

# 6CS005 High Performance Computing

## Multi-threading Problems

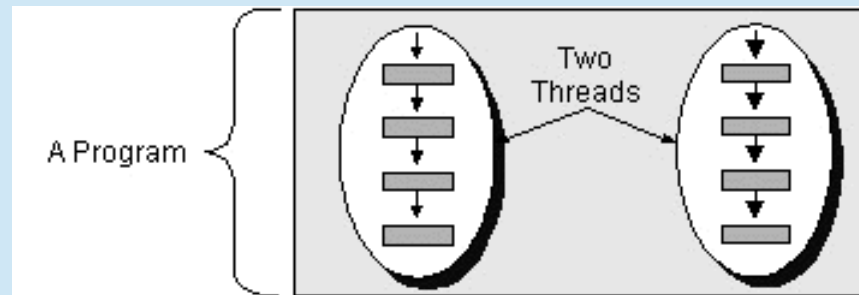
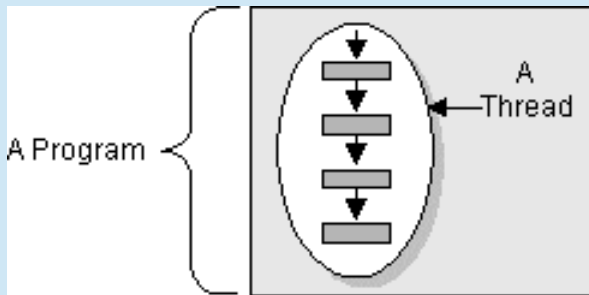
Dr Kevan Buckley

# Aims

- To be able to write concurrent programs using POSIX threads.
- To understand problems that can occur in multi-thread programs.
  - Data corruption
  - Bottlenecks
- The code presented here can be found at <http://www.scit.wlv.ac.uk/~in6659/hpc> and on Canvas.

# What is a Concurrent Program?

- A sequential program has a single thread of control.

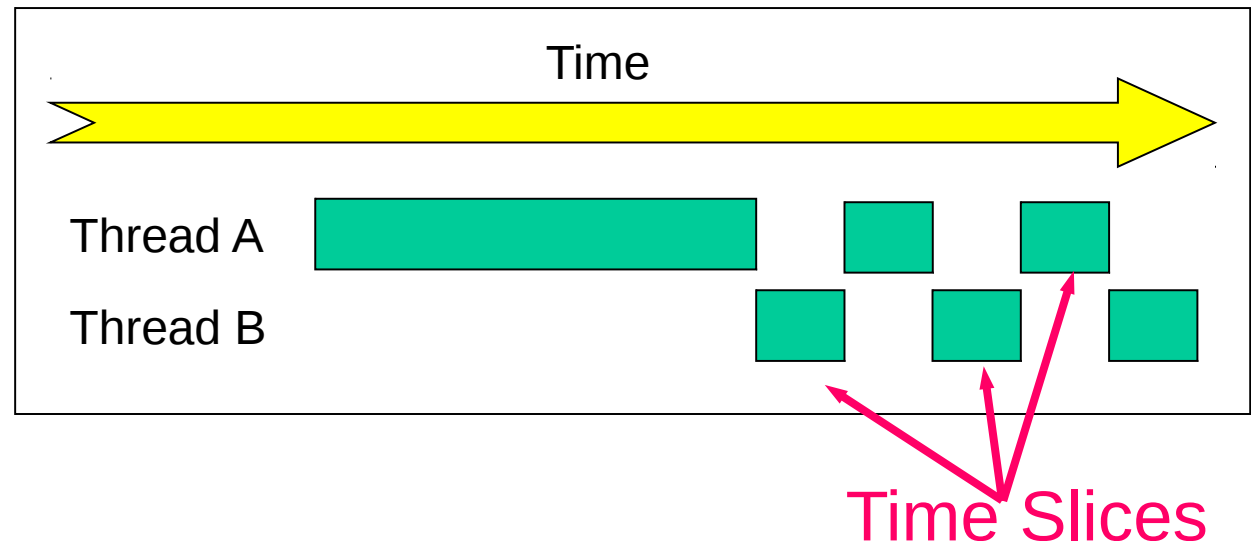


- A concurrent program has multiple threads of control allowing it to perform multiple computations in parallel and to control multiple external activities that occur at the same time.

# Concurrency and Threads

- Modern operating systems run tasks concurrently by splitting tasks into smaller chunks.
- One task is executed for a small amount of time
- Then a thread is *pre-empted*, enabling another thread to run.

Execution is  
*interleaved*



If time slices are small enough it seems like several things are happening at once

On machines with more than one processor, threads might actually run simultaneously

# Bank Account Program

```
01: /*****
02:  This program is the first in a series that looks at adding a penny to a
03:  bank account balance. It simply sets up a bank account balance as an
04:  integer and calls a function to add a penny to the account. This is done
05:  in a convoluted way (instead of just b++). This is for consistency with
06:  later versions that will demonstrate intermittent problems in the increment
07:  operation that can be accentuated when a delay is introduced.
08:
09:  Compile with:
10:
11:      cc -o penny01 penny01.c
12:
13:  Dr. Kevan Buckley, University of Wolverhampton, 2018
14:
*****/
```

# Bank Account Program

```
16: #include <stdio.h>
17: #include <unistd.h>
18:
19: void add_penny(int *balance) {
20:     int b = *balance;
21:     usleep(1000000);
22:     b = b + 1;
23:     *balance = b;
24: }
25:
26: int main(){
27:     int account = 0;
28:     add_penny(&account);
29:     printf("accumulated %dp\n", account);
30:     return 0;
31: }
```

How much money will the account contain?

