# Lab6 on POSIX Threads and Synchronization

## POSIX thread (pthread) libraries

Threads require less overhead than “forking” or spawning a new process because the system does not initialize a new system virtual memory space and environment for the process.

The purpose of using the POSIX thread library in your software is to execute software faster.

## Thread Basics:

All threads within a process share the same address space.

Threads in the same process share: process instructions, most data, open files, signals and signal handlers, current working directory, user and group id.

Each thread has a unique: thread id, set of registers, stack pointer, stack for local variables, return address, signal mask, priority, return value: error number.

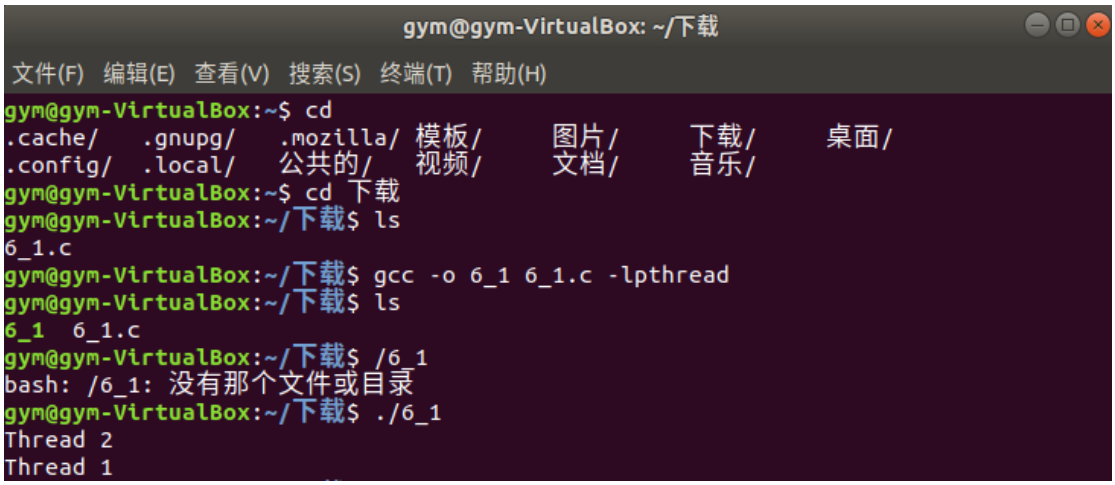
## Thread Creation and Termination:

Code：

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *print_message_function(void *prt)
{
	char *message;
	message=(char *)prt;
	printf("%s\n",message);
}
int main()
{
	pthread_t thread1,thread2;
	char *message1="Thread 1";
```

```
    char *message2="Thread 2";
    int iret1,iret2;

    //create independent threads each of which will execute function

    iret1 = pthread_create(&thread1,NULL,print_message_function,(void*)message1);
    iret2 = pthread_create(&thread2,NULL,print_message_function,(void*)message2);

    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);

    exit(0);
}
```

Snapshot:




## Thread Synchronization

The threads library provides three synchronization mechanisms:

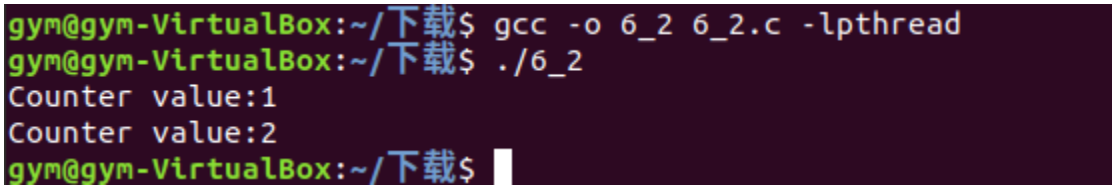
- Mutexes – Mutual exclusion lock: Block access to variables by other threads. This enforces exclusive access by a thread to a variable or set of variables.
- Joins – Make a thread wait till others are complete(terminated).
- Condition variables – data type pthread_cond_t.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *functionC(void *ptr);
pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
```

```
int main()
{
	int rc1,rc2;;
	pthread_t thread1,thread2;

	if(rc1=pthread_create(&thread1,NULL,functionC,NULL)){
		printf("Thread creation failed:%d\n",rc1);
	}

	if(rc2=pthread_create(&thread2,NULL,functionC,NULL)){
		printf("Thread creation failed:%d\n",rc2);
	}

	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);
	exit(0);
}
void *functionC(void *ptr)
{
	pthread_mutex_lock(&mutex1);
	counter++;
	printf("Counter value:%d\n",counter);
	pthread_mutex_unlock(&mutex1);
}
```

