



# [Limited] Direct Execution and Threads II

**CS 240 - The University of Illinois**

Wade Fagen-Ulmschneider

September 21, 2021

# Direct Execution

A photograph of a crowd gathered around a statue of Alma Mater, overlaid with a red filter. The statue is a woman in a long dress, standing on a pedestal. The crowd consists of many people, mostly young men, looking towards the statue. The background shows trees and a building. The text "Direct Execution" is written in large white letters across the center of the image.

# Simple CPU Operation

## CPU Instructions:

Program Counter



<code>movl 4(%esp), %eax</code>
<code>popl 0(%eax)</code>
<code>movl %esp, 4(%eax)</code>
<code>movl %ebx, 8(%eax)</code>
<code>movl %ecx, 12(%eax)</code>
<code>movl %edx, 16(%eax)</code>
<code>movl %esi, 20(%eax)</code>
<code>movl %edi, 24(%eax)</code>
<code>movl %ebp, 28(%eax)</code>
<code>...</code>

# Simple CPU Operation

## CPU Instructions:

Program Counter



<code>movl 4(%esp), %eax</code>
<code>popl 0(%eax)</code>
<code>movl %esp, 4(%eax)</code>
<code>movl %ebx, 8(%eax)</code>
<code>movl %ecx, 12(%eax)</code>
<code>movl %edx, 16(%eax)</code>
<code>movl %esi, 20(%eax)</code>
<code>movl %edi, 24(%eax)</code>
<code>movl %ebp, 28(%eax)</code>
<code>...</code>

# Direct Execution

## OS:

1. Create entry for process
2. Allocate memory for process
3. Load program into memory
4. Set up stack (argv/argc)
5. Clear registers
6. **call** main()
9. Free memory for process
10. Remove process from process list

## Process:

- 
- The diagram illustrates the execution flow of a process. A vertical grey line represents the process boundary. To the right of this line, the process steps are listed. Step 7, 'Run main()', is preceded by a light blue arrow pointing right from the vertical line. Step 8, 'return from main()', is preceded by a light blue arrow pointing left from the vertical line.
7. Run main()
  8. **return** from main()

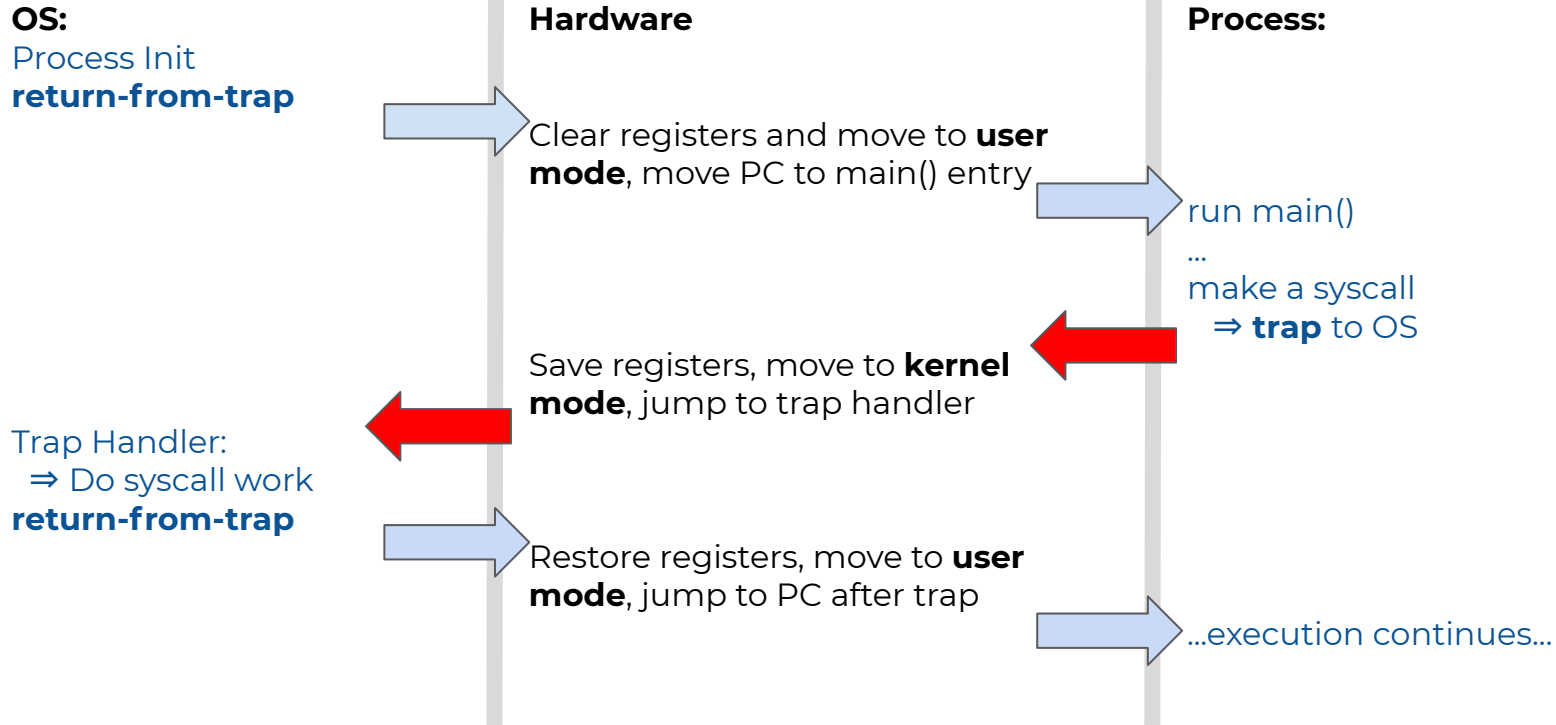
# Direct Execution

Problems?

