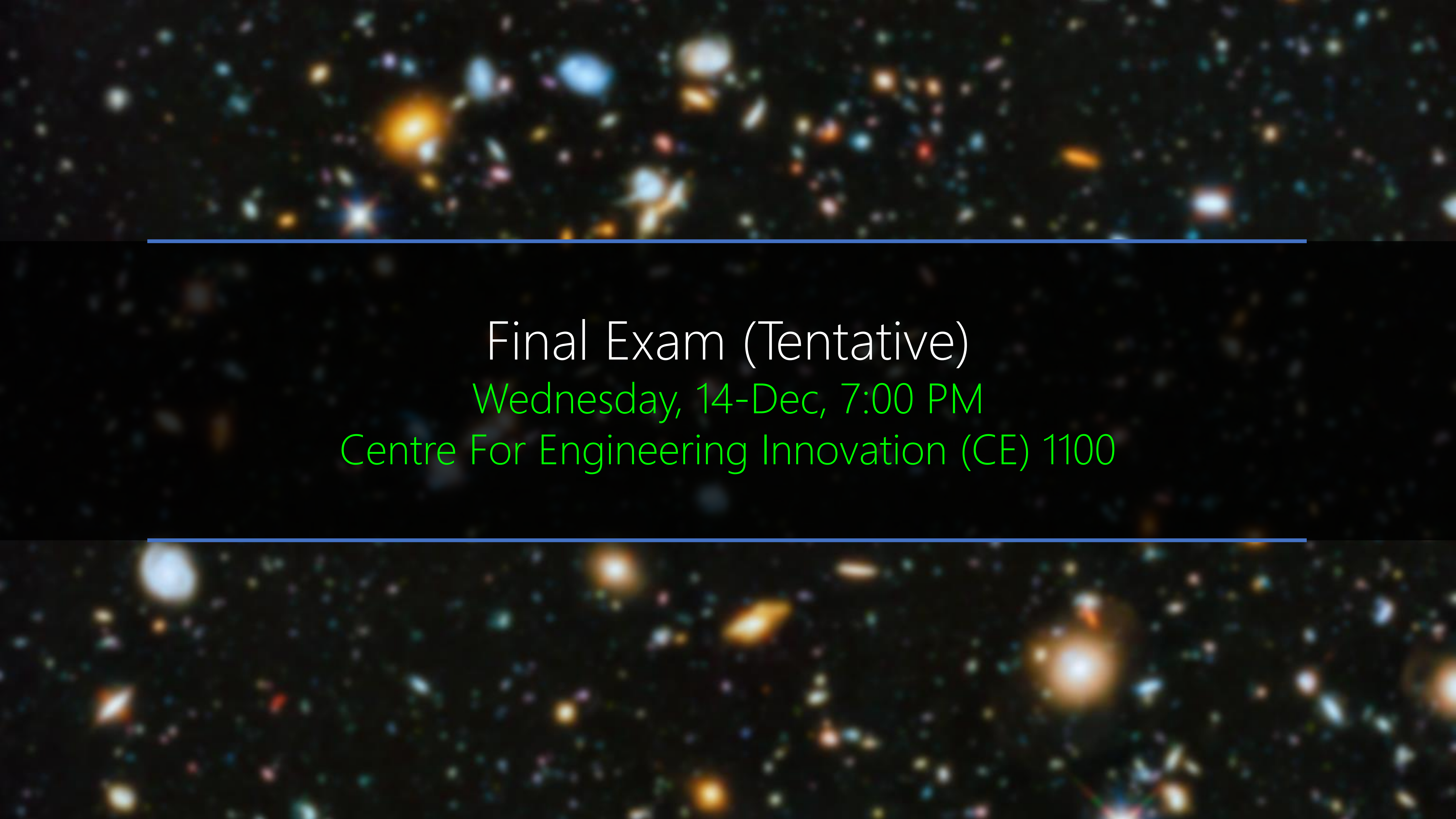
The Prestige (2006) - Christopher Nolan



Final Exam (Tentative)

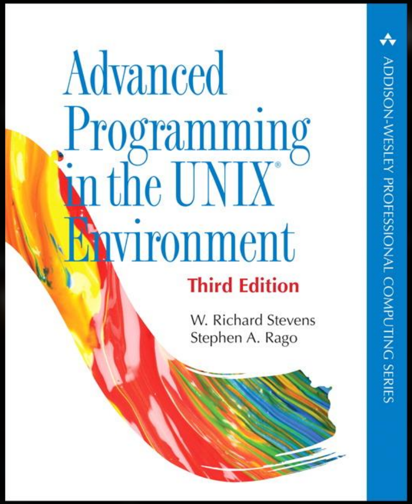
Wednesday, 14-Dec, 7:00 PM

Centre For Engineering Innovation (CE) 1100

Process Manager

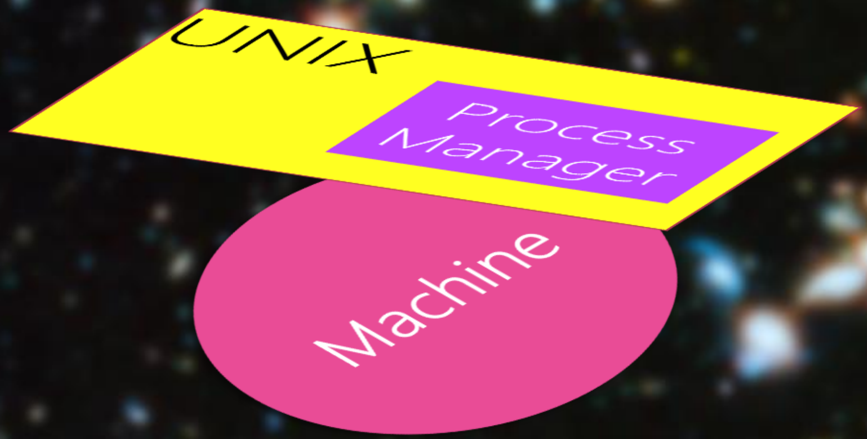
aka. Process Control

- Review: Program → Process → Run → Terminate
- Multiprocessing ✗
  - HALT
  - Context Switch
  - Fork


Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Chapter 07: Process Environment

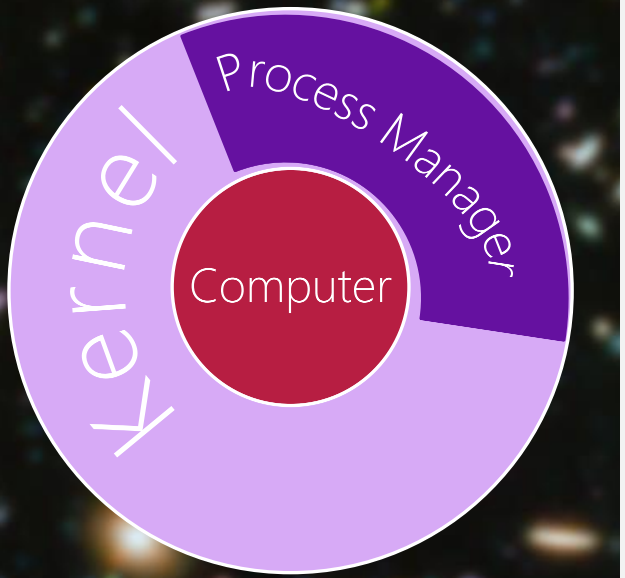
Chapter 08: Process Control



UNIX

Process Manager

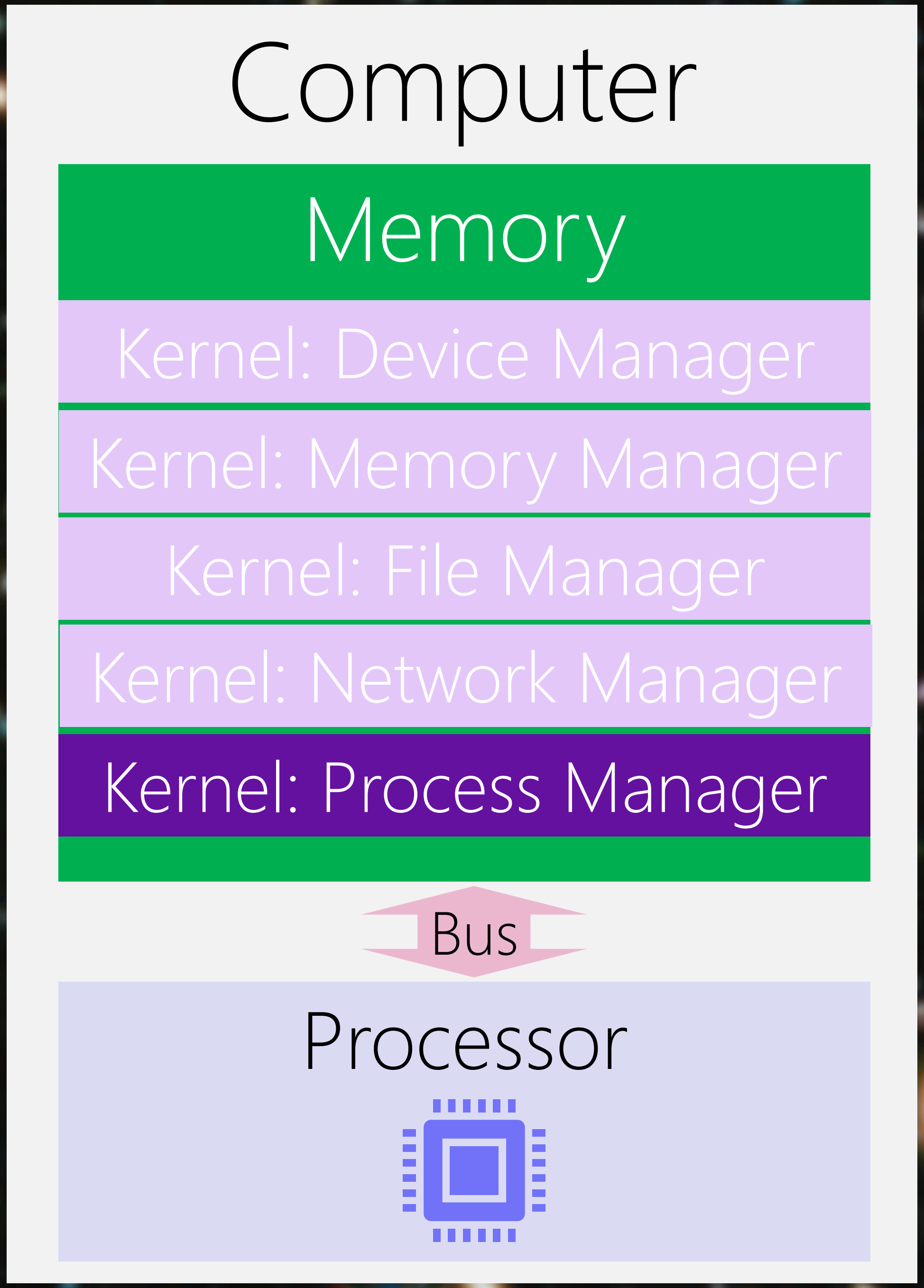
Machine



Kernel

Process Manager

Computer



Computer

Memory

Kernel: Device Manager

Kernel: Memory Manager

Kernel: File Manager

Kernel: Network Manager

Kernel: Process Manager

Bus

Processor

Program → Process → Run → Terminate

Program → Process → Run → Terminate

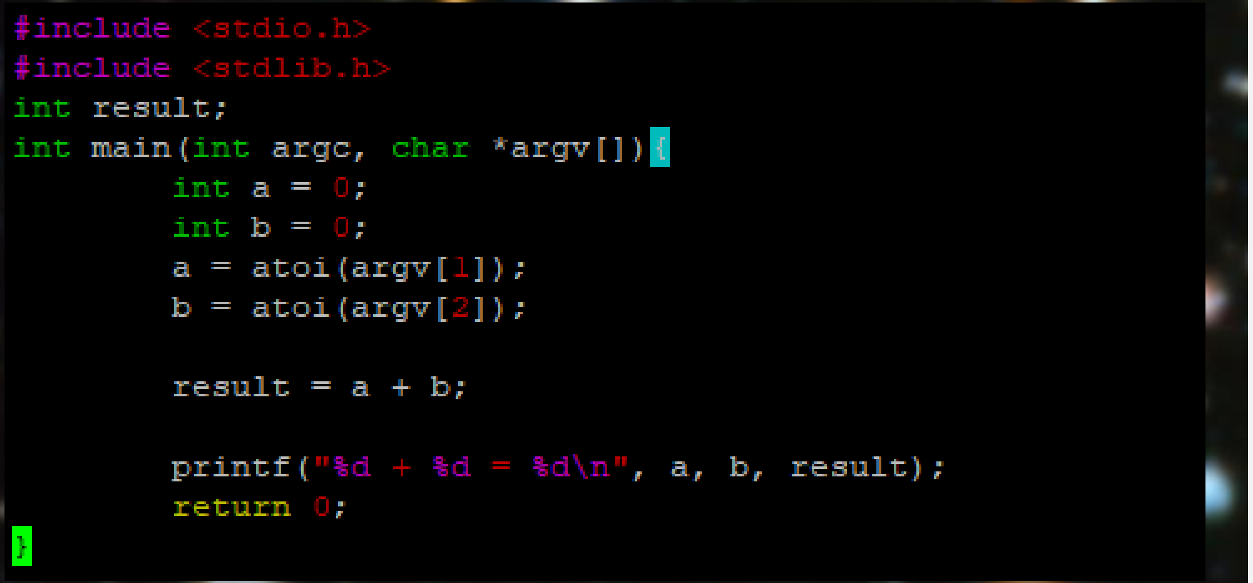
```
void main(int argc, char *argv[])
int  main(int argc, char *argv[])
```