# Running the Program

```
kev@nikola:~$ cd penny-adder/  
kev@nikola:~/penny-adder$ make  
cc -o penny01 penny01.c  
cc -o penny02 penny02.c time_diff.c -lrt -pthread  
cc -o penny03 penny03.c time_diff.c -lrt -pthread  
cc -o penny04 penny04.c time_diff.c -lrt -pthread  
cc -o penny05 penny05.c time_diff.c -lrt -pthread  
cc -o penny06 penny06.c time_diff.c -lrt -pthread  
cc -o penny07 penny07.c time_diff.c -lrt -pthread  
cc -o penny08 penny08.c time_diff.c -lrt -pthread  
cc -o penny09 penny09.c time_diff.c -lrt -pthread  
kev@nikola:~/penny-adder$ ./penny01  
accumulated 1p  
kev@nikola:~/penny-adder$
```

# Calculating The Running Time

```
17: void add_penny(int *balance) {
18:     int b = *balance;
19:     usleep(1000000);
20:     b = b + 1;
21:     *balance = b;
22: }
23:
24: int main(){
25:     struct timespec start, finish;
26:     long long int difference;
27:     int account = 0;
28:     clock_gettime(CLOCK_MONOTONIC, &start);
29:
30:     add_penny(&account);
31:
32:     clock_gettime(CLOCK_MONOTONIC, &finish);
33:     time_difference(&start, &finish, &difference);
34:     printf("accumulated %dp\n", account);
35:     printf("run lasted %9.5lfs\n", difference/1000000000.0);
36:     return 0;
37: }
```

How long will it take?

How much money will be accumulated?



# Running the Program

```
kev@nikola:~/penny-adder$ ./penny02
accumulated 1p
run lasted 1.00012s
kev@nikola:~/penny-adder$
```

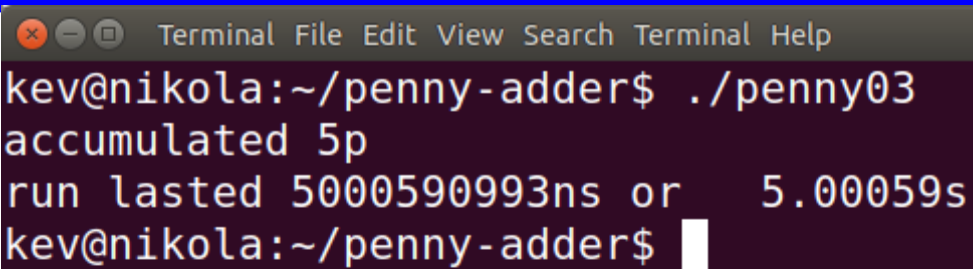
# Adding Multiple Pennies

```
29: int main(){
30:     struct timespec start, finish;
31:     int i;
32:     long long int difference;
33:     int account = 0;
34:     clock_gettime(CLOCK_MONOTONIC, &start);
35:
36:     for(i=0; i<5; i++){
37:         add_penny(&account);
38:     }
39:
40:     clock_gettime(CLOCK_MONOTONIC, &finish);
41:     time_difference(&start, &finish, &difference);
42:
43:     printf("accumulated %dp\n", account);
44:     printf("run lasted %lldns or %9.5lfs\n", difference,
difference/1000000000.0);
45:     return 0;
46: }
```

```
18: void add_penny(int *balance) {
19:     int b = *balance;
20:
21:     // 1 second delay (simulating la
22:
23:     usleep(1000000);
24:
25:     b = b + 1;
26:     *balance = b;
27: }
```

How long will it take?  
How much money will be accumulated?

# Running the Program

A terminal window with a dark background and a menu bar at the top containing 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the execution of a program named 'penny03'. The output indicates that 5p has been accumulated and the run lasted 5000590993ns or 5.00059s. The prompt 'kev@nikola:~/penny-adder\$' is visible at the end of the line.

```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny03
accumulated 5p
run lasted 5000590993ns or 5.00059s
kev@nikola:~/penny-adder$
```

# Using a Thread

```
29: int main(){
30:     struct timespec start, finish;
31:     long long int difference;
32:     int account = 0;
33:
34:     clock_gettime(CLOCK_MONOTONIC, &start);
35:
36:     pthread_t t;
37:
38:     /* start a thread to call the add_penny function */
39:     void *add_penny();
40:     pthread_create(&t, NULL, add_penny, &account);
41:
42:     /* wait for the thread to finish*/
43:     pthread_join(t, NULL);
44:
45:     clock_gettime(CLOCK_MONOTONIC, &finish);
46:     time_difference(&start, &finish, &difference);
47:     printf("accumulated %dp\n", account);
48:     printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
49:     return 0;
50: }
```

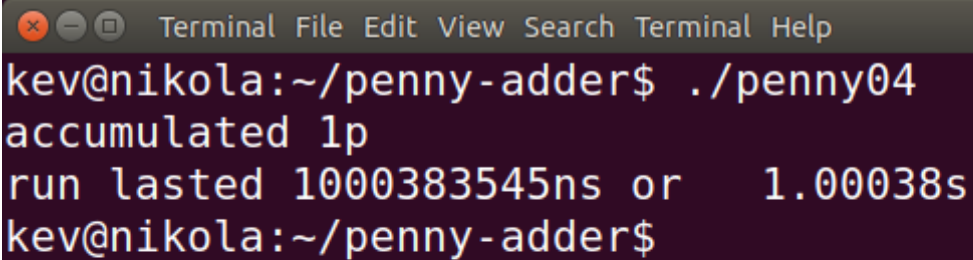
How long will it take?  
How much money will be accumulated?

