRet=pthread\_create(&thread\_2, NULL,thread\_2\_function,(void \*)ucBuff);

if(iRet == 0)

{

printf("pthread 2 created...\n");

}

pthread\_join(thread\_1, NULL);

pthread\_join(thread\_2, NULL);

sem\_destroy(&binaySemphore); /\* destroy semaphore \*/

exit(0);

}

/\* prototype for thread routine \*/

void\* thread\_1\_function(void \*ptr)

{

unsigned char\* ucBuffPtr,ucThreadBuff[10];

ucBuffPtr = (unsigned char \*) ptr;

strcpy(ucThreadBuff,ucBuffPtr);

printf("thread\_1\_function entered\n");

// while(1)

{

sem\_wait(&binaySemphore); /\* down semaphore \*/

printf("Semaphore is with %s\n",ucThreadBuff);

sleep(5);

sem\_post(&binaySemphore); /\* up semaphore \*/

sleep(1);

}

pthread\_exit(0); /\* exit thread \*/

}

void\* thread\_2\_function(void \*ptr)

{

unsigned char\* ucBuffPtr,ucThreadBuff[10];

ucBuffPtr = (unsigned char \*) ptr;

strcpy(ucThreadBuff,ucBuffPtr);

printf("thread\_2\_function entered\n");

// while(1)

{

sem\_wait(&binaySemphore); /\* down semaphore \*/

printf("Semaphore is with %s\n",ucThreadBuff);

sleep(1);

sem\_post(&binaySemphore); /\* up semaphore \*/

sleep(1);

}

pthread\_exit(0); /\* exit thread \*/

}

**Exercise 5 코드**

#include <stdio.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <mqueue.h>

#include <errno.h>

#include <unistd.h>

#include <string.h>

#define MSG\_Q\_NAME "/MSG\_Q"

#define NO\_MAX\_MSG 10

#define MAX\_MSG 1024

#define STOP\_CMD "exit"

int main(int agrc, char \*argv[]) {

mqd\_t msg\_q;

struct mq\_attr attr;

int mq\_len;

char message[MAX\_MSG];

attr.mq\_flags = 0;

attr.mq\_maxmsg = NO\_MAX\_MSG;

attr.mq\_msgsize = MAX\_MSG;

attr.mq\_curmsgs = 0;

msg\_q = mq\_open (MSG\_Q\_NAME,O\_WRONLY,S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH, &attr);

if ( -1 == msg\_q) {

perror("mq\_open");

\_exit(-1);

}

printf("Send \"Hello World\" 5 times then exit\n");

for(int i=0; i<5;i++) {

strcpy(message,"Hello World");

printf("Send> %s\n",message);

message[strlen(message)]='\0';

mq\_len = strlen(message);

if ( -1 == mq\_send(msg\_q, message, mq\_len, 0)) {

perror("mq\_send");

mq\_close(msg\_q);

mq\_unlink(MSG\_Q\_NAME);

\_exit(-1);

}

sleep(1);

}

strcpy(message,"exit");

mq\_send(msg\_q,message, mq\_len,0);

printf("mq\_writer: Exit\n");

mq\_close(msg\_q);

mq\_unlink(MSG\_Q\_NAME);

return 0;

}