University of Wolverhampton

School of Mathematics and Computer Science

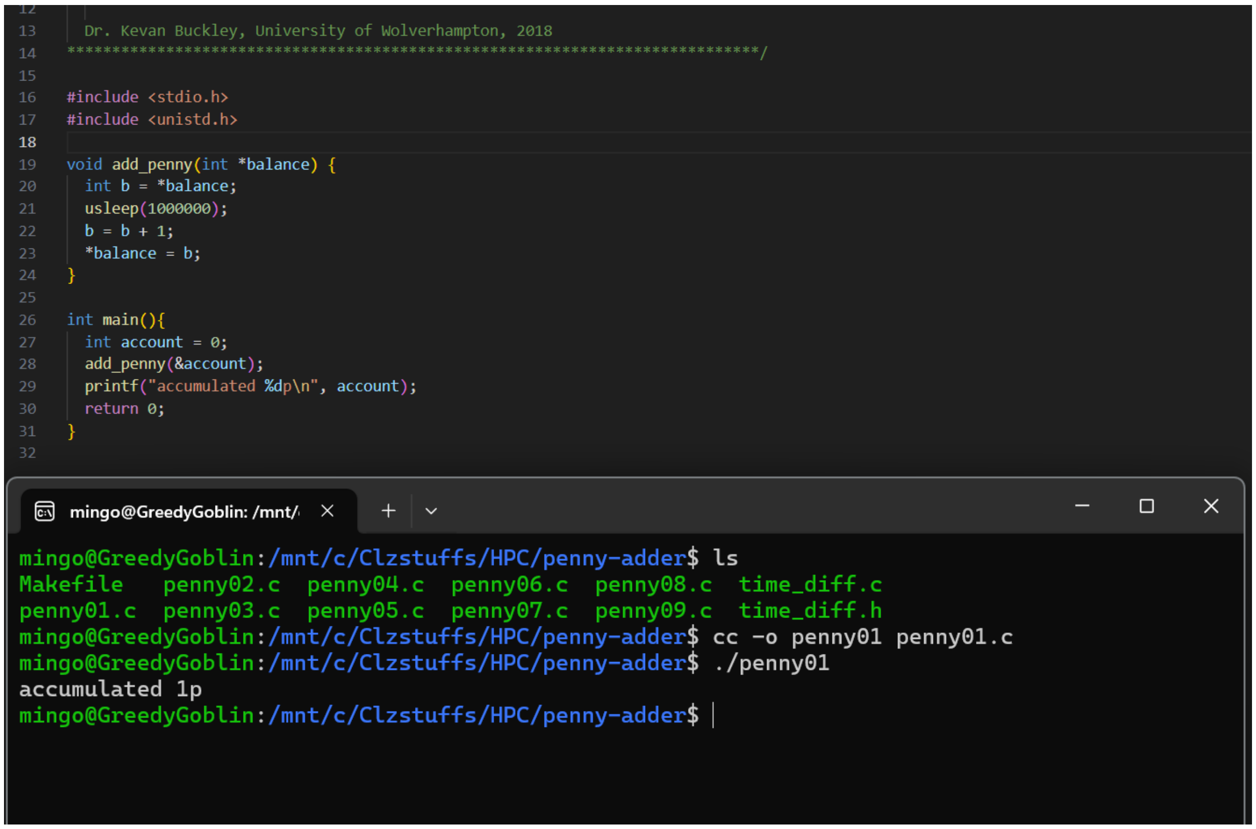
Student Number: 2407710

Name: Dhadkan K.C.

6CS005 High Performance Computing Week 4 Workshop

Penny adder

1.