# Thread Function

```
17: void *add_penny(void *balance) {  
18:     int *b = balance;  
19:     int c = *b;  
20:  
21:     // 1 second delay (simulating large calculation time)  
22:  
23:     usleep(1000000);  
24:  
25:     c = c + 1;  
26:     *b = c;  
27: }
```

How long will it take?  
How much money will be accumulated?

# Running th Program

A terminal window with a dark background and a menu bar at the top containing 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the execution of a program named 'penny04'. The output consists of three lines: 'accumulated 1p', 'run lasted 1000383545ns or 1.00038s', and a prompt for the next command.

```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny04
accumulated 1p
run lasted 1000383545ns or 1.00038s
kev@nikola:~/penny-adder$
```

# Adding Multiple Pennies With Threads

```
21: int main(){
22:     struct timespec start, finish;
23:     long long int difference;
24:     int account = 0;
25:     int i;
26:
27:     int n = 5;
28:
29:     clock_gettime(CLOCK_MONOTONIC, &start);
30:
31:     void *add_penny();
32:     pthread_t t[n];
33:     for(i=0; i<n; i++){
34:         pthread_create(&t[i], NULL, add_penny, &account);
35:     }
36:     for(i=0; i<n; i++){
37:         pthread_join(t[i], NULL);
38:     }
39:
40:     clock_gettime(CLOCK_MONOTONIC, &finish);
41:     time_difference(&start, &finish, &difference);
42:     printf("accumulated %dp\n", account);
43:     printf("run lasted %lldns or %9.5lfs\n", difference, difference/1000000000.0);
44:     return 0;
45: }
```

How long will it take?  
How much money will be accumulated?

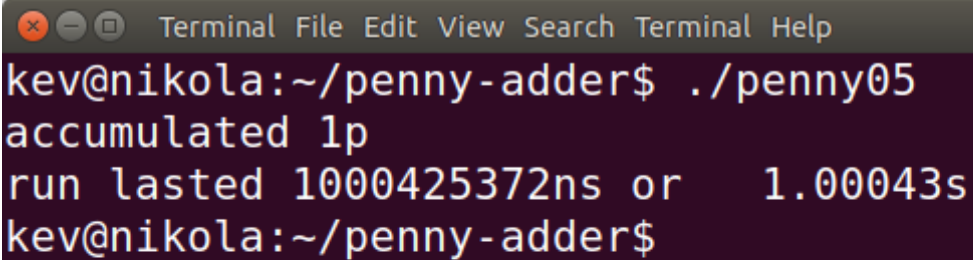
# Thread Function with Delay

```
17: void *add_penny(void *balance) {  
18:     int *b = balance;  
19:     int c = *b;  
20:  
21:     // 1 second delay (simulating large calculation time)  
22:  
23:     usleep(1000000);  
24:  
25:     c = c + 1;  
26:     *b = c;  
27: }
```

How long will it take?  
How much money will be accumulated?



# Running the Program

A terminal window with a dark background and a light-colored title bar. The title bar contains the text "Terminal File Edit View Search Terminal Help" and three window control icons (close, minimize, maximize). The terminal content shows a user named "kev" at a host named "nikola" in a directory "~ / penny-adder". The user has executed the command "./penny05". The output of the program is displayed on the next two lines: "accumulated 1p" and "run lasted 1000425372ns or 1.00043s". The prompt "kev@nikola:~/penny-adder\$" is shown again on the third line.

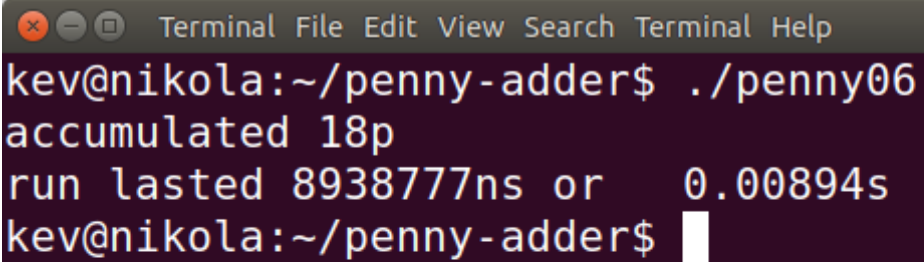
```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny05
accumulated 1p
run lasted 1000425372ns or 1.00043s
kev@nikola:~/penny-adder$
```

# A Shorter Delay

```
48: void *add_penny(int *balance) {  
49:     int b = *balance;  
50:     /* cause a short delay */  
51:     int i;  
52:     for(i=0;i<1000000;i++){  
53:     }  
54:     b = b + 1;  
55:     *balance = b;  
56: }
```

How long will it take?  
How much money will be accumulated?

