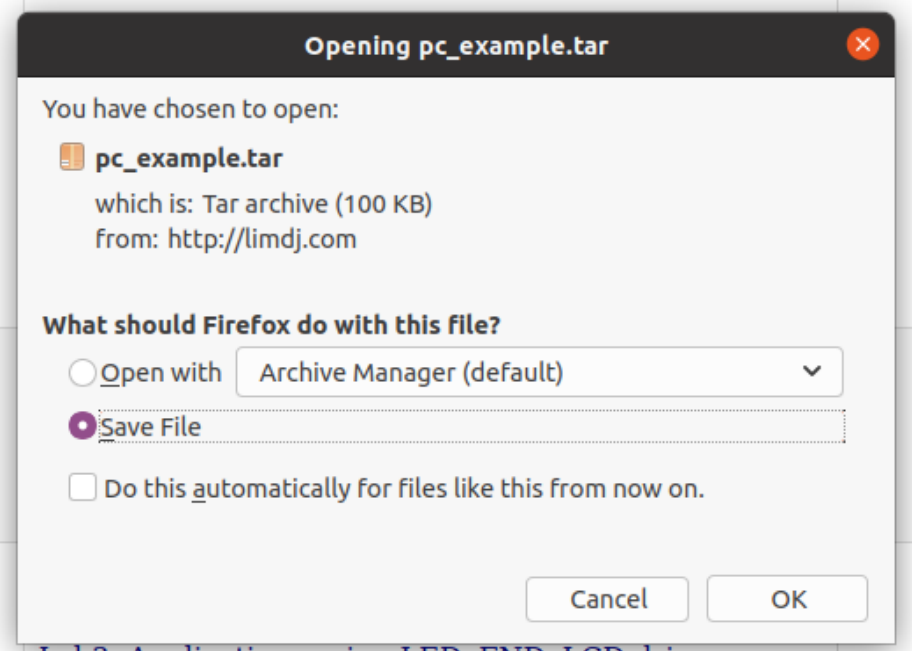
Lab 2

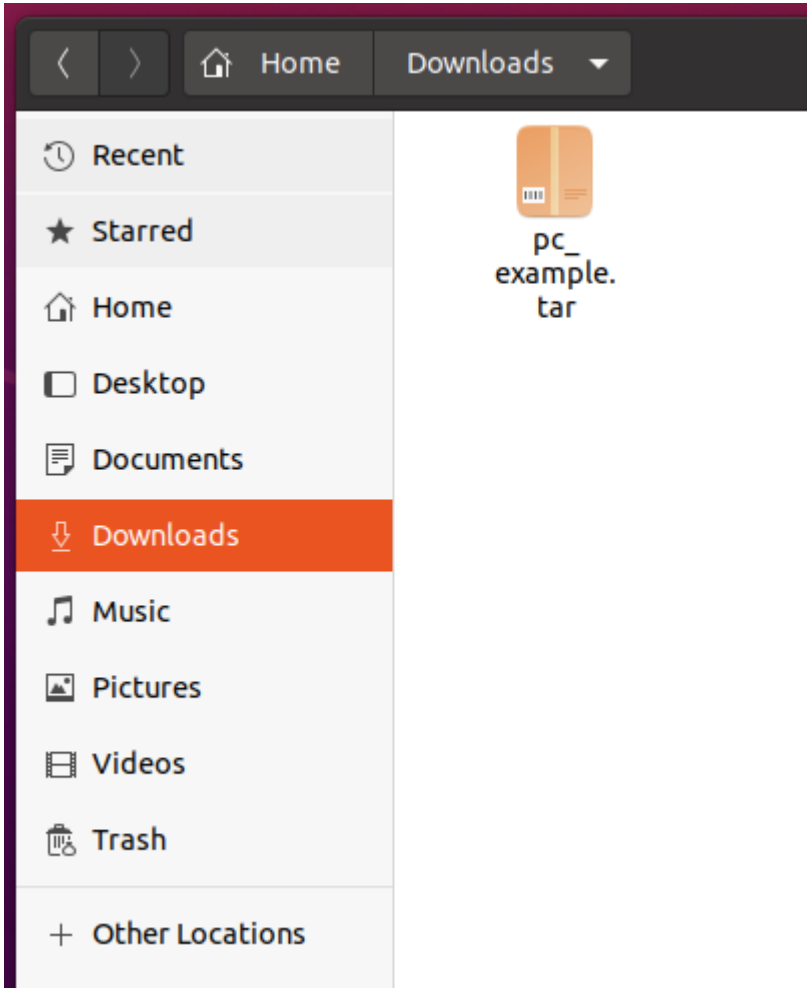
Thread, Semaphore, Mutex

Download Example

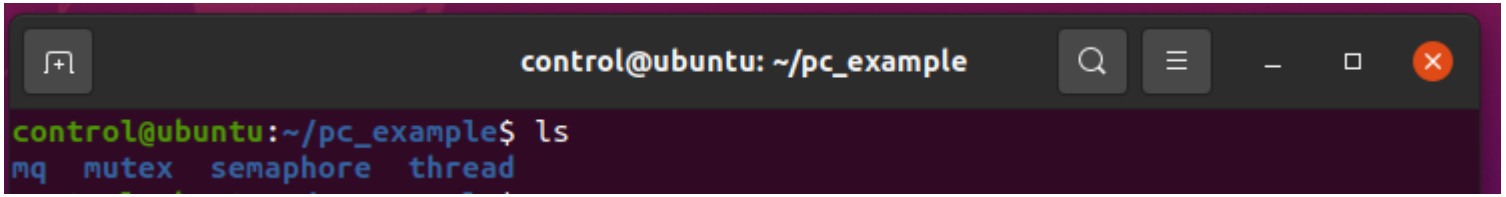
Week	강의 및 실습 내용	자료	보고서
1		Introduction.pdf Lab1.pdf Linux Fundamentals Makefile Tutorial	없음
2		RTOS.pdf Lab2.pdf pc_example.tar	제출
3	Lab3: Applications using LED, FND, LCD drivers	DeviceDriver.pdf Lab3.pdf pc_driver.tar.gz	제출



