Sean Webster

Operating Systems
Homework 2
Due 10/13/2016

Objective: The objective of this assignment was to learn thread intercommunication by created 'buyer' and 'provider' threads that exchange a number between each other.

Background: Multithreading is one of the most important concepts in operating systems. By allowing programs to be broken into multiple threads, different functions can be processed nearly simultaneously, allowing for better utilization of current architectures.

Functions Used: Functions used were: pthread_create() which creates linux threads, pthread_join(), which waits for a thread to finish, then terminates it, sem_init() which initiates a semaphore, pthread_mutex_init() which initiates a mutex, sem_destroy() which destroys a semaphore, pthread_mutex_destroy() which destroys a mutex, pthread_mutex_lock() which locks a mutex, em_post() which increases a semaphore, pthread_mutex_unlock() which unlocks a mutex, sem_wait() which decrements a semaphore, and the functions in the Queue class included in the C++ library.

Results:

```
Buyer 108: REMOVED item 4 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 110: REMOVED item 4 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 155: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 74: REMOVED item 8 from QUEUE
Provider 3: INSERTED item 5 to QUEUE
Buyer 75: REMOVED item 5 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 76: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 77: REMOVED item 8 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 78: REMOVED item 4 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 79: REMOVED item 4 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 80: REMOVED item 4 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 81: REMOVED item 4 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 82: REMOVED item 4 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 83: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 84: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 85: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 86: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 87: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 88: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 89: REMOVED item 8 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 90: REMOVED item 8 from QUEUE
Provider 0: INSERTED item 4 to QUEUE
Buyer 91: REMOVED item 4 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 92: REMOVED item 8 from QUEUE
Provider 1: INSERTED item 7 to QUEUE
Buyer 73: REMOVED item 7 from QUEUE
Provider 2: INSERTED item 8 to QUEUE
Buyer 247: REMOVED item 8 from QUEUE
[swebster@anaconda29 HW2]$
```

Figure 1: Part 1 Output

```
[swebster@anaconda29 HW2]$ ./HW2_prog2 4
Creating Provider Thread: 0 with item 5
Provider 0: INSERTED item 5 to QUEUE
Buyer 5: REMOVED item 5 from QUEUE
Provider 0: INSERTED item 5 to QUEUE
Buyer 4: REMOVED item 5 from QUEUE
Provider 0: INSERTED item 5 to QUEUE
Buyer 3: REMOVED item 5 from QUEUE
Provider 0: INSERTED item 5 to QUEUE
Buyer 2: REMOVED item 5 from QUEUE
Provider 0: INSERTED item 5 to QUEUE
Buyer 1: REMOVED item 5 from QUEUE
Provider 0: INSERTED item 5 to QUEUE
Buyer 0: REMOVED item 5 from QUEUE
[swebster@anaconda29 HW2]$
```

Figure 2: Part 2 Output

Conclusions and Observations: Thread intercommunication seems like a very useful tool. By allowing different parts of the program to work simultaneously, streamlining programs would be easy. However, in my code, I found that at the end of the program, sometimes a provider puts in another item, but is terminated because there are no buyers left to buy the item. I also found the wording in the instruction sheet very confusing, and spent a bit of time trying to understand what it was asking. Thread functions are easy to understand after looking up their documentation, as well.

The change from part 1 to 2 to make the program more efficient was to remove semaphores. Semaphores are only needed when multiple threads are accessing multiple locations, which wasn't the

case with part 2.

Readme:

Sean Webster

Homework 1 readme

Source Code Files:
HW2_prog1.cpp
HW2_prog2.cpp

Executables:
HW2_prog1
HW2_prog2

Instructions for running:
Run programs with intended number of buyer threads in arg section