# Running the Program

A screenshot of a terminal window with a dark background and light-colored text. The window has a title bar with standard macOS window controls (close, zoom, and full screen buttons) and a menu bar with the following items: Terminal, File, Edit, View, Search, Terminal, and Help. The terminal content shows a user named 'kev' at a host named 'nikola' in the directory '~/penny-adder' running a program called 'penny06'. The program outputs 'accumulated 18p' and 'run lasted 8938777ns or 0.00894s'. The prompt returns to the user.

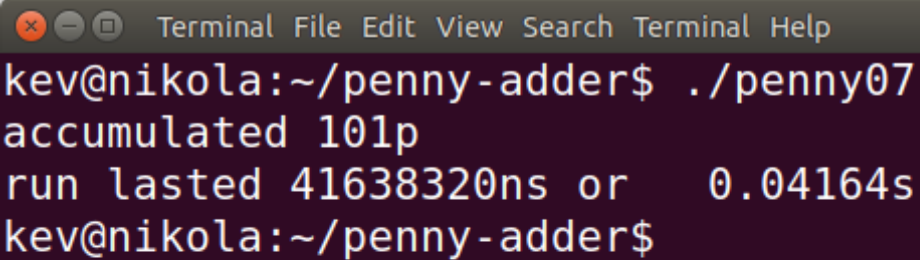
```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny06
accumulated 18p
run lasted 8938777ns or 0.00894s
kev@nikola:~/penny-adder$
```

# Initialising a Mutex

```
23: pthread_mutex_t mutex;
24:
25: void initialise_mutex() {
26:     int result = pthread_mutex_init(&mutex, NULL);
27:     if(result != 0){
28:         printf("problem initialising mutex\n");
29:         exit(EXIT_FAILURE);
30:     }
31: }
32:
33: int main(){
34:     struct timespec start, finish;
35:     long long int difference;
36:     int account = 0;
37:     int i;
38:
39:     int n = 1000;
40:     int result;
41:
42:     initialise_mutex();
43:     clock_gettime(CLOCK_MONOTONIC, &start);
44:
45:     while(i < n){
46:         pthread_mutex_lock(&mutex);
47:         account++;
48:         pthread_mutex_unlock(&mutex);
49:         i++;
50:     }
51:
52:     clock_gettime(CLOCK_MONOTONIC, &finish);
53:     difference = (finish.tv_sec - start.tv_sec) * 1000000000LL +
54:                 (finish.tv_nsec - start.tv_nsec);
55:     printf("run lasted %lldns or %9.5lfs\n", difference,
56:           (double)difference/1000000000.0);
57:
58:     pthread_mutex_destroy(&mutex);
59: }
```

How long will it take?  
How much money will be accumulated?

# Running the Program

A screenshot of a macOS Terminal window. The title bar at the top shows standard window controls (close, zoom, full screen) and the menu bar with 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows a user named 'kev' at a host named 'nikola' in the directory '~/penny-adder' running a command './penny07'. The output consists of three lines: 'accumulated 101p', 'run lasted 41638320ns or 0.04164s', and a new prompt 'kev@nikola:~/penny-adder\$'.

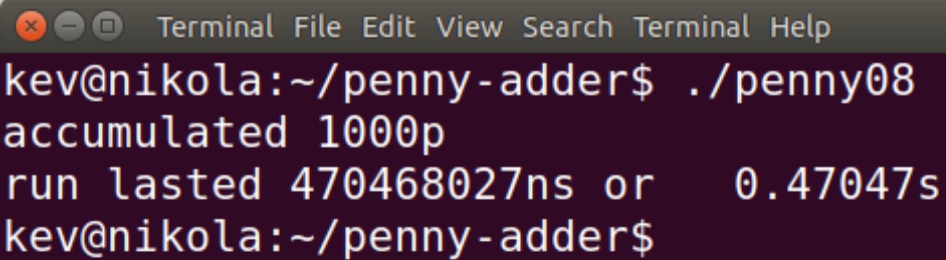
```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny07
accumulated 101p
run lasted 41638320ns or 0.04164s
kev@nikola:~/penny-adder$
```

# Adding Multiple Pennies With a Mutex

```
70: void *add_penny(void *bal) {
71:     int *balance = bal;
72:
73:     int result = pthread_mutex_lock(&mutex);
74:     if(result != 0){
75:         printf("problem locking mutex\n");
76:         exit(EXIT_FAILURE);
77:     }
78:
79:     int b = *balance;
80:     /* cause a short delay */
81:     int i;
82:     for(i=0;i<100000;i++){
83:     }
84:     b = b + 1;
85:     *balance = b;
86:
87:     result = pthread_mutex_unlock(&mutex);
88:     if(result != 0){
89:         printf("problem unlocking mutex\n");
90:     }
```

How long will it take?  
How much money will be accumulated?