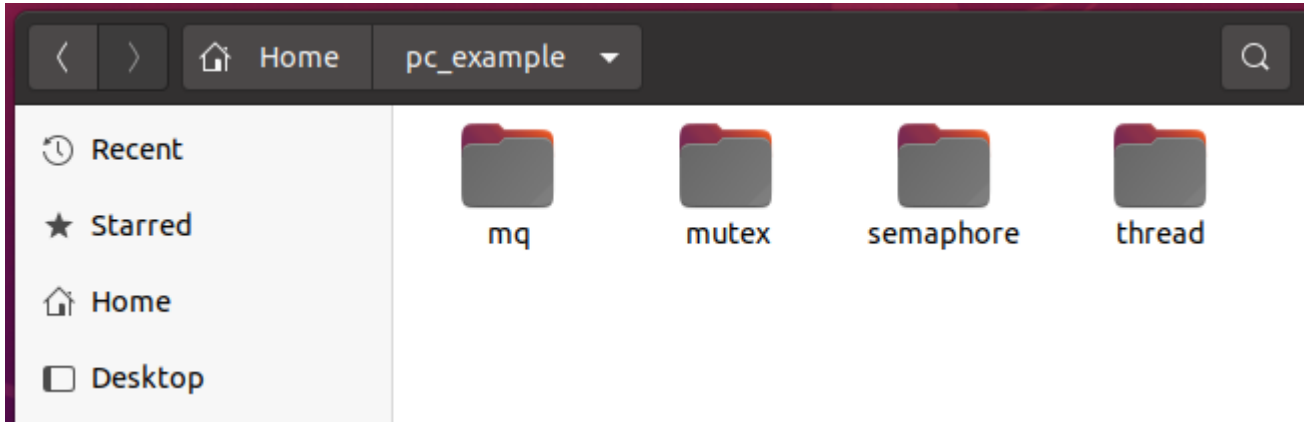
- Move to Home directory



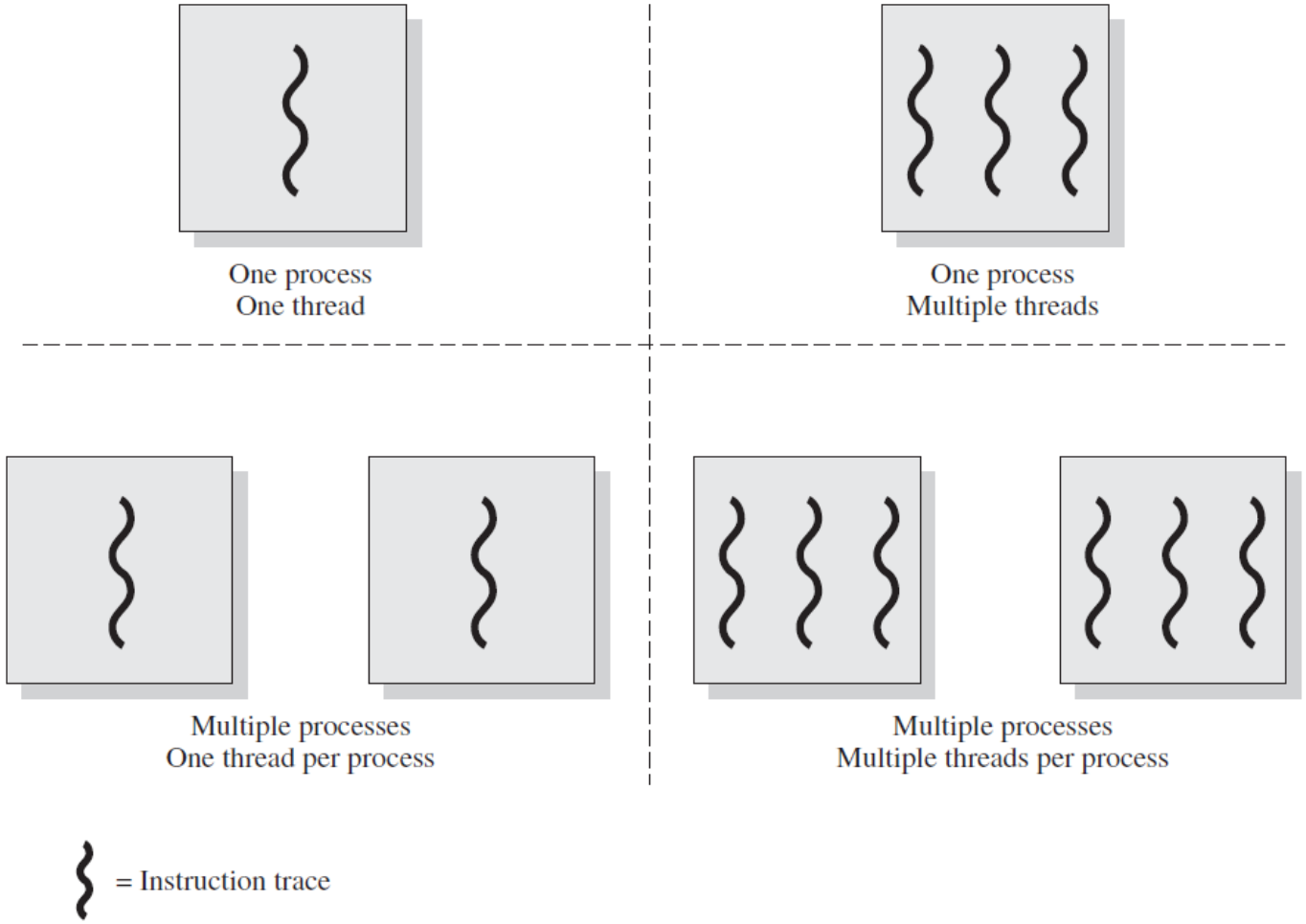
- Open Terminal
- $ tar xvf pc_example.tar
- $ cd pc_example
- $ ls



