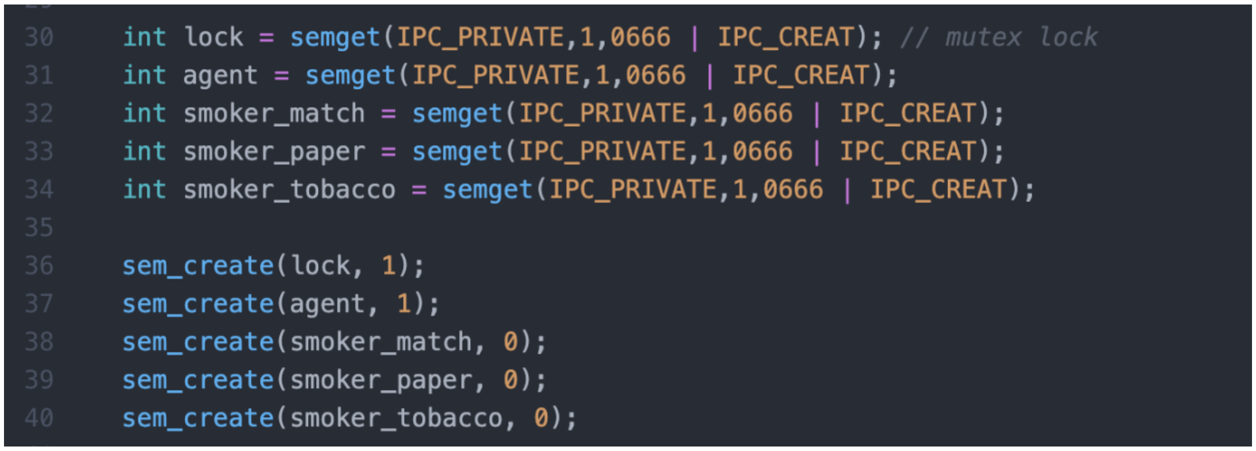
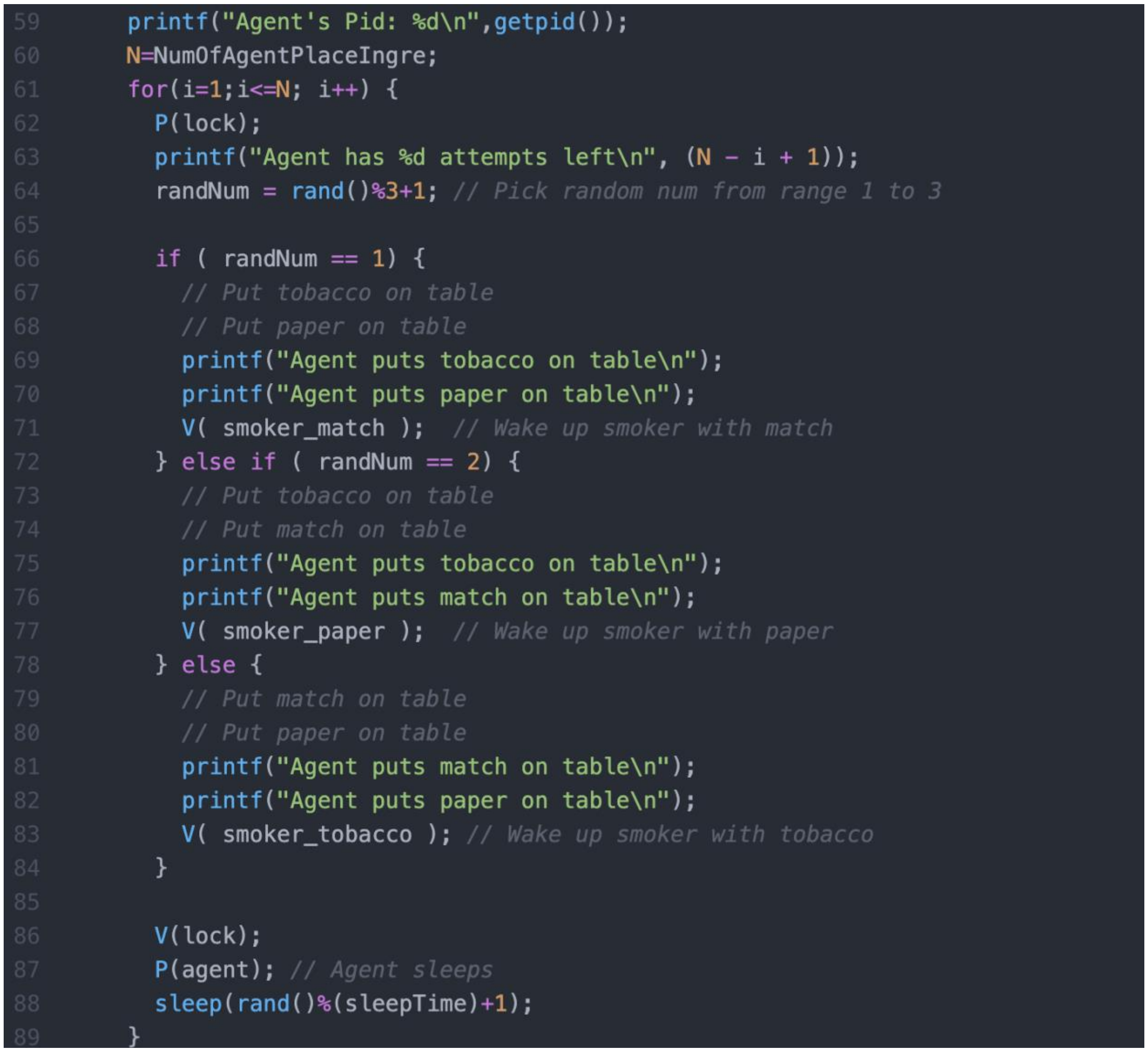
Gaurav Kuwar
CSC 332 LAB (Spring 2022)
May 15, 2022
Lab 6: Process Synchronization

