

6CS005

High Performance Computing

Week 4 Workshop

Revision on Multithreading

Student Number: 2408869

Name: Narayan Lohani

Group: L6CG15

Table of Contents

Explanation	2
Output of Program 1	4
Output of Program 2	4
Source Code of Program 1	5
Source Code of Program 2	6

Table of Figures

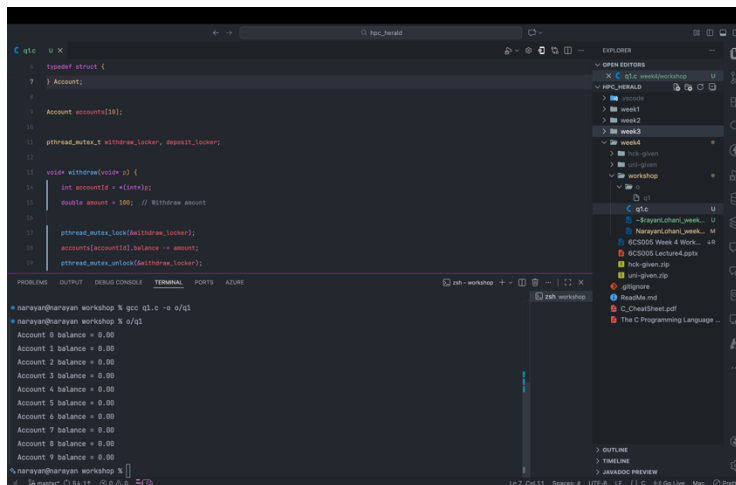
Figure 1: Output of Program 1	4
Figure 2: Output of Program 2	4
Figure 3: Source Code of Program 1: Thread Routine	5
Figure 4: Source Code of Program 1: Main function	5
Figure 5: Source Code of Program 2: Main Function	6
Figure 6: Source Code of Program 2: Thread Routine	6

Explanation

1. In a banking system, multiple users can access and modify their accounts concurrently. Each user has a balance, and they can deposit or withdraw money. Modify the code below to use a mutex to prevent race conditions during balance updates and ensure that multiple users can't simultaneously update the balance of the same account.

When multiple users try to make changes to their money either by withdrawing or depositing on the same account, critical section is formed and race condition may arise. To prevent this, whenever any thread try to make any change to the account balance, first we lock the thread using mutex, perform our desired action (withdraw or deposit), and unlock the thread. This way ensures that no balance is lost at any point.

To accomplish this task, we created two global mutex variables namely withdraw_locker and deposit_locker of type pthread_mutex_t. In the main function, we initialize both the mutexes. Then inside thread routine, we typecast and get our account number, initialize the balance to withdraw/deposit and lock the thread using corresponding locker: withdraw_locker for withdraw routine and deposit_locker for deposit routine. Then we perform the balance modification and unlock the mutex.



```
1  typedef struct {
2  } Account;
3
4  Account accounts[10];
5
6  pthread_mutex_t withdraw_locker, deposit_locker;
7
8  void withdraw(void *p) {
9      int accountId = *(int*)p;
10     double amount = 100; // Withdraw amount
11
12     pthread_mutex_lock(&withdraw_locker);
13     accounts[accountId].balance -= amount;
14     pthread_mutex_unlock(&withdraw_locker);
15 }
16
17 int main() {
18     // Initialize mutexes
19     pthread_mutex_init(&withdraw_locker, NULL);
20     pthread_mutex_init(&deposit_locker, NULL);
21
22     // Create 10 threads
23     for (int i = 0; i < 10; i++) {
24         pthread_t t;
25         pthread_create(&t, NULL, withdraw, (void*)&i);
26     }
27
28     // Wait for all threads to complete
29     for (int i = 0; i < 10; i++) {
30         pthread_join(t, NULL);
31     }
32
33     // Print final balances
34     for (int i = 0; i < 10; i++) {
35         printf("Account %d balance = %.2f\n", i, accounts[i].balance);
36     }
37
38     return 0;
39 }
```

```
narayan@narayan workshop % gcc q1.c -o a1/q1
narayan@narayan workshop % ./a1/q1
Account 0 balance = 0.00
Account 1 balance = 0.00
Account 2 balance = 0.00
Account 3 balance = 0.00
Account 4 balance = 0.00
Account 5 balance = 0.00
Account 6 balance = 0.00
Account 7 balance = 0.00
Account 8 balance = 0.00
Account 9 balance = 0.00
narayan@narayan workshop %
```