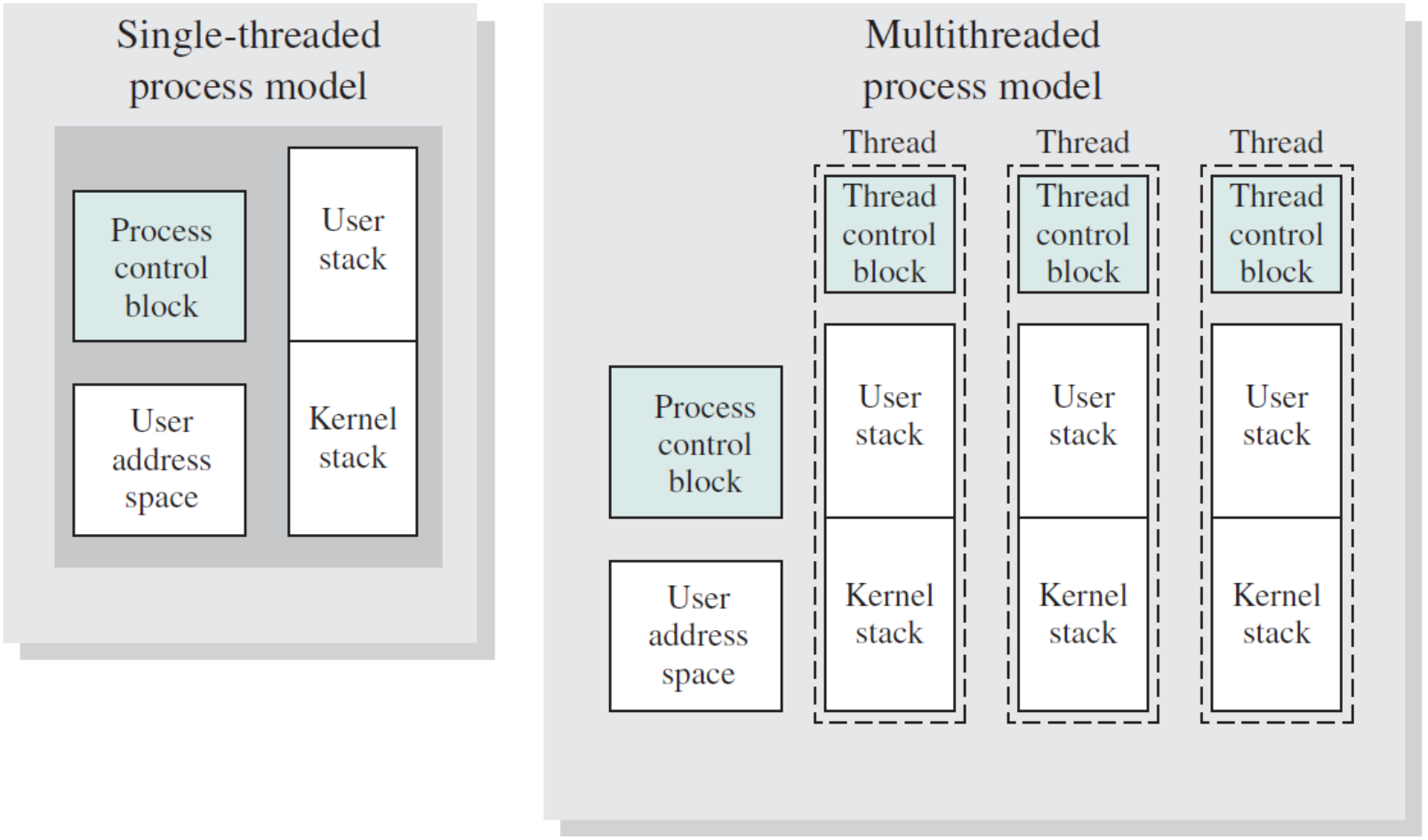
Example Program Folders



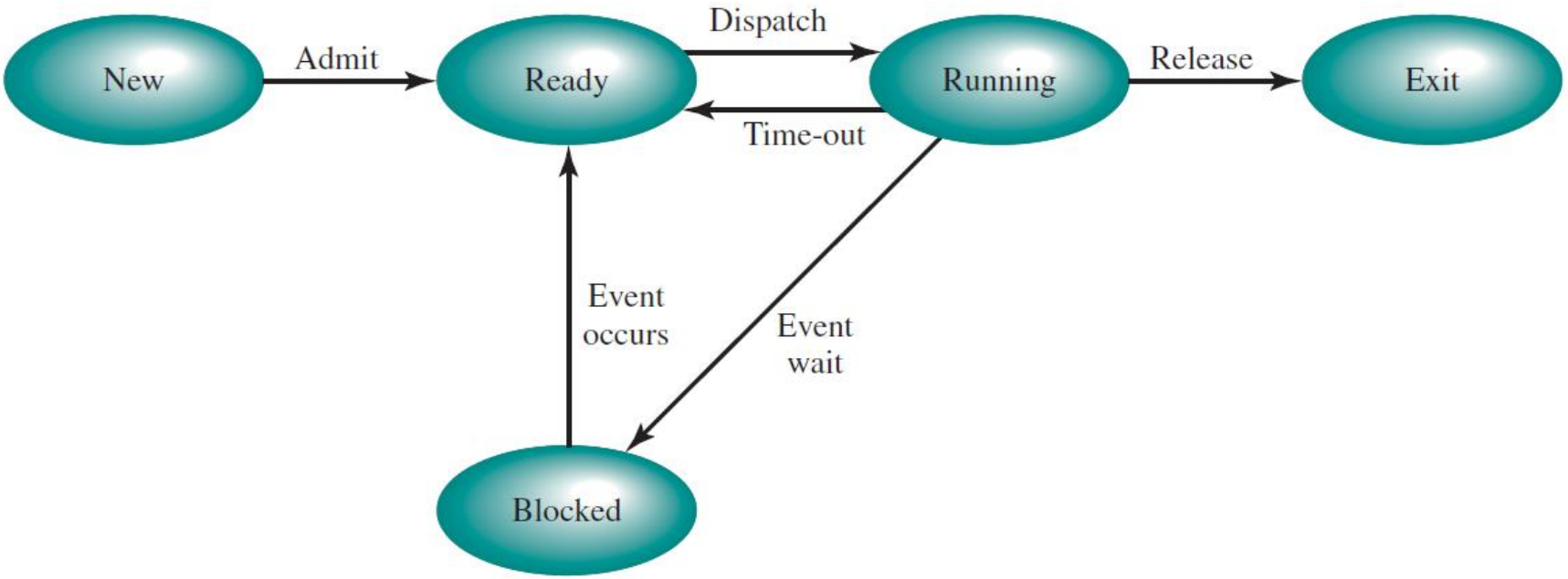
Threads and Processes



Single-Threaded and Multi-Threaded



Process Model (Five-State)



Create a new threads

#include <pthread.h>

int pthread_create(pthread_t **thread*, const pthread_attr_t **attr*, void *(*start_routine*) (void *), void **arg*);

Compile and link with *-pthread*.

