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

pthread

```
static void *ThreadEntry(void *arg)
{
     ClassName* obj = (ClassName*)arg; obj->ThreadOpeartion(); // invoke
     pthread_exit(NULL) in ThreadOperation() return NULL;
}

void MultiThread()
{ pthread_t threads[2];  for(unsigned
     int i = 0; i<2; i++)
     {  int  create  =  pthread_create(&threads[i],  NULL,  ThreadEntry,(void*)this);
           if(create != 0)
           { cerr<<"pthread_create failed"; } } for(unsigned int i = 0; i<2; i++)
     { int join = pthread_join(threads[i], NULL); if(join != 0)
           { cerr<<"pthread_join failed"; }
     }
}
```

condition variable

```
pthread_cond_t cond;
pthread_cond_init(&cond, NULL); ..
/* in thread 1 */
{ while(/* shared variable is not in state we want */)
        { pthread_cond_wait(&cond, &mtx); }
}
/* in thread 2 */
{ pthread_cond_signal(&cond);
} ..
pthread_cond_destroy(&cond);
```

mutex

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL); int
counter=0; // a shared variable
..
/* Function C */ void
functionC()
{ pthread_mutex_lock(&mutex); counter++;
      pthread_mutex_unlock(&mutex);
} ..
pthread_mutex_destroy(&mutex);
```

shared memory

```
/* before fork() */
int* shared = NULL;

shmID = shmget(IPC_PRIVATE, sizeof(int)*1, IPC_CREAT |
                                           SHM_R | SHM_W);

shared = static_cast<int *>(shmat(shmID, 0,0)); ...
/* after finishing all the job */
shmdt(shared);
shmctl(shmID, IPC_RMID, NULL);
```

semaphore

```
sem_t sem; int ret = sem_init(&sem,
1, 1); int counter = 0; // a shared
variable ..
/* Function C */
void functionC()
{ sem_wait(&sem);
        counter++;
        sem_post(&sem);
} ..
sem_destroy(&sem);
```