argo

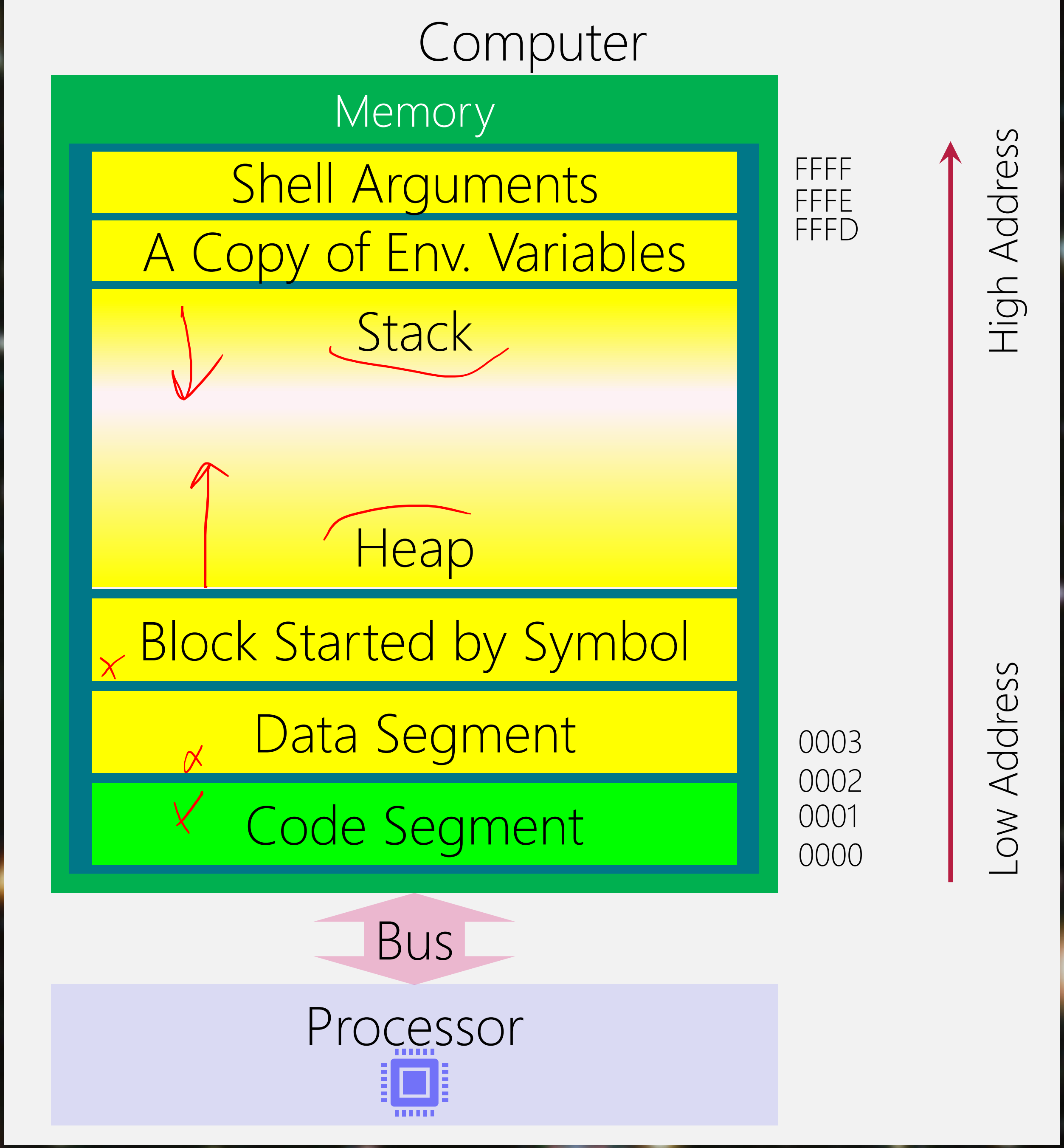
```
shell$ ./program arg1  arg2 arg3 ....
```

Into the Wild (2007) - Sean Penn

Program → Process → Run → Terminate



```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```



Computer

Memory

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

Data Segment

Code Segment

Bus

Processor

FFFF
FFFE
FFFD

0003
0002
0001
0000

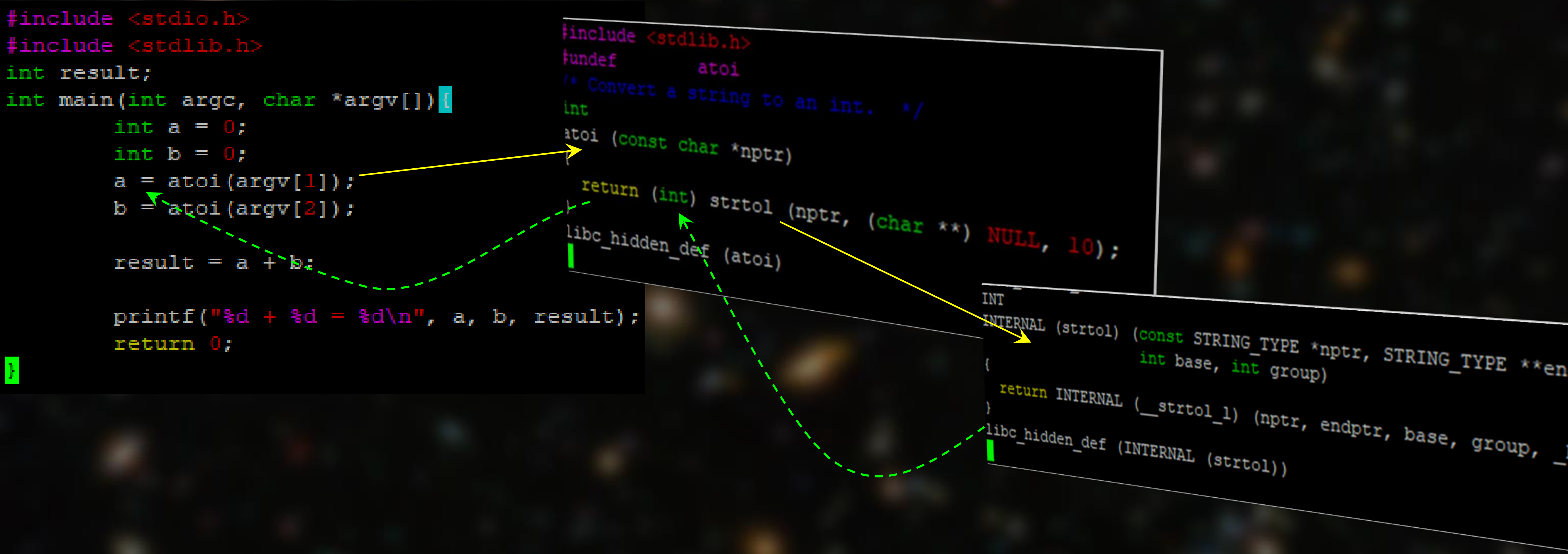
High Address

Low Address

Into the Wild (2007) - Sean Penn

Stack

Functions Arguments, Local Variables, Return Address (runtime)



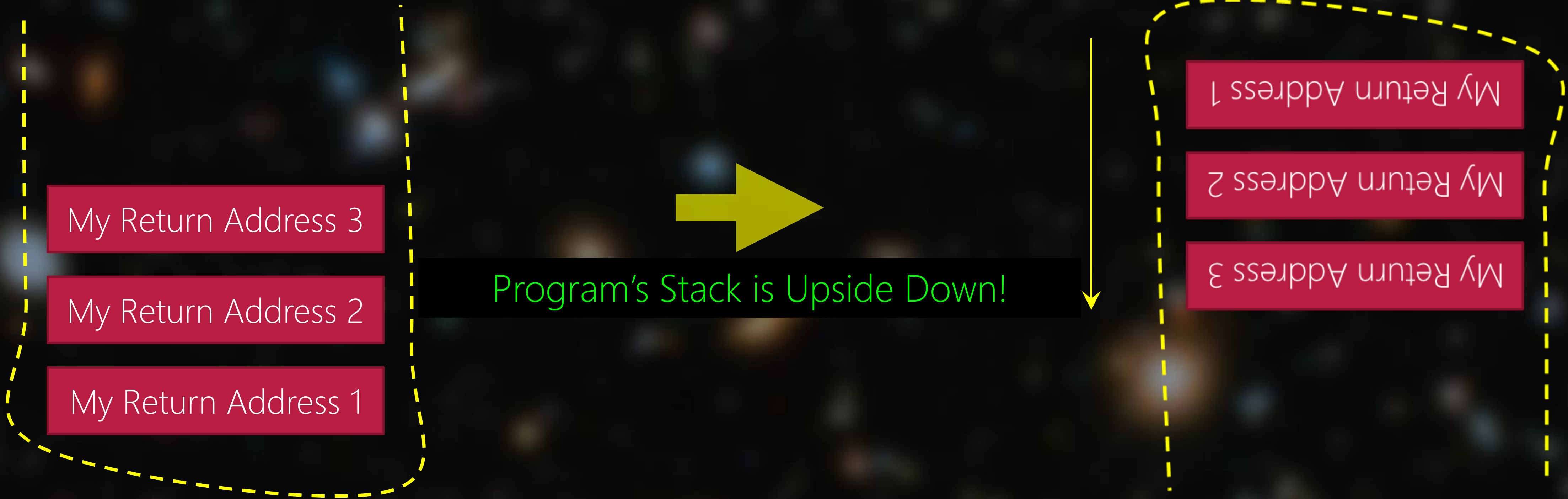
```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        result = a + b;

        printf("%d + %d = %d\n", a, b, result);
        return 0;
}
```

```
#include <stdlib.h>
#undef          atoi
/* Convert a string to an int.  */
int
atoi (const char *nptr)
{
  return (int) strtol (nptr, (char **) NULL, 10);
}
libc_hidden_def (atoi)
```

```
INT
INTERNAL (strtol) (const STRING_TYPE *nptr, STRING_TYPE **en
                   int base, int group)
{
  return INTERNAL (__strtol_l) (nptr, endptr, base, group, _
}
libc_hidden_def (INTERNAL (strtol))
```

Stack Overflow?

Functions Arguments, Local Variables, Return Address (runtime)



My Return Address 3

My Return Address 2

My Return Address 1

Program’s Stack is Upside Down!

My Return Address 1

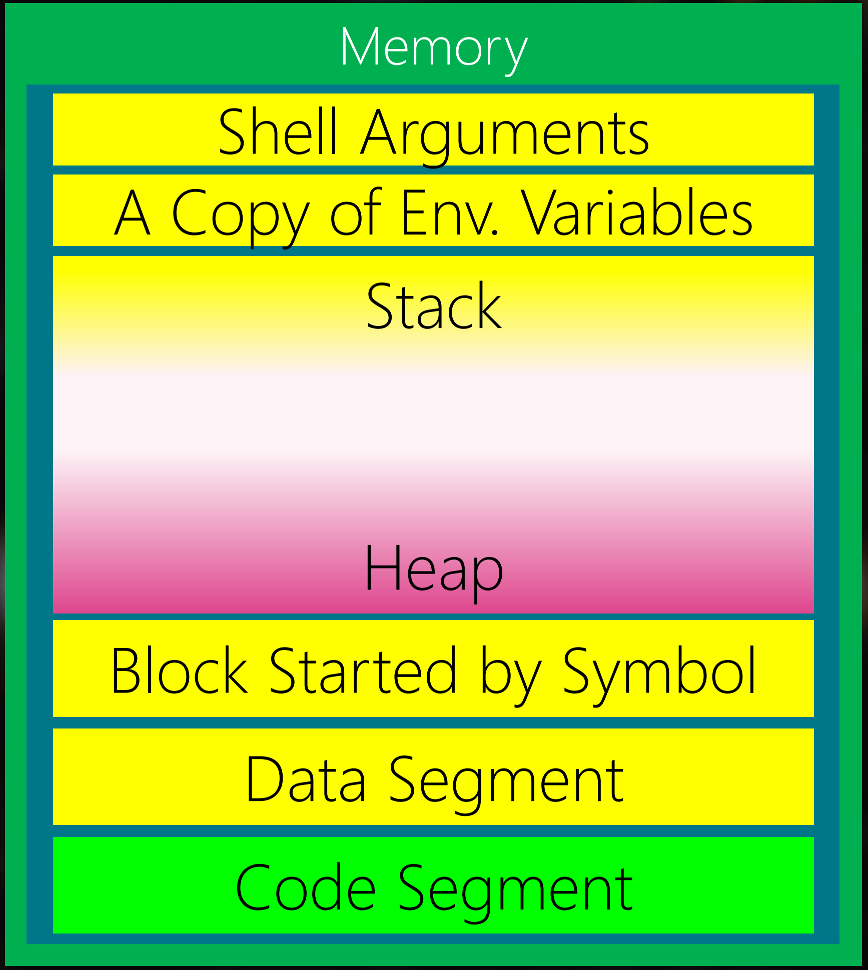
My Return Address 2

My Return Address 3

Into the Wild (2007) - Sean Penn

Heap

Dynamic memory allocation (runtime)



Memory

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

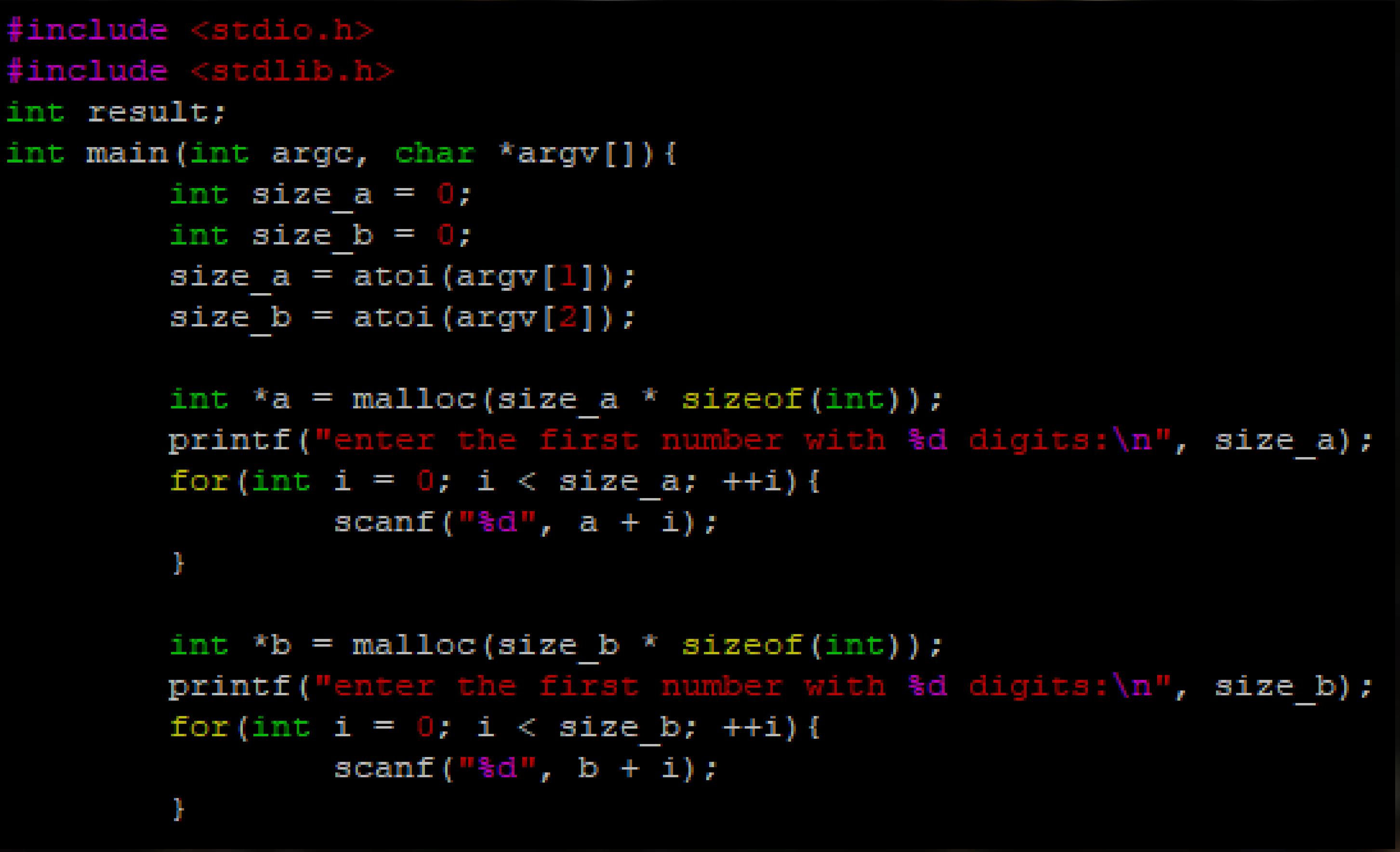
Data Segment

Code Segment

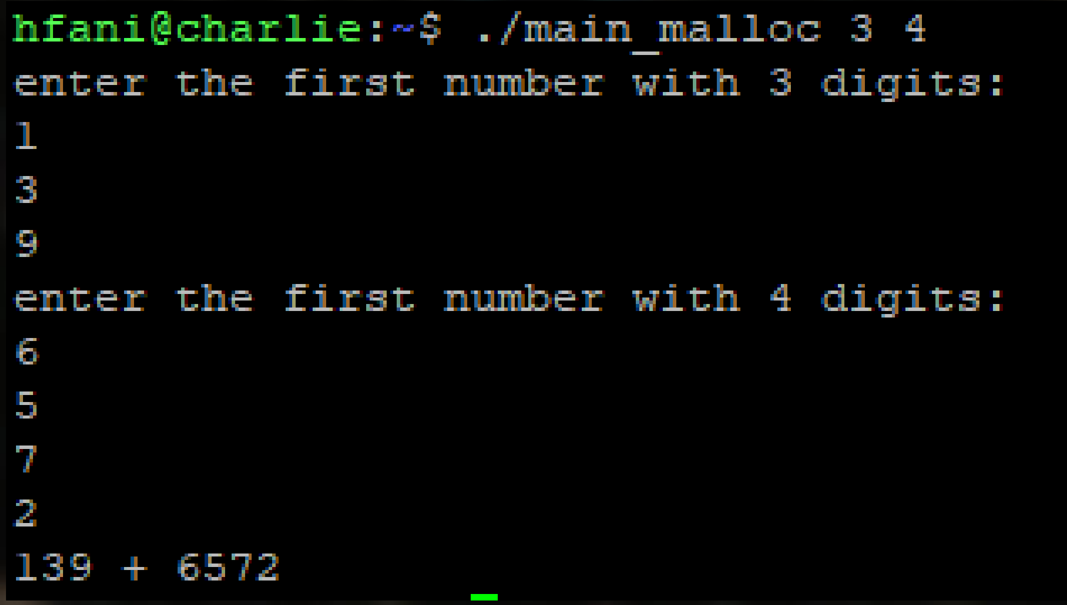
Memory Allocators by Library Routines