The **pthread_create**() function starts a new thread in the calling process. The new thread starts execution by invoking *start_routine*(); *arg* is passed as the sole argument of *start_routine*().

Wait for the thread to terminate

#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);

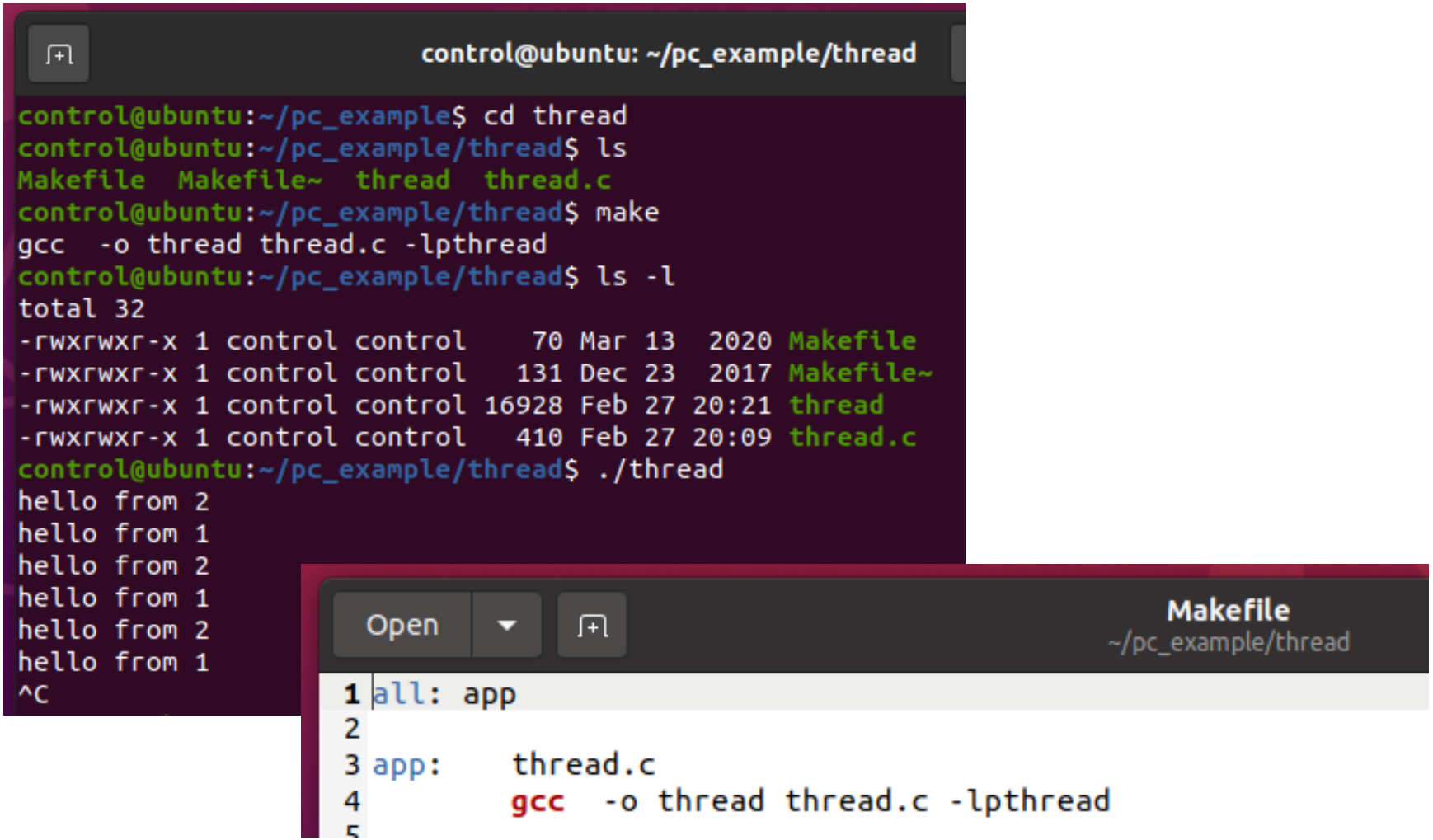
Compile and link with -pthread.

The **pthread_join**() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately.

thread.c

```
#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 main()
{
    pthread_t t1,t2;

    pthread_create(&t1,NULL,say_hello,"hello from 1");
    pthread_create(&t2,NULL,say_hello,"hello from 2");

    pthread_join(t1,NULL);
}
```