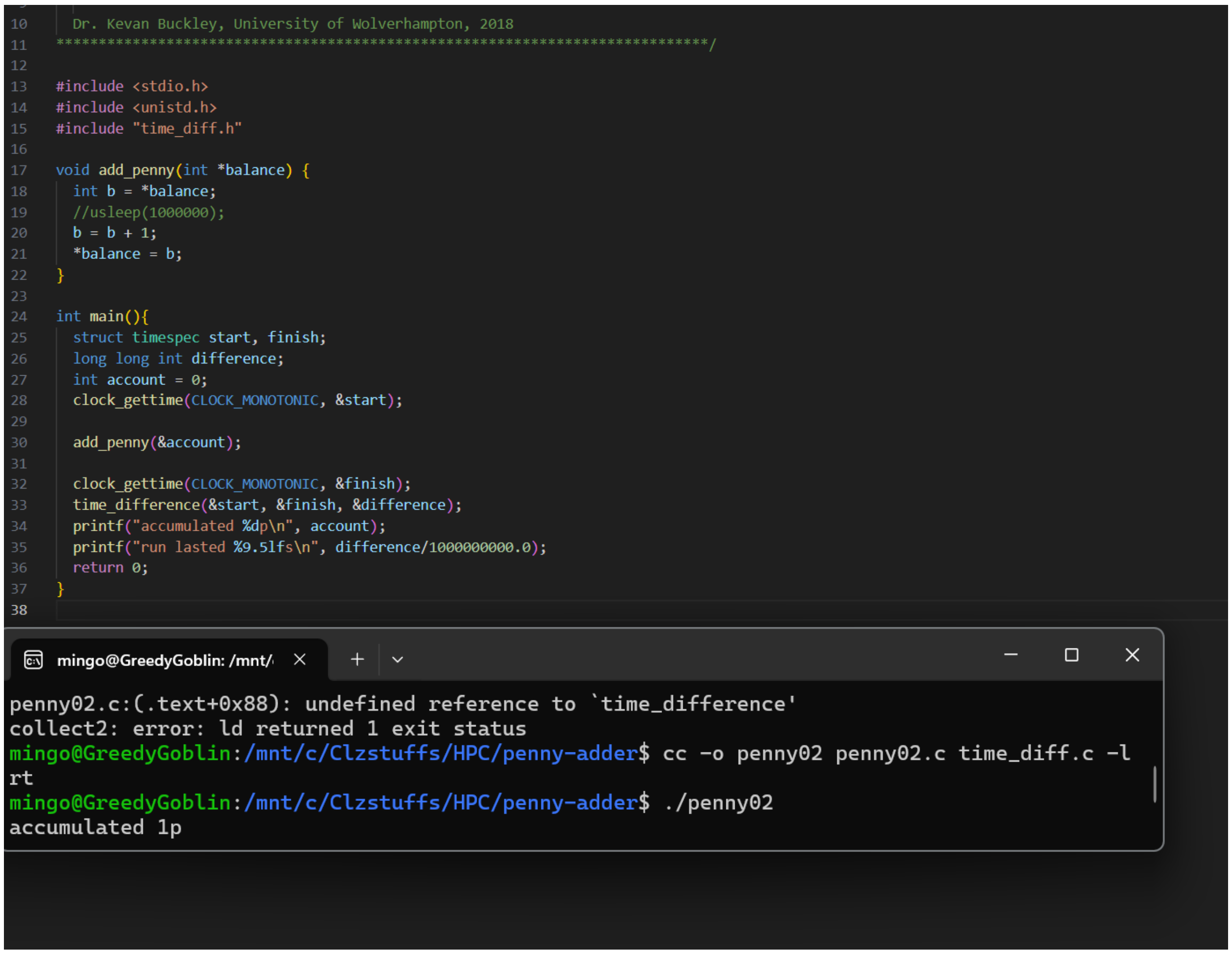


```c
  Dr. Kevan Buckley, University of Wolverhampton, 2018
*****************************************************************************/

#include <stdio.h>
#include <unistd.h>

void add_penny(int *balance) {
  int b = *balance;
  usleep(1000000);
  b = b + 1;
  *balance = b;
}

int main(){
  int account = 0;
  add_penny(&account);
  printf("accumulated %dp\n", account);
  return 0;
}
```

```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ls
Makefile   penny02.c  penny04.c  penny06.c  penny08.c  time_diff.c
penny01.c  penny03.c  penny05.c  penny07.c  penny09.c  time_diff.h
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ cc -o penny01 penny01.c
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny01
accumulated 1p
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

This program begins by including the required libraries and setting up the shared balance variable along with a mutex. The purpose of this setup is to allow multiple threads to safely update the same value without interfering with each other. By preparing the mutex early, the program ensures that later operations on the balance will be protected from race conditions.

2.

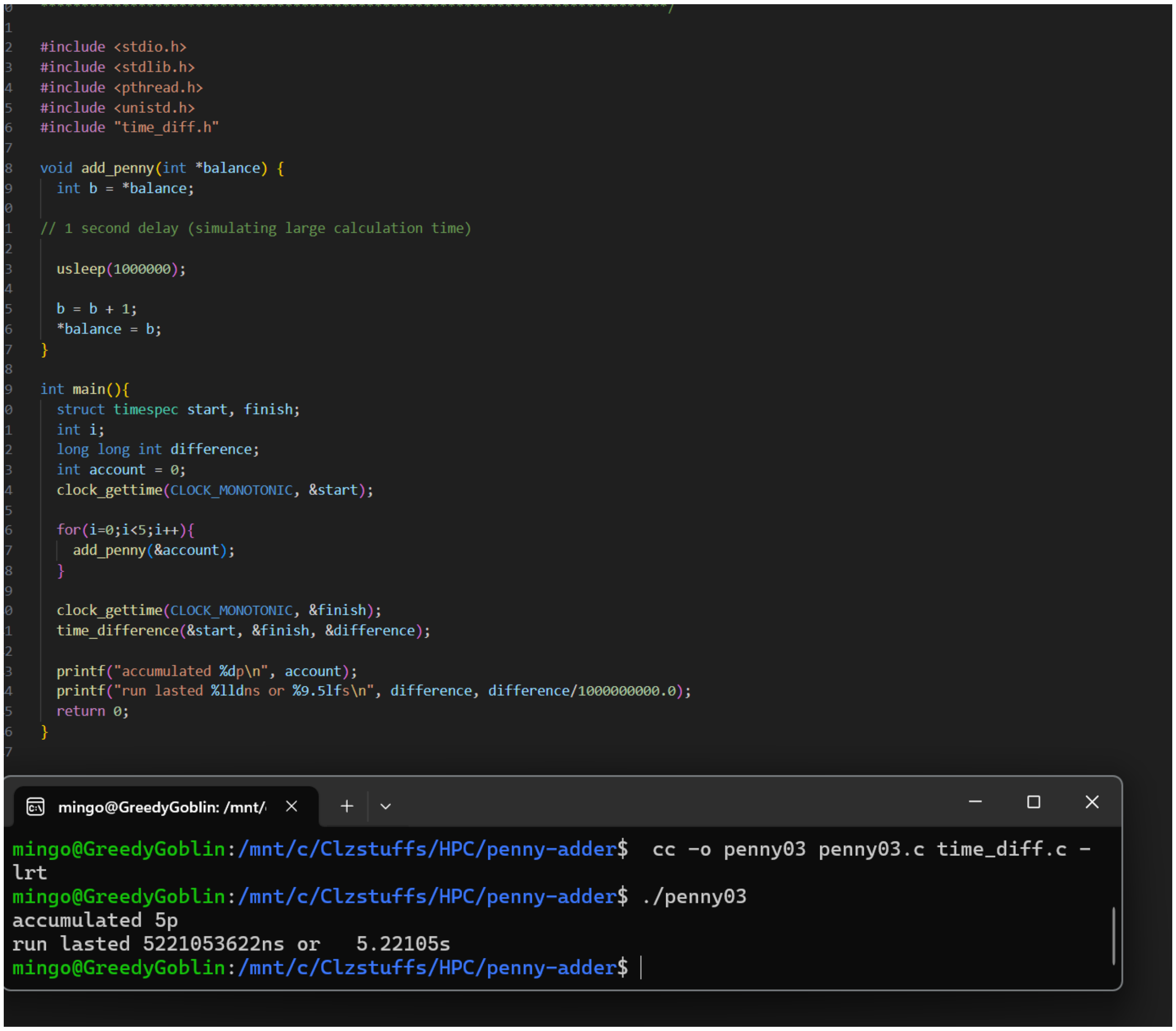


