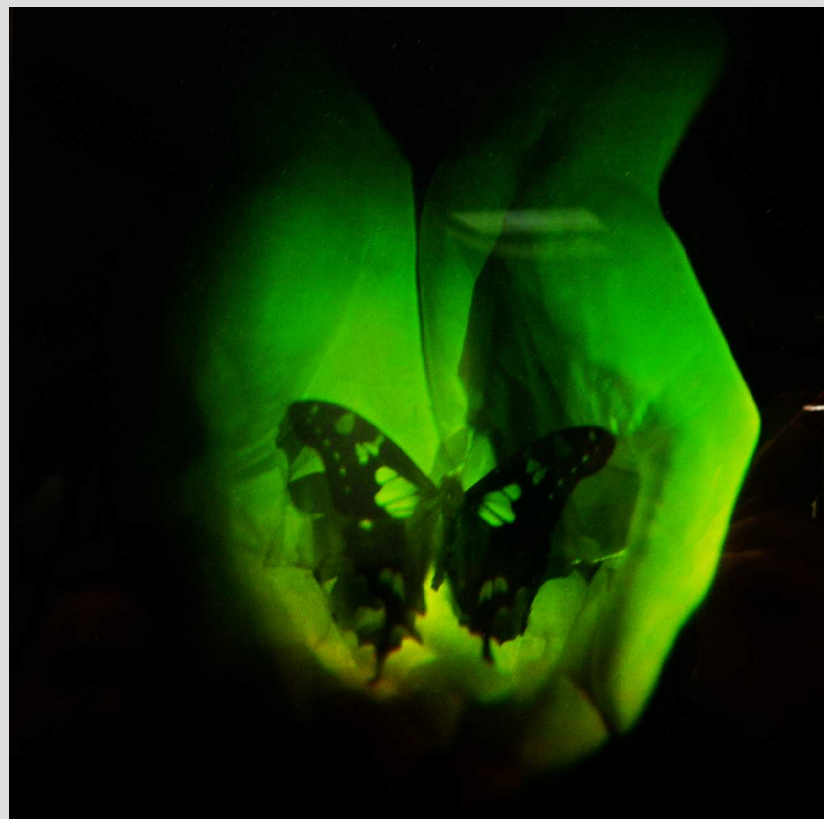


Multiprogramming and You

- Essential Differences between C and C++
- System Call Basics
- Easy Error Handling
- Forking processes
- Obtaining process ID's
- Waiting on processes
- Executing programs
- Finding system information
- Command line parameters



The Lowdown On C

- **All variables must be declared at the top of the program. In particular, the following will not work:**
 - `for(int i = 0; i < 10; i++)`
- Variables are (very!) weakly typed. It is common to recast variables types several times throughout a program.
- C does not have references. You will need to use pointers.
- Pointers are very common. Pointers to pointers are also common. In fact, learn everything you can about pointers.
- C also does not have classes. You can create structs, but these cannot contain any functions.

The Terrible, Terrible C-Strings

By far, the largest source of errors in EECS 338 programs is the improper use of C-strings. C-strings are not objects; they are pointers to the first character in a block of memory. Watch out for the following!

<i>Task</i>	<i>Mistake</i>	<i>Solution</i>
Comparison	<code>if(str1 == str2)</code>	<code>strncmp</code>
Setting one string equal to another	<code>str1 = str2;</code>	<code>strncpy</code>
Concatenation	<code>str1 = str2 + str3;</code>	<code>strncat</code>

When System Calls...

System calls really are not as scary as most people think. I've only ever seen them eat one person, and he deserved it.

Most systems calls are pretty predictable. If you satisfy their demands and they are able to fulfill their tasks, then they will return 0 or a positive value to let you know that they are pleased. If something goes wrong, they will return -1 and possibly eat your computer.

Pierre? perror()?

When something goes wrong with a system call, `perror()` can be used to print an error message to standard error. Note that this function does not cause the program to exit.

`perror()` accepts one argument that serves as a label for the error message. For instance, calling `perror("Yo!")` will cause the system to print something such as:

```
Yo!: Permission denied
```