<u>**Lab 6 - Report**</u>
<u>**Semaphore version**</u>

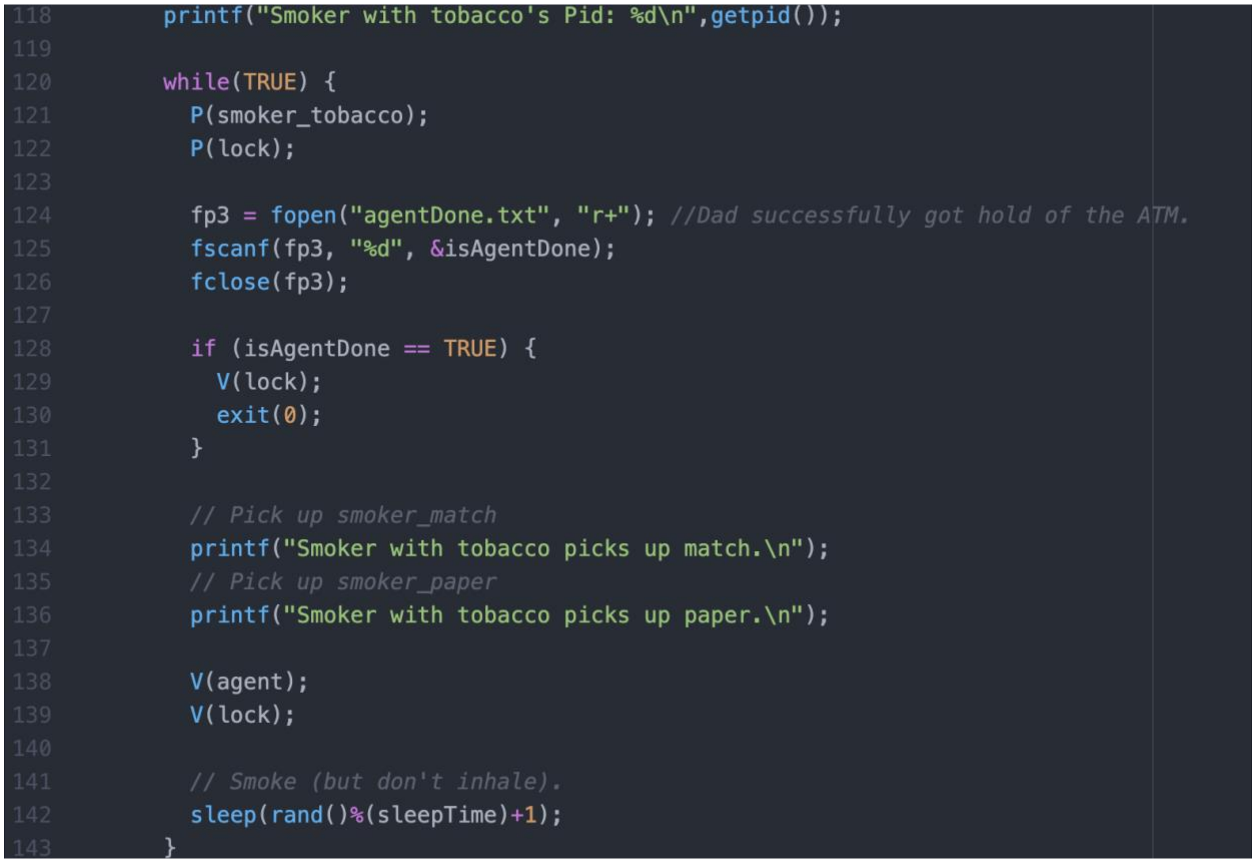


```
    int lock = semget(IPC_PRIVATE,1,0666 | IPC_CREAT); // mutex lock
    int agent = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
    int smoker_match = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
    int smoker_paper = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);
    int smoker_tobacco = semget(IPC_PRIVATE,1,0666 | IPC_CREAT);

    sem_create(lock, 1);
    sem_create(agent, 1);
    sem_create(smoker_match, 0);
    sem_create(smoker_paper, 0);
    sem_create(smoker_tobacco, 0);
```

I use 5 semaphores, and generally the code follows the pseudo code given. Lock is the mutex lock, agent wakes up or puts agent process to sleep. The three smoker semaphores allow agent to wake the specific smoker process up, when its ingredients are available.


```
    printf("Agent's Pid: %d\n",getpid());
    N=NumOfAgentPlaceIngre;
    for(i=1;i<=N; i++) {
      P(lock);
      printf("Agent has %d attempts left\n", (N - i + 1));
      randNum = rand()%3+1; // Pick random num from range 1 to 3

      if ( randNum == 1) {
        // Put tobacco on table
        // Put paper on table
        printf("Agent puts tobacco on table\n");
        printf("Agent puts paper on table\n");
        V( smoker_match );  // Wake up smoker with match
      } else if ( randNum == 2) {
        // Put tobacco on table
        // Put match on table
        printf("Agent puts tobacco on table\n");
        printf("Agent puts match on table\n");
        V( smoker_paper );  // Wake up smoker with paper
      } else {
        // Put match on table
        // Put paper on table
        printf("Agent puts match on table\n");
        printf("Agent puts paper on table\n");
        V( smoker_tobacco ); // Wake up smoker with tobacco
      }

      V(lock);
      P(agent); // Agent sleeps
      sleep(rand()%(sleepTime)+1);
    }
```

The agent process (just like the pseudo code, but runs N times only). It locks the process, and randomly selects the ingredients and wakes up the appropriate smoker process, then unlocks and goes to sleep.