Build and Makefile



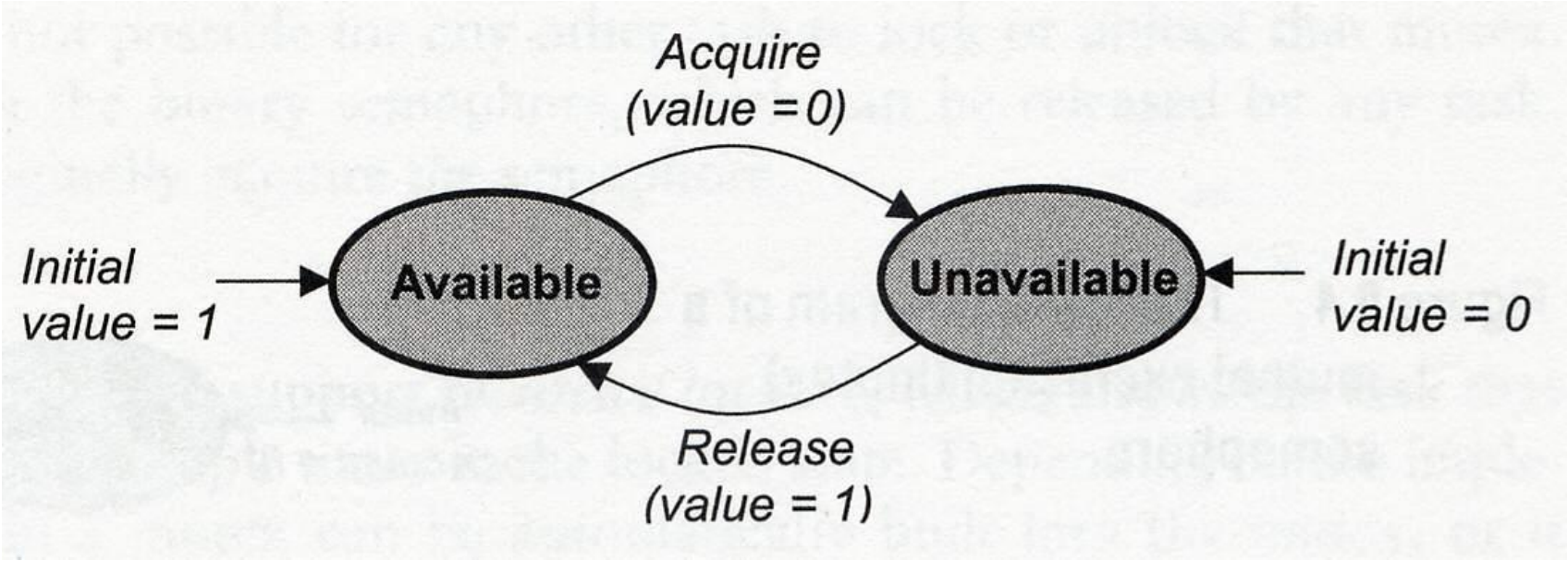
Exercise 1

- 예제 프로그램 thread.c 를 수정하여 다음을 만족하는 프로그램을 작성한다.

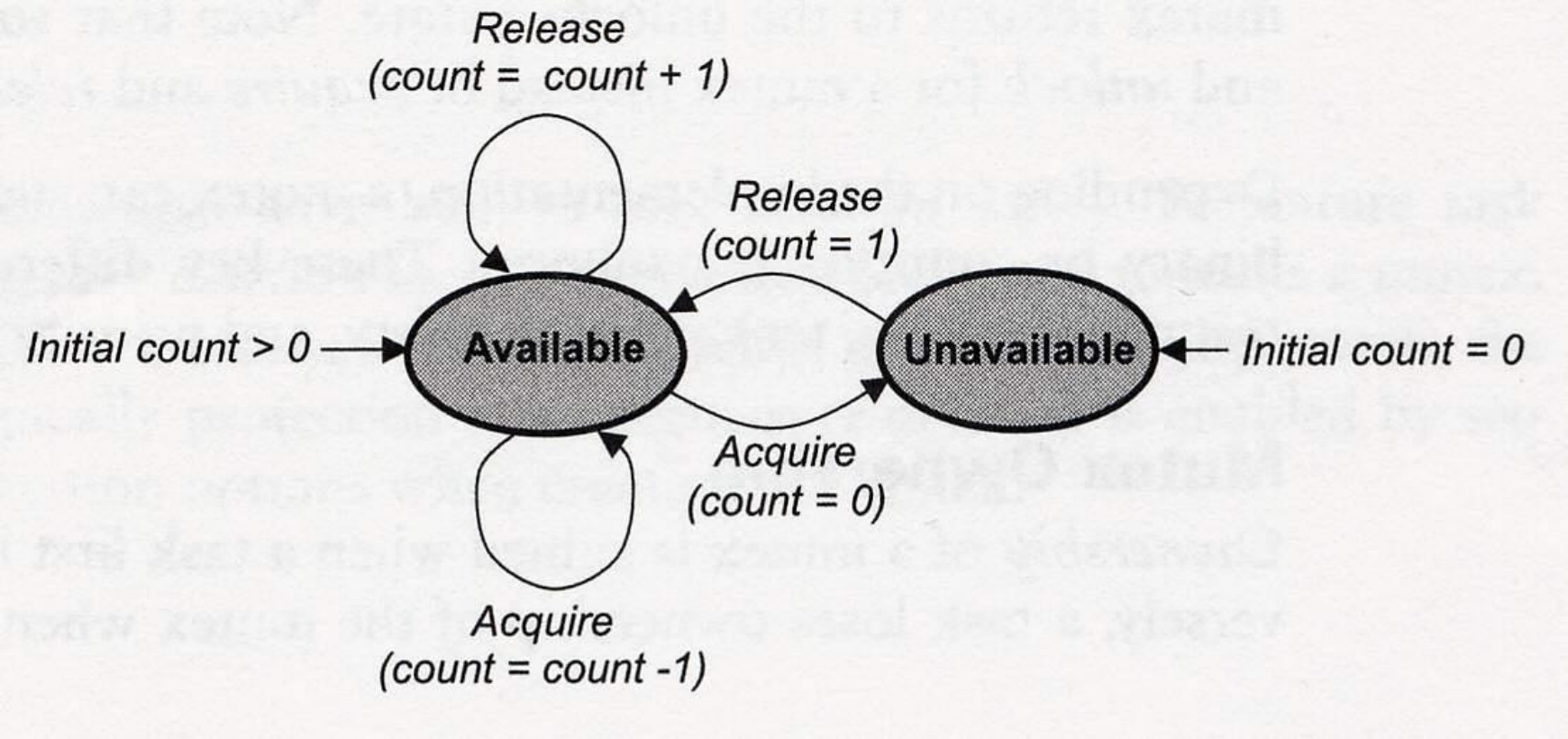
- 2개의 thread를 생성하고 thread 1은 1초 간격으로, thread 2는 2초 간격으로 출력 메시지를 프린트 한다.

Binary Semaphore

- Value: 0 unavailable/empty
- Value: 1 available/full



Counting Semaphore



Initialize a semaphore

#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);

Link with -pthread.

sem_init() initializes the unnamed semaphore at the address pointed to by sem. The ***value*** argument specifies the initial value for the semaphore.

The ***pshared*** argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.

If ***pshared*** has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads (e.g., a global variable, or a variable allocated dynamically on the heap).

Lock a semaphore

#include <semaphore.h>

int sem_wait(sem_t *sem);

Link with -pthread.

sem_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement (i.e., the semaphore value rises above zero), or a signal handler interrupts the call.