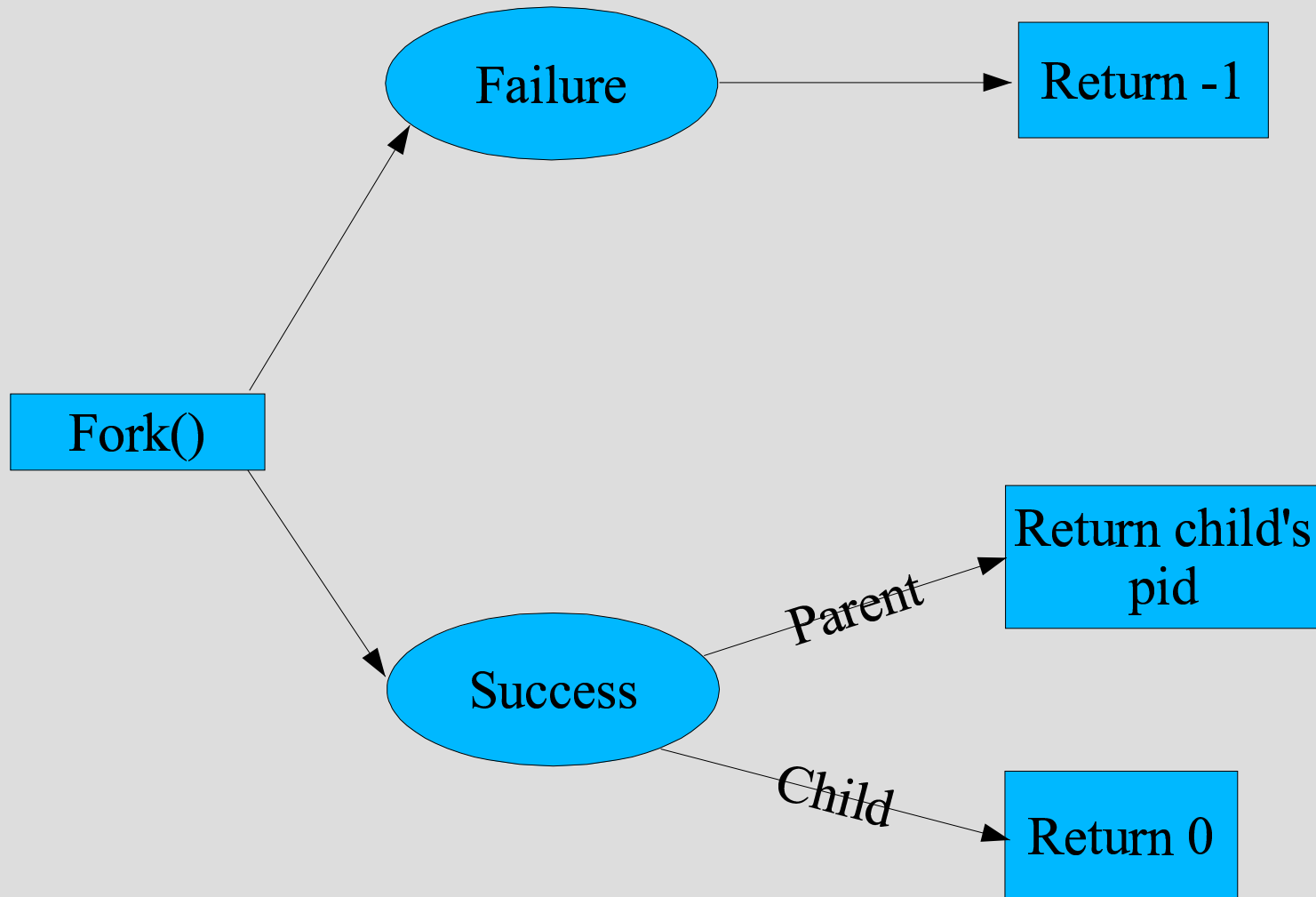


Tip: Easy Error Checking

Since Unix system calls behave in such a predictable manner, one can easily streamline error checking by creating “pipelined” error functions

```
int Error(int result)
{
    if(result == -1)
    {
        perror("Error");
        exit(EXIT_FAILURE);
    }
    return result;
}
.
.
.
pid_t id = blError(fork());
```

What the fork()?



Obtaining Process ID Info

`pid_t getpid(void)`: Returns the id of the current process

`pid_t getppid(void)`: Returns the id of the parent process

`pid_t getpgid(0)`: Returns the group id of the current process

Timing Processes

`unsigned int sleep(unsigned int seconds)`

Number of seconds remaining
before sleep was interrupted.

Number of seconds that you want
the process to sleep.

`pid_t wait(int *status)`

The id of the process that
terminated.

The status of the process
that just terminated.

The exec()-utive Suite

`execXX(...)`



`{l, v}`

`l`: Command and its parameters are listed in the program

`v`: Command and its parameters are contained in a vector

`{p, (blank)}`

`p`: The execution will search the PATH

: Paths must be explicit or relative to current directory

Exiting a Process

```
void exit(int status)
```



The process will terminate
and return the value of
status

Finding System Information

`cuserid(NULL)`: Returns the current username

`gethostname(char *buffer, int len)`: This will store the hostname of the computer in buffer assuming that len is long enough. This can be assured by setting len to `MAXHOSTNAMELEN` as defined in `<sys/params.h>`

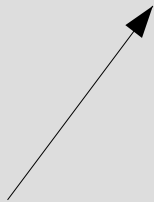
`getcwd(char *buffer, int len)`: Same idea as above, except this function returns the current working directory of the process. Set len to `MAXPATHLEN`.

Command Line Parameters


Syntax:

```
int main(int argc, char **argv)
{ ... }
```

The total number
of parameters
passed to the
program



The strings that
form the parameters



Example:

```
$ ./ls -l /mnt/cdrom
```

```
argc = 3
```

```
argv[0] = "./ls"
```

```
argv[1] = "-l"
```

```
argv[2] = "/mnt/cdrom"
```