```
#include <stdlib.h>
void *malloc(size_t size)
void *realloc(void *ptr, size_t newsize)
```

Size is dynamic during runtime
Value is dynamic during runtime



```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int size_a = 0;
        int size_b = 0;
        size_a = atoi(argv[1]);
        size_b = atoi(argv[2]);

        int *a = malloc(size_a * sizeof(int));
        printf("enter the first number with %d digits:\n", size_a);
        for(int i = 0; i < size_a; ++i){
                scanf("%d", a + i);
        }

        int *b = malloc(size_b * sizeof(int));
        printf("enter the first number with %d digits:\n", size_b);
        for(int i = 0; i < size_b; ++i){
                scanf("%d", b + i);
        }
```



```
hfani@charlie:~$ ./main_malloc 3 4
enter the first number with 3 digits:
1
3
9
enter the first number with 4 digits:
6
5
7
2
139 + 6572
```

Process Identifier (pid)

Non-negative

Unique among processes (live programs)

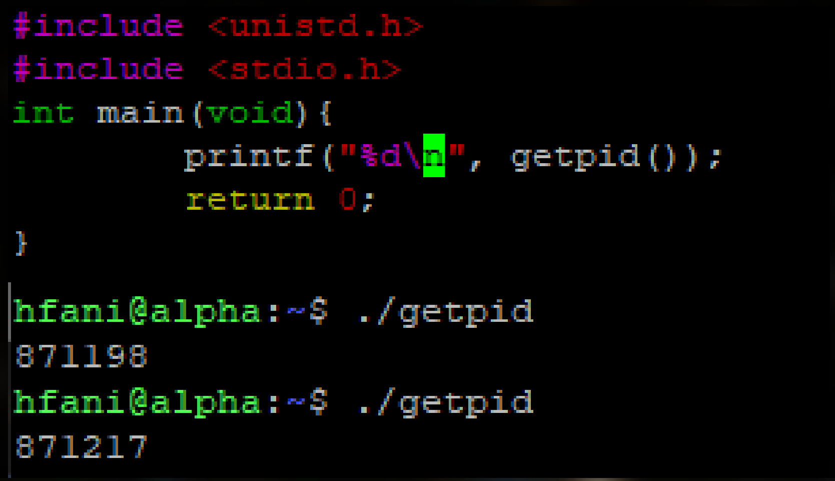
Not an identifier! It can be reused (delay reuse)

Process Identifier by System Call

getpid()

```
#include <unistd.h>
pid_t getpid(void);
```

Return process ID of calling process

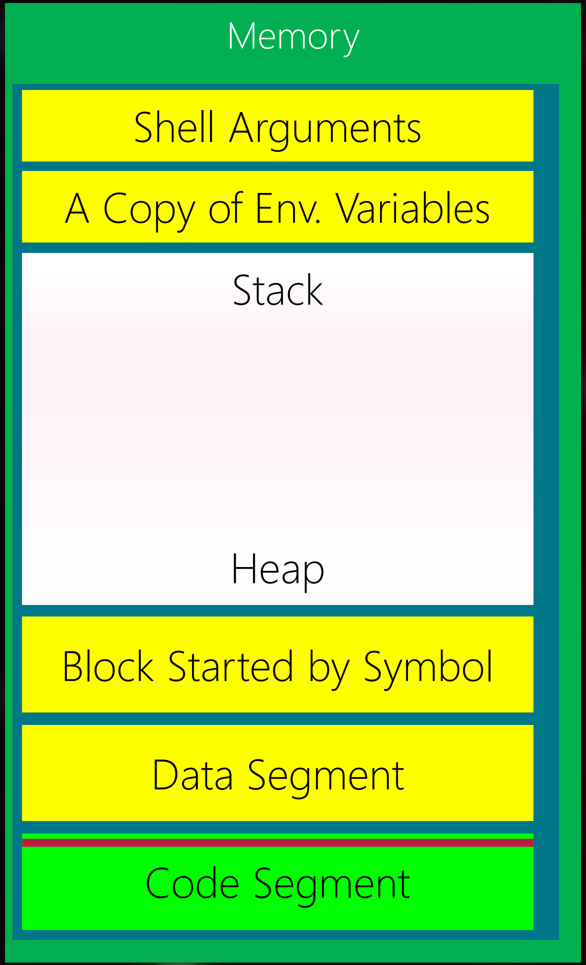


```
#include <unistd.h>
#include <stdio.h>
int main(void){
        printf("%d\n", getpid());
        return 0;
}
```

```
hfani@alpha:~$ ./getpid
871198
hfani@alpha:~$ ./getpid
871217
```

Into the Wild (2007) - Sean Penn

Program → Process → Run → Terminate



Memory

Shell Arguments

A Copy of Env. Variables

Stack

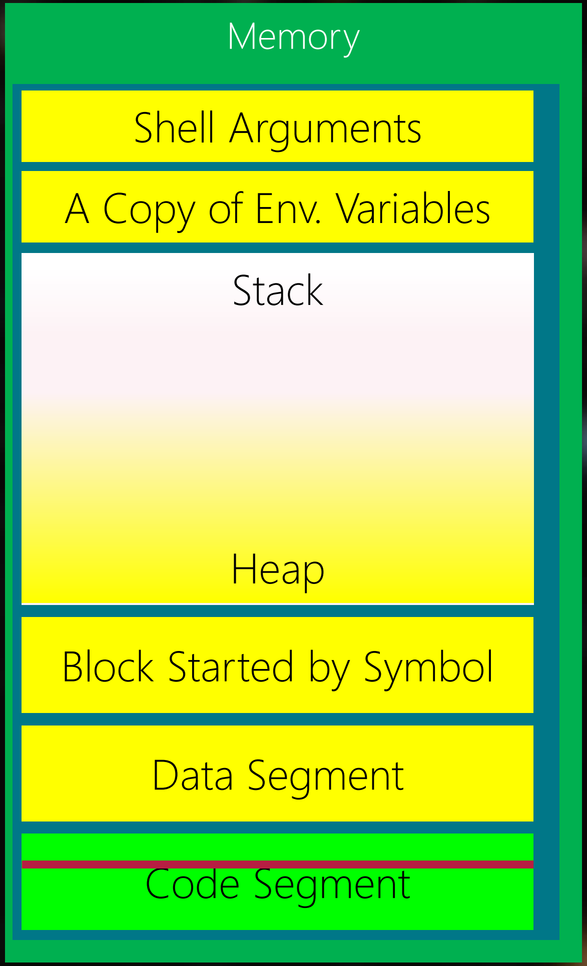
Heap

Block Started by Symbol

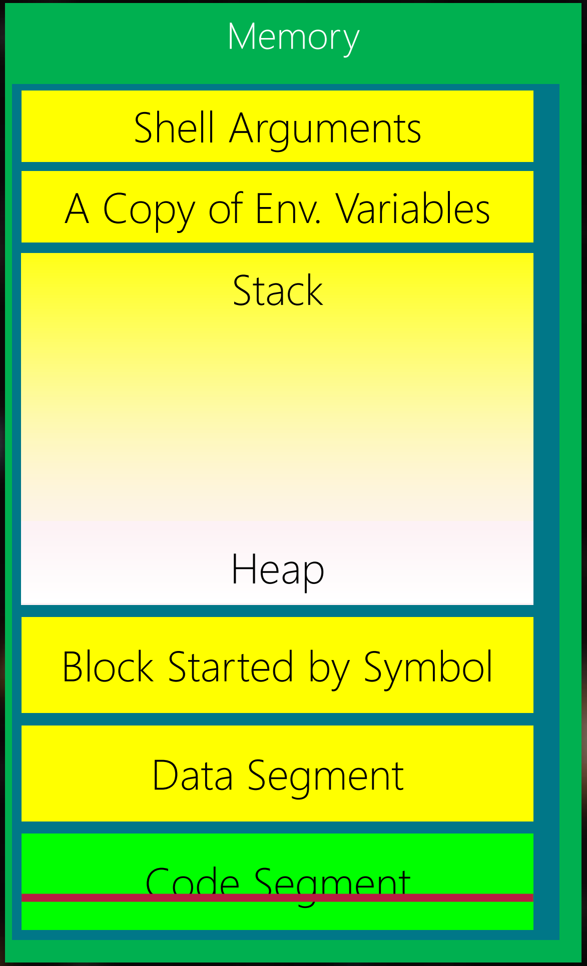
Data Segment

Code Segment

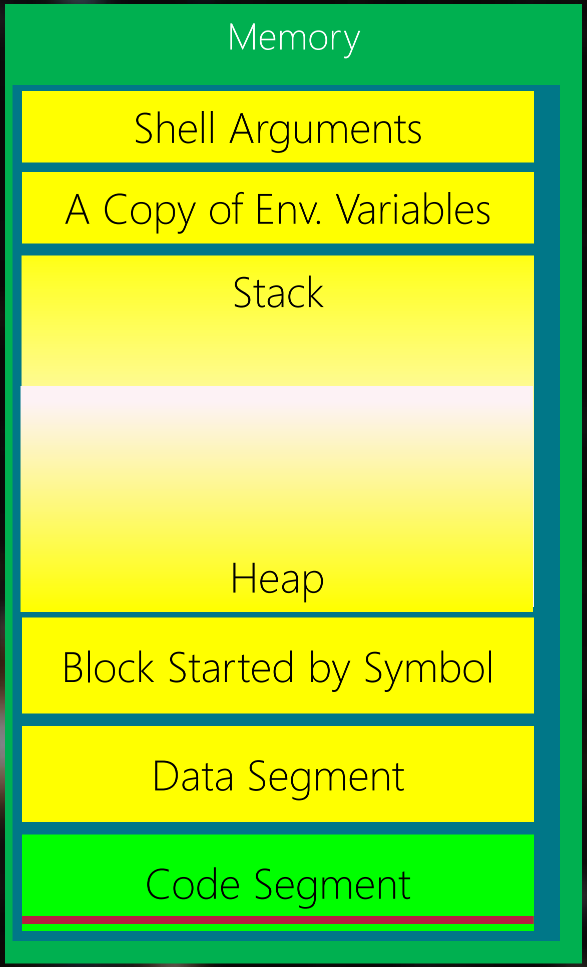
Program → Process → Run → Terminate



Program → Process → Run → Terminate



Program → Process → Run → Terminate



Memory

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

Data Segment

Code Segment

Into the Wild (2007) - Sean Penn

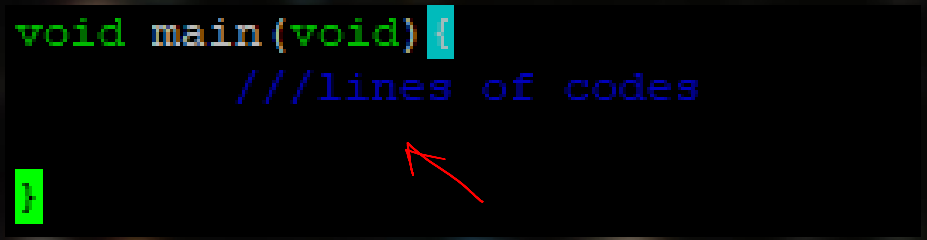
Program → Process → Run → Terminate

C has exit status (code)

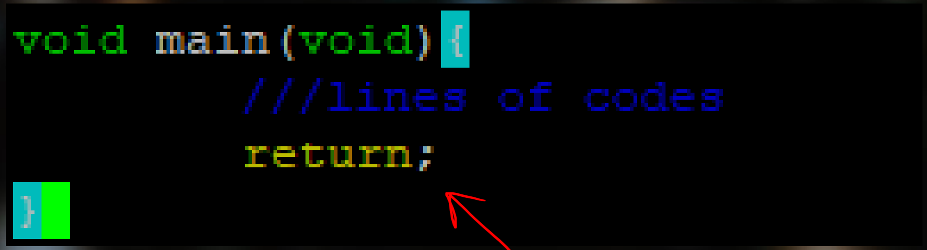
Normal vs. Abnormal Exits

C has exit status (code)

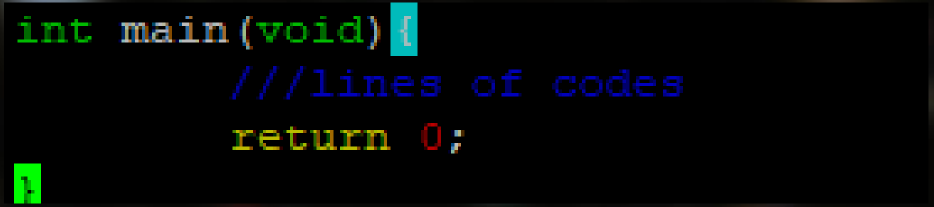
Normal



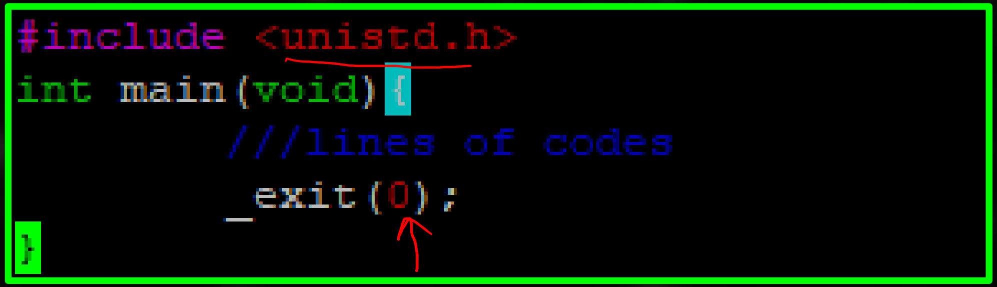
```
void main(void){
        ///lines of codes

