Advance Operating Systems

Assignment-2

Due date: 10th March 2019

Question-1) Consider the following C codes that calls fork(). Run this program and explain the output. Then **strace** each executable and explain its outcome.

```
1.

int main()
{
   int i;
   for (i = 0; i < 3; i++) {
       if (fork() == 0) {
           printf("Child sees i = %d\n", i);
           exit(1);
       } else {
           printf("Parent sees i = %d\n", i);
       }
    }
}
```

```
2.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

int main() {

       int fd;

       close( 2 );

       printf( "HI\n" );

       fflush( stdout );

       fd = open( "newfile.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );

       printf( "==> %d\n", fd );

       printf( "WHAT?\n" );

       fprintf( stderr, "ERROR\n" );

       fflush( stdout );

       int rc = fork();
```

```
        if ( rc == 0 ) {
                printf( "AGAIN?\n" );
                fprintf( stderr, "ERROR ERROR\n" );
        }
        else {
                wait( NULL );
        }
        printf( "BYE\n" );
        fprintf( stderr, "HELLO\n" );
        return EXIT_SUCCESS;
}
```

3.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
int main() {
        int fd;
        close( 2 );
        close( 1 );
        printf( "HI\n" );
        fflush( stdout );
        fd = open( "newfile.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );
        printf( "==> %d\n", fd );
        printf( "WHAT?\n" );
        fprintf( stderr, "ERROR\n" );
        fflush( stdout );

        int rc = fork();

        if ( rc == 0 ) {

                printf( "AGAIN?\n" );

                fprintf( stderr, "ERROR ERROR\n" );

        }

        else {

                wait( NULL );
        }

         printf( "BYE\n" );

         fprintf( stderr, "HELLO\n" );

          return EXIT_SUCCESS;

}
```

4.

```
int main(void) {

        int stuff = 7;

        pid_t pid = fork();

         printf("Stuff is %d\n", stuff);

        if (pid == 0)

             stuff = 6;

}
```

5.

```
int main(void){

        int* stuff = malloc(sizeof(int)*1);

         *stuff = 7;

        pid_t pid = fork();

        printf("Stuff is %d\n", *stuff);

         if (pid == 0)
```

```
                        *stuff = 6

}
```

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

#define CHILDREN 8   /* based on a true story.... */
#define MAX_SUM 1000000

int sum;

/* function executed by each thread */
void * whattodo( void * arg );

int main() {
  pthread_t tid[CHILDREN];   /* keep track of the thread IDs */
  int i, rc;

   sum=0;
  /* create all the child threads */
  for ( i = 0 ; i < CHILDREN ; i++ )   {

    rc = pthread_create( &tid[i], NULL, whattodo, &i );
                           /*                      ^^^^
                                       all threads get the same
                                        statically allocated memory address */
}

  /* wait for child threads to terminate */
  for ( i = 0 ; i < CHILDREN ; i++ )
  {
    rc = pthread_join( tid[i], NULL );   /* BLOCKING CALL */


  }

  printf( "MAIN: All threads terminated\n" );

  return EXIT_SUCCESS;
}

void * whattodo( void * arg )
```

```
{
    int i;
       int t = *(int *)arg;
    unsigned int tid = (unsigned int)pthread_self();
    int my_sum = sum;
    for ( i = 0 ; i < MAX_SUM ; i++ ){
    my_sum =  my_sum +1;
    /* printf( "THREAD %u:    sum: %d \n", tid, sum); */
   }
    sum= sum + my_sum;
  printf( "THREAD %u:   My final sum: %d \n",
          tid,sum );

  return NULL;
}
```

Question2) Run the following code and answer the following question:

Notice that ***mynum*** is a global variable.