# Running the Program

A screenshot of a macOS Terminal window. The title bar at the top shows standard window controls (close, zoom, full screen) and the menu bar with 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows a user named 'kev' at a host named 'nikola' in the directory '~/penny-adder' running a program called './penny08'. The program outputs 'accumulated 1000p' and 'run lasted 470468027ns or 0.47047s'.

```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny08
accumulated 1000p
run lasted 470468027ns or 0.47047s
kev@nikola:~/penny-adder$
```

# A Mutex Combined With A Delay

```
70: void *add_penny(void *bal) {
71:     int *balance = bal;
72:     printf("one\n");
73:
74:     int result = pthread_mutex_lock(&mutex);
75:     if(result != 0){
76:         printf("problem locking mutex\n");
77:         exit(EXIT_FAILURE);
78:     }
79:
80:     printf("two\n");
81:
82:     int b = *balance;
83:     /* cause a substantial delay */
84:     usleep(1000000);
85:     b = b + 1;
86:     *balance = b;
87:
88:     printf("three\n");
89:
90:     result = pthread_mutex_unlock(&mutex);
91:     if(result != 0){
92:         printf("problem unlocking mutex\n");
93:         exit(EXIT_FAILURE);
94:     }
95: }
```

How long will it take?  
How much money will be accumulated?



# Running the Program

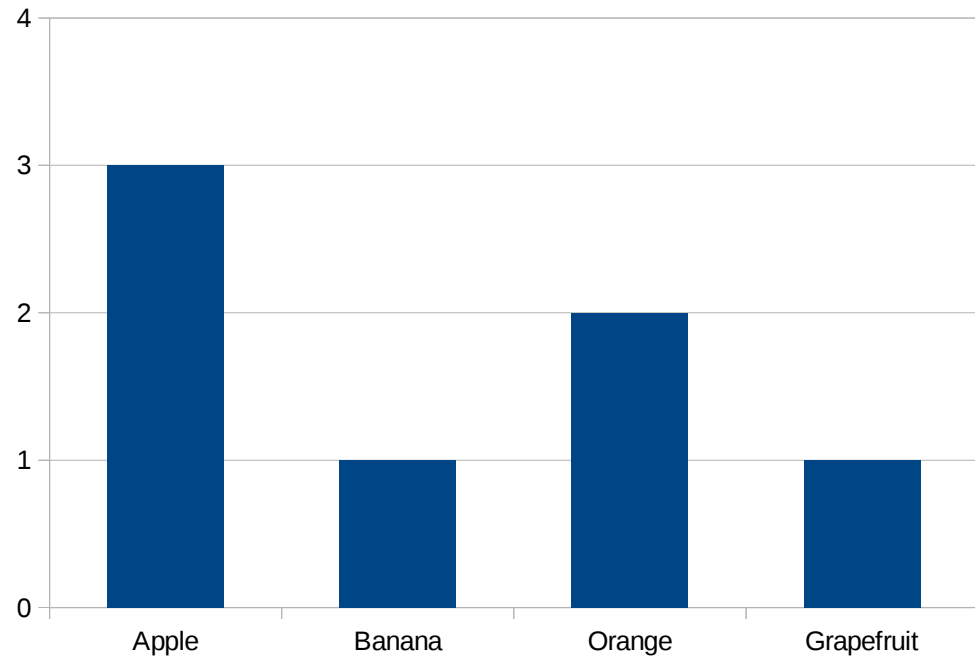
```
Terminal File Edit View Search Terminal Help
kev@nikola:~/penny-adder$ ./penny09
one
two
one
one
one
one
one
one
one
one
one
one
three
two
three
two
three
two
three
two
three
two
three
two
```

```
Terminal File Edit View Search Terminal Help
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 10002883333ns or 10.00288s
kev@nikola:~/penny-adder$
```

# Histograms

Apple  
Apple  
Orange  
Banana  
Apple  
Orange  
Grapefruit

Apple	3
Banana	1
Orange	2
Grapefruit	1



# time\_diff.c

```
16 int time_difference(struct timespec *start, struct timespec *finish,
17                     long long int *difference) {
18     long long int ds = finish->tv_sec - start->tv_sec;
19     long long int dn = finish->tv_nsec - start->tv_nsec;
20
21     if(dn < 0 ) {
22         ds--;
23         dn += 1000000000;
24     }
25     *difference = ds * 1000000000 + dn;
26     return !(*difference > 0);
27 }
28
29
30 void capture_start_time(struct timespec *start){
31     if ((clock_gettime(CLOCK_MONOTONIC, start)) != 0) {
32         fprintf(stderr, "start time could not be set\n");
33         exit(EXIT_FAILURE);
34     }
35 }
36
37 void capture_finish_time(struct timespec *finish){
38     if (clock_gettime(CLOCK_MONOTONIC, finish) != 0) {
39         fprintf(stderr, "finish time could not be set\n");
40         exit(EXIT_FAILURE);
41     }
42 }
```

