

between you and your classroom
neighbors, how many different languages
do y'all speak?

CS 340

Threading

clicker.cs.illinois.edu

Q1

~Code~
340



Updates

1. MP5 Allocator Steps 1-6 due next tuesday

2. HW 5 out today

HW 4-5

Agenda

1. Processes Review

~~2.~~ Threading

~~3.~~ Synchronization in programming

Review

- A process is ... *running instance of a program*
- A single core CPU can run ... *1 process at a time*
- a million inst. a second
- On a single core CPU, processes run ...
Concurrently
- The OS ... *schedules and loads processes in*
and out of the CPU

What prints?

main.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 unsigned int calc(int n){
5     unsigned int ret = 1;
6     for(int i = n, i > 0; i--){
7         ret = i * ret;
8     }
9     return ret;
10 }
11
12 int main(int argc, char *argv[]) {
13     sleep(3); // Pauses for 3 seconds
14     int intValue = argv[1][0] - '0';
15     printf("%i", calc(intValue));
16 }
```

4.3.2 = 24

gcc main.c
./a.out 4

→ a.out

argv[1] → "4"

clicker.cs.illinois.edu

Q2

~Code~
340



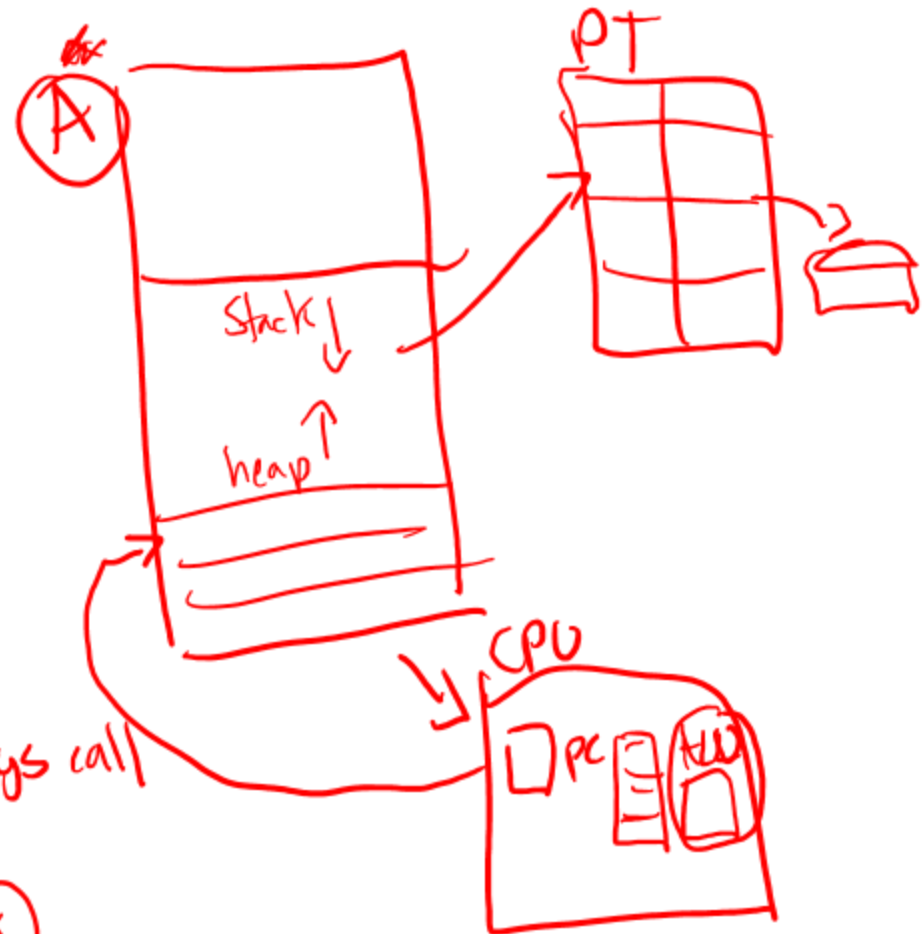
main.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 unsigned int calc(int n){
5     unsigned int ret = 1;
6     → for(int i = n; i > 0; i--){
7         → ret = i * ret;
8     }
9     return ret;
10 }
11
12 → int main(int argc, char *argv[]) {
13     sleep(3); // Pauses for 3 seconds
14     int intValue = argv[1][0] - '0';
15     printf("%i", calc(intValue));
16 }
```

gcc main.c

./a.out 4

→ machine instr.
→ start process (A)