```
  Dr. Kevan Buckley, University of Wolverhampton, 2018
*****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include "time_diff.h"

void add_penny(int *balance) {
  int b = *balance;
  //usleep(1000000);
  b = b + 1;
  *balance = b;
}

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  clock_gettime(CLOCK_MONOTONIC, &start);

  add_penny(&account);

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %9.5lfs\n", difference/1000000000.0);
  return 0;
}
```

```
penny02.c:(.text+0x88): undefined reference to `time_difference'
collect2: error: ld returned 1 exit status
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ cc -o penny02 penny02.c time_diff.c -l
rt
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny02
accumulated 1p
```

This code then initializes the mutex, checking whether it was successfully created. If the mutex fails to initialize, the program exits because thread synchronization would not be guaranteed. This step ensures that every update to the shared balance during execution will be done safely and consistently.

3.

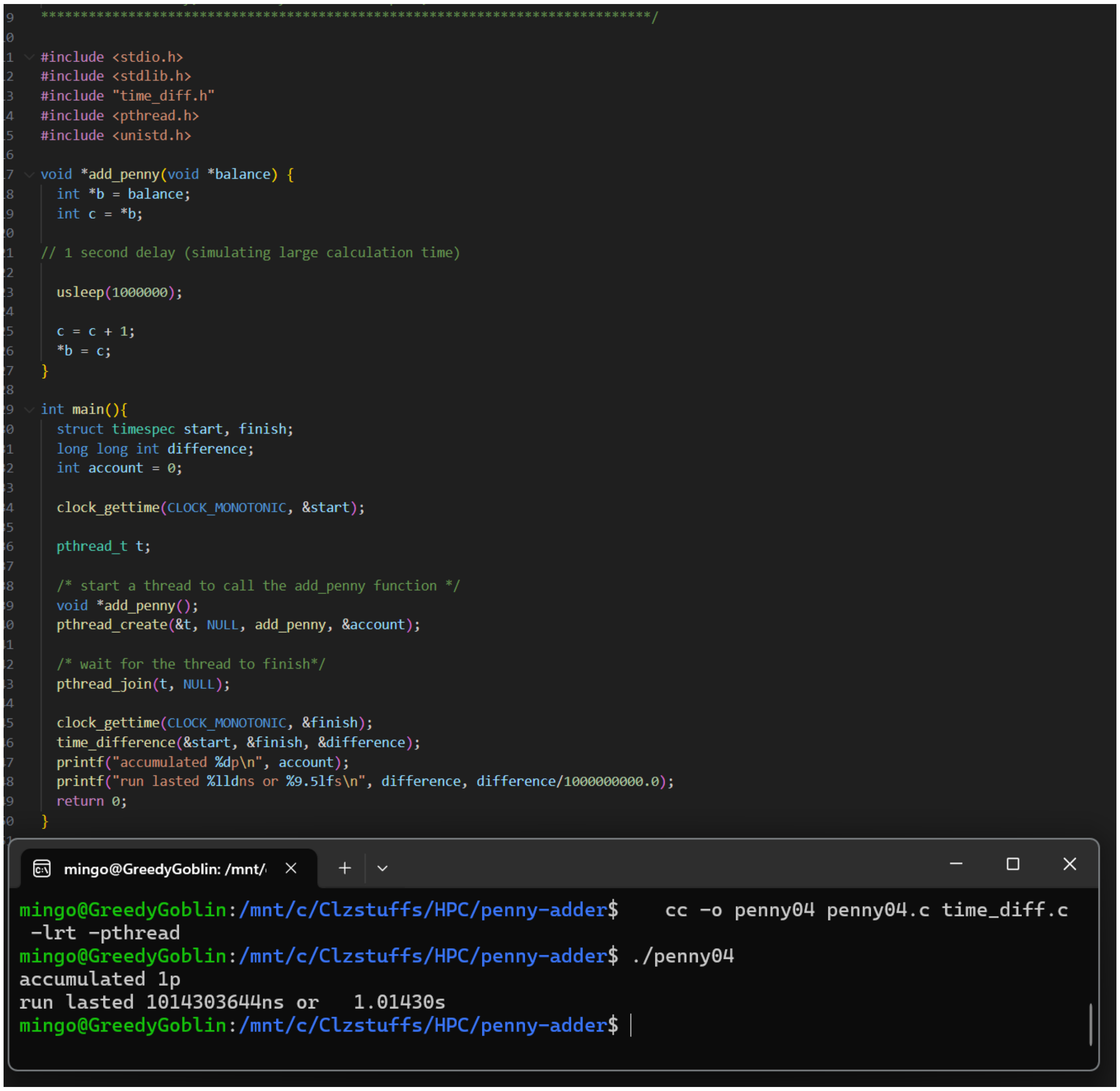


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include "time_diff.h"

void add_penny(int *balance) {
  int b = *balance;

// 1 second delay (simulating large calculation time)

  usleep(1000000);

  b = b + 1;
  *balance = b;
}

int main(){
  struct timespec start, finish;
  int i;
  long long int difference;
  int account = 0;
  clock_gettime(CLOCK_MONOTONIC, &start);

  for(i=0;i<5;i++){
    add_penny(&account);
  }

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);

  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
  return 0;
}
```

```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$  cc -o penny03 penny03.c time_diff.c -
lrt
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny03
accumulated 5p