# histogram\_common.c

```
89 int n_records = 100000;
90 unsigned char data[] = {
91     215, 100, 200, 204, 233, 50, 85, 196, 71, 141, 122, 160, 93, 131, 243, 234, 162, 183, 36, 155,
92     4, 62, 35, 205, 40, 102, 33, 27, 255, 55, 131, 214, 156, 75, 163, 134, 126, 249, 74, 197,
93     134, 197, 102, 228, 72, 90, 206, 235, 17, 243, 134, 22, 49, 169, 227, 89, 16, 5, 117, 16,
94     60, 248, 230, 217, 68, 138, 96, 194, 131, 170, 136, 10, 112, 238, 238, 184, 72, 189, 163, 90,
95     176, 42, 112, 225, 212, 84, 58, 228, 89, 175, 244, 150, 168, 219, 112, 236, 101, 208, 175, 233,
96     123, 55, 243, 235, 37, 225, 164, 110, 158, 71, 201, 78, 114, 57, 48, 70, 142, 106, 43, 232,
97     26, 32, 126, 194, 252, 239, 175, 98, 191, 94, 75, 59, 149, 62, 39, 187, 32, 203, 42, 190,
98     19, 243, 13, 133, 45, 61, 204, 187, 168, 247, 163, 194, 23, 34, 133, 20, 17, 52, 118, 209,
99     146, 193, 13, 40, 255, 52, 227, 32, 255, 13, 222, 18, 1, 236, 152, 46, 41, 100, 233, 209,
100    91, 141, 148, 115, 175, 25, 135, 193, 77, 254, 147, 224, 191, 161, 9, 191, 213, 236, 223, 212,
101    250, 190, 231, 251, 170, 127, 41, 212, 227, 19, 166, 63, 161, 58, 179, 81, 84, 59, 18, 162,
102    57, 166, 130, 248, 71, 139, 184, 28, 120, 151, 241, 115, 86, 217, 111, 0, 88, 153, 213, 59,
103    172, 123, 123, 78, 182, 46, 159, 10, 105, 178, 172, 163, 88, 47, 155, 160, 187, 84, 189, 51,
104    235, 175, 167, 65, 136, 22, 66, 224, 175, 23, 28, 92, 147, 151, 170, 73, 198, 73, 84, 48,
105    251, 0, 211, 84, 48, 111, 245, 235, 195, 178, 31, 175, 98, 198, 241, 234, 220, 52, 203, 140,
106    76, 231, 232, 223, 127, 147, 41, 70, 221, 126, 118, 217, 126, 74, 46, 175, 186, 35, 154, 126,
107    214, 185, 45, 56, 127, 31, 35, 92, 83, 238, 232, 159, 214, 209, 126, 85, 100, 168, 155, 66,
108    38, 18, 27, 165, 93, 73, 84, 23, 109, 239, 149, 67, 168, 195, 124, 40, 226, 160, 132, 53,
109    142, 109, 212, 100, 62, 83, 186, 163, 252, 86, 229, 34, 105, 1, 200, 198, 75, 29, 221, 184,
110    12, 114, 252, 181, 53, 121, 221, 24, 25, 98, 77, 168, 207, 33, 13, 13, 117, 199, 177, 113,
111    30, 150, 148, 135, 152, 92, 77, 227, 122, 43, 156, 134, 158, 152, 59, 212, 17, 25, 236, 43,
112    123, 57, 211, 74, 91, 224, 88, 208, 168, 9, 65, 199, 160, 214, 78, 56, 50, 156, 28, 172,
113    200, 184, 51, 102, 80, 111, 59, 98, 136, 39, 142, 3, 97, 97, 78, 188, 66, 166, 141, 235,
091 int expected_results[] = {
092     404, 389, 376, 394, 376, 342, 364, 364, 383, 396,
093     412, 409, 394, 409, 405, 383, 379, 401, 377, 400,
094     383, 410, 386, 383, 418, 416, 406, 349, 390, 388,
095     393, 372, 386, 386, 400, 384, 404, 355, 400, 361,
096     398, 371, 389, 383, 406, 414, 364, 389, 418, 391,
097     404, 396, 390, 397, 375, 389, 387, 392, 368, 430,
098     407, 387, 380, 380, 383, 352, 386, 413, 435, 413,
099     358, 453, 436, 409, 419, 393, 423, 398, 407, 372,
100     399, 353, 370, 389, 399, 376, 395, 439, 412, 379,
```

# histogram\_common.c

```
10 /*
11  This function clears the histogram bins.
12 */
13
14 void clear_bins(int *bins) {
15     int i;
16     for(i=0;i<256;i++){
17         bins[i]=0;
18     }
19 }
20
```

# histogram\_common.c

```
21 /*
22  This function displays the histogram on the screen.
23 */
24
25 void output_results(int *bins) {
26     int i, j, k;
27     printf("\nresults\n=====\n");
28     for(i=0;i<32;i++){
29         for(j=0;j<8;j++){
30             k = (8 * i) + j;
31             printf("[%3d:%5d]", k, bins[k]);
32         }
33         printf("\n");
34     }
35 }
36
```

# histogram\_common.c

```
38  This function verifies that the total number of records counted in the  
39  histogram bins is equal to the number of records that should have been  
40  processed.  
41  */  
42  
43 void verify_correct_number_of_records_was_processed(int *bins) {  
44     int i;  
45     int count = 0;  
46     for(i=0;i<256;i++){  
47         count += bins[i];  
48     }  
49  
50     printf("\n%d records were found in bins\n", count);  
51 }  
--
```

# histogram\_common.c

```
54  This function checks that the histogram computed is equal to the  
55  known results.  
56  */  
57  
58  void verify_results(int *expected, int *actual) {  
59      int i;  
60      int error_count = 0;  
61  
62      for(i=0;i<256;i++){  
63          if(actual[i] != expected[i]){  
64              error_count++;  
65          }  
66      }  
67  
68      if(!error_count) {  
69          printf("\nresults verified okay\n");  
70      } else {  
71          printf("\n%d errors were found\n", error_count);  
72      }  
73 }  
..
```