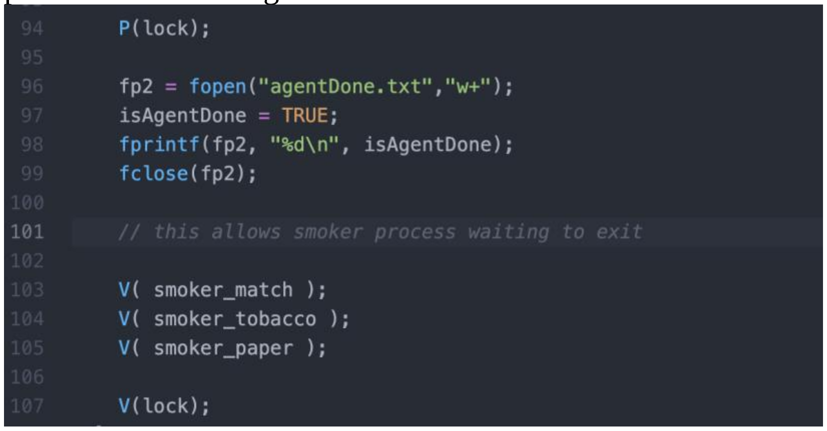


```
        printf("Smoker with tobacco's Pid: %d\n",getpid());

        while(TRUE) {
          P(smoker_tobacco);
          P(lock);

          fp3 = fopen("agentDone.txt", "r+"); //Dad successfully got hold of the ATM.
          fscanf(fp3, "%d", &isAgentDone);
          fclose(fp3);

          if (isAgentDone == TRUE) {
            V(lock);
            exit(0);
          }

          // Pick up smoker_match
          printf("Smoker with tobacco picks up match.\n");
          // Pick up smoker_paper
          printf("Smoker with tobacco picks up paper.\n");

          V(agent);
          V(lock);

          // Smoke (but don't inhale).
          sleep(rand()%(sleepTime)+1);
        }
```

All the smoker processes are relatively the same. First the smoker waits for the agent to wake it up. The it locks the process, it then checks the agentDone file for the value of isAgentDone, it exits if agent is done (I explain in further in the next section). The smoker then wakes agent, and unlocks.

<u>How do all the processes exit?</u>

Since, our program is not infinite like the pseudo code, I had to find a way to exit all the processes once the agent was done.



```
    P(lock);

    fp2 = fopen("agentDone.txt","w+");
    isAgentDone = TRUE;
    fprintf(fp2, "%d\n", isAgentDone);
    fclose(fp2);

    // this allows smoker process waiting to exit

    V( smoker_match );
    V( smoker_tobacco );
    V( smoker_paper );

    V(lock);
```

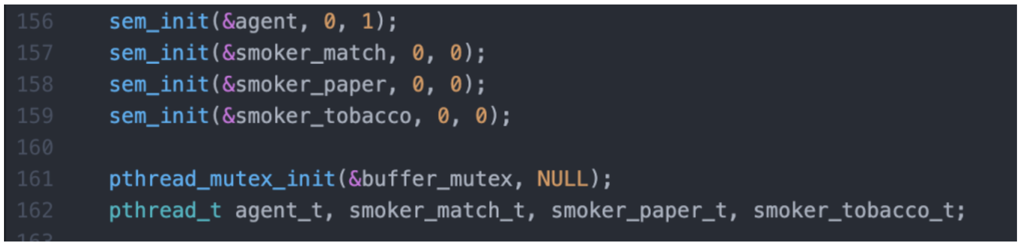
This section of code runs after agent process loop is complete. It first locks the process, it sets the agentDone.txt file value to TRUE (we create this file at the start of the code). Then it wakes up every smoker process, because they could be waiting for the agent. And unlocks.

Every smoker process checks the agentDone.txt, file and if its TRUE it will unlock the smoker process and exit the process.

This allows all the process to exit once agent process is done.