run lasted 5221053622ns or   5.22105s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

The program sets up timing variables and a loop count so it can measure how long the penny-adding process takes. It prepares to record the time before and after the threaded work. This lets the program show not just the final outcome but also how long the synchronized operations required.

4.

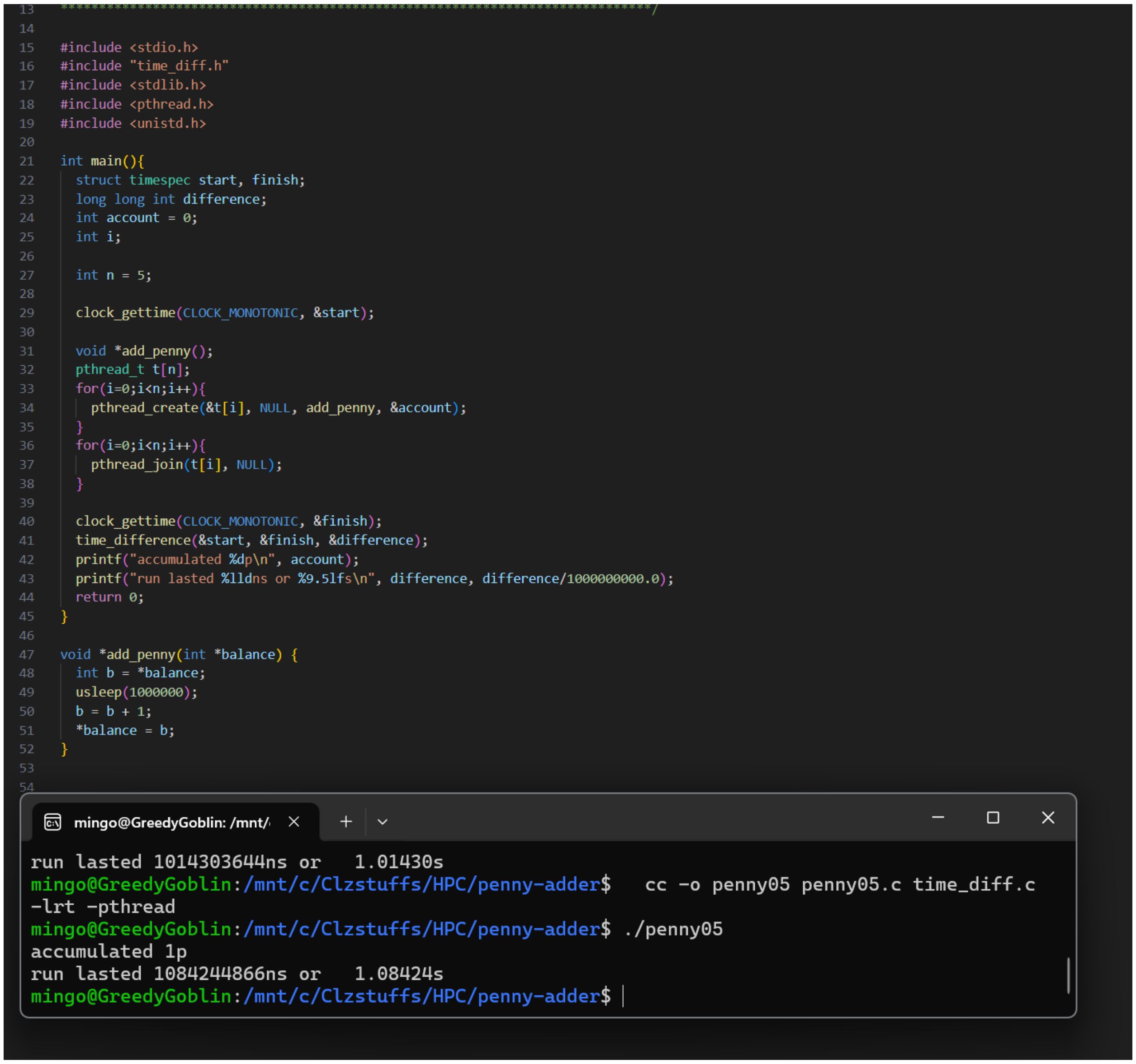


```
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "time_diff.h"
#include <pthread.h>
#include <unistd.h>

void *add_penny(void *balance) {
    int *b = balance;
    int c = *b;

// 1 second delay (simulating large calculation time)

    usleep(1000000);

    c = c + 1;
    *b = c;
}

int main(){
    struct timespec start, finish;
    long long int difference;
    int account = 0;

    clock_gettime(CLOCK_MONOTONIC, &start);

    pthread_t t;

    /* start a thread to call the add_penny function */
    void *add_penny();
    pthread_create(&t, NULL, add_penny, &account);

    /* wait for the thread to finish*/
    pthread_join(t, NULL);

    clock_gettime(CLOCK_MONOTONIC, &finish);
    time_difference(&start, &finish, &difference);
    printf("accumulated %dp\n", account);
    printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
    return 0;
}
```

```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$    cc -o penny04 penny04.c time_diff.c
 -lrt -pthread
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny04
accumulated 1p
run lasted 1014303644ns or   1.01430s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

The code creates a thread that will run the add_penny function. The program checks whether the thread was made correctly and then waits for it to finish using pthread_join. This ensures the penny-adding logic fully completes before the program prints any final results.