Child is incrementing the ascii value of ***mynum***

Parent is not

Child and Parent can take turns running

Why does child print CHILD0, CHILD1, CHILD2, etc whereas parent prints PARENT0, PARENT0, PARENT0, etc? Remember ***mynum*** is a global variable.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>

char mynum='0';
int main(void)
{
   int i;
   pid_t fork_return;
   static char buffer[10];
   fork_return = fork();
   if (fork_return == 0)
   {
      strcpy(buffer, "CHILD"); /*in the child process*/
      for (i=0; i<5; ++i) /*both processes do this*/
      {
         mynum=i + '0';
         sleep(1); /*5 times each*/
         write(1, buffer, strlen(buffer));
         write(1, &mynum, 1);
         write(1, "\n", 1);
      }
      return 0;
   }
   else if (fork_return > 0)
   {
      strcpy(buffer, "PARENT"); /*in the parent process*/
      for (i=0; i<5; ++i) /*both processes do this*/
      {
         sleep(1); /*5 times each*/
         write(1, buffer, strlen(buffer));
         write(1, &mynum, 1);
         write(1, "\n", 1);
      }
      return 0;
   }
   else
   {
      write (1, "Error\n", strlen("Error\n"));
   }
}
```

Question-3:

```
using System;
using System.Threading;

public class PipeClient
{
   private static int numClients = 4;
   static int[] code = { 4, 6, 3, 1, 5, 0, 2 };

   public static void Main(string[] Args)
{
  int i;
  Thread[] tid = new Thread[7];

  for (i = 0; i < 7; i++){
   tid[i] = new Thread(PipeClient.textthread);
   tid[i].Start(i);
  }
  return;
}

public static void textthread(object o1){
  int n= (int) o1;
  switch (n)
  {
   case 0:
     Console.WriteLine("A semaphore S is an integer-valued variable which can take only non-
negative\n");
     Console.WriteLine("values. Exactly two operations are defined on a semaphore:\n\n");
     break;

   case 1:
     Console.WriteLine("Signal(S): If there are processes that have been suspended on this
semaphore,\n");
     Console.WriteLine(" wake one of them, else S := S+1.\n\n");
     break;

   case 2:
     Console.WriteLine("Wait(S): If S>0 then S:=S-1, else suspend the execution of this process.\n");
     Console.WriteLine(" The process is said to be suspended on the semaphore S.\n\n");
     break;

   case 3:
     Console.WriteLine("The semaphore has the following properties:\n\n");
     break;

   case 4:
     Console.WriteLine("1. Signal(S) and Wait(S) are atomic instructions. In particular, no\n");
     Console.WriteLine(" instructions can be interleaved between the test that S>0 and the\n");
```

```
      Console.WriteLine(" decrement of S or the suspension of the calling process.\n\n");
      break;

    case 5:
      Console.WriteLine("2. A semaphore must be given an non-negative initial value.\n\n");
      break;

    case 6:
      Console.WriteLine("3. The Signal(S) operation must waken one of the suspended processes.
The\n");
      Console.WriteLine(" definition does not specify which process will be awakened.\n\n");
      break;
  }

}
}
```

There are a couple of problems with this program. First of all, the threads do not have a chance to complete, because the main program terminates without waiting for them. Second, the threads are not synchronized and therefore the text output is garbled.

Your task is to add semaphores to this program to synchronize the threads. You may declare the semaphores as static objects.

Question-4:

A barbershop consists of a waiting room with *n* chairs and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.

(1) Write a program to coordinate the barber and the customers.

```
using System;

using System.Threading;



namespace MultithreadingApplication {

class ThreadCreationProgram {

  static int customers1 = 0;

  static int  n =2;
static void barber() {
        while(true) {
                customers1 = customers1 - 1;
                cut_hair();
        }
}

 public static void customer() {
        if (customers1 < n) {
                customers1 = customers1 + 1;
                get_haircut();
        }
        else { //do nothing (leave) when all chairs are used.
                Console.WriteLine("do nothing (leave) when all chairs are used");

        }
 }
        public static void cut_hair(){
                Console.WriteLine("Barber is cutting hair");
                Thread.Sleep(500);