Source Code of Program 1

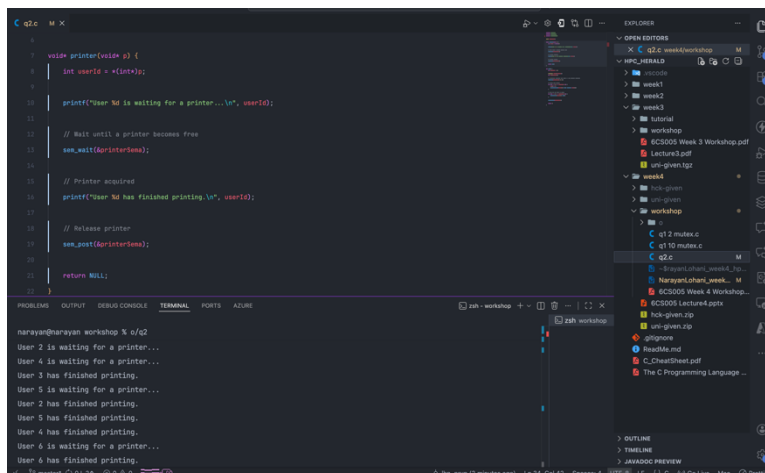
Output of Program 1

2. A printer is shared among multiple users. Each user can either print a document or wait if the printer is in use. There are only 2 printers in the office. Use semaphores to manage the printer access, ensuring that no more than 2 users can print at the same time.

When many users try to use the office printers at the same time, a critical section is formed because only a limited number of printers are available. If all users try to print together, the system needs a way to make some users wait until a printer becomes free. To handle this, we use a semaphore that keeps track of how many printers are available. Since there are two printers, the semaphore is initialized with the value 2.

Whenever a thread wants to print a document, it first performs a `sem_wait`, which decreases the semaphore value if a printer is free. If both printers are busy, the thread automatically waits. After finishing the printing task, the thread performs `sem_post`, which increases the semaphore value and lets another waiting user print.

This approach ensures that at most two users can print at the same time, preventing printer conflicts while still allowing multiple users to work efficiently.



```
1 // Source code for Program 2: Managing printer access using semaphores.
2 // There are 2 printers available.
3 // Users are represented by threads.
4 // The program ensures that no more than 2 users can print at the same time.
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <unistd.h>
9 #include <semaphore.h>
10
11 // Semaphore to manage printer access (initialized to 2)
12 static sem_t sem;
13
14 // Function to simulate a user printing a document
15 void print_document(int user_id) {
16     // Wait until a printer becomes free
17     sem_wait(&sem);
18
19     // Printer acquired
20     printf("User %d is waiting for a printer...\n", user_id);
21
22     // Simulate printing time (sleep for 2 seconds)
23     sleep(2);
24
25     // Release printer
26     sem_post(&sem);
27
28     printf("User %d has finished printing.\n", user_id);
29 }
30
31 // Main function
32 int main() {
33     // Initialize semaphore to 2
34     sem_init(&sem, 0, 2);
35
36     // Create 6 threads (users)
37     pthread_t threads[6];
38     for (int i = 0; i < 6; i++) {
39         pthread_create(&threads[i], NULL, print_document, (void *) (i + 1));
40     }
41
42     // Wait for all threads to finish
43     for (int i = 0; i < 6; i++) {
44         pthread_join(threads[i], NULL);
45     }
46
47     return 0;
48 }
```

Terminal Output:

```
narayan@narayan:~/workspace % ./a.out
User 1 is waiting for a printer...
User 2 is waiting for a printer...
User 3 is waiting for a printer...
User 4 is waiting for a printer...
User 5 is waiting for a printer...
User 6 is waiting for a printer...
User 1 has finished printing.
User 2 has finished printing.
User 3 has finished printing.
User 4 has finished printing.
User 5 has finished printing.
User 6 has finished printing.
```

Source Code of Program 2

Output of Program 2

Output of Program 1

```
[narayan@narayan workshop % gcc q1.c -o o/q1
[narayan@narayan workshop % o/q1
Account 0 balance = 0.00
Account 1 balance = 0.00
Account 2 balance = 0.00
Account 3 balance = 0.00
Account 4 balance = 0.00
Account 5 balance = 0.00
Account 6 balance = 0.00
Account 7 balance = 0.00
Account 8 balance = 0.00
Account 9 balance = 0.00
narayan@narayan workshop % >
```