# histogram00

```
24 int main() {
25     struct timespec start_time, finish_time;
26     long long int time_elapsed;
27     int bins[256];
28
29     capture_start_time(&start_time);
30
31     clear_bins(bins);
32     calculate_histogram(bins);
33
34     capture_finish_time(&finish_time);
35
36     output_results(bins);
37     verify_correct_number_of_records_was_processed(bins);
38     verify_results(expected_results, bins);
39
40     time_difference(&start_time, &finish_time, &time_elapsed);
41
42     printf("run took %0.9lfs\n", (time_elapsed/1.0e9));
43
44     return 0;
45 }
```

# histogram00

```
16  
17 void calculate_histogram(int *bins) {  
18     int i;  
19     for(i=0;i<n_records;i++){  
20         bins[data[i]]++;  
21     }  
22 }  
23 }
```

```
kevan@aaargh:~/prep/week04/src/histogram$ ./histogram00
```

```
results
```

```
=====
```

```
[ 0: 404][ 1: 389][ 2: 376][ 3: 394][ 4: 376][ 5: 342][ 6: 364][ 7: 364]
[ 8: 383][ 9: 396][10: 412][11: 409][12: 394][13: 409][14: 405][15: 383]
[16: 379][17: 401][18: 377][19: 400][20: 383][21: 410][22: 386][23: 383]
[24: 418][25: 416][26: 406][27: 349][28: 390][29: 388][30: 393][31: 372]
[32: 386][33: 386][34: 400][35: 384][36: 404][37: 355][38: 400][39: 361]
[40: 398][41: 371][42: 389][43: 383][44: 406][45: 414][46: 364][47: 389]
[48: 418][49: 391][50: 404][51: 396][52: 390][53: 397][54: 375][55: 389]
[56: 387][57: 392][58: 368][59: 430][60: 407][61: 387][62: 380][63: 380]
[64: 383][65: 352][66: 386][67: 413][68: 435][69: 413][70: 358][71: 453]
[72: 436][73: 409][74: 419][75: 393][76: 423][77: 398][78: 407][79: 372]
[80: 399][81: 353][82: 370][83: 389][84: 399][85: 376][86: 395][87: 439]
[88: 412][89: 379][90: 404][91: 374][92: 392][93: 393][94: 366][95: 377]
[96: 374][97: 395][98: 402][99: 380][100: 422][101: 407][102: 379][103: 398]
[104: 376][105: 410][106: 376][107: 392][108: 374][109: 409][110: 415][111: 382]
[112: 411][113: 398][114: 379][115: 385][116: 383][117: 374][118: 421][119: 371]
[120: 359][121: 403][122: 373][123: 396][124: 365][125: 365][126: 382][127: 383]
[128: 352][129: 399][130: 367][131: 439][132: 401][133: 418][134: 407][135: 403]
[136: 392][137: 373][138: 385][139: 374][140: 389][141: 365][142: 414][143: 415]
[144: 360][145: 384][146: 387][147: 381][148: 400][149: 410][150: 400][151: 406]
[152: 385][153: 395][154: 373][155: 381][156: 419][157: 362][158: 383][159: 399]
[160: 424][161: 379][162: 394][163: 401][164: 371][165: 426][166: 376][167: 375]
[168: 383][169: 370][170: 405][171: 402][172: 372][173: 404][174: 364][175: 419]
[176: 390][177: 376][178: 368][179: 405][180: 393][181: 386][182: 402][183: 393]
[184: 420][185: 388][186: 380][187: 364][188: 412][189: 383][190: 411][191: 357]
[192: 412][193: 377][194: 346][195: 389][196: 380][197: 371][198: 393][199: 408]
[200: 386][201: 425][202: 392][203: 338][204: 373][205: 382][206: 380][207: 365]
[208: 379][209: 394][210: 379][211: 378][212: 415][213: 394][214: 352][215: 378]
[216: 417][217: 403][218: 407][219: 388][220: 390][221: 433][222: 352][223: 394]
[224: 398][225: 407][226: 397][227: 409][228: 419][229: 378][230: 387][231: 359]
[232: 406][233: 384][234: 403][235: 385][236: 411][237: 418][238: 408][239: 371]
[240: 384][241: 386][242: 392][243: 422][244: 377][245: 399][246: 364][247: 381]
[248: 362][249: 379][250: 393][251: 383][252: 381][253: 400][254: 434][255: 404]
```

```
100000 records were found in bins
```

```
results verified okay
```

```
run took 0.000511236s
```

# histogram01 – Single Thread

```
19 typedef struct {
20     int *bins;
21     unsigned char *data;
22     int start_record;
23     int n_records;
24 } param_t;
25
26 void *thread_function(param_t *params){
27     int i, j;
28
29     for(i=params->start_record;i<params->start_record + params->n_records;i++){
30         j = data[i];
31         params->bins[j]++;
32     }
33 }
--
```

```

35 int main() {
36     struct timespec start_time, finish_time;
37     long long int time_elapsed;
38
39     capture_start_time(&start_time);
40
41     int *bins = malloc(sizeof(int) * 256);
42     clear_bins(bins);
43
44     param_t params;
45     params.bins = bins;
46     params.data = data;
47     params.n_records = n_records;
48     params.start_record = 0;
49
50     void *thread_result;
51     pthread_t t;
52     pthread_create(&t, NULL, thread_function, &params);
53     pthread_join(t, &thread_result);
54     capture_finish_time(&finish_time);
55
56     output_results(bins);
57     verify_correct_number_of_records_was_processed(bins);
58     verify_results(expected_results, bins);
59
60     if(time_difference(&start_time, &finish_time, &time_elapsed) != 0){
61         fprintf(stderr, "error: start time is after finish time\n");
62         return EXIT_FAILURE;
63     }
64
65     printf("run took %0.9lfs\n", (time_elapsed/1.0e9));
66
67     return 0;
68 }

