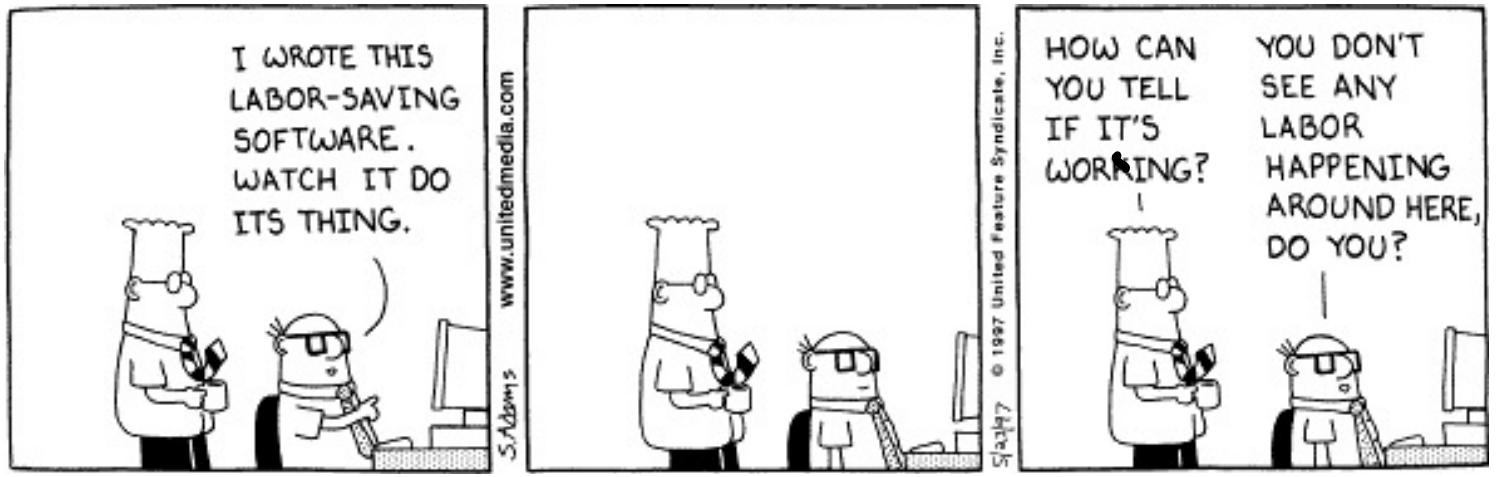
Annotated slides from class (but not many, since tablet wasn't working): PART 2

Week 6: Processes

CS211: Programming and Operating Systems

Thursday, 18 March 2021



CS211
Week 6: Processes

Start of ...

PART 2: Process Creation

In this section, we'll see how to create a process in C using the `fork()` function. Unfortunate, this won't work under Windows/codeblocks. So use one of the online compilers, such as `https://repl.it` or `onlinegdb.com`

Memory = "variables"

A parent process creates children processes, which, in turn create other processes, forming a tree of processes.

After a parent creates a subprocess it may:

- execute[1] ***concurrently*** with the child
  or
- ***wait*** until child terminates before it continues.

The parent may share all, some or none of its resources with the child (resources include memory space, open files, the terminal, etc.)

It is usually the case that the child will share the parent's memory only in the sense that it receives a copy.
The child can then mimic the parents execution, or it might over-write (or "*over-lay*") its memory space with other instructions.

[1] "Execute" in this context means "run" or "preform operations", as in "to execute a plan"

All processes have a unique **Process Identification Number** – ***PID*** for short. If we create a subprocess in a C program using the `fork()` function, a new process is created:

- The new process run concurrently with its parent, unless we instruct the parent to `wait()`.
- The subproc is given a copy of the parents memory space.
- At the time of creation, the two processes are almost identical, except that the `fork()` returns the child's PID to the parent and 0 to the child.

In order to use this function, we must include the *`unistd.h`* header file. This provides various functions including

- `fork()` [Creating new child procs]
- `getpid()` [Returns my PID]
- `getppid()` [Returns my parent's PID]

01Fork.c ~~01Fork.c~~

```
1  // An example of forking a process

3  #include <unistd.h>
   #include <stdio.h>
5  #include <stdlib.h>


   int main(void )
9  {
     int pid1, mypid;

     pid1 = fork();
13   mypid = getpid();

15   printf("I am %d\t", mypid);
     printf("Fork returned %d\n", pid1);
17   return(0);
   }
```

[As of now, two procs are running]

When I compile and run this (e.g., on https://www.onlinegdb.com/) I get something like

```
I am 7791. Fork returned 0
I am 7790. Fork returned 7791
```

[Output from the child]

[Output from parent]

IMPORTANT: *unistd.h* is not included in the installation of `code::blocks` on Windows. Try

- https://www.onlinegdb.com/online_c_compiler
- https://www.jdoodle.com/c-online-compiler
- https://paiza.io/projects/
- https://rextester.com/l/c_online_compiler_gcc
- But not https://www.tutorialspoint.com/compile_c_online.php or http://www.compileonline.com/ or https://www.codechef.com/. Also problematic: https://ideone.co

02Fork2.c

```
// An example of forking two processes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void )
{
  int pid1, pid2, mypid;

  pid1 = fork();
  pid2 = fork();
  mypid = getpid();

  printf("I am %d\t", mypid);
  printf("1st fork returned %d\t", pid1);
  printf("2nd fork returned %d\n", pid2);
  return(0);
}
```

Running that we might get:

```
I am 7802. 1st Fork returned 7803. 2nd Fork returned 7805
I am 7803. 1st Fork returned 0. 2nd Fork returned 7804
I am 7804. 1st Fork returned 0. 2nd Fork returned 0
I am 7805. 1st Fork returned 7803. 2nd Fork returned 0
```

Discuss: Why do we get this output?

We get 4 lines of output because
1. First just the parent is running.
2. It calls pid1=fork(), so now there are two
3. Each of those calls "pid2 = fork()", so now there are 4.

The parent knows the child's PID because it is returned by `fork()`. The child can find out its parent's PID, by using the `getppid()` function:

06ParentsPID.c

```
 6  int main(void )
    {
 8    int pid1;
      pid1 = fork();
10    printf("I am %d\t", getpid());
      printf("fork returned %5d\t", pid1);
12    printf("My parent is %d\n", getppid());
      return(0);
14  }
```

OUTPUT:

```
I'm proc 7825. fork() returned 0. My partent is 7824
I'm proc 7824. fork() returned 7825. My partent is 5394
```

[Child]

[Parent]