```c
/* 4) A word processor and a spreadsheet process are trying to access a printer that is shared
      among several processes. Devise a mechanism to ensure that the two processes cooperate
      in an orderly manner to avoid inconsistent system state. */

// In multiprogramming environment, several processes (here word and excel) may compete
//   for a finite number of resources(here single instance of printer)
// A process requests resources; if the resources are not available at that time, the
//   process enters a waiting state
// Sometimes, a waiting process is never again able to change state, because the resources
//   it has requested are held by other waiting processes , an inconsistent system state

// A printer cannot be simultaneously shared by several processes
// The mutual-exclusion condition must hold for nonsharable resources

// Process must request a resource before using it and must release the resource after using it

// Request and release of resources that are not managed by the operating system can be
//   accomplished with semaphores or through acquisition and release of a mutex lock
// Semaphore is a counter used to provide access to a shared data object for multiple processes

// So Printer process : creates one semaphore with value 1 (as one instance of printer resource)
// Word and Excel will use same semaphore for mututal exclusion

/* To obtain a shared resource(printer), a process(word, excel) needs to do the following:
Request -
   1. Test the semaphore that controls the resource
   2. If the value of the semaphore is positive, the process can use the resource
      In this case, the process decrements the semaphore value by 1, indicating that it has used
      one unit of the resource
   3. Otherwise, if the value of the semaphore is 0, the process goes to sleep until the
      semaphore value is greater than 0
      When the process wakes up, it returns to step 1
Release -
   When a process is done with a shared resource that is controlled by a semaphore, the
     semaphore value is incremented by 1
     If any other processes are asleep, waiting for the semaphore, they are awakened */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define PERMS 0666
#define PRINTERKEY 654321L // Assume as unique key, input of semget(), or use ftok()
 int main() // Please add error check for all API calls
  { // semget() creates System V semaphore set, int semget(key_t key, int nsems, int semflg);
    // nsems=1, for one resource type, semflg=IPC_CREAT, let printer create the semaphore
    int semid = semget( PRINTERKEY, 1 , IPC_CREAT | PERMS );//with PERMS permission

    // semctl() - semaphore control operations, int semctl(int semid, int semnum, int cmd, ...);
    // Perform cmd operation on semaphore set identified by semid, on the 0-th semaphore of
    int status = semctl(semid, 0, SETVAL , 1 ); //that set , initialize as one instance of resource

    return(0);
  } // 1. Run Printer program to create semaphore
```

```
// Assume this is word program

// Printer process has created the semaphore, with key 654321L
// A semaphore set consisting of a single member and initialized value to 1

// Word process should get the details of same semaphore using API -
//   semget, semctl, semop, check value of semaphore and take decision

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>

#define PERMS 0666
#define PRINTERKEY 654321L /* Same key that was used by printer to create semaphore*/

 int main() // Please add error check for all API calls
  { // Get semaphore id of PRINTERKEY , using semget(), IPC_EXCL - to get existing
   int semid = semget( PRINTERKEY, 1 , IPC_EXCL );            // semaphore identifier

   setbuf(stdout, NULL);

   // Use semop API with sembuf structure for semaphore operations

   // int semop(int semid, struct sembuf *sops, size_t nsops);
   // The function semop atomically performs an array of operations on a semaphore set
   // The nops argument specifies the number of operations (elements) in the array.