        }
        public static void get_haircut(){
                Console.WriteLine("get hair cut for some time");
                Thread.Sleep(500);
```

```
            }



            static void Main(string[] args) {

        Console.WriteLine("In Main: Creating the Child thread");

         Thread childThread1 = new Thread(barber);

         Thread childThread2 = new Thread(customer);

         Thread childThread3 = new Thread(customer);

         Thread childThread4 = new Thread(customer);

         Thread childThread5 = new Thread(customer);



         childThread1.Start();

         childThread2.Start();

         childThread3.Start();

         childThread4.Start();

         childThread5.Start();

         Console.ReadKey();

      }

   }

}
```

Question-5:

The Cigarette-Smokers Problem. Consider a system with three *smoker* processes and one *agent* process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobaccor, paper, and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats. Write a program to synchronize the agent and the smokers.

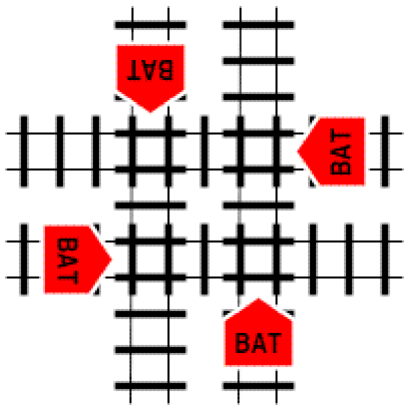
```
using System;
using System.Threading;



namespace MultithreadingApplication {

class ThreadCreationProgram {

  static Semaphore semlock = new Semaphore(1,1);

  public static void agent(){

              while(true){

                  semlock.WaitOne();

                  Random random = new Random();



                          int num = random.Next(1,3);  // Pick a random number from 1-3

                          if ( num == 1 ) {

                                      Console.WriteLine("Put tobacco on table");

                                      Console.WriteLine("Put paper on table");

                                      // Wake up smoker with match

                          } else if ( num == 2 ) {

                                      Console.WriteLine("Put tobacco on table");

                                      Console.WriteLine("Put paper on table");

                                      // Wake up smoker with paper

                          } else {

                                      Console.WriteLine("Put tobacco on table");

                                      Console.WriteLine("Put paper on table");

                                      // Wake up smoker with tobacco

                          }

                          semlock.Release();

                 // Agent sleeps

              }
```

```
 }
public static void Smoker1(){
            while(true){
                        Console.WriteLine("Smoker-1");
                        Console.WriteLine("Smoker-1: Pick up match");
                        Console.WriteLine("Smoker-1: Pick up paper");
                        Console.WriteLine("Smoker-1: (but don't inhale).");
                        Thread.Sleep(100);
                        // Wakeup Agent
       }
 }
public static void Smoker2(){
            while(true){
                        Console.WriteLine("Smoker-2");
                        Console.WriteLine("Smoker-2:Pick up tobacco");
                        Console.WriteLine("Smoker-2:Pick up paper");
                        Console.WriteLine("Smoker-2:(but don't inhale).");
                        Thread.Sleep(100);
                        // Wakeup Agent


       }
 }
public static void Smoker3(){
            while(true){
                        Console.WriteLine("Smoker-3");
                        Console.WriteLine("Smoker-3:Pick up match");
                        Console.WriteLine("Smoker-3:Pick up tobacco");
                        Console.WriteLine("Smoker-3: (but don't inhale).");

                        Thread.Sleep(100);

                        // Wakeup Agent

            }

  }



            static void Main(string[] args) {

      Console.WriteLine("In Main: Creating the Child thread");

         Thread agentthread = new Thread(agent);

         Thread Smoker1thread = new Thread(Smoker1);

         Thread Smoker2thread = new Thread(Smoker2);

         Thread Smoker3thread = new Thread(Smoker3);

         agentthread.Start();

         Smoker1thread.Start();