5.

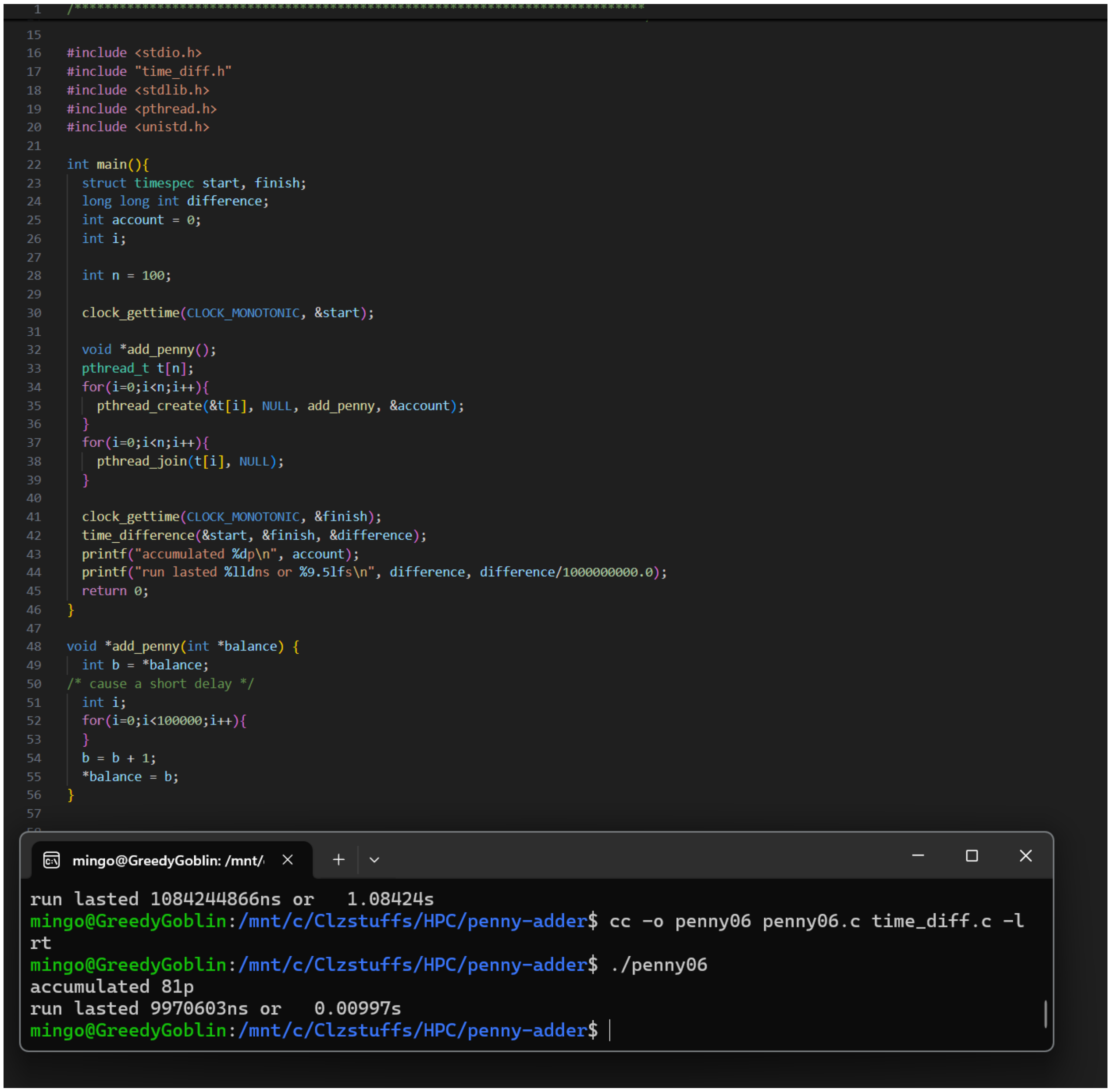


```c
****************************************************************************/

#include <stdio.h>
#include "time_diff.h"
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 5;

  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    pthread_create(&t[i], NULL, add_penny, &account);
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
  return 0;
}

void *add_penny(int *balance) {
  int b = *balance;
  usleep(1000000);
  b = b + 1;
  *balance = b;
}
```

```
run lasted 1014303644ns or   1.01430s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$   cc -o penny05 penny05.c time_diff.c
-lrt -pthread
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny05
accumulated 1p
run lasted 1084244866ns or   1.08424s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

The program captures the current time immediately before and after the threaded operation. By subtracting these two timestamps, it can calculate how long the thread took to complete all penny additions. This timing helps demonstrate how locking, unlocking, and delays affect execution speed.

6.



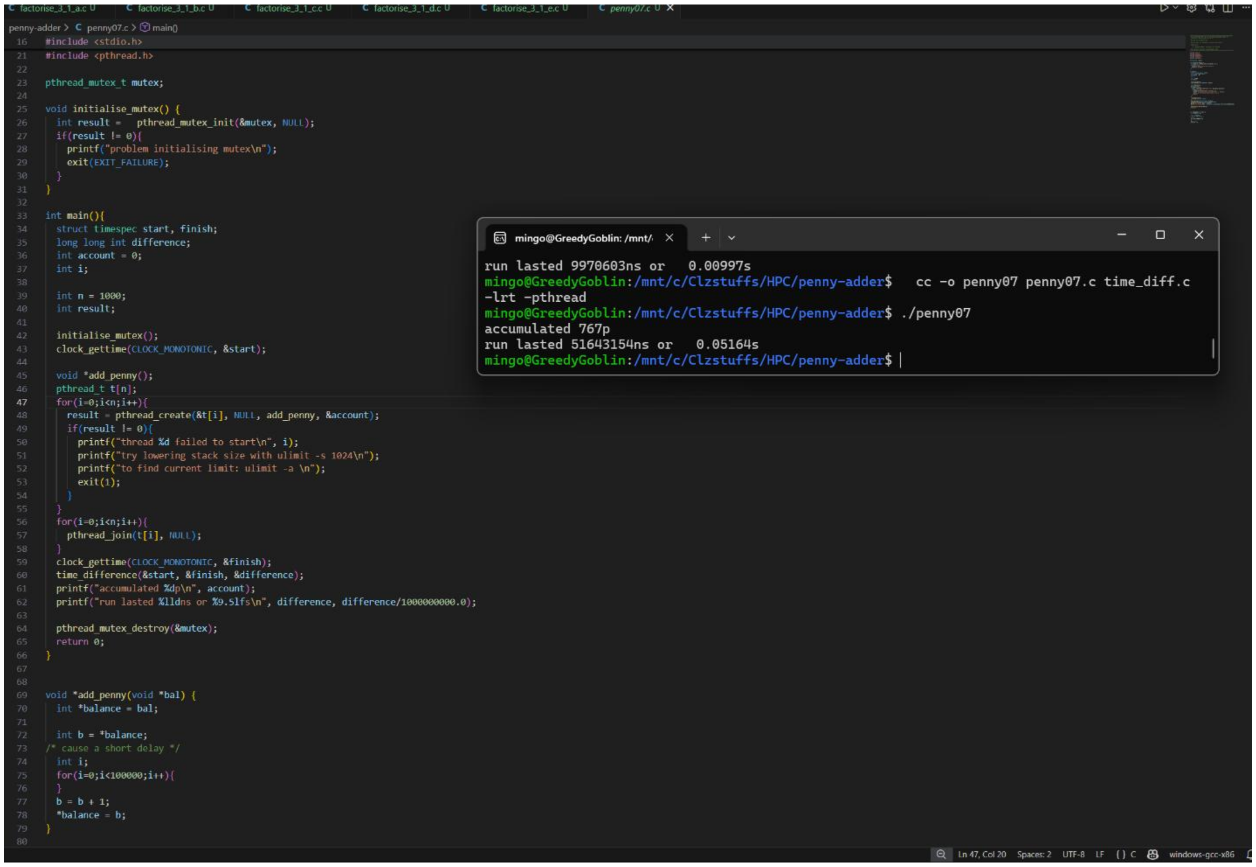
```c
/*****************************************************************************