```

```
100000 records were found in bins  
results verified okay  
run took 0.000846368s  
kevan@aaargh:~/prep/week04/src/histogram$
```

# histogram02 – two threads

```
43 param_t params1, params2;
44 params1.bins = bins;
45 params1.data = data;
46 params1.n_records = n_records/2;
47 params1.start_record = 0;
48
49 params2.bins = bins;
50 params2.data = data;
51 params2.n_records = n_records/2;
52 params2.start_record = n_records/2;
53
54 void *thread_result;
55
56 pthread_t t1, t2;
57 pthread_create(&t1, NULL, thread_function, &params1);
58 pthread_create(&t2, NULL, thread_function, &params2);
59 pthread_join(t1, &thread_result);
60 pthread_join(t2, &thread_result);
61 capture_finish_time(&finish_time);
62
```

```
96595 records were found in bins
```

```
256 errors were found
```

```
run took 0.001707159s
```



# histogram04 – four threads

```
43 param_t params1, params2, params3, params4;
44 params1.bins = bins;
45 params1.data = data;
46 params1.n_records = n_records/4;
47 params1.start_record = 0;
48
49 params2.bins = bins;
50 params2.data = data;
51 params2.n_records = n_records/4;
52 params2.start_record = n_records/4;
53
54 params3.bins = bins;
55 params3.data = data;
56 params3.n_records = n_records/4;
57 params3.start_record = n_records/2;
58
59 params4.bins = bins;
60 params4.data = data;
61 params4.n_records = n_records/4;
62 params4.start_record = 3*n_records/4;
63
64 void *thread_result;
65 pthread_t t1, t2, t3, t4;
66 pthread_create(&t1, NULL, thread_function, &params1);
67 pthread_create(&t2, NULL, thread_function, &params2);
68 pthread_create(&t3, NULL, thread_function, &params3);
69 pthread_create(&t4, NULL, thread_function, &params4);
70 pthread_join(t1, &thread_result);
71 pthread_join(t2, &thread_result);
72 pthread_join(t3, &thread_result);
73 pthread_join(t4, &thread_result);
```

```
92388 records were found in bins  
256 errors were found  
run took 0.001359008s
```

# histogram04b – Mutex to the Rescue

```
26 pthread_mutex_t mutex;
27
28 void *thread_function(param_t *params){
29     int i, j;
30
31
32     for(i=params->start_record;i<params->start_record + params->n_records;i++){
33         pthread_mutex_lock(&mutex);
34         j = data[i];
35         params->bins[j]++;
36         pthread_mutex_unlock(&mutex);
37     }
38 }
```

```
100000 records were found in bins  
results verified okay  
run took 0.035905188s
```

# histogram04c – Independent Histograms

```
42
43  int *bins1 = malloc(sizeof(int) * 256);
44  int *bins2 = malloc(sizeof(int) * 256);
45  int *bins3 = malloc(sizeof(int) * 256);
46  int *bins4 = malloc(sizeof(int) * 256);
47  clear_bins(bins1);
48  clear_bins(bins2);
49  clear_bins(bins3);
50  clear_bins(bins4);
```

# histogram04c – Independent Histograms

```
52  param_t params1, params2, params3, params4;
53  params1.bins = bins1;
54  params1.data = data;
55  params1.n_records = n_records/4;
56  params1.start_record = 0;
57
58  params2.bins = bins2;
59  params2.data = data;
60  params2.n_records = n_records/4;
61  params2.start_record = n_records/4;
62
63  params3.bins = bins3;
64  params3.data = data;
65  params3.n_records = n_records/4;
66  params3.start_record = n_records/2;
67
68  params4.bins = bins4;
69  params4.data = data;
70  params4.n_records = n_records/4;
71  params4.start_record = 3*n_records/4;
72
```

# histogram04c – Independent Histograms

```
74
75  pthread_create(&t1, NULL, thread_function, &params1);
76  pthread_create(&t2, NULL, thread_function, &params2);
77  pthread_create(&t3, NULL, thread_function, &params3);
78  pthread_create(&t4, NULL, thread_function, &params4);
79  pthread_join(t1, NULL);
80  pthread_join(t2, NULL);
81  pthread_join(t3, NULL);
82  pthread_join(t4, NULL);
83
84  for(i=0;i<256;i++){
85      bins1[i] = bins1[i] + bins2[i] + bins3[i] + bins4[i];
86  }
87
```

```
100000 records were found in bins  
results verified okay  
run took 0.000622621s
```



# Summary

- We have learnt about race conditions and interference.
  - Multiple threads working on the same data can produce unpredictable results.
- Mutual exclusion (mutex) can protect a critical section.
  - Good – it can avoid data corruption
  - Bad – creates bottlenecks
- Clever algorithm design can avoid mutexes

# Next

- Today we saw issues with multithreading using just 4 threads
- Next week we will see what happens when we try to use thousands of threads to create a histogram of the number of occurrences in “War and Peace”
  - When there is a critical section accessible only by one thread, there can be serious consequences if 1 thread holds back 1000s of others.