         Smoker2thread.Start();

         Smoker3thread.Start();

         Console.ReadKey();

      }

   }

}
```

The smoker immediately sleeps. When the agent puts the two items on the table, then the agent will wake up the appropriate smoker. The smoker will then grab the items, and wake the agent. While the smoker is smoking, the agent can place two items on the table, and wake a different smoker (if the items placed aren't the same). The agent sleeps immediately after placing the items out. This is something like the producer-consumer problem except the producer can only produce 1 item (although a choice of 3 kinds of items) at a time.

<u>Question-6: (BONUS)</u>

In a heroic effort to meet increasingly tighter shipping deadlines, a company has employed a *Bidirectional Autonomous Trolley* (BAT) system to move products from its warehouses to the delivery trucks. Each BAT is a mobile platform that travels on separate tracks back and forth between a warehouse and a truck. Because the goods are fragile, the tracks are perfectly leveled, which requires the placement of level crossings between warehouses. At most one BAT can cross at a time. Traffic at the crossing arriving from the right has the right of way. But this presents a problem, as the company soon found out when a simultaneous shipment of plastic penguins and black umbrellas caused the system to come to a grinding halt. Two BATs with the shipments and two other BATs returning to the warehouses were deadlocked at a level crossing:



In our system each BAT is controlled by a separate thread. It is your task to create BATMAN: a BAT Manager that prevents deadlock at a crossing. Part of the solution is to use one mutex lock for the crossing, so that at most one BAT at a time can pass. The mutex lock will act as a monitor for the BAT operations at the crossing. Besides the mutex lock, we will also need a set of condition variables to control the synchronization of the BATs.

We need a condition variable per BAT to queue BATs arriving from one direction (NorthQueue, EastQueue, SouthQueue, WestQueue). For example, when a BAT from North is already at the crossing, a second BAT from North will have to wait.

Another type of condition variable is needed to let BATs from the right have precedence to cross (NorthFirst, EastFirst, SouthFirst, WestFirst). However, using this rule can cause starvation. To prevent starvation, when a BAT is waiting to cross but BATs continuously arriving from the right have the right of way, we will let a BAT that just passed the crossing signal a waiting BAT on his left.

When deadlock occurs the BAT Manager must signal one of the BATs to proceed, e.g. the BAT from North. You will need a counter for each direction to keep track of the number of BATs waiting in line.

The program must take a string of 'n', 's', 'e', 'w' from the command line indicating a sequence of arrivals of BATs from the four directions. For example:

```
$ ./batman nsewwewn
BAT 4 from West arrives at crossing
BAT 2 from South arrives at crossing
BAT 1 from North arrives at crossing
BAT 3 from East arrives at crossing
DEADLOCK: BAT jam detected, signalling North to go
BAT 1 from North leaving crossing
BAT 3 from East leaving crossing
BAT 2 from South leaving crossing
BAT 4 from West leaving crossing
BAT 6 from East arrives at crossing
BAT 5 from West arrives at crossing
BAT 8 from North arrives at crossing
BAT 5 from West leaving crossing
BAT 6 from East leaving crossing
BAT 8 from North leaving crossing
BAT 7 from West arrives at crossing
BAT 7 from West leaving crossing
```

Note: the ordering of the above arrivals and departures may vary between runs and implementations. You don't need to produce exactly the same output.

To summarize:

- *BATs arriving from the same direction line up behind the first BAT already at the crossing;*
- *BATs arriving from the right always have the right of way (unless the waiting BAT receives a signal to go);*
- *Deadlock has to be prevented*
- *Starvation has to be prevented*

To implement the solution, analyze the actions of a BAT arriving from a particular direction under different circumstances, i.e. another BAT is already at the crossing, there is a BAT at the right, and/or there is a BAT at the left. Determine what to do when it arrives at the crossing and it must wait in line or wait for the BAT at his right, what to do when it passed the crossing, and what to do when it leaves the crossing.