Lab 6 Report

Tasks:

Write two programs to synchronize the agent and smoker processes: one using semaphores and another using pthread libraries.

Cigarette Smoker's Problem:

There are four processes in this problem: three smoker processes and an agent process. Each of the smoker processes will make a cigarette and smoke it. To make a cigarette requires tobacco, paper, and matches. Each smoker process has one of the three items. I.e., one process has tobacco, another has paper, and a third has matches. The agent has an infinite supply of all three. The agent places two of the three items on the table, and the smoker that has the third item makes the cigarette. Synchronize the processes.

Solution:

Semaphore Implementation

My semaphore implementation can be viewed in the 'Smokers_sem.c' file. It begins by creating semaphores that will be used to give access to a specific process. Since there are four processes (agent, tobacco smoker, paper smoker, and match smoker) with the addition of the lock semaphore, there will be five semaphores in total. After creating the semaphores (**lines 27-31**), I initialized the semaphores in **lines 35-39** where all semaphores are initialized to 0, except for the lock semaphore, as stated in the problem solution.

```
20
21  int main() {
22    // Setting up variables
23    int pid, status;
24    int counter = 0; // remove this to be more truthful to the problem
25
26    // Setting up the semaphors
27    int lock_sem = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);
28    int agent_sem = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);
29    int tobacco_sem = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);
30    int paper_sem = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);
31    int matches_sem = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT);
32
33    // Initializing semaphors
34    // All semaphores except for lock_sem are initialized to 0. "lock is initialized to 1"
35    sem_create(lock_sem, 1);
36    sem_create(agent_sem, 0);
37    sem_create(tobacco_sem, 0);
38    sem_create(paper_sem, 0);
39    sem_create(matches_sem, 0);
40
```

Then, I created each individual child process. The first process I created was the agent process (**lines 41-70**) by following the same procedure as the solution's pseudo code. However, for testing purposes I set a limit of 10 for the loop. I added comments to easily fix this and make the code closer to the actual problem solution. The way this function works is by first choosing a random number from 1 to 3. This number will be used to select one of the three smoker processes. When a smoker process is chosen, the code will print out that the agent places the missing materials into the table. Then it will call a V operator to give access to the chosen smoker process.

My Code:

```
40
41  // Create child processes that will do the updates
42  if ((pid = fork()) == -1) {
43    //fork failed!
44    perror("fork");
45    exit(1);
46  }
47
48  //First Child Process. The agent smoker process
49  if (pid == 0) {
50    while (counter < 10) { // while (1) { // replace with this to be more truthful to the problem
51      P(lock_sem); // lock sem sleeps
52      int randNum = (rand() % 3) + 1; // Pick a random number from 1-3
53      if (randNum == 1) { // if random number = 1, put tobacco and paper on table
54        printf("\nAgent places tobacco on the table\n");
55        printf("Agent places paper on the table.\n");
56        V(matches_sem); // Wake up smoker with match
57      } else if (randNum == 2) {
58        printf("\nAgent places tobacco on the table\n");
59        printf("Agent places match on the table.\n");
60        V(paper_sem); // Wake up smoker with paper
61      } else {
62        printf("\nAgent places match on the table.\n");
63        printf("Agent places paper on the table.\n");
64        V(tobacco_sem); // Wake up smoker with tobacco
65      }
66      V(lock_sem); // lock sem wakes up
67      P(agent_sem); // Agent sleeps
68      counter += 1;
69    }
70  } else {
71
```

After the agent process was created, I moved on to the smoker processes: the tobacco smoker (**lines 72-89**), the paper smoker process (**lines 91-108**), and the match smoker process (**lines 110-127**). Each of the processes were created similar to the solution pseudo code. The way that these work is by first calling the P operator to set the respective smoker process and the lock process to sleep. Then it states what happens in this step. Finally, the process will wake the agent process and the lock process up, and the process continues.