#include <stdio.h>
#include "time_diff.h"
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 100;

  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    pthread_create(&t[i], NULL, add_penny, &account);
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }

  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
  return 0;
}

void *add_penny(int *balance) {
  int b = *balance;
/* cause a short delay */
  int i;
  for(i=0;i<100000;i++){
  }
  b = b + 1;
  *balance = b;
}
```

mingo@GreedyGoblin: /mnt/

```
run lasted 1084244866ns or   1.08424s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ cc -o penny06 penny06.c time_diff.c -l
rt
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny06
accumulated 81p
run lasted 9970603ns or   0.00997s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

The add_penny function is where each penny is added. The code locks the mutex, prints the current balance as one, two, or three depending on its value, performs a deliberate delay, then increments the balance. The mutex is unlocked afterwards so the next update can happen safely. This shows how synchronized access prevents errors when multiple updates occur.

7.

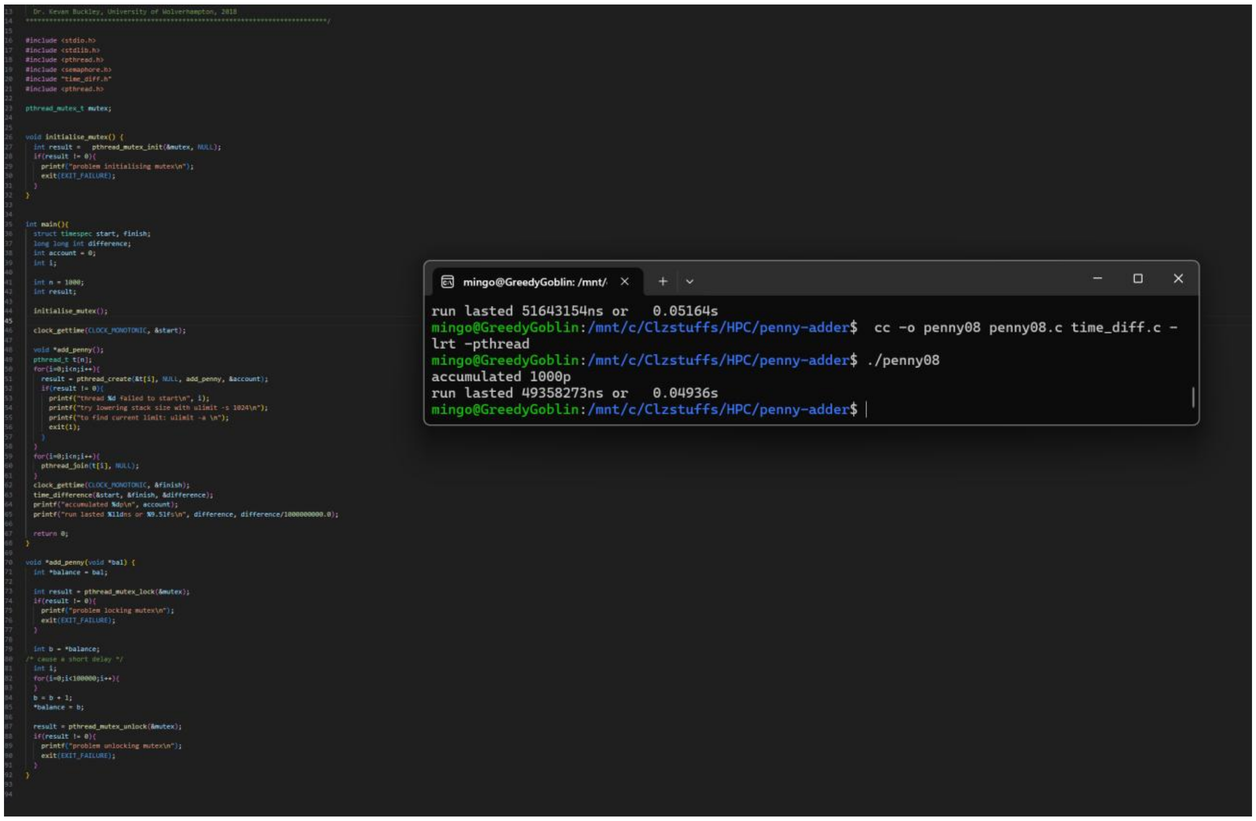


```c
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;

void initialise_mutex() {
  int result =   pthread_mutex_init(&mutex, NULL);
  if(result != 0){
    printf("problem initialising mutex\n");
    exit(EXIT_FAILURE);
  }
}

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 1000;