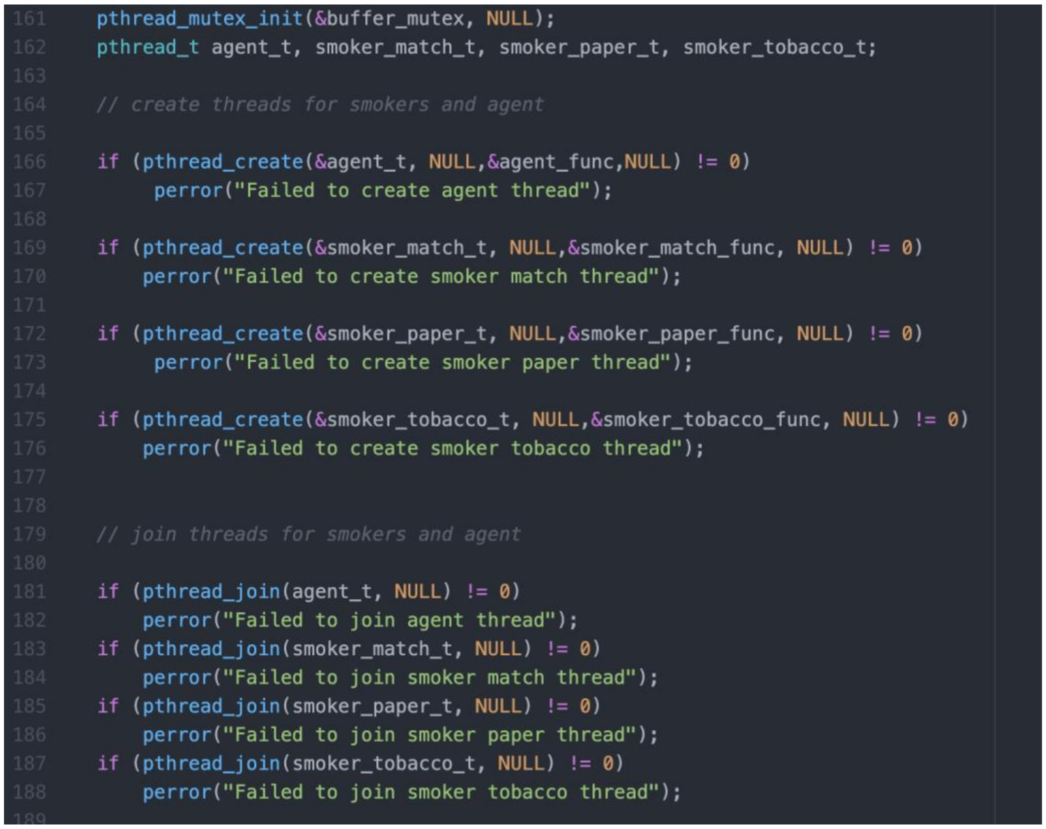
pthread version

I used the same semaphores as used in the semaphore version, except the lock semaphore is done using pthread's builtin mutex.