main.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 unsigned int calc(int n){
5     unsigned int ret = 1;
6     for(int i = n; i > 0; i--){
7         ret = i * ret;
8     }
9     return ret;
10 }
11
12 int main(int argc, char *argv[]) {
13     sleep(3); // Pauses for 3 seconds
14     int intValue = argv[1][0] - '0';
15     printf("%i", calc(intValue));
16 }
```

gcc main.c

Ter 1

./a.out 3



Ter 2

./a.out 4



Process Vs. Threads

Process

- own VM
- own PC
- own reg values

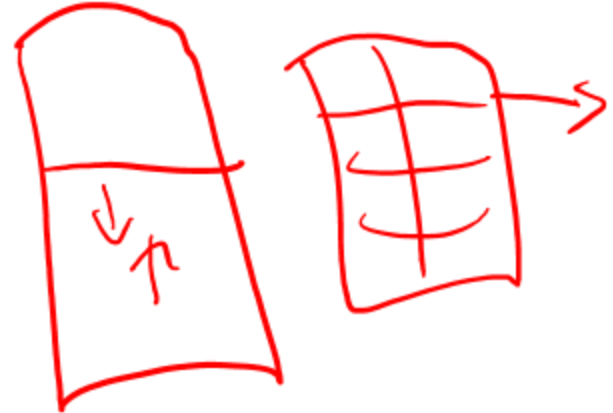
- UI responsiveness
- handle many user requests
- shared computation

Thread

within a process ← 1 thread or more

- Thread
- own PC
 - own reg values

shared VM



Thread Example

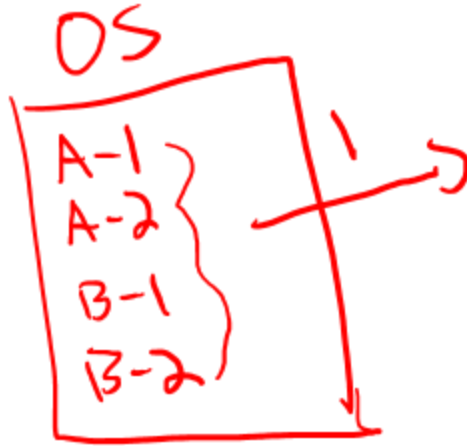
```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

```
gcc main.c
./a.out
```



Scheduling

- Works on a thread level



A1 → A2
faster

A1 → B2

B2 → A2 Slow

Threads
VM → own
stack
frame

```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

Ⓐ ./a.out
Ⓑ ./a.out

processes

What are the max number of threads that could be running concurrently if I run this program from 5 terminals at the same time?

→ 1-2

clicker.cs.illinois.edu

Q3

~Code~
340



```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

10 Threads

How many different page tables would the computer need to store to run this program from 5 terminals concurrently?

clicker.cs.illinois.edu

Q4

~Code~
340



5

0x6BF → PA 0x806

```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

With a 1 core CPU how many processes could I run in parallel?

clicker.cs.illinois.edu

Q5

~Code~
340



```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

1

With a 1 core CPU how many threads could I run in parallel?

1

```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* thread_function(void* arg) {
5     printf("Hello from the new thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     // Create a new thread
13     if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
14         perror("Failed to create thread");
15         return 1;
16     }
17
18     printf("Hello from the main thread!\n");
19
20     // Wait for the new thread to finish
21     pthread_join(thread, NULL);
22
23     printf("Thread has finished.\n");
24 }
```

clicker.cs.illinois.edu

Q6

~Code~
340



Threading Example

```
main.c
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4
5  void *compute(void* arg) {
6      int* ptr = (int*)arg;
7      printf("my value is %i\n", *ptr);
8      return NULL;
9  }
10
11 int main() {
12     pthread_t threads[2];
13     int args[2] = {0, 1};
14     for (int i = 0; i < 2; i++) {
15         if (pthread_create(&threads[i], NULL, compute, &(args[i])) != 0) {
16             perror("Failed to create thread");
17             return 1;
18         }
19     }
20
21     //wait for both threads to finish
22     for (int i = 0; i < 2; i++) {
23         pthread_join(threads[i], NULL);
24     }
25 }
26 }
```

3 threads

here

0, 1

What will this code print?

```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 void *sort(void* arg) {
6     int *ptr = (int*)arg;
7     sleep(*ptr);
8     printf("%i ", *ptr);
9     return NULL;
10 }
11
12 int main() {
13     int arr[6] = {5, 10, 1, 4, 2, 7};
14     pthread_t threads[6];
15
16     for (int i = 0; i < 6; i++) {
17         if (pthread_create(&threads[i], NULL, sort, &(arr[i])) != 0) {
18             perror("Failed to create thread");
19             return 1;
20         }
21     }
22
23     //wait for both threads to finish
24     for (int i = 0; i < 6; i++) {
25         pthread_join(threads[i], NULL);
26     }
27 }
```

Handwritten annotations on the code:

