Assignment

A. Write a program in which producer process creates a child process, child process takes input to generate two matrices and put in the shared memory and parent process reads the matrices in the shared memory and then calculates the product matrix using multiple threads and stores in the shared memory, a separate client process reads product matrix and prints on the console.

B. Write a program that creates three child processes and three pipes to communicate with each process. Each child process reads from different serial line and sends the characters read back to the parent process through a pipe. The parent process outputs all characters received on the console. A child terminates when two newline characters are received consecutively. The parent terminates after all three children have terminated. (hint: send-pipe and receive-pipe primitives can be used)

C. The reader writer problem: A number of readers may simultaneously be reading from a file. Only one writer at a time may write to file, and no reader can be reading while a writer is writing. Using semaphores, Write solution to the reader writers problem that gives priority to writers.

D. Modify the socket-based date server (Figure 3.21) in Chapter 3 so that the server services each client request in a separate thread.

E. Banker's Algorithm

For this project, you will write a multithreaded program that implements the banker's algorithm discussed in Section 7.5.3. Several customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied. This programming assignment combines three separate topics:
(1) multithreading, (2) preventing race conditions, and (3) deadlock avoidance.

The Banker

The banker will consider requests from *n* customers for *m* resources types. As outlined in Section 7.5.3. The banker will keep track of the resources using the following data structures:

```
/* these may be any values >= 0 */
#define NUMBER OF CUSTOMERS 5
#define NUMBER OF RESOURCES 3
/* the available amount of each resource */
int available[NUMBER OF RESOURCES];
/*the maximum demand of each customer */
int maximum[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];
/* the amount currently allocated to each customer */
int allocation[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];
```

/* the remaining need of each customer */
int need[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES];

The Customers

Create *n* customer threads that request and release resources from the bank. The customers will continually loop, requesting and then releasing random numbers of resources. The customers' requests for resources will be bounded by their respective values in the need array. The banker will grant a request if it satisfies the safety algorithm discussed in the class. If a request does not leave the system in a safe state, the banker will deny it. Function prototypes for requesting and releasing resources are as follows:

int request resources(int customer num, int request[]);
int release resources(int customer num, int release[]);

These two functions should return 0 if successful (the request has been granted) and –1 if unsuccessful. Multiple threads (customers) will concurrently access shared data through these two functions. Therefore, access must be controlled through mutex locks to prevent race conditions.

Implementation

You should invoke your program by passing the number of resources of each type on the command line. For example, if there were three resource types, with ten instances of the first type, five of the second type, and seven of the third type, you would invoke your program follows:

```
./a.out 10 5 7
```

The available array would be initialized to these values. You may initialize the maximum array (which holds the maximum demand of each customer) using any method you find convenient.