# Protection Levels

A photograph of a crowd gathered around a statue of Alma Mater, overlaid with a red filter. The statue is the central focus, with the words "ALMA MATER" visible on its base. The background shows a large group of people, some looking towards the statue. The entire image is tinted with a solid red color.

# Limited Direct Execution





# Facilitate Multiple Applications

**Big Idea:** Don't give apps direct access to hardware!

# Protection Levels / “Rings”

Ring 0:

# Protection Levels / “Rings”

Ring 3:

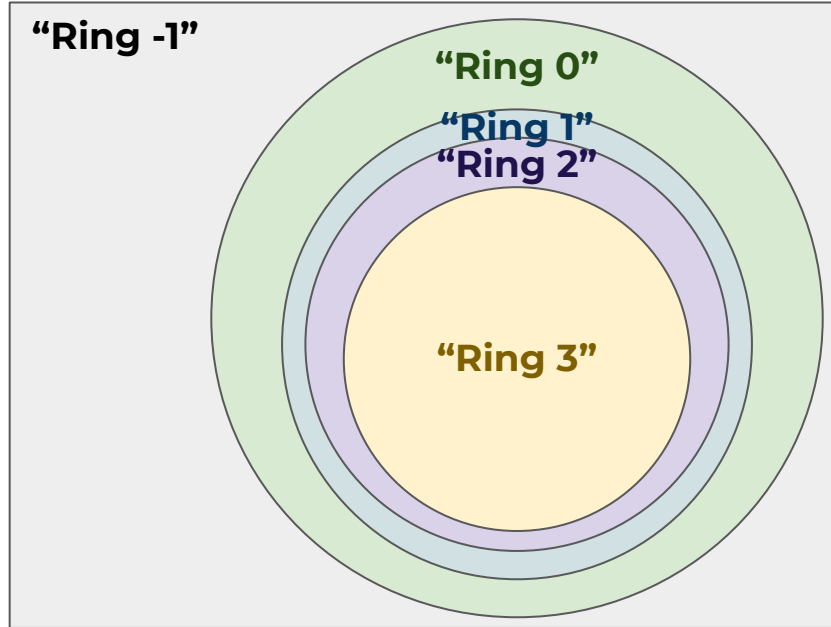
# Protection Levels / “Rings”

Rings 1 and 2:

# Protection Levels / “Rings”

Ring -1:

# Protection Levels / “Rings”



# Trapping to the OS

A photograph of a crowd gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The text "Trapping to the OS" is centered in white. The statue is the central focus, with a crowd of people in the foreground and background. The background shows bare trees, suggesting a winter or late autumn setting. The overall mood is somber or reflective.

# Trap Mechanisms

## #1: System Calls



# Trap Mechanisms

#2:

# Trap Mechanisms

#3:

The background image shows a crowd of people gathered for a ceremony. In the center, a statue of a woman in a long dress stands on a pedestal. The scene is overlaid with a semi-transparent orange filter. The text 'Five State Process/Thread Model' is centered in white, bold font.

# Five State Process/Thread Model



A photograph of a crowd gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The statue is the central focus, standing on a pedestal. The crowd is visible in the foreground and background, though somewhat blurred. The text '08/thread-count.c' is overlaid in white on the image.

**08/thread-count.c**

## thread-count.c

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int ct = 0;
6
7 void *thread_start(void *ptr) {
8     int countTo = *((int *)ptr);
9
10    int i;
11    for (i = 0; i < countTo; i++) {
12        ct = ct + 1;
13    }
14
15    return NULL;
16 }
17
```

```
18 int main(int argc, char *argv[]) {
19     // Parse Command Line:
20     if (argc != 3) {
21         printf("Usage: %s <countTo> <thread count>\n",
argv[0]);
22         return 1;
23     }
24
25     const int countTo = atoi(argv[1]);
26     if (countTo == 0) { printf("Valid `countTo` is
required.\n"); return 1; }
27
28     const int thread_ct = atoi(argv[2]);
29     if (thread_ct == 0) { printf("Valid thread count is
required.\n"); return 1; }
30
```

```
31 // Create threads:
32 int i;
33 pthread_t tid[thread_ct];
34 for (i = 0; i < thread_ct; i++) {
35     pthread_create(&tid[i], NULL,
36                   thread_start, (void *)&countTo);
37 }
38 // Join threads:
39 for (i = 0; i < thread_ct; i++) {
40     pthread_join(tid[i], NULL);
41 }
42
43 // Display result:
44 printf("Final Result: %d\n", ct);
45 return 0;
46 }
```



**Q1:** What do we expect when we run this program?

**Q2:** What is the output of this program when it's running as:

```
./count 100 2
```

**Q3:** What is the output of this program when it's running as:

```
./count 100 16
```

**Q4:** What is the output of this program when it's running as:

```
./count 1000000 2
```

**Q5:** What is the output of this program when it's running as:

```
./count 1000000 16
```

**Q6:** What is going on???