Figure 1: Output of Program 1

Output of Program 2

```
[narayan@narayan workshop % gcc q2.c -o o/q2
q2.c:32:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
  32 |     sem_init(&printerSema, 0, 2);
      |     ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/semaphore.h:55:1: note: declared here
  55 | int sem_init(sem_t *, int, unsigned int) __deprecated;
      | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/semaphore.h:223:1: note: __attribute__((deprecated))
  223 | #define __deprecated __attribute__((deprecated))
      | 
q2.c:45:5: warning: 'sem_destroy' is deprecated [-Wdeprecated-declarations]
  45 |     sem_destroy(&printerSema);
      |     ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/semaphore.h:53:1: note: declared here
  53 | int sem_destroy(sem_t *) __deprecated;
      | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/semaphore.h:223:1: note: __attribute__((deprecated))
  223 | #define __deprecated __attribute__((deprecated))
      | 
2 warnings generated.
[narayan@narayan workshop % o/q2
User 0 is waiting for a printer...
User 0 has finished printing.
User 1 is waiting for a printer...
User 1 has finished printing.
User 3 is waiting for a printer...
User 2 is waiting for a printer...
User 2 has finished printing.
User 4 is waiting for a printer...
User 4 has finished printing.
User 3 has finished printing.
User 5 is waiting for a printer...
User 5 has finished printing.
User 6 is waiting for a printer...
User 6 has finished printing.
User 7 is waiting for a printer...
User 7 has finished printing.
User 8 is waiting for a printer...
User 8 has finished printing.
User 9 is waiting for a printer...
User 9 has finished printing.
narayan@narayan workshop %
```

Figure 2: Output of Program 2

Source Code of Program 1

```
hpc_herald - q1.c

1  #include <stdio.h>
2  #include <pthread.h>
3
4  typedef struct {
5      int accountNumber;
6      double balance;
7  } Account;
8
9  Account accounts[10];
10
11 pthread_mutex_t withdraw_locker, deposit_locker;
12
13 void* withdraw(void* p) {
14     int accountId = *(int*)p;
15     double amount = 100; // Withdraw amount
16
17     pthread_mutex_lock(&withdraw_locker);
18     accounts[accountId].balance -= amount;
19     pthread_mutex_unlock(&withdraw_locker);
20
21     return NULL;
22 }
23
24 void* deposit(void* p) {
25     int accountId = *(int*)p;
26     double amount = 100; // Deposit amount
27
28     pthread_mutex_lock(&withdraw_locker);
29     accounts[accountId].balance += amount;
30     pthread_mutex_unlock(&withdraw_locker);
31
32     return NULL;
33 }
34
```

Figure 3: Source Code of Program 1: Thread Routine

```
hpc_herald - q1.c

35 int main() {
36     pthread_t threads[20];
37     int ids[10];
38
39     pthread_mutex_init(&withdraw_locker, NULL);
40     pthread_mutex_init(&deposit_locker, NULL);
41     // Create multiple threads to simulate transactions on the same account
42     for (int i = 0; i < 10; i++) {
43         ids[i] = i;
44         pthread_create(&threads[i], NULL, withdraw, &ids[i]);
45         pthread_create(&threads[i + 10], NULL, deposit, &ids[i]);
46     }
47     for (int i = 0; i < 20; i++) {
48         pthread_join(threads[i], NULL);
49     }
50     // Print final balance
51     for (int i = 0; i < 10; i++) {
52         printf("Account %d balance = %.2f\n", i, accounts[i].balance);
53     }
54
55     pthread_mutex_destroy(&withdraw_locker);
56     pthread_mutex_destroy(&deposit_locker);
57
58     return 0;
59 }
```

Figure 4: Source Code of Program 1: Main function

Source Code of Program 2

```
hpc_herald - q2.c

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4
5  sem_t printerSema;
6
7  void* printer(void* p) {
8      int userId = *(int*)p;
9
10     printf("User %d is waiting for a printer...\n", userId);
11
12     // Wait until a printer becomes free
13     sem_wait(&printerSema);
14
15     // Printer acquired
16     printf("User %d has finished printing.\n", userId);
17
18     // Release printer
19     sem_post(&printerSema);
20
21     return NULL;
22 }
23
```

Figure 6: Source Code of Program 2: Thread Routine

```
hpc_herald - q2.c

24 int main() {
25     int num_users = 10;
26
27     pthread_t users[num_users];
28     int ids[num_users];
29
30     // Initialize semaphore with value 2 : two printers available
31     sem_init(&printerSema, 0, 2);
32
33     // Create user threads
34     for (int i = 0; i < num_users; i++) {
35         ids[i] = i;
36         pthread_create(&users[i], NULL, printer, &ids[i]);
37     }
38
39     // Wait for all users to finish
40     for (int i = 0; i < num_users; i++) {
41         pthread_join(users[i], NULL);
42     }
43
44     sem_destroy(&printerSema);
45
46     return 0;
47 }
48
```

Figure 5: Source Code of Program 2: Main Function