}
```



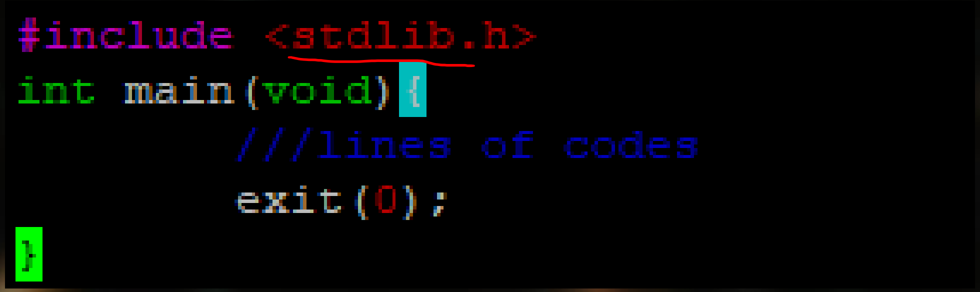
```
void main(void){
        ///lines of codes
        return;
}
```



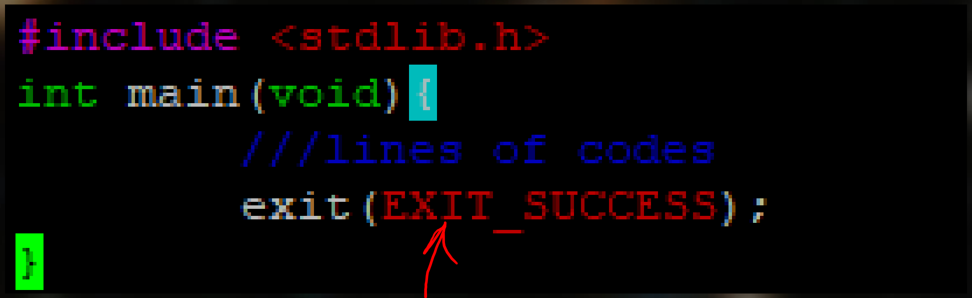
```
int main(void){
        ///lines of codes
        return 0;
}
```



```
#include <unistd.h>
int main(void){
        ///lines of codes
        _exit(0);
}
```



```
#include <stdlib.h>
int main(void){
        ///lines of codes
        exit(0);
}
```



```
#include <stdlib.h>
int main(void){
        ///lines of codes
        exit(EXIT_SUCCESS);
}
```

C has exit status (code)

Normal

Clean up procedure

- Flushes unwritten buffered data.
- Closes all open file descriptors.
- Frees the memory used by its code, data, stack, heap, …
- Returns an integer exit status to the kernel. 0

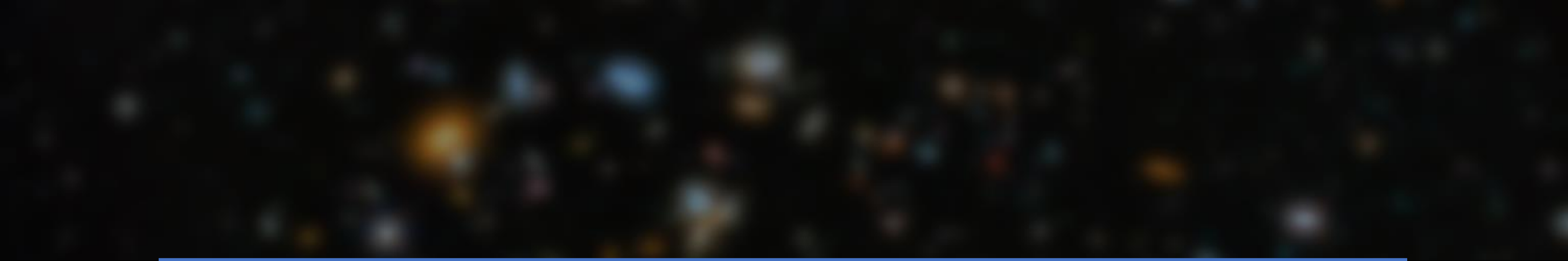
C has exit status (code)

Abnormal

exit(2)

- Any non-zero number less than 256
- Receiving a `SIGNAL`

`e.g., SIGABRT` raised by `abort()`



Process Manager

aka. Process Control

- Review: Program → Process → Run → Terminate
- Multiprocessing
  - HALT
  - Context Switch
  - Fork

Multiprocessing

aka multiprogramming

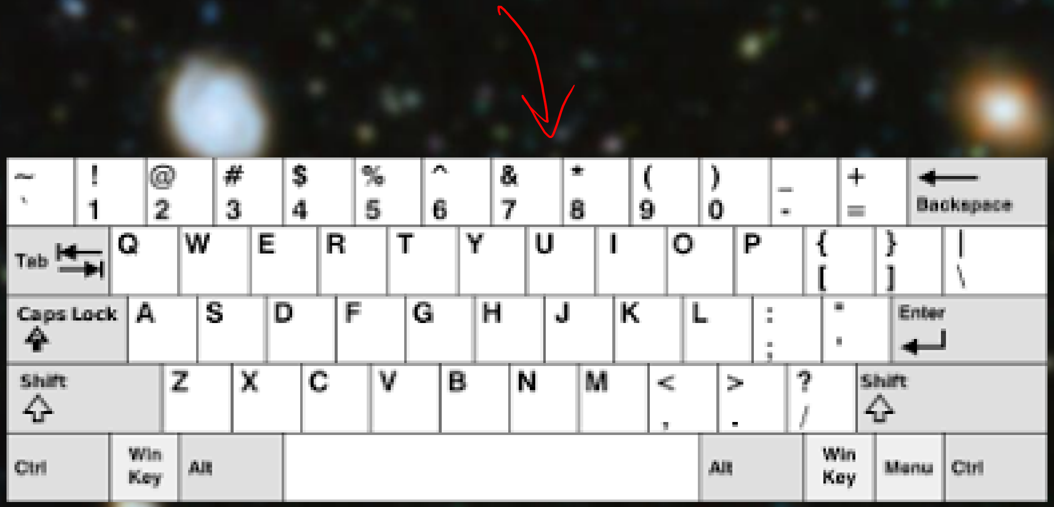
Single Processor ~~Multiprocessor~~

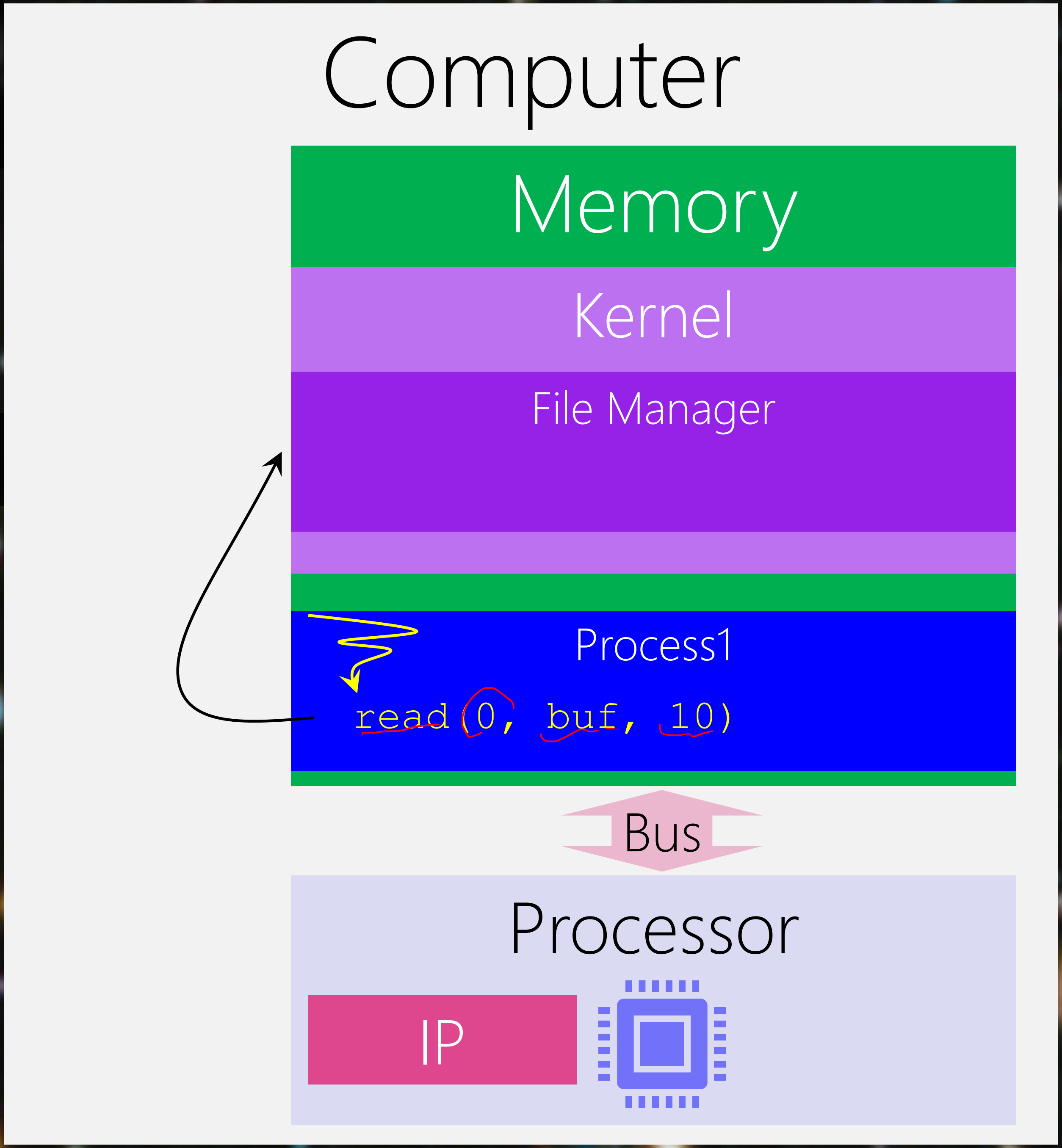
Week#2

Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?





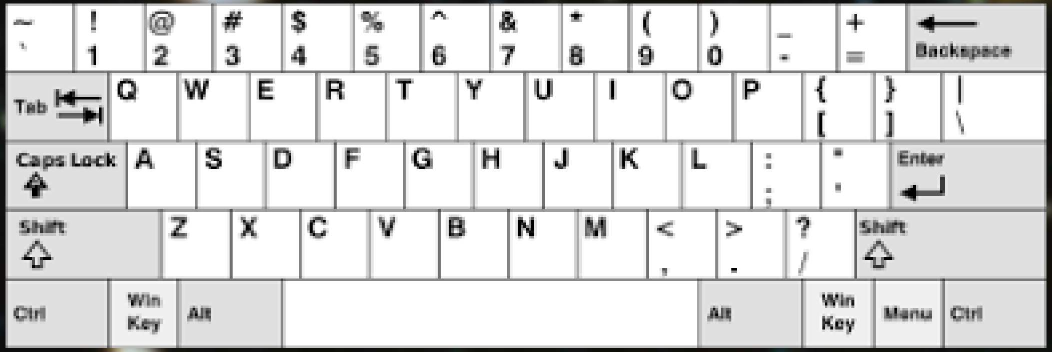
Week#2

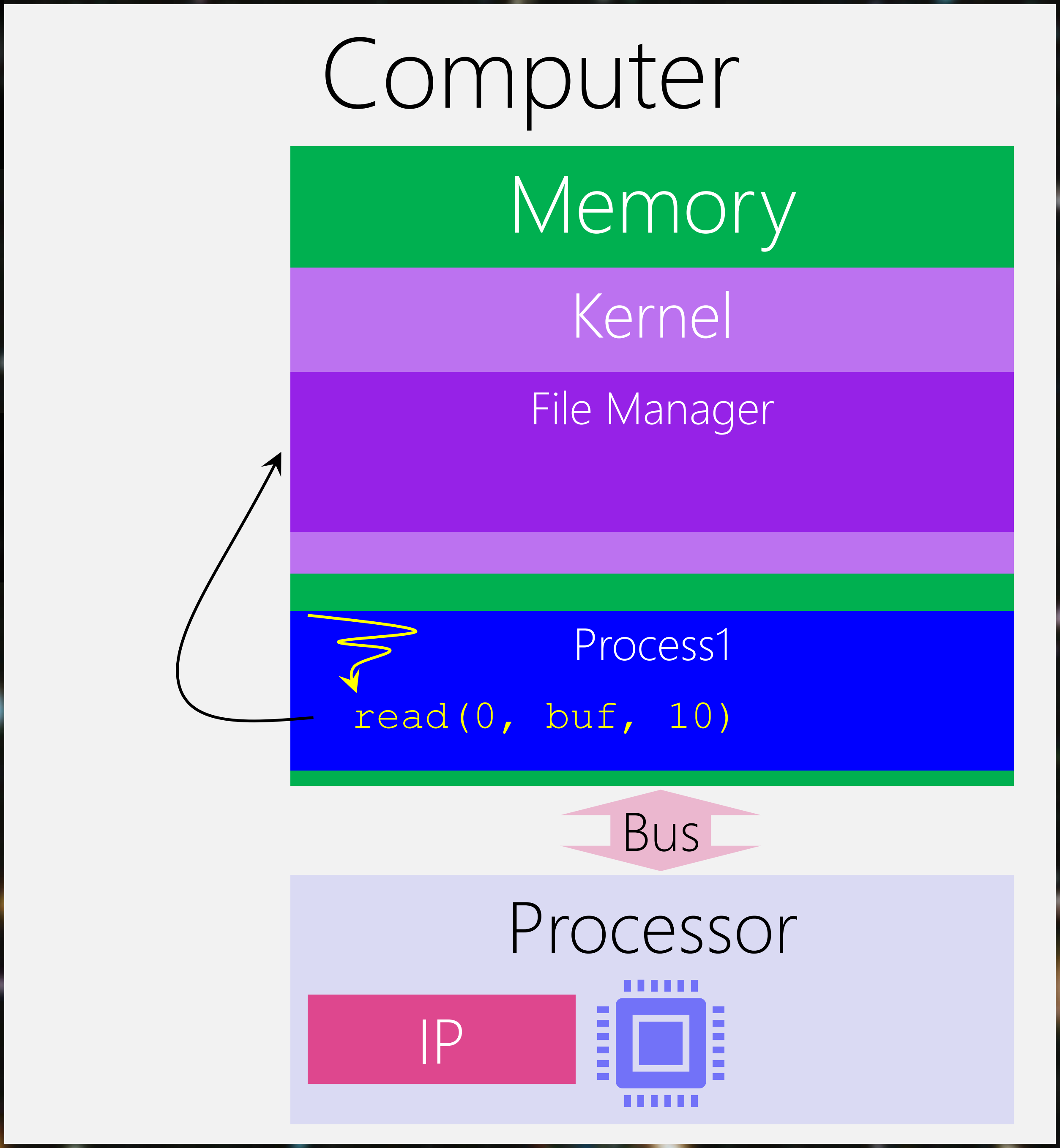
Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?

What is the processor doing?


~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl


Computer

Memory

Kernel

File Manager

Process1