Inputs:
./HW2_prog1 258
./HW2_prog2 4

Part 1:

```
/*****************************************************
 * Sean Webster
 * Operating Systems
 * Homework 2
 * Due 10/13
 *
 *
 * ************************************************
 * HW2_prog1.cpp
     * Creates N buyer threads and 4 provider threads, where
     * buyer threads 'take' the number out of a queue that
     * is filled by provider threads
     *
     * Code from the examples given in class was used as a
     * template for this assignment
     * ************************************************/

    #include<stdio.h>
    #include<stdlib.h>
    #include<unistd.h>
    #include<pthread.h>
    #include<semaphore.h>
    #include<signal.h>
    #include<time.h>
    #include<queue>
    #include <stdlib.h>


    int BUYER_NUM;
    #define PROVIDER_NUM 4


    int index_counter = 0;

    // global variable, all threads can acess

    void *thread_Insert(void *arg);    // function for sending
    void *thread_Remove(void *arg);    // function for receiving

    sem_t bin_sem;          // semaphore
    pthread_mutex_t mutx;   // mutex

    std::queue<int> theQueue;    // Queue
    //int theItem = 0;

    int counter = 0;


    int main(int argc, char **argv)
    {
      //for(int m = 0; m < argc; m++)
         // {
        /* initialize random seed: */
        srand (time(NULL));

        BUYER_NUM = strtol(argv[1], NULL, 10) + 2;
        const int BUY_NUM = BUYER_NUM;
```

```
58.         pthread_t pID[PROVIDER_NUM];
59.         pthread_t bID[BUY_NUM];
60.         void *thread_result;
61.         int state1, state2;
62.         state1 = pthread_mutex_init(&mutx, NULL);
63.         state2 = sem_init(&bin_sem, 0 ,0);
64.         //mutex initialization
65.         //semaphore initialization, first value = 0
66.
67.         if(state1||state2!=0)
68.           puts("Error mutex & semaphore initialization!!!");
69.
70.
71.
72.         // Create provider threads
73.         for(long int j = 0; j < PROVIDER_NUM; j++)
74.         {
75.           pthread_create(&pID[j], NULL, thread_Insert, (void*)j);
76.
77.         }
78.         sleep(1);
79.
80.         // Create buyer threads
81.         for(long int i = 0; i < BUYER_NUM; i++)
82.         {
83.           pthread_create(&bID[i], NULL, thread_Remove, (void*)i);
84.         }
85.
86.         // Waiting buyer threads to terminate
87.         for(int k = 0; k < BUYER_NUM; k++)
88.         {
89.           //printf("Buyers executed: %d\n", k + 1);
90.           pthread_join(bID[k], &thread_result);
91.         }
92.         // Waiting buyer threads to terminate
93.         for(int l = 0; l < PROVIDER_NUM; l++)
94.         {
95.           //printf("PROVIDER executed: %d\n", l + 1);
96.           pthread_join(pID[l], &thread_result);
97.         }
98.
99.         sem_destroy(&bin_sem);    // destroy semaphore
100.        pthread_mutex_destroy(&mutx);  // destroy mutex
101.        //printf("Final Index: %d\n", index_counter);
102.
103.         // }
104.        return 0;
105.      }
106.
107.
108.      // Provider inserts item into queue
109.      void *thread_Insert(void *arg)
110.      {
111.        int theGoods = rand() % 10 + 1;
112.        printf("Creating Provider Thread: %d with item %d\n", (int*)arg,
  theGoods);
113.
114.        //for(i = 0;i < PROVIDER_NUM; i++)
```

```
115.         for(;counter < BUYER_NUM -1;)
116.         {
117.           pthread_mutex_lock(&mutx);
118.
119.           if(theQueue.empty() && !index_counter)
120.           //if(!theItem)
121.           {
122.           sleep(.5);
123.           index_counter++;
124.           //printf("index added - index: %d - counter: %d\n", index_counter,
  counter);
125.           theQueue.push(theGoods);
126.           //theItem = theGoods;
127.           printf("Provider %d: INSERTED item %d to QUEUE\n", (int*)arg,
  theGoods);
128.           sem_post(&bin_sem);     // semaphore to increase
129.           }
130.           else
131.           {
132.           sleep(.5);
133.           }
134.           pthread_mutex_unlock(&mutx);
135.         }
136.       }
137.
138.       // Thread decreases items
139.       void *thread_Remove(void *arg)
140.       {
141.         bool isDone = 0;
142.
143.         //printf("Creating Buyer Thread: %d\n", (int*)arg);
144.
145.         for(;isDone < 1;)
146.         {
147.           if(theQueue.empty())
148.           sleep(.5);
149.           else
150.           {
151.           sem_wait(&bin_sem);     //decrease index_counter
152.           pthread_mutex_lock(&mutx);
153.           sleep(.5);
154.           counter++;
155.           printf("Buyer %d: REMOVED item %d from QUEUE\n", (int*)arg,
  theQueue.front());
156.           //printf("threads executed: %d \n", counter);
157.           theQueue.pop();
158.           //theItem = 0;
159.           //printf("index subbed - index: %d - counter: %d\n", index_counter,
  counter);
160.           index_counter--;
161.
162.           pthread_mutex_unlock(&mutx);
163.           isDone = 1;
164.           }
165.       }
166.     }
```