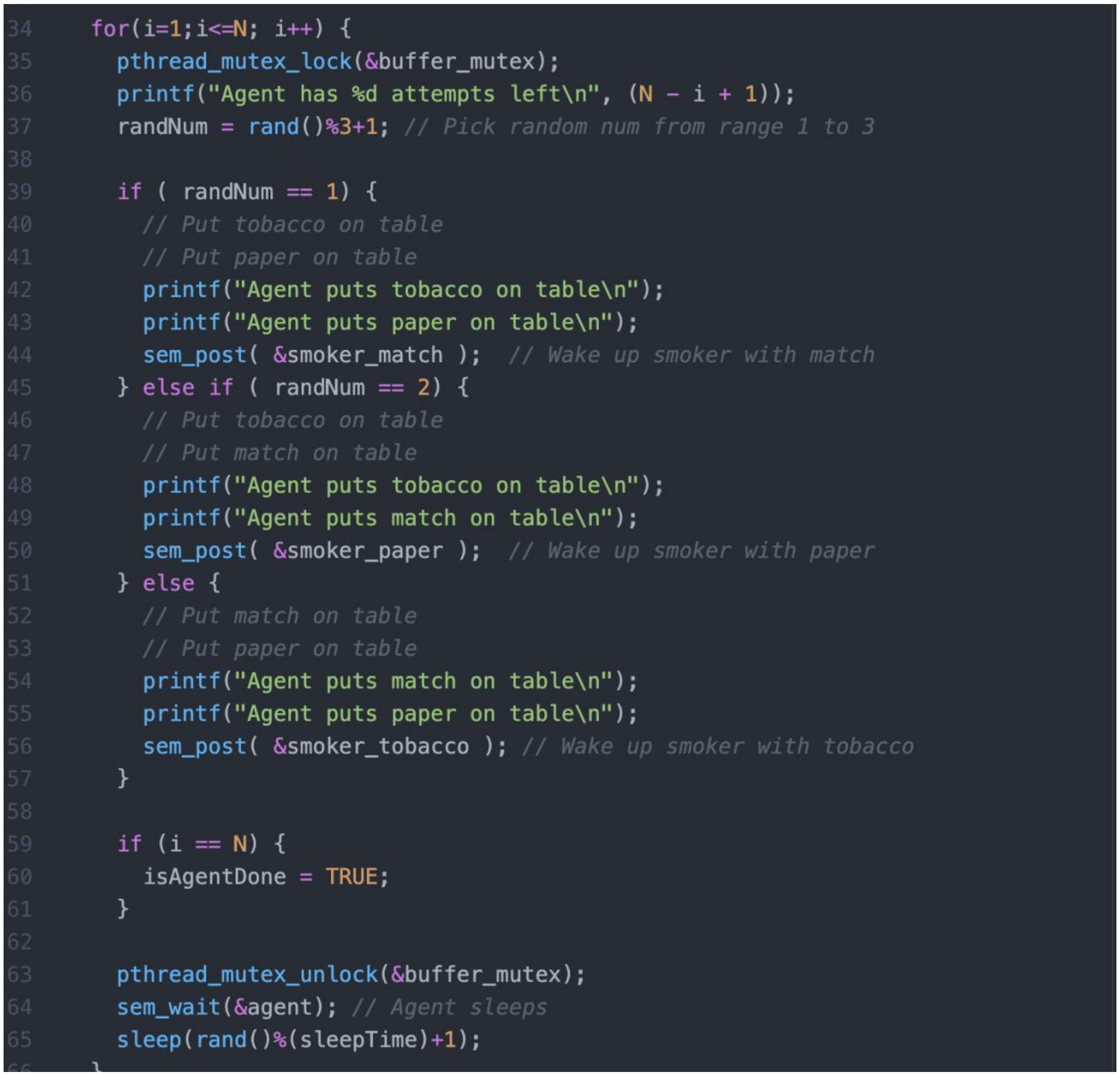


```c
sem_init(&agent, 0, 1);
sem_init(&smoker_match, 0, 0);
sem_init(&smoker_paper, 0, 0);
sem_init(&smoker_tobacco, 0, 0);

pthread_mutex_init(&buffer_mutex, NULL);
pthread_t agent_t, smoker_match_t, smoker_paper_t, smoker_tobacco_t;
```

In this version, we use threads, so I create a thread for agent and every smoker and call the appropriate function. Then use join to wait and wait for threads to finish.