Unlock a semaphore

#include <semaphore.h>

int sem_post(sem_t *sem);

Link with -pthread.

sem_post() increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait(3) call will be woken up and proceed to lock the semaphore.

semaphore.c(1)

```
#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];
    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");
    }
```

semaphore.c(2)

```
    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);
}
```

semaphore.c(3)

```
/* 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(1);
        sem_post(&binaySemphore);       /* up semaphore */
        sleep(1);
    }
    pthread_exit(0); /* exit thread */
}
```

semaphore.c(4)

```
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 2

- 예제 프로그램 semaphore.c 를 수정하여 다음을 만족하는 프로그램을 작성한다.
- 각 thread function 에서 while(1)을 삭제한다.
- thread_1 이 semaphore를 획득한 후, 5초 후에 thread_2 가 semaphore를 획득하도록 하고, 시간을 측정해서 정상 동작하는지 확인 한다.
- 10초 후로 바꾸어서 동일하게 확인한다.

Exercise 3

- 앞의 예제 에서 삭제 했던 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);
   }
```

Exercise 3(계속)

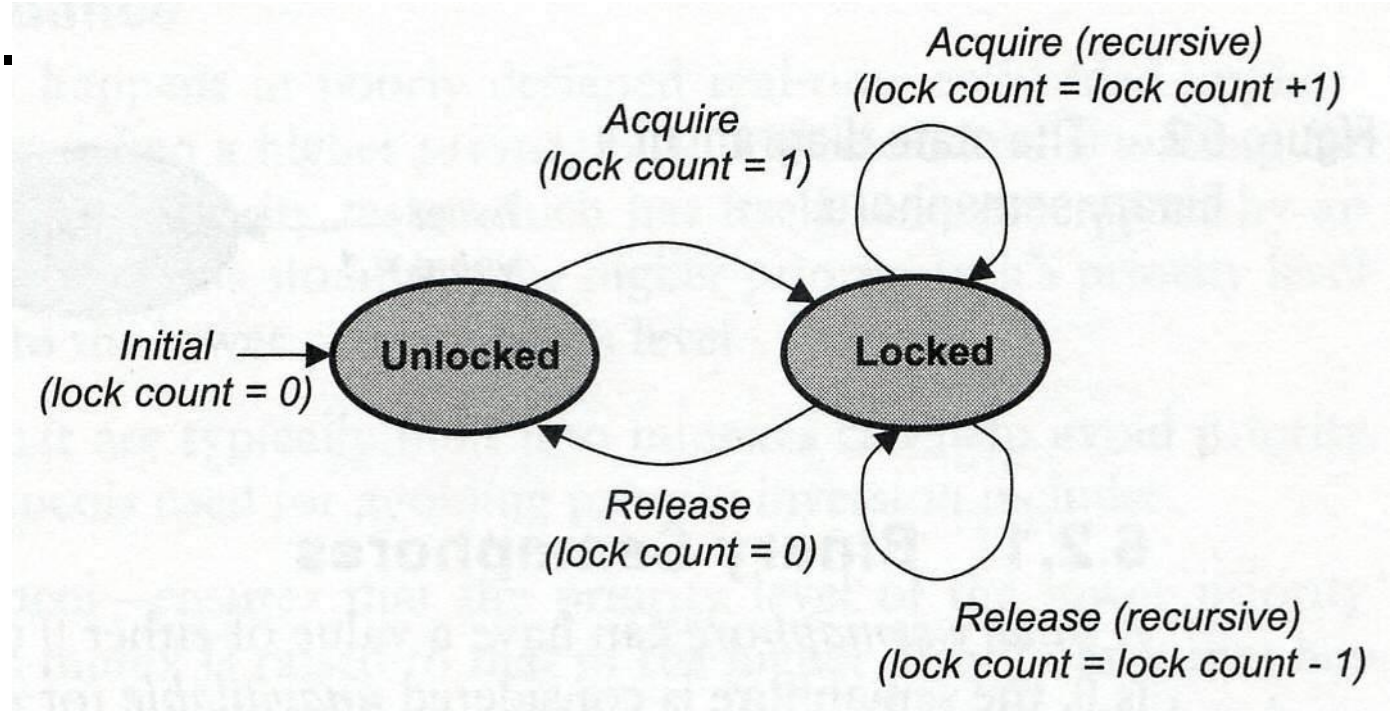
- thread_2의 while loop를 아래와 같이 수정한다.

```
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.

Mutual Exclusion (Mutex) Semaphore

- A special binary semaphore that supports ownership, recursive access, task deletion safety, priority inversion avoidance protocol.



Initialize a mutex

#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex,
const pthread_mutexattr_t *attr);

Link with -pthread.

The **pthread_mutex_init**() function initialises the mutex referenced by mutex with attributes specified by attr. If attr is NULL, the default mutex attributes are used; the effect is the same as passing the address of a default mutex attributes object. Upon successful initialisation, the state of the mutex becomes initialised and unlocked.

Destroy a mutex

#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);

Link with -pthread.

The **pthread_mutex_destroy**() function destroys the mutex object referenced by mutex; the mutex object becomes, in effect, uninitialised. An implementation may cause pthread_mutex_destroy() to set the object referenced by mutex to an invalid value. A destroyed mutex object can be re-initialised using pthread_mutex_init(); the results of otherwise referencing the object after it has been destroyed are undefined.

It is safe to destroy an initialised mutex that is unlocked. Attempting to destroy a locked mutex results in undefined behaviour.

Lock a mutex

#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);

Link with -pthread.

The mutex object referenced by mutex is locked by calling **pthread_mutex_lock**(). If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

Unlock a mutex

#include <pthread.h>

int pthread_mutex_unlock(pthread_mutex_t *mutex);

Link with -pthread.

The **pthread_mutex_unlock**() function releases the mutex object referenced by mutex. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by mutex when pthread_mutex_unlock() is called, resulting in the mutex becoming available, the scheduling policy is used to determine which thread shall acquire the mutex.

mutex.c(1)

```
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t lock;
int shared_data;

void *thread_function(void *arg)
{
	int i;
	for (i=0;i<10000;i++) {
		pthread_mutex_lock(&lock);
		shared_data++;
		pthread_mutex_unlock(&lock);
		usleep(1000);
	}
	return NULL;
}
```

mutex.c(2)