Snapshot:




## Condition Variables:

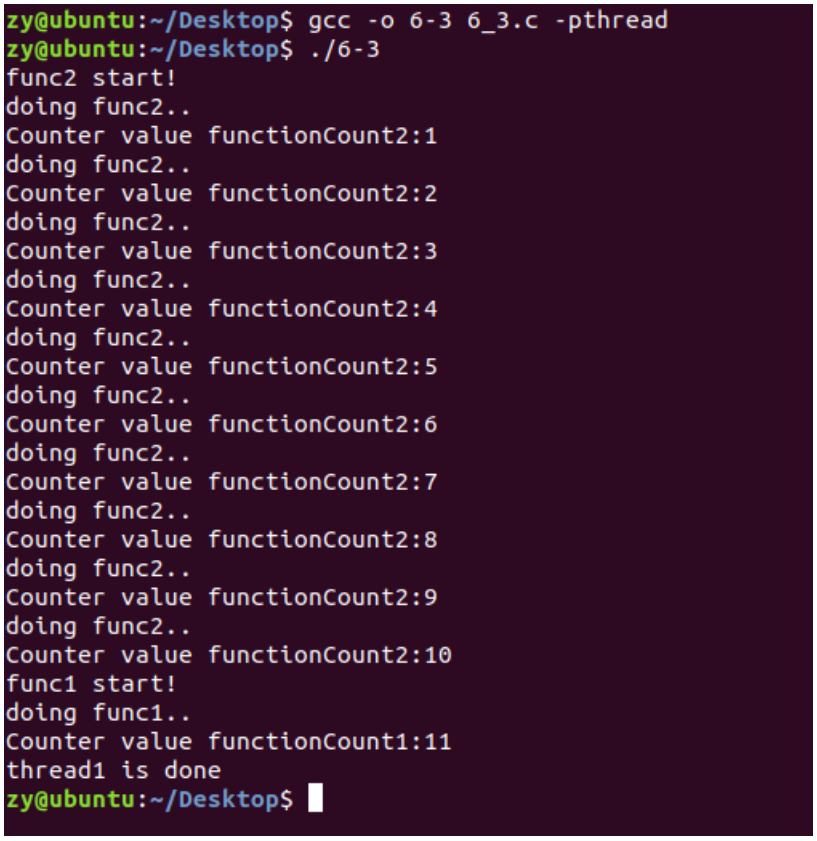
Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t condition_mutex =PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition_cond = PTHREAD_COND_INITIALIZER;

void *functionCount1();
```

```
void *functionCount2();
int ccount = 0;
#define COUNT_DONE 10
#define COUNT_HALT1 3
#define COUNT_HALT2 6
main()
{
	pthread_t thread1,thread2;

	pthread_create(&thread1,NULL,&functionCount1,NULL);
	pthread_create(&thread2,NULL,&functionCount2,NULL);
	pthread_join(thread1,NULL);
	printf("thread1 is done\n");
	pthread_join(thread2,NULL);
	exit(0);
}