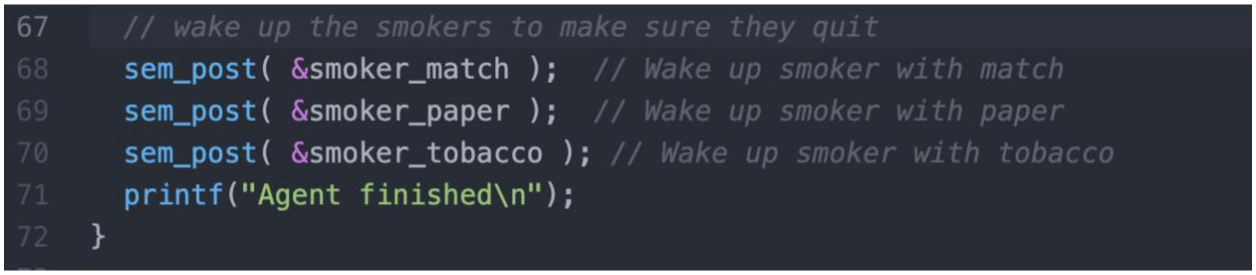


```c
pthread_mutex_init(&buffer_mutex, NULL);
pthread_t agent_t, smoker_match_t, smoker_paper_t, smoker_tobacco_t;

// create threads for smokers and agent

if (pthread_create(&agent_t, NULL,&agent_func,NULL) != 0)
     perror("Failed to create agent thread");

if (pthread_create(&smoker_match_t, NULL,&smoker_match_func, NULL) != 0)
    perror("Failed to create smoker match thread");

if (pthread_create(&smoker_paper_t, NULL,&smoker_paper_func, NULL) != 0)
     perror("Failed to create smoker paper thread");

if (pthread_create(&smoker_tobacco_t, NULL,&smoker_tobacco_func, NULL) != 0)
    perror("Failed to create smoker tobacco thread");


// join threads for smokers and agent

if (pthread_join(agent_t, NULL) != 0)
    perror("Failed to join agent thread");
if (pthread_join(smoker_match_t, NULL) != 0)
    perror("Failed to join smoker match thread");
if (pthread_join(smoker_paper_t, NULL) != 0)
    perror("Failed to join smoker paper thread");
if (pthread_join(smoker_tobacco_t, NULL) != 0)
    perror("Failed to join smoker tobacco thread");
```