   /* struct sembuf
      unsigned short sem_num;// semaphore number, member # in set (0, 1, ..., nsems-1)
      short          sem_op; // semaphore operation , operation (negative, 0, or positive)
      short          sem_flg;// operation flags , IPC_NOWAIT, SEM_UNDO    */

   // If sem_op is :
   // 1. negative, we want to obtain resources that the semaphore controls
   // 2. 0, calling process wants to wait until the semaphore's value becomes 0
   // 3. positive, we want to return resource allocated to the process

   // If sem_flg is SEM_UNDO, handles the case of a process that terminates without
   //   releasing its resource

   // Assume semop call is successful,
   // If the semaphore's value is less than the absolute value of sem_op
   //    then the resources are not available, process waits / goes to sleep
   // Else access the resource
   // Then release resource

   // To allocate the resource, we call semop with a sem_op of -1; -1 because process will be
   // using one resource, one resource being taken from available resource

   // To release the resource, we perform a sem_op of +1; +1 because process is releasing
  // resource, one resource being added back to available resources

  // The semop function operates atomically ( Atomicity - A of ACID ) ; it does either all the
  // operations in the array or none of them

  struct sembuf semBufVar;
  semBufVar.sem_num = 0;
  semBufVar.sem_op = -1; // Request resource
  semBufVar.sem_flg = SEM_UNDO;

  printf("\n Word process Requested printer resource on - "); system("date");

  if ( semop(semid, &semBufVar, 1 ) != -1 )
   {
    printf("\n Printer Allocated to Word process on - "); system("date");
    printf("\n Word process Using resource ");
    sleep(10); // Say word process is using the printer for 10 seconds
   }

  semBufVar.sem_op = +1;     // Release / Return resource
  printf("\n\n Printer resource released by Word process on - "); system("date");
  semop(semid, &semBufVar, 1 );

  // What will be the semaphore value After last process releases resource

  return 0;
 }
/* 1. Run Printer program to create semaphore

  2. Run word process followed by Excel process or vice-versa

  ./02WordProcess

  Word process Requesting printer resource on - Wed Aug 30 22:05:52 IST 2017

  Printer Allocated to Word process on - Wed Aug 30 22:05:52 IST 2017

  Word process Using resource

  Printer resource released by Word process on - Wed Aug 30 22:06:02 IST 2017 */
```

```
// Assume this is excel program

// Printer process has created the semaphore, with key 654321L
// A semaphore set consisting of a single member and initialized value to 1

// Excel process should get the details of same semaphore using API -
//   semget, semctl, semop, check value of semaphore and take decision

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>

#define PERMS 0666
#define PRINTERKEY 654321L /* Same key that was used by printer to create semaphore*/

 int main() // Please add error check for all API calls
  { // Get semaphore id of PRINTERKEY , using semget(), IPC_EXCL - to get existing
   int semid = semget( PRINTERKEY, 1 , IPC_EXCL );           // semaphore identifier

   setbuf(stdout, NULL);

   // Use semop API with sembuf structure for semaphore operations

   // int semop(int semid, struct sembuf *sops, size_t nsops);
   // The function semop atomically performs an array of operations on a semaphore set
   // The nops argument specifies the number of operations (elements) in the array.

   /* struct sembuf
     unsigned short sem_num;// semaphore number, member # in set (0, 1, ..., nsems-1)
     short          sem_op; // semaphore operation , operation (negative, 0, or positive)
     short          sem_flg;// operation flags , IPC_NOWAIT, SEM_UNDO    */

   // If sem_op is :
   // 1. negative, we want to obtain resources that the semaphore controls
   // 2. 0, calling process wants to wait until the semaphore's value becomes 0
   // 3. positive, we want to return resource allocated to the process

   // If sem_flg is SEM_UNDO, handles the case of a process that terminates without
   //   releasing its resource

   // Assume semop call is successful,
   // If the semaphore's value is less than the absolute value of sem_op
   //    then the resources are not available, process waits / goes to sleep
   // Else access the resource
   // Then release resource

   // To allocate the resource, we call semop with a sem_op of -1;
   // -1 because it will be using on resource

   // To release the resource, we perform a sem_op of +1
   // +1 because resource is being release and added back to available resources

   // The semop function operates atomically ( Atomicity - A of ACID ) ; it does either all the
   // operations in the array or none of them

   struct sembuf semBufVar;
   semBufVar.sem_num = 0;
   semBufVar.sem_op = -1; // Request resource
   semBufVar.sem_flg = SEM_UNDO;

   printf("\n Excel process Requested printer resource on - "); system("date");

   if ( semop(semid, &semBufVar, 1 ) != -1 )
    {
     printf("\n Printer Allocated to Excel process on - "); system("date"); // Do some work
     printf("\n Excel process Using resource ");
     sleep(10); // Say excel process is using the printer for 10 seconds
    }

   semBufVar.sem_op = +1;     // Release / Return resource
   printf("\n\n Printer resource released by Excel process on - "); system("date");
   semop(semid, &semBufVar, 1 );

   // What will be the semaphore value After last process releases resource

   return 0;
  }
/* 1. Run Printer program to create semaphore

  2. Run Excel process followed by Word process or vice-versa

  ./03ExcelProcess

  Excel process Requested printer resource on - Wed Aug 30 22:05:55 IST 2017

  Printer Allocated to Excel process on - Wed Aug 30 22:06:02 IST 2017

  Excel process Using resource

  Printer resource released by Excel process on - Wed Aug 30 22:06:12 IST 2017 */

/* key  ,  key_t ftok(const char *pathname, int proj_id);
  ftok - convert a pathname and a project identifier to a System V IPC key
  key_t type System V IPC  key,  suitable for use with msgget, semget or shmget

  int status = semctl(semid, 0, GETVAL); Get value of semid semaphore */
```