- A red arrow points from the `sleep(*ptr);` line (line 7) to the text `sleep(5)`.
- A red arrow points from the `sleep(*ptr);` line (line 7) to the `5` in the array `{5, 10, 1, 4, 2, 7}`.
- The array `{5, 10, 1, 4, 2, 7}` is enclosed in a red box, with red arrows pointing to each element.
- The `&(arr[i])` expression in the `pthread_create` call (line 17) is enclosed in a red box, with a red arrow pointing to it.

clicker.cs.illinois.edu

Q7

~Code~
340



main.c

```

2 #include <pthread.h>
3 #include <unistd.h>
4
5 int arr[2] = {1, 1};
6 int counter = 0;
7
8 void *compute(void* arg) {
9     for(int i = 0; i < 2; i++){
10         if(arr[i] == 0){
11             printf("changing index %i\n", i);
12             arr[i] = counter;
13             counter++;
14         }
15     }
16     return NULL;
17 }

```

```

18
19 int main() {
20     pthread_t threads[2];
21
22     for (int i = 0; i < 2; i++) {
23         if (pthread_create(&threads[i], NULL, compute, NULL) != 0) {
24             perror("Failed to create thread");
25             return 1;
26         }
27     }
28
29     //wait for both threads to finish
30     for (int i = 0; i < 2; i++) {
31         pthread_join(threads[i], NULL);
32     }
33
34     for (int i = 0; i < 2; i++) {
35         printf("%i ", arr[i]);
36     }
37 }

```

(A)

(B)

race condition

clicker.cs.illinois.edu

Q8

~Code~
340



Could this code print "3 2"

3 2 ?

yes

Synchronization

Idea - take turns accessing critical sections
(share resource)

Mutex - lock - A gets and releases

Mutex

Sharing
resources

indiv.
resources

```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER; // static init
6
7 int arr[2] = {0, 0};
8 int counter = 1;
9
10 void *compute(void* arg) {
11     pthread_mutex_lock(&m);
12     for(int i = 0; i < 2; i++){
13         if(arr[i] == 0){
14             printf("changing index %i\n", i);
15             arr[i] = counter;
16             counter++;
17         }
18     }
19     pthread_mutex_unlock(&m);
20
21     return NULL;
22 }
23
24 int main() {
25     pthread_t threads[2];
26
27     for (int i = 0; i < 2; i++) {
28         if (pthread_create(&threads[i], NULL, compute, NULL) != 0) {
29             perror("Failed to create thread");
30             return 1;
31         }
32     }
33     //wait for both threads to finish
34     for (int i = 0; i < 2; i++) {
35         pthread_join(threads[i], NULL);
36     }
37
38     for (int i = 0; i < 2; i++) {
39         printf("%i ", arr[i]);
40     }
41 }
```

get the lock function

unlock

Synchronization

Idea - take turns
condition variable

conditional_var cv;
Mutex m;

- get_mutex(m) ←

while (complicated - cond_not_met()) {
 → cond_wait(cv, m)
 ↑ ↑

3
↓
have lock [do things
broadcast(cv);
unlock(m);]

Conditional Variable

```
5 pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
6 pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
7
8 int ready = 0;
9 int data = 0;
10
11 void* worker(void* arg) {
12     pthread_mutex_lock(&m);
13     while (!ready) {
14         pthread_cond_wait(&cv, &m);
15     }
16     int value = data;
17     pthread_mutex_unlock(&m);
18     printf("worker got data = %d\n", value);
19     return NULL;
20 }
21
22
23 int main(void) {
24     pthread_t t;
25     pthread_create(&t, NULL, worker, NULL);
26
27     // doing work to produce the data
28     sleep(1);
29
30     pthread_mutex_lock(&m);
31     data = 42;
32     ready = 1;
33     pthread_cond_signal(&cv);
34     pthread_mutex_unlock(&m);
35
36     pthread_join(t, NULL);
37     return 0;
38 }
```

Handwritten annotations in red:

- Arrows pointing to lines 5, 6, 8, and 9.
- Arrows pointing to lines 12, 13, 14, and 15, with the text "give up m" and "wait" written next to them.
- An arrow pointing to line 17, with the text "unlock" written next to it.
- An arrow pointing to line 34, with the text "unlock m" written next to it.

Buffet Example



Do I potentially need a mutex for handling multiple processes running on my computer? Why or why not?

clicker.cs.illinois.edu

Q9

~Code~
340



Don't
Share

VM

NO

Deadlock

(A)



(B)



Alice - gets A → hold
on B

Bob - gets B → Hold
on A

11
~

Deadlock - Dining Philosophers

What does deadlock look like in code?

infinite loop

clicker.cs.illinois.edu

Q10

~Code~
340