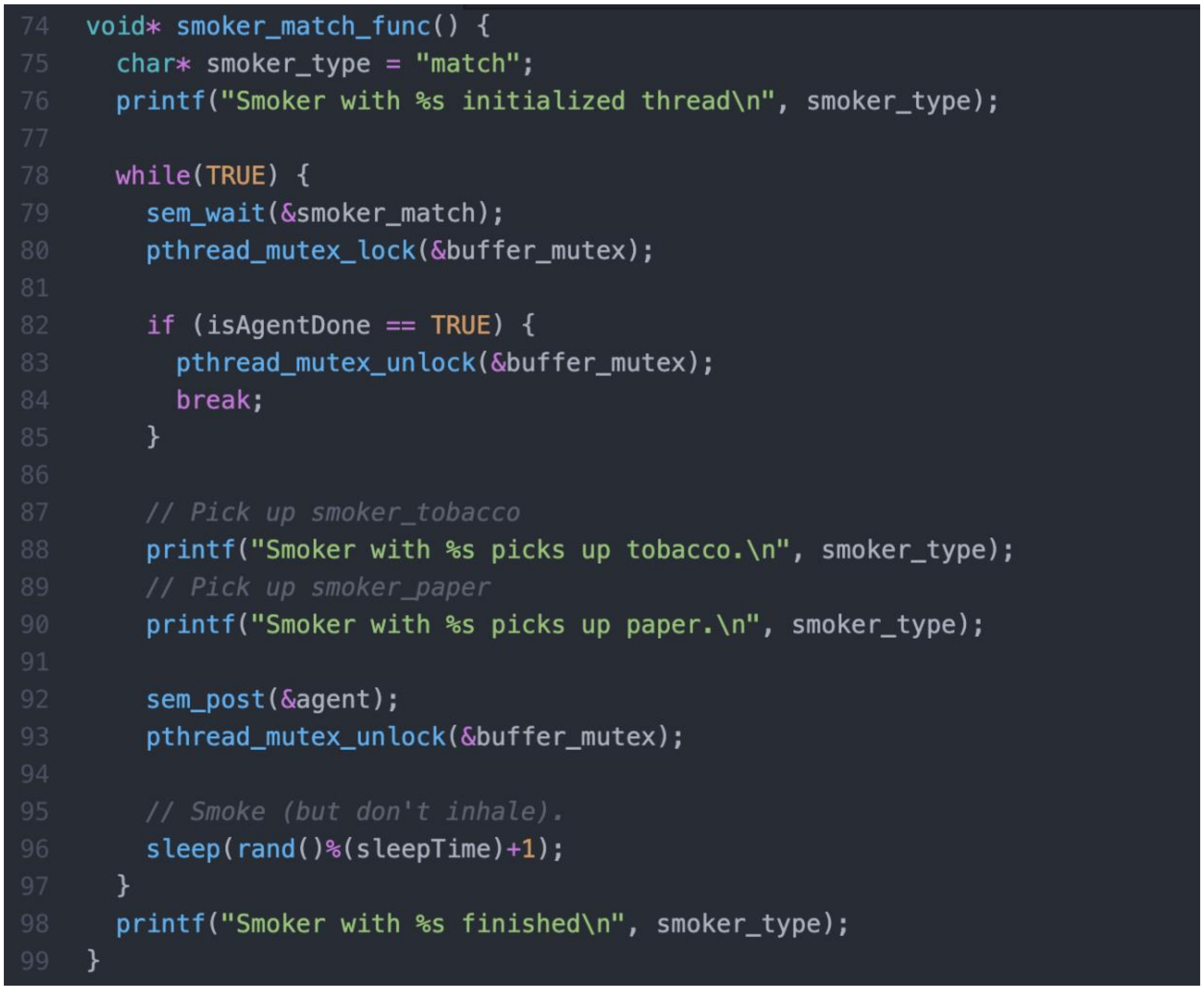


```c
  for(i=1;i<=N; i++) {
    pthread_mutex_lock(&buffer_mutex);
    printf("Agent has %d attempts left\n", (N - i + 1));
    randNum = rand()%3+1; // Pick random num from range 1 to 3

    if ( randNum == 1) {
      // Put tobacco on table
      // Put paper on table
      printf("Agent puts tobacco on table\n");
      printf("Agent puts paper on table\n");
      sem_post( &smoker_match );  // Wake up smoker with match
    } else if ( randNum == 2) {
      // Put tobacco on table
      // Put match on table
      printf("Agent puts tobacco on table\n");
      printf("Agent puts match on table\n");
      sem_post( &smoker_paper );  // Wake up smoker with paper
    } else {
      // Put match on table
      // Put paper on table
      printf("Agent puts match on table\n");
      printf("Agent puts paper on table\n");
      sem_post( &smoker_tobacco ); // Wake up smoker with tobacco
    }

    if (i == N) {
      isAgentDone = TRUE;
    }

    pthread_mutex_unlock(&buffer_mutex);
    sem_wait(&agent); // Agent sleeps
    sleep(rand()%(sleepTime)+1);
  }
```

I loop through the agent N times, where N is the number of times agent places ingredient on the table. Once agent is finished I call sem_post for every smoker semaphore to make sure they all exit.



```c
  // wake up the smokers to make sure they quit
  sem_post( &smoker_match );  // Wake up smoker with match
  sem_post( &smoker_paper );  // Wake up smoker with paper
  sem_post( &smoker_tobacco ); // Wake up smoker with tobacco
  printf("Agent finished\n");
}
```



```c
void* smoker_match_func() {
  char* smoker_type = "match";
  printf("Smoker with %s initialized thread\n", smoker_type);

  while(TRUE) {
    sem_wait(&smoker_match);
    pthread_mutex_lock(&buffer_mutex);

    if (isAgentDone == TRUE) {
      pthread_mutex_unlock(&buffer_mutex);
      break;
    }

    // Pick up smoker_tobacco
    printf("Smoker with %s picks up tobacco.\n", smoker_type);
    // Pick up smoker_paper
    printf("Smoker with %s picks up paper.\n", smoker_type);

    sem_post(&agent);
    pthread_mutex_unlock(&buffer_mutex);

    // Smoke (but don't inhale).
    sleep(rand()%(sleepTime)+1);
  }
  printf("Smoker with %s finished\n", smoker_type);
}
```

The smoker works the same way as in the semaphore version, but now there is no need for a separate file, since all threads share the isAgentDone value, and exit when agent is done.