My Code:

```
72      // Parent Process. Fork off another child process.
73      if ((pid = fork()) == -1) {
74        //fork failed!
75        perror("fork");
76        exit(1);
77      }
78
79      if (pid == 0) { // Second child process. Tobacco smoker
80        while(1) {
81          P(tobacco_sem);  // tobacco sem sleeps right away
82          P(lock_sem); // lock sem sleeps
83          printf("Tobacco smoker picks up matches from the table.\n");
84          printf("Tobacco smoker picks up paper from the table.\n");
85          V(agent_sem); // wake the agent
86          V(lock_sem); // wake the lock sem
87          printf("Tobacco smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
88        }
89      } else {
90
```

```
91          // Parent Process. Fork off another child process.
92          if ((pid = fork()) == -1) {
93            //fork failed!
94            perror("fork");
95            exit(1);
96          }
97
98          if (pid == 0) { // Third child process. Paper smoker
99            while(1) {
100             P(paper_sem);  // Sleep right away
101             P(lock_sem); // lock sem sleeps
102             printf("Paper smoker picks up tobacco from the table.\n");
103             printf("Paper smoker picks up matches from the table.\n");
104             V(agent_sem); // wake the agent
105             V(lock_sem); // wake the lock sem
106             printf("Paper smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
107           }
108         } else {
109
```

```
110         // Parent Process. Fork off another child process.
111         if ((pid = fork()) == -1) {
112           //fork failed!
113           perror("fork");
114           exit(1);
115         }
116
117         if (pid == 0) { // Fourth child porcess. Matches smoker
118           while(1){
119             P(matches_sem);  // Sleep right away
120             P(lock_sem); // lock sem sleeps
121             printf("Matches smoker picks up tobacco from the table.\n");
122             printf("Matches smoker picks up paper from the table.\n");
123             V(agent_sem); // wake the agent
124             V(lock_sem); // wake the lock sem
125             printf("Matches smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
126           }
127         } else {
```

Finally, just like in the previous lab, the process has to end (**lines 129-132**)

```
127             } else {
128
129               // Process ends
130               pid = wait(&status);
131               printf("\nProcess with pid = %d exited with the status %d. \n", pid, status);
132             }
133           }
134         }
135       }
136     }
```

Pthread Implementation

My pthread implementation can be viewed in the 'Smokers_pthread.c' file. It begins by defining the agent function (containing the agent's code procedure) and smokers function (containing the three smoker code procedures). Afterwards, I defined the pthreads values to be used and the mutex values (**lines 22-34**). Since there are four processes, there are four pthreads who each get a mutex value. With the addition of the lock mutex value it means there will be five mutex values overall.

```
14  #include <stdio.h>
15  #include <stdlib.h>
16  #include <pthread.h>
17  #include <unistd.h>
18  #include <time.h>
19  #include <sys/wait.h>
20  #include <sys/types.h>
21
22  // Defining processes
23  void *agent(); // define agent function
24  void *smokers(void *arg); // define smoker function
25
26  // Defnining pthread values
27  pthread_t agent_pthread, tobacco_pthread, paper_pthread, match_pthread;
28
29  // Defining the mutex values
30  pthread_mutex_t lock_mutex = PTHREAD_MUTEX_INITIALIZER;
31  pthread_mutex_t agent_mutex = PTHREAD_MUTEX_INITIALIZER;
32  pthread_mutex_t tobacco_mutex = PTHREAD_MUTEX_INITIALIZER;
33  pthread_mutex_t paper_mutex = PTHREAD_MUTEX_INITIALIZER;
34  pthread_mutex_t match_mutex = PTHREAD_MUTEX_INITIALIZER;
35
```

Within the main function (**lines 36-61**), the code first locks and unlocks the respective mutex values. Then, it creates the pthreads and their connection to their respective function. Since all three smoker processes are within the smokers function, I added an ID parameter that indicates which smoker process was randomly chosen. Then, the agent function is joined and once its process is finished the mutex values are destroyed.

```
36  int main(){
37    srand(time(0)); // ensure randomization
38
39    // locking and unlocking the mutex objects
40    pthread_mutex_unlock(&lock_mutex);
41    pthread_mutex_lock(&agent_mutex);
42    pthread_mutex_lock(&tobacco_mutex);
43    pthread_mutex_lock(&paper_mutex);
44    pthread_mutex_lock(&match_mutex);
45
46    // creating new threads for each process
47    pthread_create(&agent_pthread, NULL, agent, NULL);
48    pthread_create(&match_pthread, NULL, smokers, (void *)1); // match pthread id = 1
49    pthread_create(&paper_pthread, NULL, smokers, (void *)2); // paper pthread id = 2
50    pthread_create(&tobacco_pthread, NULL, smokers, (void *)3); // tobacco pthread id = 3
51
52    pthread_join(agent_pthread, NULL);
53
54    // destroy mutex values after code is finished
55    pthread_mutex_destroy(&agent_mutex);
56    pthread_mutex_destroy(&tobacco_mutex);
57    pthread_mutex_destroy(&paper_mutex);
58    pthread_mutex_destroy(&match_mutex);
59    pthread_mutex_destroy(&lock_mutex);
60    return 0;
61  }
62
```

For the agent function I began by making a counter variable that serves as a limit for the code to not run infinitely. However, to make the code more accurate to the actual problem, I have added comments to make a quick switch and work as intended. Afterwards, I followed the same approach as the solution pseudo code. The way the agent function works is that it begins by getting a random number from 1 to 3. This number will be used as the smoker ID for the smoker procedure to be done. The agent function will then display the materials it is providing and the corresponding procedure will run. The only difference being using mutex lock and unlock commands instead of the P and V functions in the semaphore implementation. After the while loop is finished, the lock mutex is locked, the smoker processes are canceled, and the lock mutex is finally unlocked to return to the main function.