Part 2:

```
/**************************************************
 * Sean Webster
 * Operating Systems
 * Homework 2
 * Due 10/13
 *
 *
 * ************************************************
 * HW2_prog2.cpp
     * Creates N buyer threads and 4 provider threads, where
     * buyer threads 'take' the number out of a queue that
     * is filled by provider threads
     *
     * Code from the examples given in class was used as a
     * template for this assignmen
     * ************************************************/

    #include<stdio.h>
    #include<stdlib.h>
    #include<unistd.h>
    #include<pthread.h>
    #include<semaphore.h>
    #include<signal.h>
    #include<time.h>
    #include<queue>
    #include <stdlib.h>


    int BUYER_NUM;
    #define PROVIDER_NUM 1


    int index_counter = 0;

    // global variable, all threads can acess

    void *thread_Insert(void *arg);    // function for sending
    void *thread_Remove(void *arg);    // function for receiving

    sem_t bin_sem;          // semaphore
    pthread_mutex_t mutx;   // mutex

    std::queue<int> theQueue;    // Queue
    //int theItem = 0;

    int counter = 0;


    int main(int argc, char **argv)
    {
      //for(int m = 0; m < argc; m++)
         // {
        /* initialize random seed: */
        srand (time(NULL));

```

```
56.         BUYER_NUM = strtol(argv[1], NULL, 10) + 2;
57.         const int BUY_NUM = BUYER_NUM;
58.         pthread_t pID[PROVIDER_NUM];
59.         pthread_t bID[BUY_NUM];
60.         void *thread_result;
61.         int state1, state2;
62.         state1 = pthread_mutex_init(&mutx, NULL);
63.         //state2 = sem_init(&bin_sem, 0 ,0);
64.         //mutex initialization
65.         //semaphore initialization, first value = 0
66.
67.         if(state1=0)
68.           puts("Error mutex & semaphore initialization!!!");
69.
70.
71.
72.         // Create provider threads
73.         for(long int j = 0; j < PROVIDER_NUM; j++)
74.         {
75.           pthread_create(&pID[j], NULL, thread_Insert, (void*)j);
76.
77.         }
78.         sleep(1);
79.
80.         // Create buyer threads
81.         for(long int i = 0; i < BUYER_NUM; i++)
82.         {
83.           pthread_create(&bID[i], NULL, thread_Remove, (void*)i);
84.         }
85.
86.         // Waiting buyer threads to terminate
87.         for(int k = 0; k < BUYER_NUM; k++)
88.         {
89.           //printf("Buyers executed: %d\n", k + 1);
90.           pthread_join(bID[k], &thread_result);
91.         }
92.         // Waiting buyer threads to terminate
93.         for(int l = 0; l < PROVIDER_NUM; l++)
94.         {
95.           //printf("PROVIDER executed: %d\n", l + 1);
96.           pthread_join(pID[l], &thread_result);
97.         }
98.
99.         //sem_destroy(&bin_sem);  // destroy semaphore
100.        pthread_mutex_destroy(&mutx);  // destroy mutex
101.        //printf("Final Index: %d\n", index_counter);
102.
103.         // }
104.        return 0;
105.      }
106.
107.
108.      // Provider inserts item into queue
109.      void *thread_Insert(void *arg)
110.      {
111.        int theGoods = rand() % 10 + 1;
112.        printf("Creating Provider Thread: %d with item %d\n", (int*)arg,
  theGoods);
```

```
113.
114.         //for(i = 0;i < PROVIDER_NUM; i++)
115.         for(;counter < BUYER_NUM -1;)
116.         {
117.           pthread_mutex_lock(&mutx);
118.
119.           if(theQueue.empty() && !index_counter)
120.           //if(!theItem)
121.           {
122.           sleep(.5);
123.           index_counter++;
124.           //printf("index added - index: %d - counter: %d\n", index_counter,
  counter);
125.           theQueue.push(theGoods);
126.           //theItem = theGoods;
127.           printf("Provider %d: INSERTED item %d to QUEUE\n", (int*)arg,
  theGoods);
128.           //sem_post(&bin_sem);   // semaphore to increase
129.           }
130.           else
131.           {
132.           sleep(.5);
133.           }
134.           pthread_mutex_unlock(&mutx);
135.         }
136.       }
137.
138.       // Thread decreases items
139.       void *thread_Remove(void *arg)
140.       {
141.         bool isDone = 0;
142.
143.         //printf("Creating Buyer Thread: %d\n", (int*)arg);
144.
145.         for(;isDone < 1;)
146.         {
147.           if(theQueue.empty())
148.           sleep(.5);
149.           else
150.           {
151.           //sem_wait(&bin_sem);   //decrease index_counter
152.           pthread_mutex_lock(&mutx);
153.           sleep(.5);
154.           counter++;
155.           printf("Buyer %d: REMOVED item %d from QUEUE\n", (int*)arg,
  theQueue.front());
156.           //printf("threads executed: %d \n", counter);
157.           theQueue.pop();
158.           //theItem = 0;
159.           //printf("index subbed - index: %d - counter: %d\n", index_counter,
  counter);
160.           index_counter--;
161.
162.           pthread_mutex_unlock(&mutx);
163.           isDone = 1;
164.           }
165.       }
166.     }
```