```
int main(void)
{
	pthread_t thread_ID;
	void *exit_status;
	int i;

	pthread_mutex_init(&lock,NULL);
	pthread_create(&thread_ID, NULL, thread_function, NULL);
	sleep(1);
	for(i=0;i<10;i++) {
		pthread_mutex_lock(&lock);
		printf("\rShared integer's value = %d before\n", shared_data);
		sleep(1);
		printf("\rShared integer's value = %d after\n", shared_data);
		pthread_mutex_unlock(&lock);
		sleep(1);
	}
	printf("\n");

	pthread_join(thread_ID, &exit_status);

	pthread_mutex_destroy(&lock);
	return 0;
}
```

Exercise 4

- 예제 프로그램 mutex.c 를 실행하여 출력을 관찰한다.
- 예제 프로그램 mutex.c 를 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.

Message Queue

```
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
```

mqd_t mq_open(const char *name, int oflag);
mqd_t mq_open(const char *name, int oflag, mode_t mode,
struct mq_attr *attr);

Link with **-lrt**.

mq_open() creates a new POSIX message queue or opens an existing queue. The queue is identified by name.

Message Queue

mode

If the queue name is to be created, then the new queue has the file permission bits as specified in mode. If any bits are set other than file permission bits, they are ignored. Read and write permissions are analogous to receive and send permissions, respectively; execute permissions are ignored.

Message Queue

attr

Like mode, this argument is only examined if the queue name is to be created. It's a pointer to an mq_attr structure that's to contain the attributes for the new queue. If this is NULL, the following default attributes are used (provided that no defaults were specified when starting the message queue server):

Variable	Default value
mq_maxmsg	1024
mq_msgsize	4096
mq_flags	0

According to the POSIX 1003.4 documentation, if the attr pointer is non-NULL, then the new queue adopts the mq_maxmsg and mq_msgsize of attr. However, no mention is made of consultation of the mq_flags field. In the QNX implementation, the mq_flags field is consulted. This is due to the fact that some of the extended QNX options cannot be changed at runtime, and must be specified on creation. See the entry for mq_setattr() for the meaning of these flags.

Message Queue

#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr,
size_t msg_len, unsigned int msg_prio);

Link with -lrt.

mq_send() adds the message pointed to by *msg_ptr* to the message queue referred to by the message queue descriptor *mqdes*. The *msg_len* argument specifies the length of the message pointed to by *msg_ptr*; this length must be less than or equal to the queue's *mq_msgsize* attribute. Zero-length messages are allowed.

Message Queue

#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr,
size_t msg_len, unsigned int *msg_prio);

Link with -lrt.

mq_receive() removes the oldest message with the highest priority from the message queue referred to by the message queue descriptor *mqdes*, and places it in the buffer pointed to by *msg_ptr*. The *msg_len* argument specifies the size of the buffer pointed to by *msg_ptr*; this must be greater than or equal to the *mq_msgsize* attribute of the queue (see mq_getattr(3)).

server.c(1)

```
#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;
  char message[MAX_MSG];
  int mq_len;

  attr.mq_flags = 0;
  attr.mq_maxmsg = NO_MAX_MSG;
  attr.mq_msgsize = MAX_MSG;
  attr.mq_curmsgs = 0;
```

server.c(2)

```
  msg_q = mq_open (MSG_Q_NAME,O_CREAT|O_RDONLY,S_IRUSR | S_IWUSR |
      S_IRGRP | S_IROTH, &attr);
  if ( -1 == msg_q) {
    perror("mq_open");
  _exit(-1);
  }

  do {
    bzero(message, MAX_MSG);
    mq_len = mq_receive(msg_q, message, MAX_MSG, NULL);
    if ( -1 == mq_len) {
      perror("mq_receive");
      mq_close(msg_q);
      mq_unlink(MSG_Q_NAME);
      _exit(-1);
      }
      printf("Received> %s\n", message);
  } while (!(0 == strcmp(message,STOP_CMD)));
  printf("mq_reader: Exit\n");
  mq_close(msg_q);
  mq_unlink(MSG_Q_NAME);
  return 0;
}
```

client.c(1)

```
#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;
```

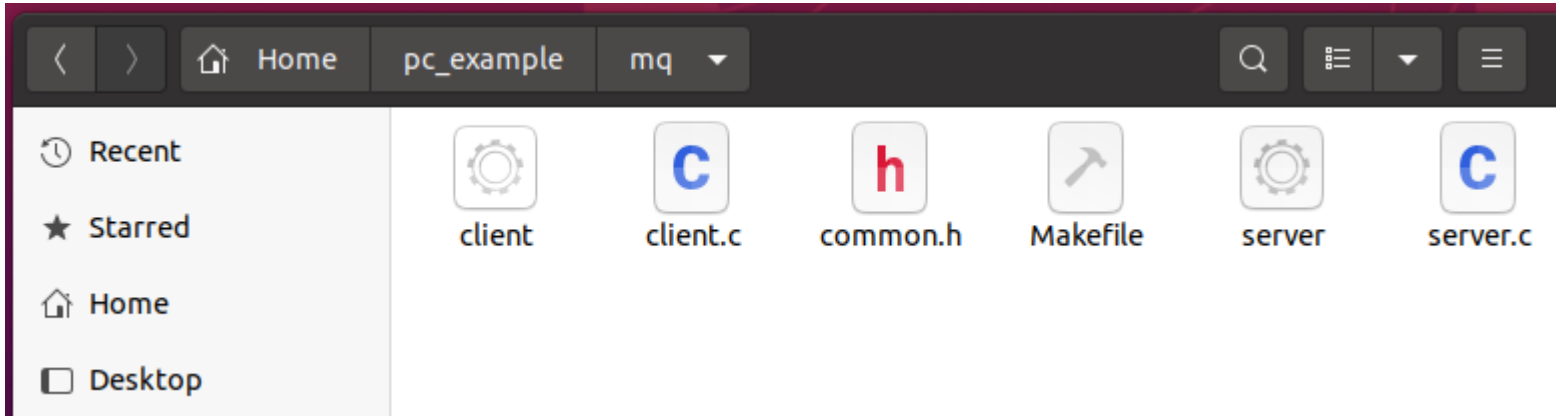
client.c(2)

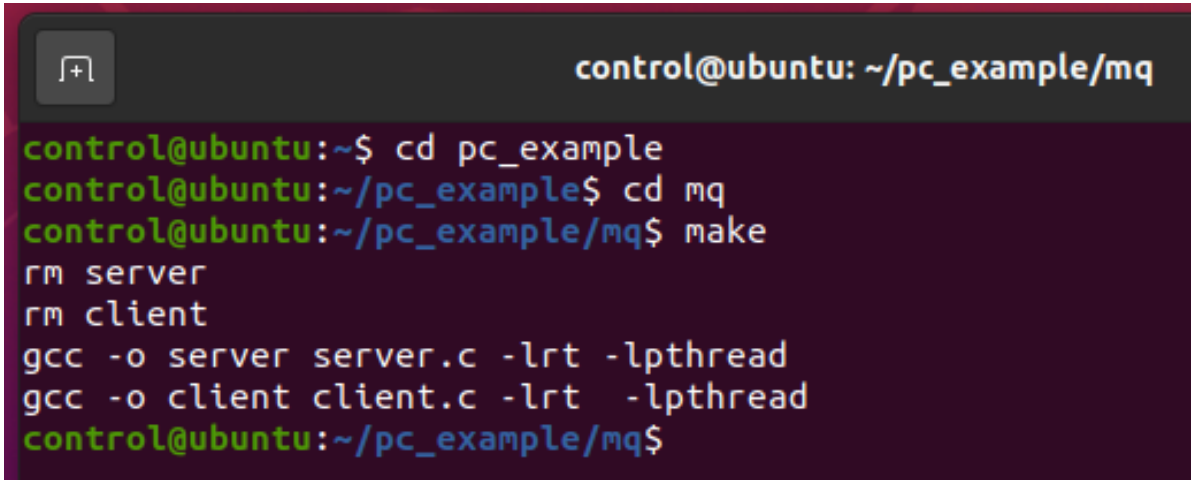
```
msg_q = mq_open (MSG_Q_NAME,O_WRONLY,S_IRUSR | S_IWUSR | S_IRGRP |
    S_IROTH, &attr);
 if ( -1 == msg_q) {
   perror("mq_open");
   _exit(-1);
 }