```
read(0, buf, 10)
```

Bus

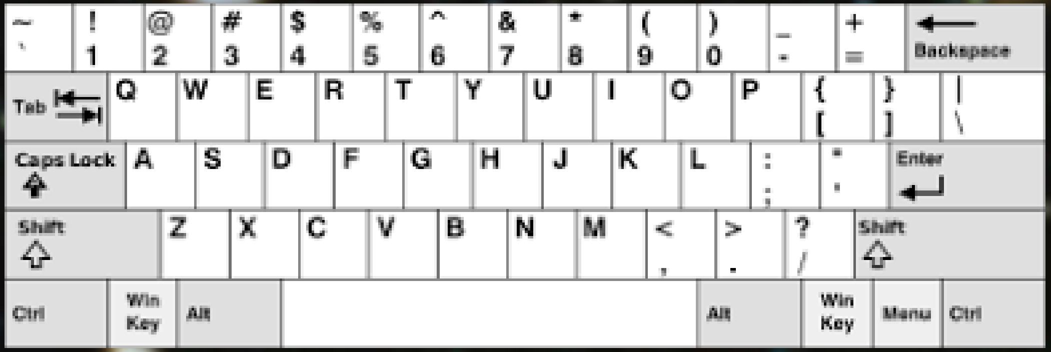
Processor

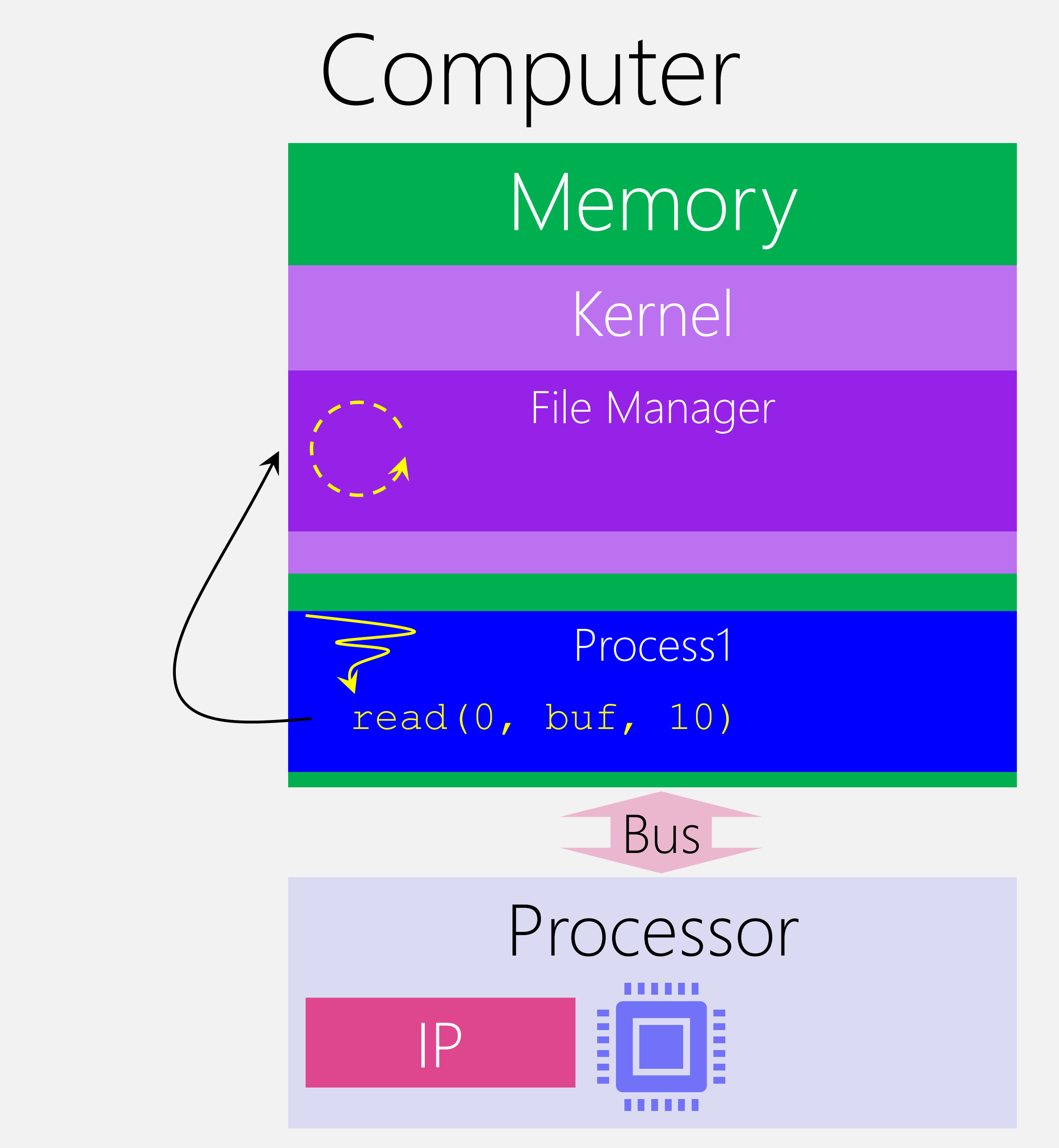
IP

Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?
What is the processor doing?
A) Busy waiting by the File Manager





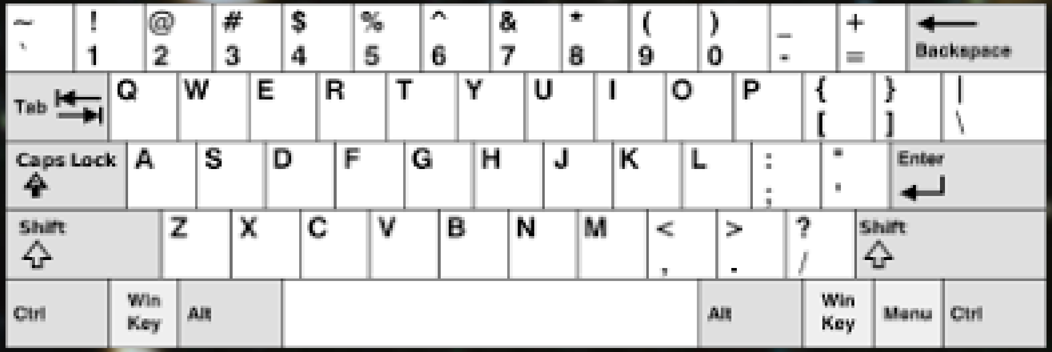
Week#2

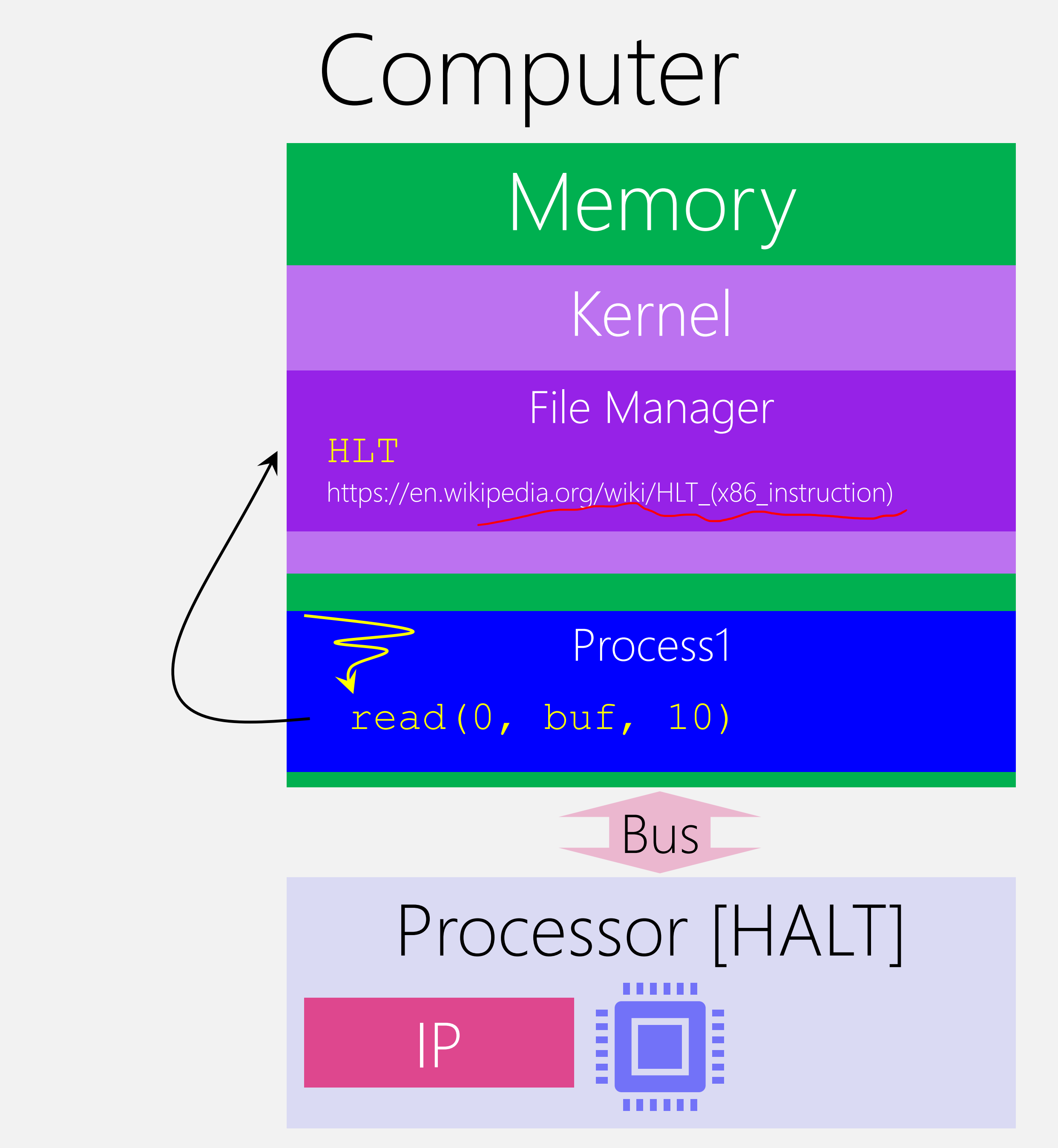
Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?

What is the processor doing?

B) HALT State





Computer

Memory

Kernel

File Manager

HLT

https://en.wikipedia.org/wiki/HLT_(x86_instruction)

Process1

```
read(0, buf, 10)
```

Bus

Processor [HALT]

IP

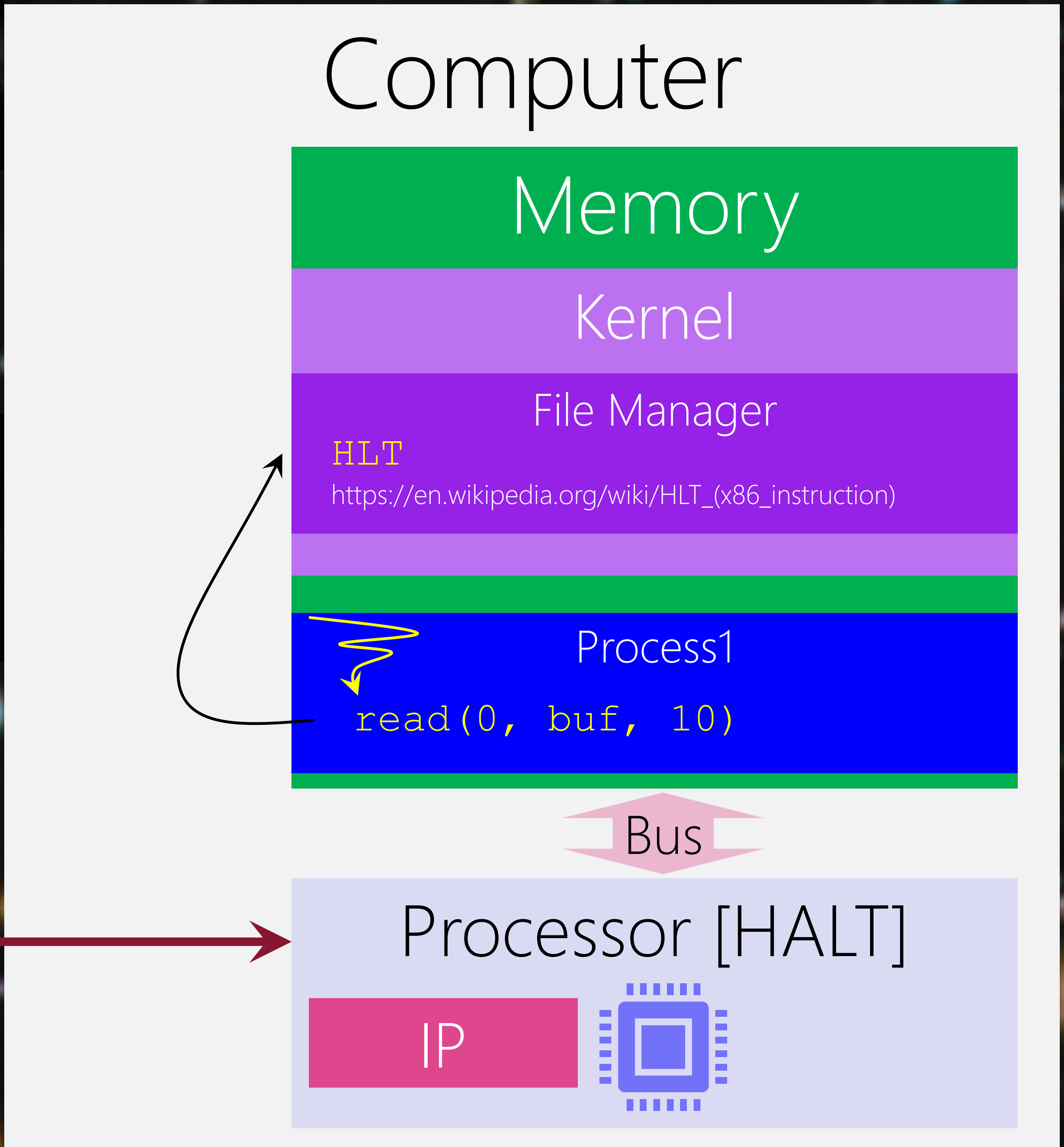
Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?

What is the processor doing?

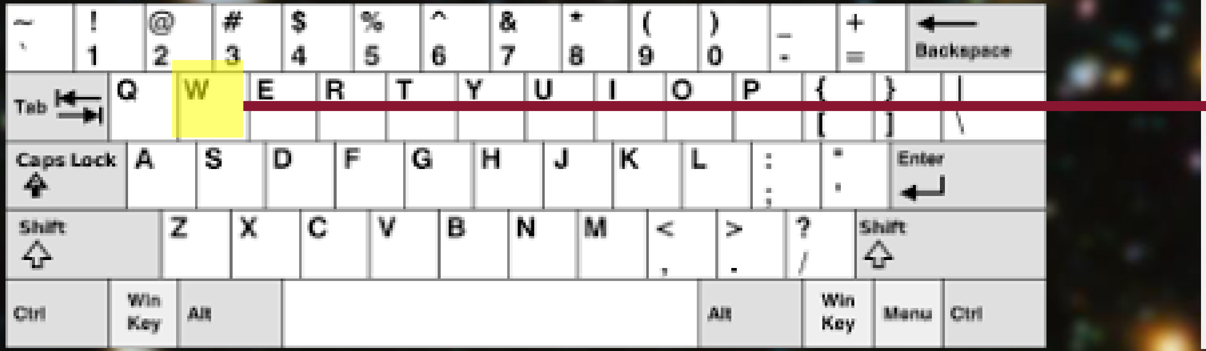
B) HALT State until an external shock!