void *functionCount1()
{
	printf("func1 start!\n");
	for(;;){
		pthread_mutex_lock(&condition_mutex);
		while(ccount>=COUNT_HALT1&&ccount<=COUNT_HALT2){
			pthread_cond_wait(&condition_cond,&condition_mutex);
		}
		printf("doing func1..\n");
		pthread_mutex_unlock(&condition_mutex);
		pthread_mutex_lock(&count_mutex);
		ccount++;
		printf("Counter value functionCount1:%d\n",ccount);
		pthread_mutex_unlock(&count_mutex);
		if(ccount>=COUNT_DONE) return (NULL);
	}
}

void *functionCount2()
{
	printf("func2 start!\n");
	for(;;){
		pthread_mutex_lock(&condition_mutex);
		if(ccount<COUNT_HALT1||ccount>COUNT_HALT2){
			pthread_cond_signal(&condition_cond);
		}
		printf("doing func2..\n");
```

```
            pthread_mutex_unlock(&condition_mutex);
            pthread_mutex_lock(&count_mutex);
            ccount++;
            printf("Counter value functionCount2:%d\n",ccount);
            pthread_mutex_unlock(&count_mutex);
            if(ccount>=COUNT_DONE) return (NULL);
        }
}
```

Snapshot:



```
zy@ubuntu:~/Desktop$ gcc -o 6-3 6_3.c -pthread
zy@ubuntu:~/Desktop$ ./6-3
func2 start!
doing func2..
Counter value functionCount2:1
doing func2..
Counter value functionCount2:2
doing func2..
Counter value functionCount2:3
doing func2..
Counter value functionCount2:4
doing func2..
Counter value functionCount2:5
doing func2..
Counter value functionCount2:6
doing func2..
Counter value functionCount2:7
doing func2..
Counter value functionCount2:8
doing func2..
Counter value functionCount2:9
doing func2..
Counter value functionCount2:10
func1 start!
doing func1..
Counter value functionCount1:11
thread1 is done
zy@ubuntu:~/Desktop$
```

Analyze:

There is a deadlock appeared. The mutex is locked by thread2, but thread2 give a signal to thread1 by pthread_cond_signal. Then thread1 wants to lock the mutex, which has been locked by thread2. So deadlock appeared.

## POSIX Message Queue

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
```

```
#include<mqueue.h>

#define MAX_MSG 15
#define OVER 37
#define MQ_NAME "/mqname"
#define MQ_SIZE 10
#define PMODE 0666

//to open the message queue
void init_queue(mqd_t *mq_desc,int open_flags)
{
	struct mq_attr attr;

	attr.mq_maxmsg = MQ_SIZE;
	attr.mq_msgsize=sizeof(int);
	attr.mq_flags=0;

	//open the mq
	//*mq_desc = mq_open(MQ_NAME,open_flags,&attr);
	*mq_desc = mq_open(MQ_NAME,open_flags,PMODE,&attr);
	if(*mq_desc ==(mqd_t)-1){
		perror("Mq opening failed");
		exit(-1);
	}
}

// to add an integer to the message queue
void put_integer_in_mq(mqd_t mq_desc,int data)
{
	int status;

	//send message
	status = mq_send(mq_desc,(char*)&data,sizeof(int),1);
	if(status == -1)
		perror("mq_send failed");
}

// to get an integer from message queue
int get_integer_from_mq(mqd_t mq_desc)
{
	ssize_t num_bytes_received=0;
	int data=0;

	//receive an int from mq
```

```
	num_bytes_received = mq_receive(mq_desc,(char*)&data,sizeof(int),NULL);
	if(num_bytes_received==-1)
		perror("mq_receive failed");
	return (data);
}

// code of a sending process
void sender()
{
	int n;
	mqd_t mqfd;
	printf("i am in sender\n");
	init_queue(&mqfd,O_CREAT|O_WRONLY);
	for(n=0;n<MAX_MSG;n++){
		printf("%d-->\n",n);
		put_integer_in_mq(mqfd,n);
	}
	put_integer_in_mq(mqfd,OVER);

	mq_close(mqfd);
	return ;
}

//code of receiving process
void receiver()
{
	int d;
	mqd_t mqfd;

	printf("i am in receiver\n");
	init_queue(&mqfd,O_RDONLY);
	while(1){
		d = get_integer_from_mq(mqfd);
		if(d==OVER) break;
		printf("-->%d\n",d);
	}
	mq_close(mqfd);
}

int main(void)
{
	int pid;

	//create two processes: one for sending, one for receiving
```

```
    pid = fork();
    if(pid!=0){
        receiver();
        mq_unlink(MQ_NAME);
    }
    else{
        sender();
    }
}
```