 printf("Enter \"exit\"  to stop: \n");
 do {
   bzero(message, MAX_MSG);
   printf("Send> ");
   fgets(message,sizeof(message), stdin);
   message[strlen(message)-1]='\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);
   }
 } while (!(0 == strcmp(message, STOP_CMD)));
```

client.c(3)

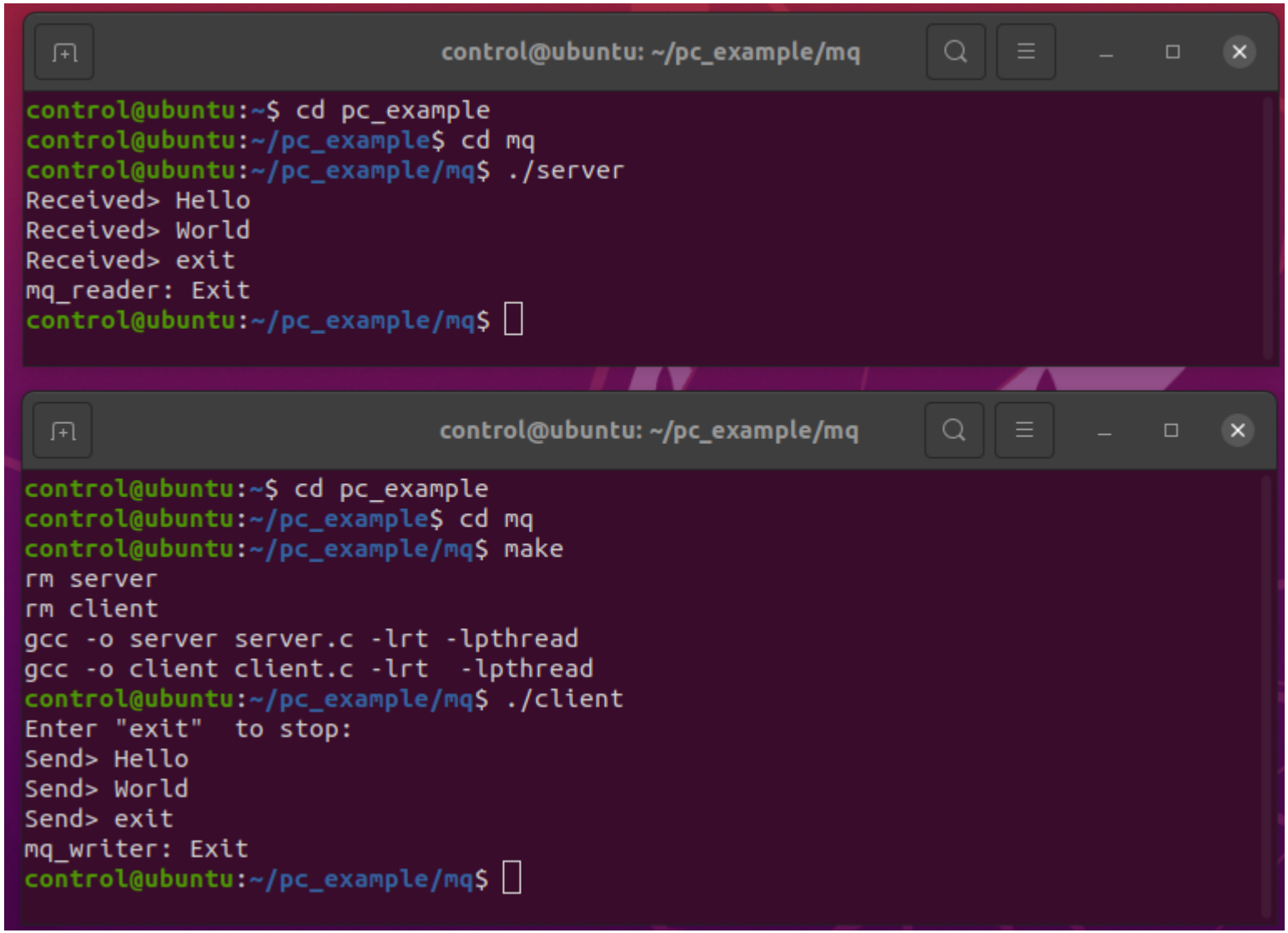
```
  printf("mq_writer: Exit\n");
  mq_close(msg_q);
  mq_unlink(MSG_Q_NAME);
  return 0;
}
```

Message Queue Example





Run server and client



Exercise 5

- “Hello World” 라는 메시지를 1초에 한번씩 총 5회 보낸 후 “exit” 을 보내는 client 프로그램을 작성해서 실행한다.