```
HLT
```

https://en.wikipedia.org/wiki/HLT_(x86_instruction)

```
read(0, buf, 10)
```


~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl

Week#2

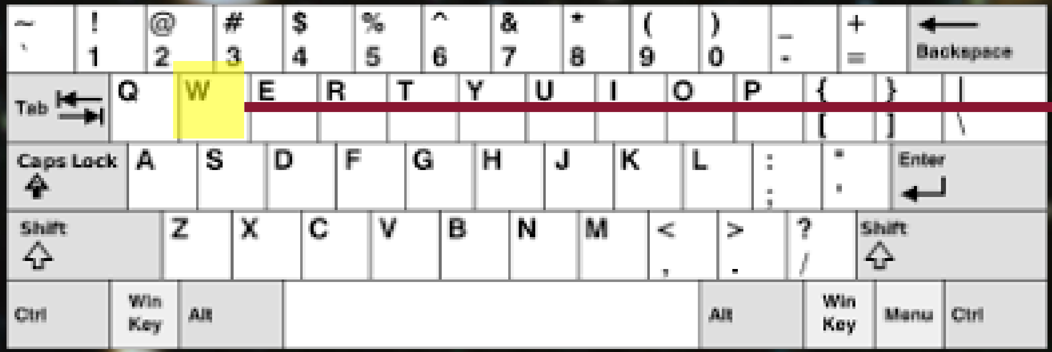
Interrupt Request (IRQ)

Interrupt Request Handler

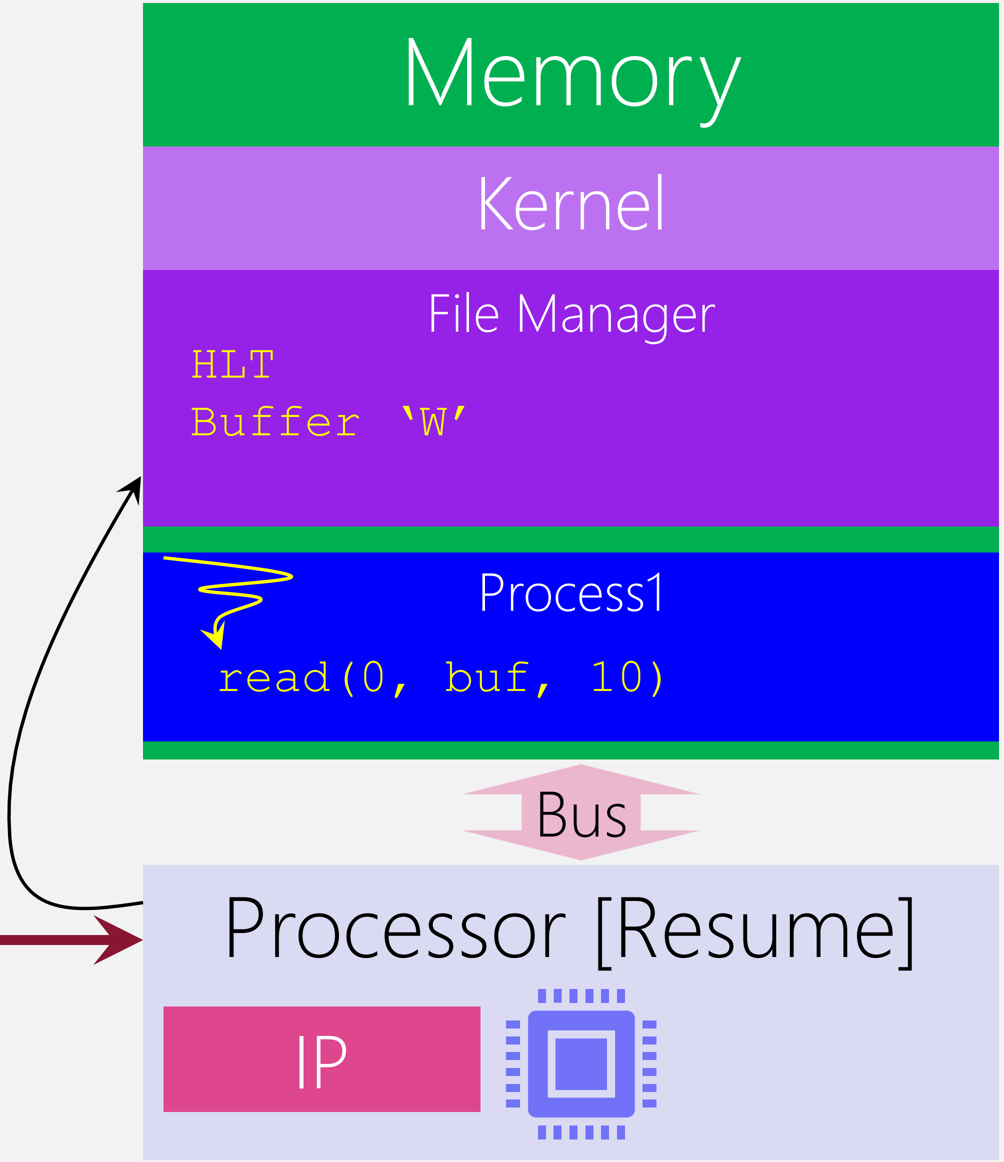
What is happening next?

What is the processor doing?

- Resume normal operation



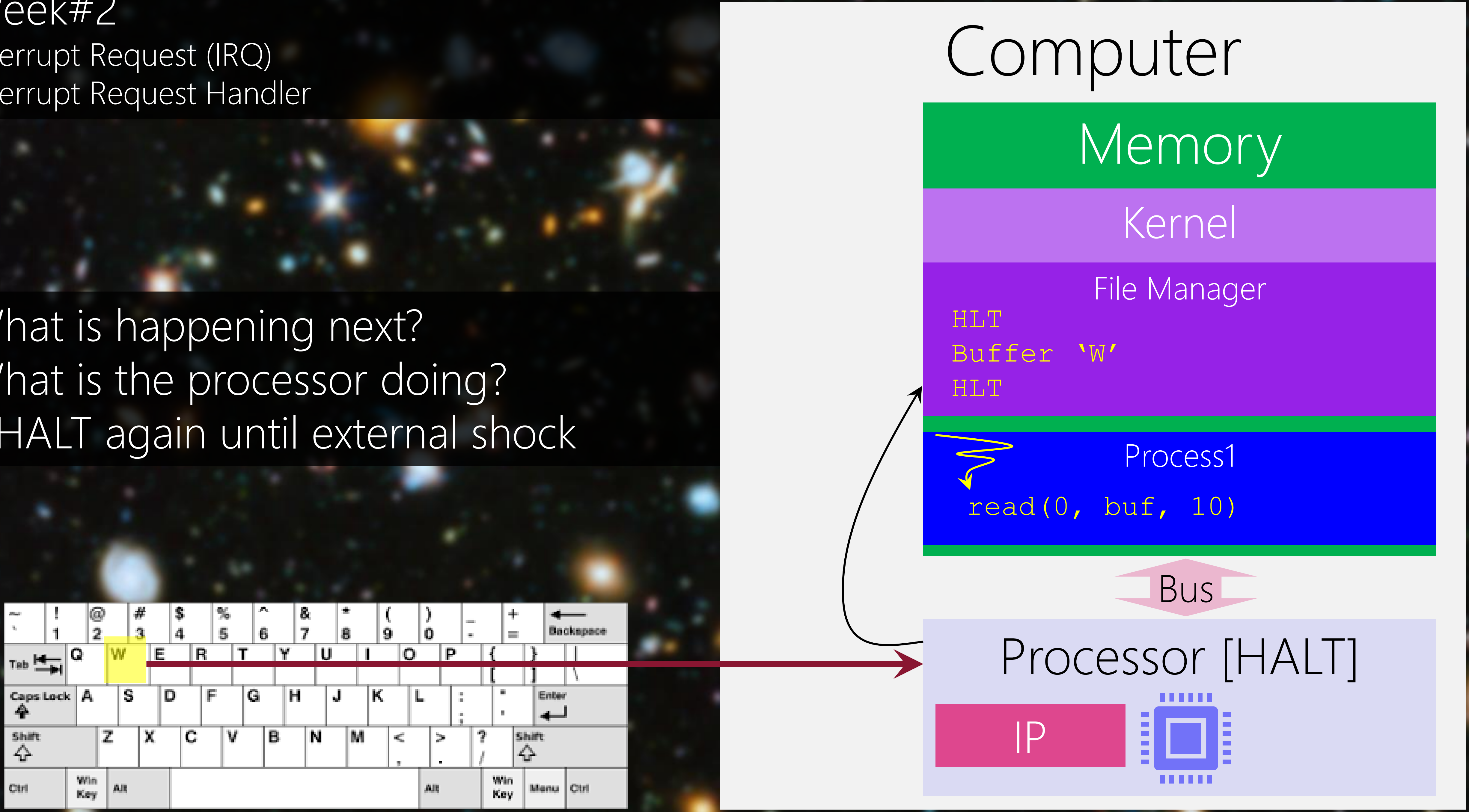
Computer



Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?
What is the processor doing?
- HALT again until external shock



Week#2

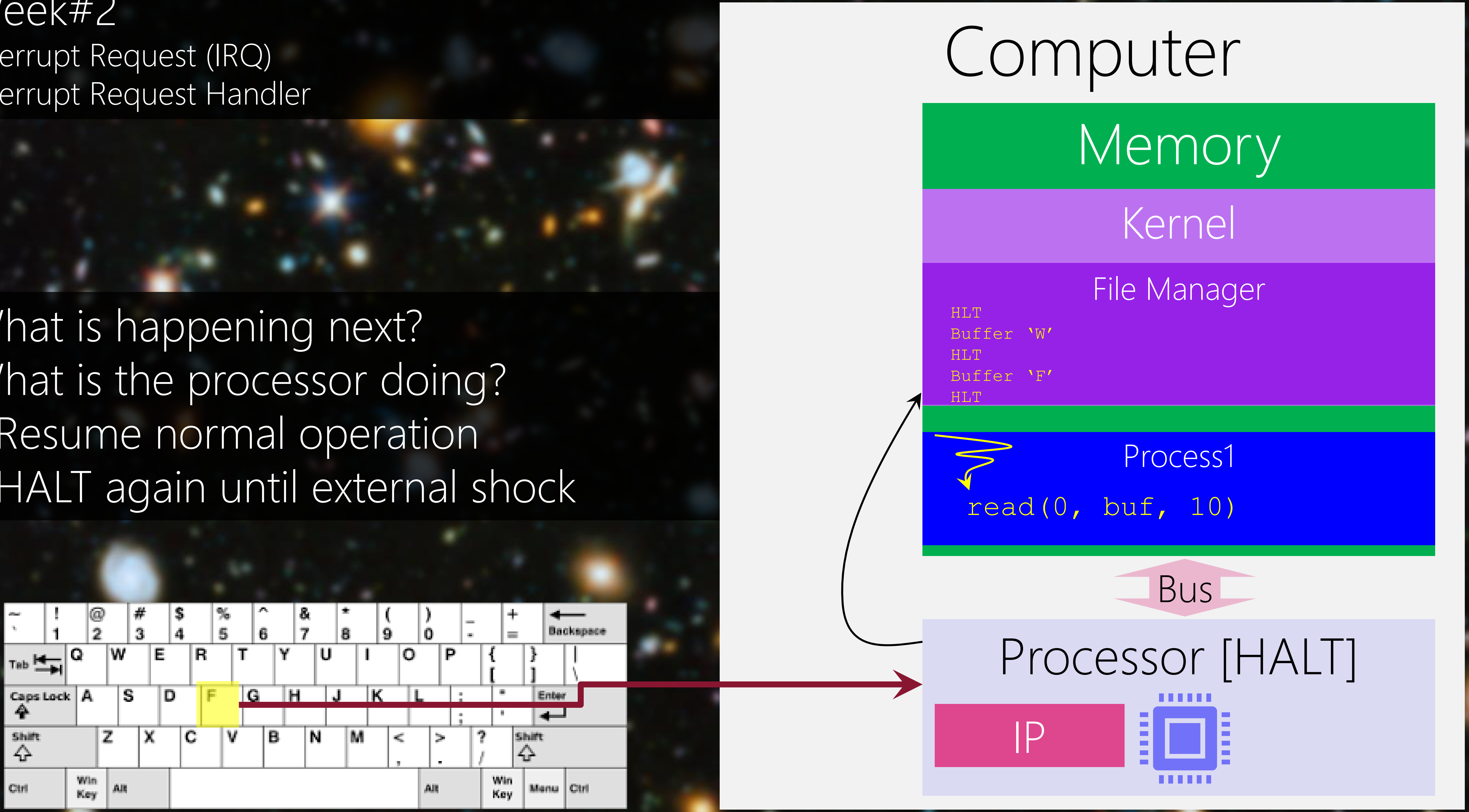
Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?

What is the processor doing?

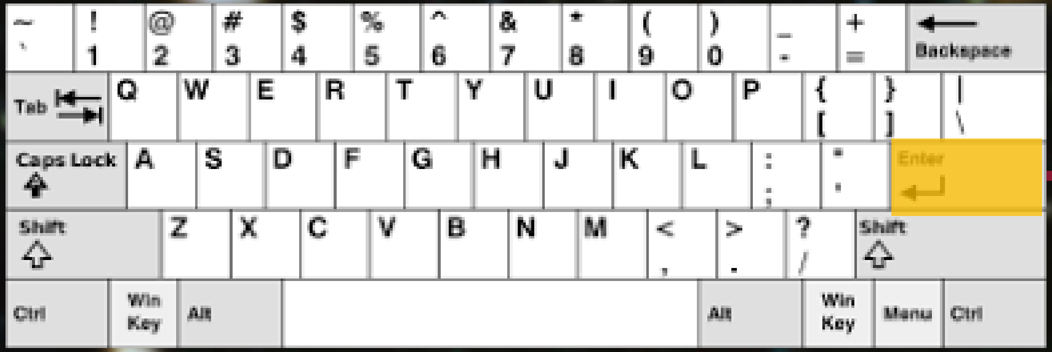
- Resume normal operation
- HALT again until external shock



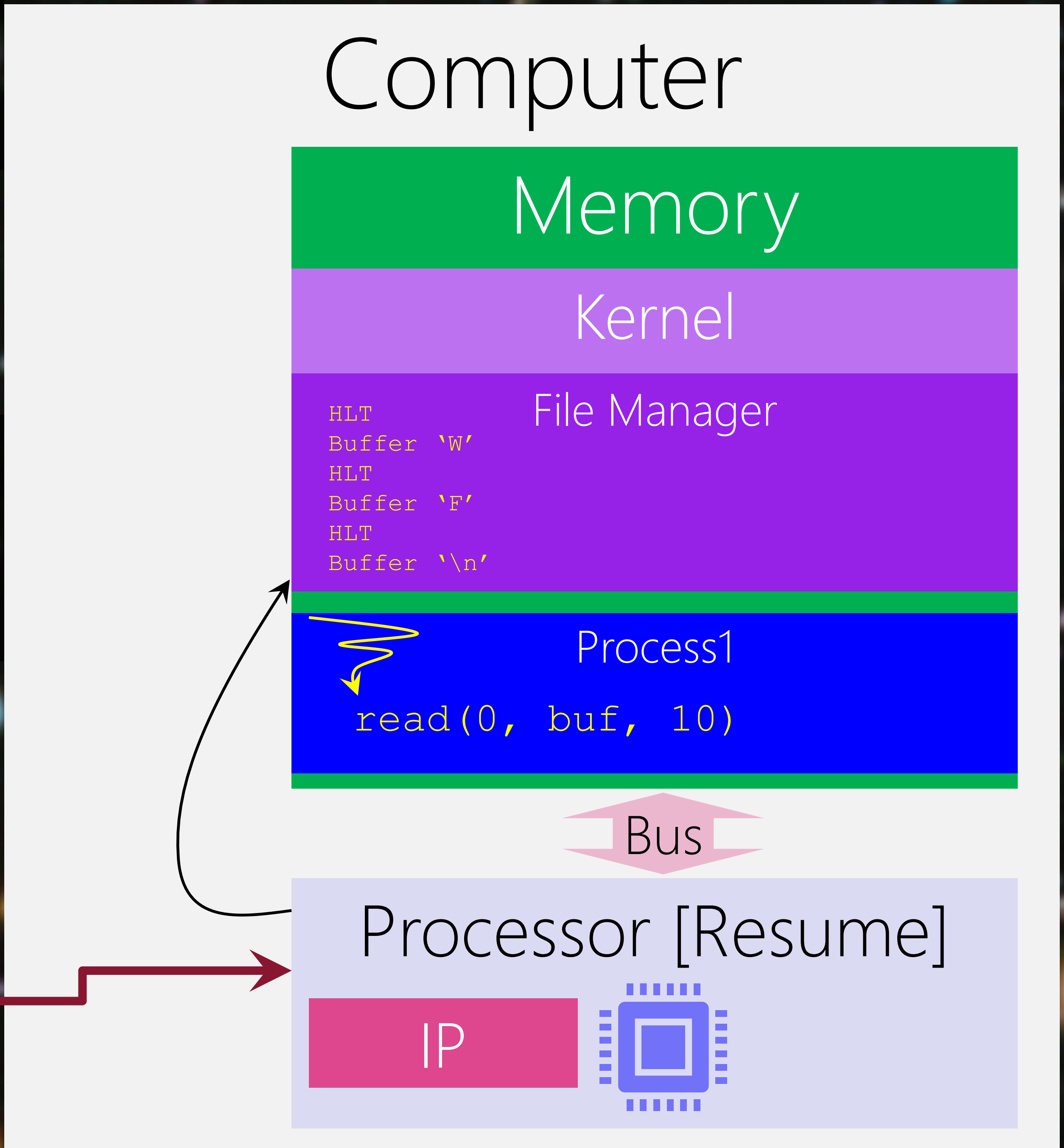
Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?
What is the processor doing?
- Resume normal operation



~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl



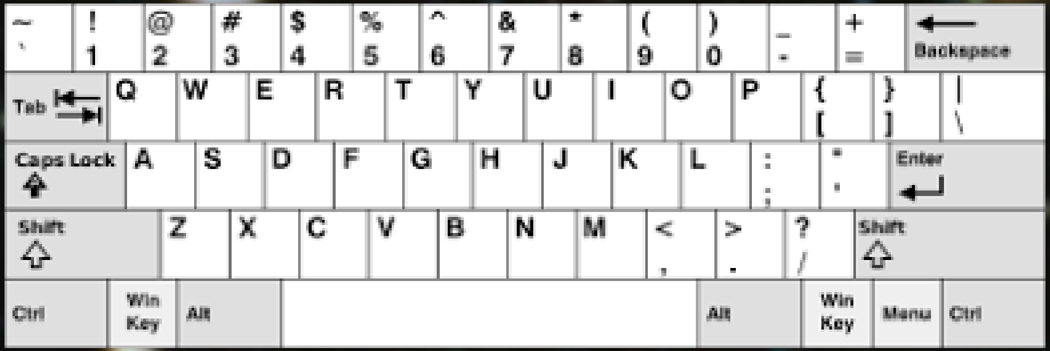
```
HLT
Buffer 'W'
HLT
Buffer 'F'
HLT
Buffer '\n'
```

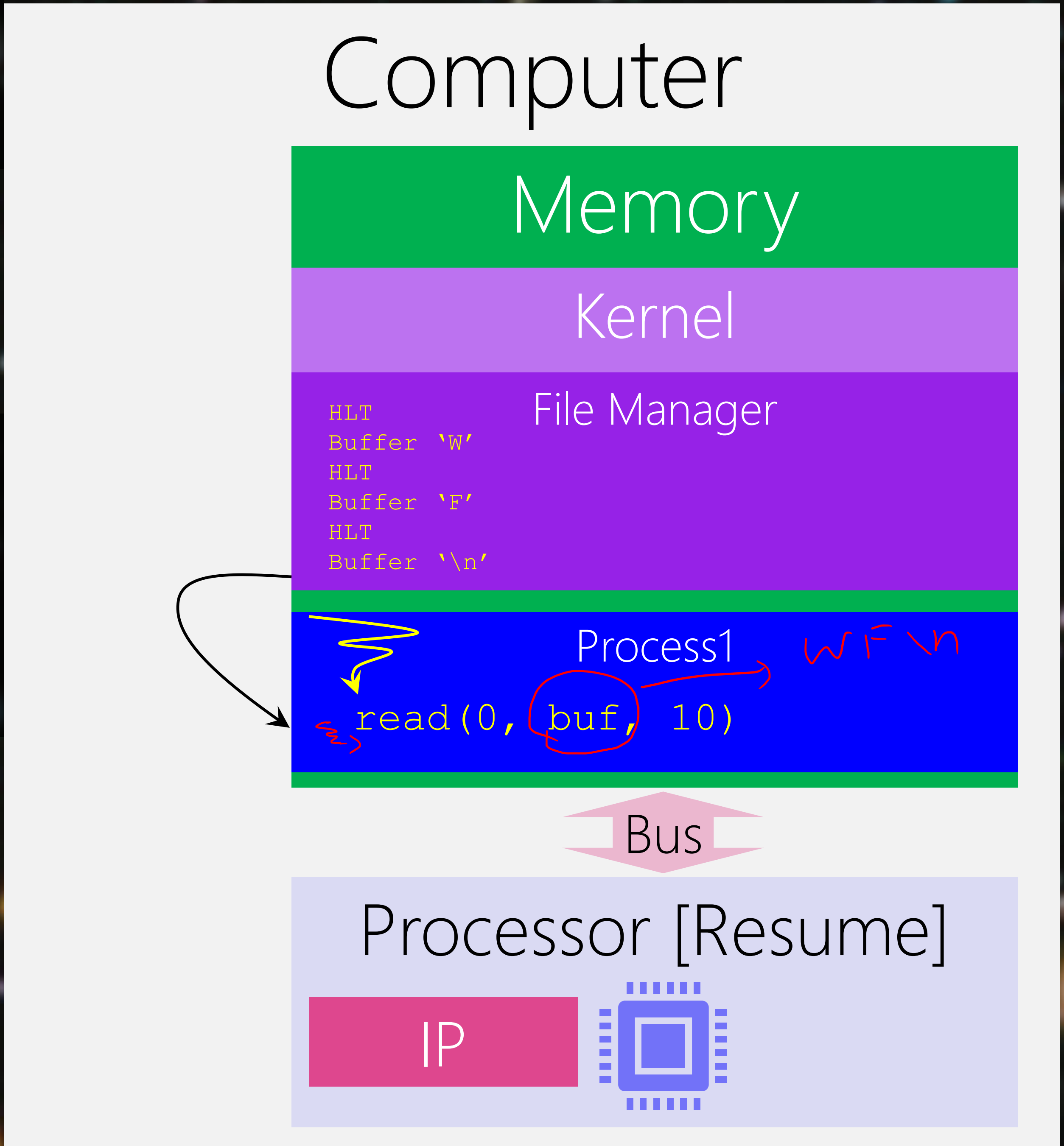
```
read(0, buf, 10)
```

Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?
What is the processor doing?
- Resume normal operation
- Give control to the process