  int result;

  initialise_mutex();
  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    result = pthread_create(&t[i], NULL, add_penny, &account);
    if(result != 0){
      printf("thread %d failed to start\n", i);
      printf("try lowering stack size with ulimit -s 1024\n");
      printf("to find current limit: ulimit -a \n");
      exit(1);
    }
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }
  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);

  pthread_mutex_destroy(&mutex);
  return 0;
}


void *add_penny(void *bal) {
  int *balance = bal;

  int b = *balance;
/* cause a short delay */
  int i;
  for(i=0;i<100000;i++){
  }
  b = b + 1;
  *balance = b;
}
```

```
run lasted 9970603ns or   0.00997s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$  cc -o penny07 penny07.c time_diff.c
-lrt -pthread
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny07
accumulated 767p
run lasted 51643154ns or   0.05164s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

This part of the code contains the delay loop inside add_penny. The loop intentionally slows down each iteration so the effect of locking becomes visible in the output. Because of this, the printed values appear many times, making it clear how often the balance is read and changed during execution.

8.



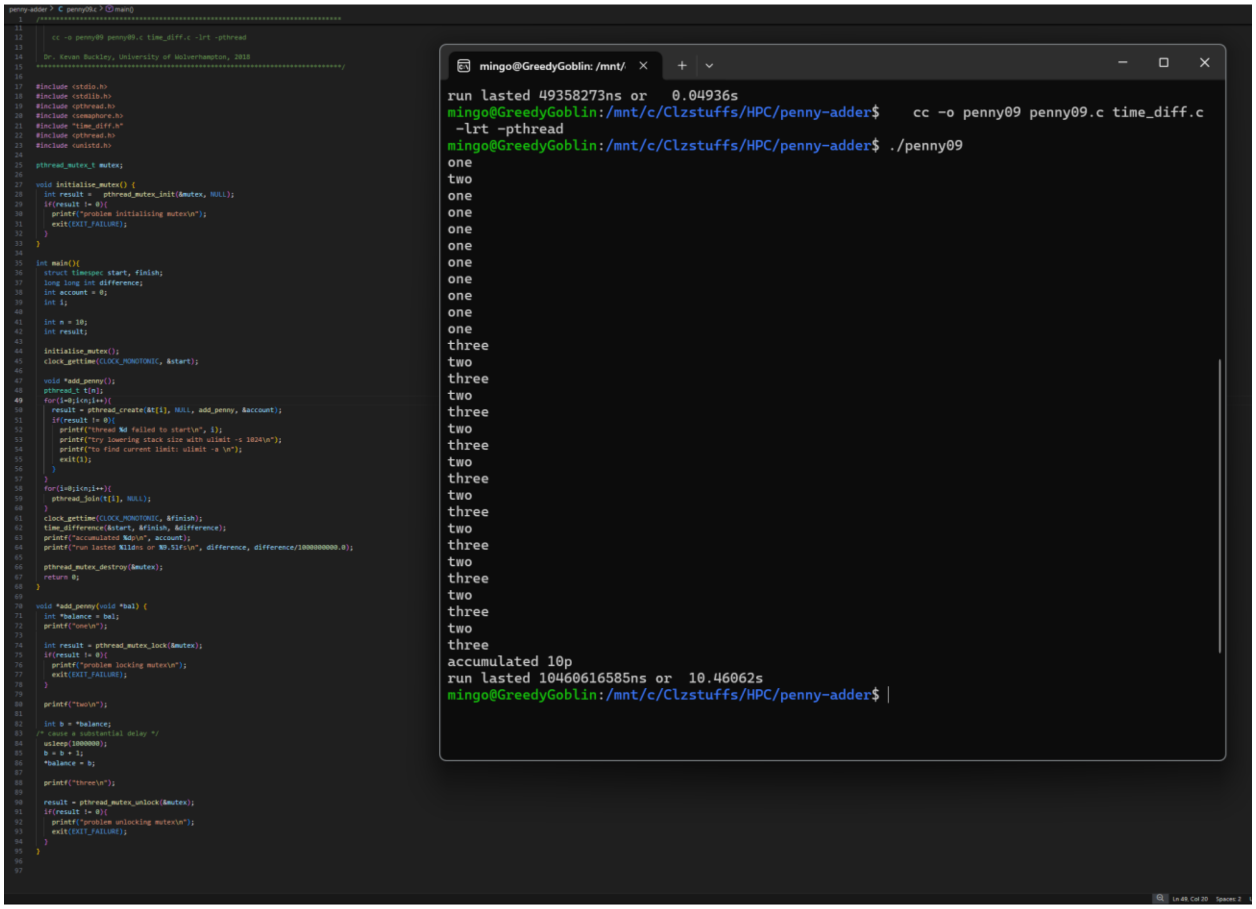
```c
  Dr. Kevan Buckley, University of Wolverhampton, 2018
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include "time_diff.h"
#include <pthread.h>