Solution Pseudo Code:

```
do forever {
  P( lock );
  randNum = rand( 1, 3 ); // Pick a random number from 1-3
  if ( randNum == 1 ) {
     // Put tobacco on table
     // Put paper on table
     V( smoker_match );  // Wake up smoker with match
   } else if ( randNum == 2 ) {
     // Put tobacco on table
     // Put match on table
     V( smoker_paper );  // Wake up smoker with paper
   } else {
     // Put match on table
     // Put paper on table
     V( smoker_tobacco ); } // Wake up smoker with tobacco
   V( lock );
   P( agent );  //  Agent sleeps
  }  // end forever loop
```

My Code:

```c
// Function to define agent process
void *agent(){
  int counter = 0; // remove this to be more truthful to the problem

  while (counter < 10){ // while (1) { // replace with this to be more truthful to the problem
    pthread_mutex_lock(&lock_mutex); // lock the lock_pthread
    int randNum = (rand() % 3) + 1; // Pick a random number from 1-3

    // Check which process was randomly chosen
    if (randNum == 1) { // Matches smoker was chosen
      printf("\nAgent places tobacco on the table\n");
      printf("Agent places paper on the table.\n");
      pthread_mutex_unlock(&match_mutex); // Wake up smoker with match
    } else if (randNum == 2) { // Paper smoker was chosen
      printf("\nAgent places tobacco on the table\n");
      printf("Agent places match on the table.\n");
      pthread_mutex_unlock(&paper_mutex); // Wake up smoker with paper
    } else { // Tobacco smoker was chosen
      printf("\nAgent places match on the table\n");
      printf("Agent places paper on the table.\n");
      pthread_mutex_unlock(&tobacco_mutex); // Wake up smoker with tobacco
    }

    pthread_mutex_unlock(&lock_mutex);
    pthread_mutex_lock(&agent_mutex); // Agent sleeps
    counter += 1;
  }

  pthread_mutex_lock(&lock_mutex);

  // sends a cancellation request to a thread
  pthread_cancel(paper_pthread);
  pthread_cancel(tobacco_pthread);
  pthread_cancel(match_pthread);
  pthread_mutex_unlock(&lock_mutex); // unlock the lock
  return 0;
}
```

Finally, I created the smokers function which contains each of the specific smoker procedures. These procedures are separated by an ID that is carried over from the agent function. The ID will be taken and the corresponding smoker procedure will take place. Once again, the code for each procedure is similar to the solution pseudo code. The only difference is that V and P functions are replaced with mutex lock and unlock commands. First, the respective mutex value to the procedure is locked along with the lock mutex. Then, the procedure is printed and the agent mutex as well as the lock mutex are unlocked. The code will then return to the agent function.

My Code:

```
100
101  void *smokers(void *arg){
102    int ID = (int) arg; // get ID of chosen process
103
104    if (ID == 1) { // if ID = 1, matches smoker was chosen
105      while(1){
106        pthread_mutex_lock(&match_mutex);
107        pthread_mutex_lock(&lock_mutex);
108        printf("Matches smoker picks up tobacco from the table.\n");
109        printf("Matches smoker picks up paper from the table.\n");
110        printf("Matches smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
111        pthread_mutex_unlock(&agent_mutex); // unlock the agent
112        pthread_mutex_unlock(&lock_mutex); // unlock the lock
113      }
114
115    } else if (ID == 2) { // if ID = 2, paper smoker was chosen
116      while(1){
117        pthread_mutex_lock(&paper_mutex);  // Sleep right away
118        pthread_mutex_lock(&lock_mutex);
119        printf("Paper smoker picks up tobacco from the table.\n");
120        printf("Paper smoker picks up matches from the table.\n");
121        printf("Paper smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
122        pthread_mutex_unlock(&agent_mutex); // unlock the agent
123        pthread_mutex_unlock(&lock_mutex); // unlock the lock
124      }
125
126    } else { // if ID = 3, tobacco smoker was chosen
127      while(1) {
128        pthread_mutex_lock(&tobacco_mutex);  // Sleep right away
129        pthread_mutex_lock(&lock_mutex);
130        printf("Tobacco smoker picks up matches from the table.\n");
131        printf("Tobacco smoker picks up paper from the table.\n");
132        printf("Tobacco smoker makes a cigarette and goes to sleep.\n"); // Smoke (but don't inhale).
133        pthread_mutex_unlock(&agent_mutex); // unlock the agent
134        pthread_mutex_unlock(&lock_mutex); // unlock the lock
135      }
136    }
137    return 0;
138  }
```

Outputs

A sample of my outputs for my semaphore implementation can be viewed in the ‘sem_output.txt’ file, and a sample output for my pthread implementation can be viewed in the ‘pthread_output.txt’ file.