```
HLT
Buffer 'W'
HLT
Buffer 'F'
HLT
Buffer '\n'
```

```
read(0, buf, 10)
```

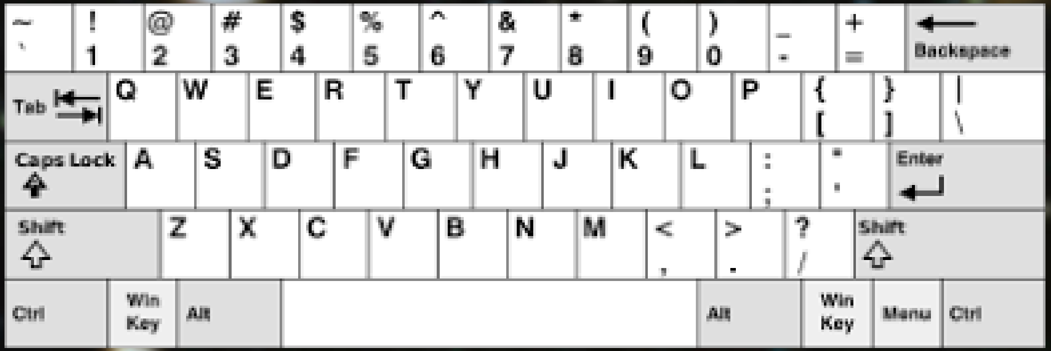
Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

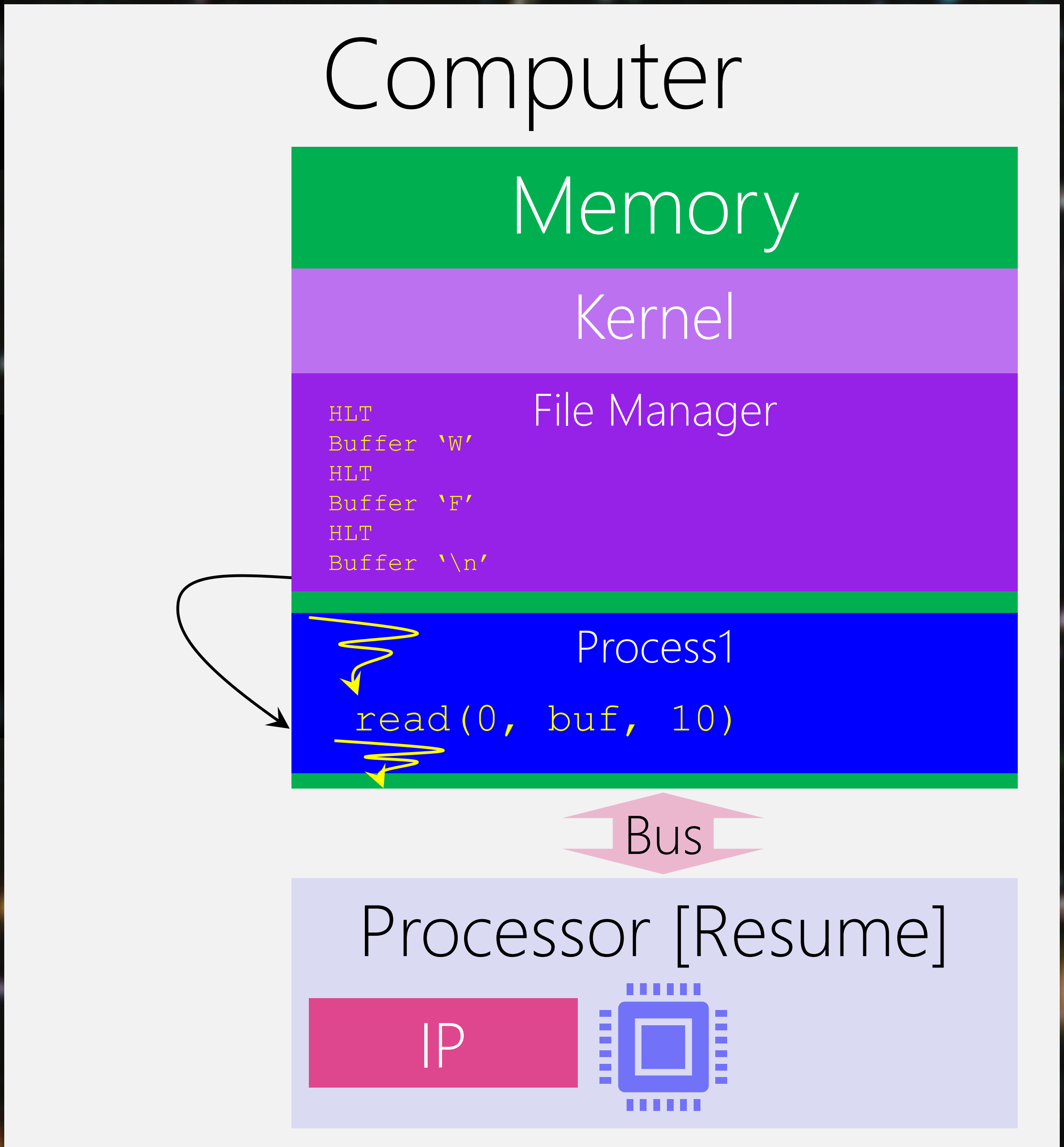
What is happening next?

What is the processor doing?

- Resume normal operation
- Give control to the process



~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl



Computer

Memory

Kernel

File Manager

