Recitation 1: Multitasking

Kai Mast
## Threads vs. Processes

<table>
<thead>
<tr>
<th></th>
<th>Threads</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How to start?</strong></td>
<td><code>pthread_create()</code></td>
<td><code>fork()</code> (+<code>exec()</code>)</td>
</tr>
<tr>
<td><strong>Own Address Space?</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Can share memory?</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Can execute concurrently?</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

(and other differences not relevant for the following...)
A primer on concurrency

```c
int main() {
    int i = fork();
    if(i == 0)
        printf("I’m the child! \n");
    else
        printf("I’m the parent! \n");
    return 0;
}
```

Two possible outputs:

```
bash:~ > ./a.out
I’m the child
I’m the parent
```

Or

```
bash:~ > ./a.out
I’m the parent
I’m the child
```

Why?
A primer on concurrency

```c
int main() {
    printf("one \n");
    int i = fork();
    if(i == 0)
        printf("two \n")
    return 0;
}
```

Only one possible output:

```
bash:~ > ./a.out
one
two
```

Why?
A visualization of concurrency

Parent

Child

one

fork()

two

Arrow implies “happens before”
Question 2a)

Multiprocessing.
int result = 0;

int main() {
    int i;
    for (i = 0; i < 2; i ++){
        fork();
        result++;
        printf("result = %d\n", result);
    }
    printf("result = %d\n", result);
    return 0;
}
Step 1

Note: This is only one possible schedule
Step 2

Note: This is only one possible schedule
Note: This is only one possible schedule
Question 2b)

Multithreading.
Do not feel threatened by threads

```c
int result = 0;
pthread_t tid[2];

void *inc_result(void *ignore) {
    result++;
    printf("result = %d\n", result);
    fflush(stdout);
    return NULL;
}

int main() {
    for (int i = 0; i < 2; i ++){
        pthread create(&tid[i], NULL, &inc_result, NULL);
        result++;
        printf('result = %d\n', result);
    }
    printf("result = %d\n", result);
    return 0;
}
```

- **result** is a global variable
- Each child-thread will execute this function
- Creates a new thread in the same process
Step 1

Note: This is only one possible schedule
Step 2

Note: This is only one possible schedule