Snapshot:

```
gym@gym-VirtualBox:~/下载$ gcc -Wall -o 6_4 6_4.c -lrt
6_4.c: In function ‘main’:
6_4.c:92:16: warning: implicit declaration of function ‘fork’ [-Wimplicit-functi
on-declaration]
     pid = (int)fork();
                ^~~~
gym@gym-VirtualBox:~/下载$
```

The code lose the head file of fork(). So we add

#include <sys/types.h>

#include <unistd.h>

Result:

```
gym@gym-VirtualBox:~/下载$ ./6_4
i am in receiver
Mq opening failed: No such file or directory
gym@gym-VirtualBox:~/下载$ i am in sender
0-->
1-->
2-->
3-->
4-->
5-->
6-->
7-->
8-->
9-->
10-->
```

```
gym@gym-VirtualBox:~/下载$ ./6_4
i am in receiver
11-->
-->0
12-->
-->1
13-->
-->2
14-->
-->3
-->4
-->5
-->6
-->7
-->8
-->9
-->10
-->11
-->12
-->13
-->14
gym@gym-VirtualBox:~/下载$ i am in sender
0-->
1-->
2-->
3-->
4-->
5-->
6-->
7-->
8-->
9-->
10-->
gym@gym-VirtualBox:~/下载$
```

```
i am in receiver
-->0
-->1
-->2
-->3
-->4
-->5
-->6
-->7
-->8
-->9
-->10
i am in sender
0-->
1-->
2-->
3-->
4-->
5-->
6-->
7-->
8-->
9-->
10-->
11-->
-->0
-->1
-->2
-->3
-->4
-->5
-->6
-->7
-->8
-->9
-->10
-->11
-->11
-->12
-->13
-->14
gym@gym-VirtualBox:~/下载$ 12-->
13-->
14-->
gym@gym-VirtualBox:~/下载$
```

```
gym@gym-VirtualBox:~/下载$ ./6_4
i am in receiver
Mq opening failed: No such file or directory
gym@gym-VirtualBox:~/下载$ i am in sender
0-->
1-->
2-->
3-->
4-->
5-->
6-->
7-->
8-->
9-->
10-->
```

Error: No such file!

# POSIX Semaphore

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
#include<pthread.h>

int x=0;
sem_t m;

void *thread(void *arg)
{
	// critical section
	sem_wait(&m);//lock the mutex m
	x=x+1;
	sem_post(&m);//unlock
	return (NULL);
}

int main()
{
	pthread_t tid[10];
	int i;
	// semaphor m should be initialized by 1
	if(sem_init(&m,0,1)==-1){
		perror("Could not initialize mylock semaphore");
		exit(2);
	}

	//create 10 threads
	for(i=0;i<10;i++){
		if(pthread_create(&tid[i],NULL,thread,NULL)<0){
			perror("Error:thread cannot be created");
			exit(1);
		}
```

```
    }

    //wait for all created thread to terminate
    for(i=0;i<10;i++)pthread_join(tid[i],NULL);

    printf("Final value of x is %d\n",x);
    exit(0);
}
```

gcc -o 6_5 6_5.c -pthread

```
gym@gym-VirtualBox:~/下载$ gcc -Wall -o 6_5 6_5.c -pthread
gym@gym-VirtualBox:~/下载$ ./6_5
Final value of x is 10
gym@gym-VirtualBox:~/下载$
```