pthread_mutex_t mutex;


void initialise_mutex() {
  int result =   pthread_mutex_init(&mutex, NULL);
  if(result != 0){
    printf("problem initialising mutex\n");
    exit(EXIT_FAILURE);
  }
}


int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 1000;
  int result;

  initialise_mutex();

  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    result = pthread_create(&t[i], NULL, add_penny, &account);
    if(result != 0){
      printf("thread %d failed to start\n", i);
      printf("try lowering stack size with ulimit -s 1024\n");
      printf("to find current limit: ulimit -a \n");
      exit(1);
    }
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }
  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);

  return 0;
}

void *add_penny(void *bal) {
  int *balance = bal;

  int result = pthread_mutex_lock(&mutex);
  if(result != 0){
    printf("problem locking mutex\n");
    exit(EXIT_FAILURE);
  }

  int b = *balance;
/* cause a short delay */
  int i;
  for(i=0;i<100000;i++){
  }
  b = b + 1;
  *balance = b;

  result = pthread_mutex_unlock(&mutex);
  if(result != 0){
    printf("problem unlocking mutex\n");
    exit(EXIT_FAILURE);
  }
}
```

mingo@GreedyGoblin: /mnt/

```
run lasted 51643154ns or   0.05164s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$  cc -o penny08 penny08.c time_diff.c -
lrt -pthread
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny08
accumulated 1000p
run lasted 49358273ns or   0.04936s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ 
```

The program finishes by printing the final accumulated balance after all increments are completed. Since the thread runs ten times, the final result is 10p. It also prints the total time taken, showing how long the synchronized and delayed operations required to finish.

9.



```c
/*****************************************************************************

   cc -o penny09 penny09.c time_diff.c -lrt -pthread

  Dr. Kevan Buckley, University of Wolverhampton, 2018
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include "time_diff.h"
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mutex;

void initialise_mutex() {
  int result =   pthread_mutex_init(&mutex, NULL);
  if(result != 0){
    printf("problem initialising mutex\n");
    exit(EXIT_FAILURE);
  }
}

int main(){
  struct timespec start, finish;
  long long int difference;
  int account = 0;
  int i;

  int n = 10;
  int result;

  initialise_mutex();
  clock_gettime(CLOCK_MONOTONIC, &start);

  void *add_penny();
  pthread_t t[n];
  for(i=0;i<n;i++){
    result = pthread_create(&t[i], NULL, add_penny, &account);
    if(result != 0){
      printf("thread %d failed to start\n", i);
      printf("try lowering stack size with ulimit -s 1024\n");
      printf("to find current limit: ulimit -a \n");
      exit(1);
    }
  }
  for(i=0;i<n;i++){
    pthread_join(t[i], NULL);
  }
  clock_gettime(CLOCK_MONOTONIC, &finish);
  time_difference(&start, &finish, &difference);
  printf("accumulated %dp\n", account);
  printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);

  pthread_mutex_destroy(&mutex);
  return 0;
}

void *add_penny(void *bal) {
  int *balance = bal;
  printf("one\n");

  int result = pthread_mutex_lock(&mutex);
  if(result != 0){
    printf("problem locking mutex\n");
    exit(EXIT_FAILURE);
  }

  printf("two\n");

  int b = *balance;
/* cause a substantial delay */
  usleep(1000000);
  b = b + 1;
  *balance = b;

  printf("three\n");

  result = pthread_mutex_unlock(&mutex);
  if(result != 0){
    printf("problem unlocking mutex\n");
    exit(EXIT_FAILURE);
  }
}
```

```
run lasted 49358273ns or   0.04936s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$    cc -o penny09 penny09.c time_diff.c
 -lrt -pthread
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$ ./penny09
one
two
one
one
one
one
one
one
one
one
one
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
two
three
accumulated 10p
run lasted 10460616585ns or  10.46062s
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/penny-adder$
```

The output shows repeated prints of one, two, and three because the thread repeatedly checks and updates the balance while the delay loop is running. The slow increments make each stage appear many times. At the end, the program prints accumulated 10p and the total runtime, confirming that all increments happened correctly under the mutex protection.