```
HLT
Buffer ‘W’
HLT
Buffer ‘F’
HLT
Buffer ‘\n’
```

Process1

```
read(0, buf, 10)
```

Bus

Processor [Resume]

Whether Busy Waiting or HALT

Waste of Processor

Single Processor ~~Multiprocessor~~

Whether Busy Waiting or HALT

Share it with another process

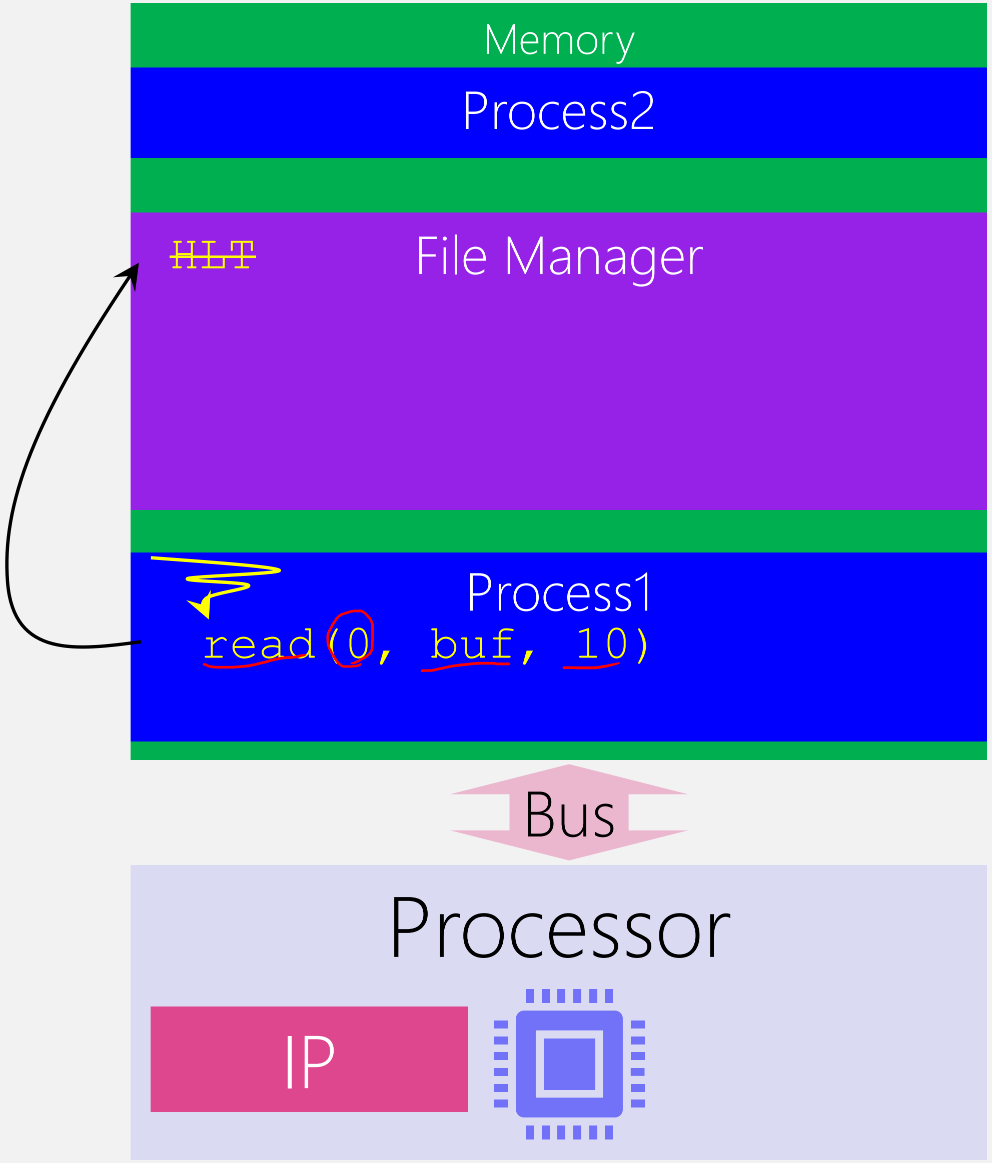
Single Processor ~~Multiprocessor~~

Whether Busy Waiting or HALT

Processor Sharing → Time Sharing/Slicing → Scheduling

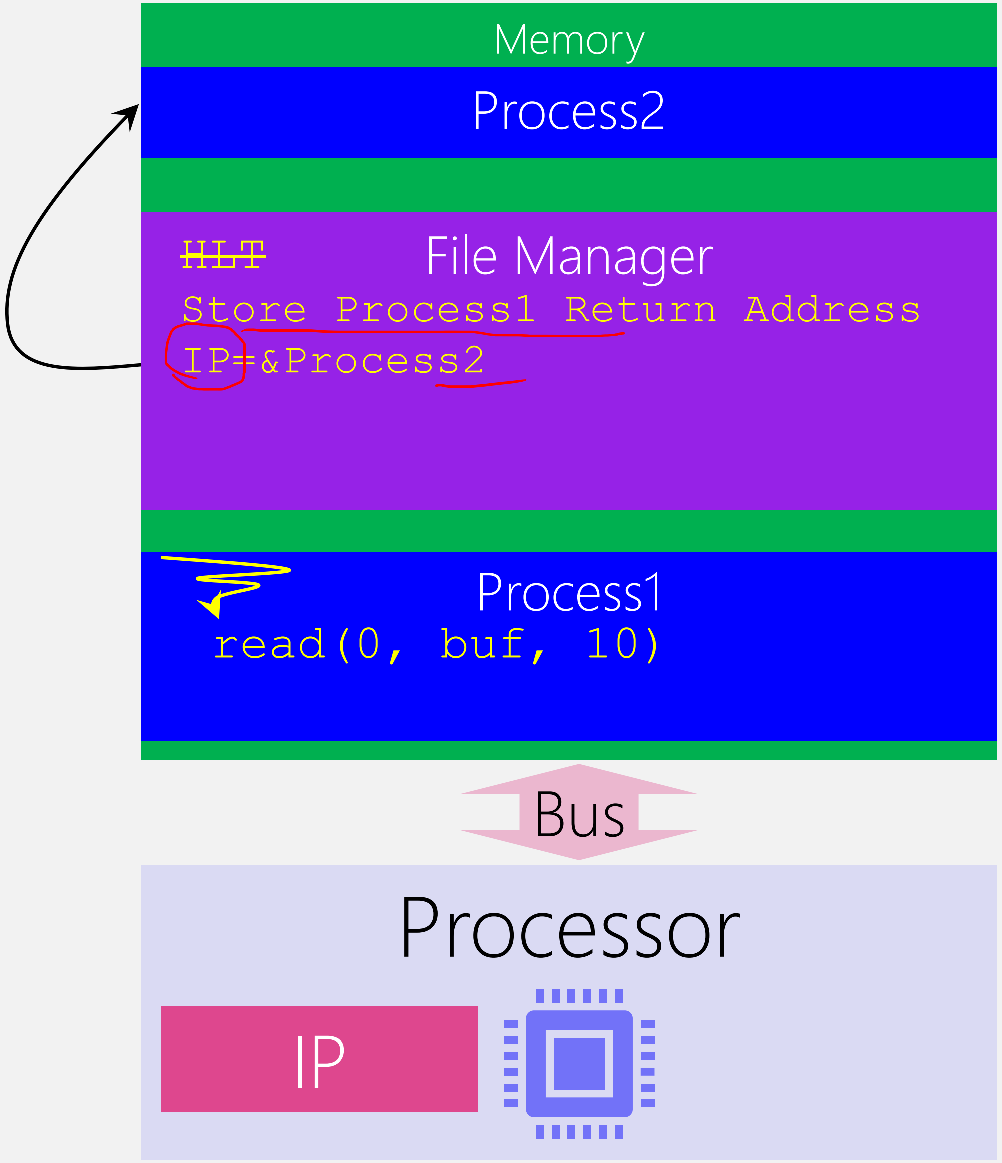
Single Processor ~~Multiprocessor~~

Computer



```
read(0, buf, 10)
```

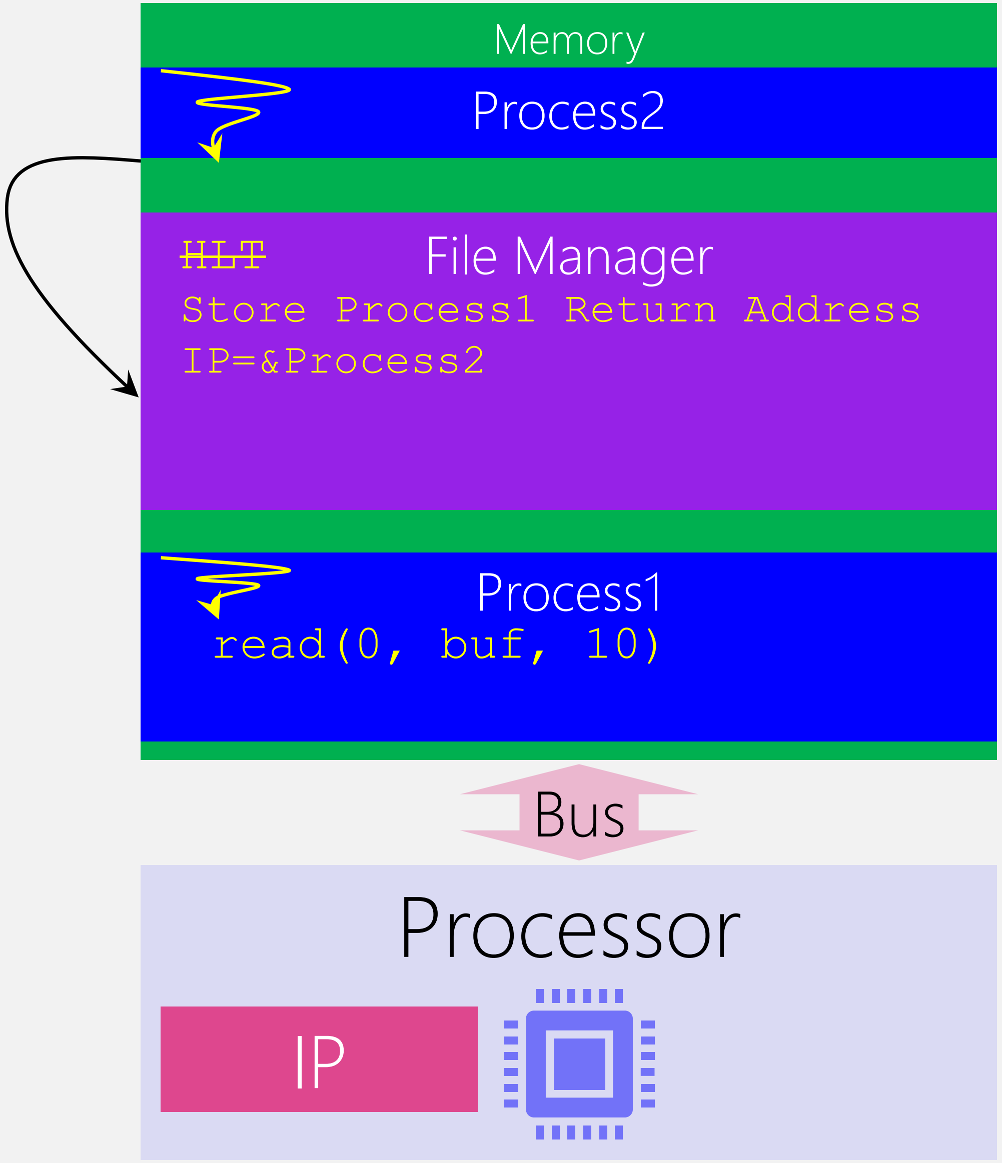
Computer



```
~~HLT~~
Store Process1 Return Address
IP=&Process2
```

```
read(0, buf, 10)
```

Computer



Memory

Process2

~~HLT~~

File Manager

```
Store Process1 Return Address
IP=&Process2
```

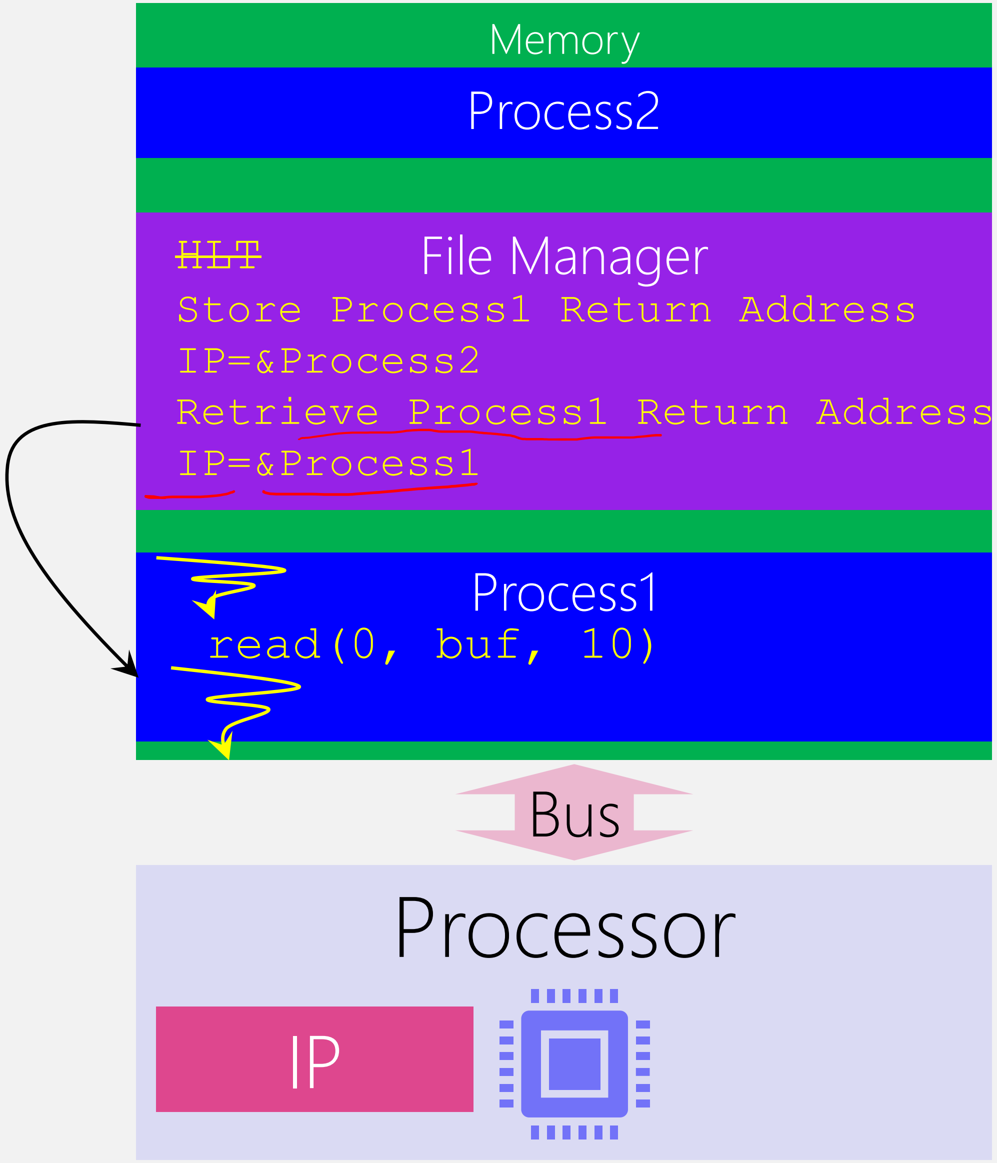
Process1

```
read(0, buf, 10)
```

Bus

Processor

Computer



Memory

Process2

~~HLT~~ File Manager

```
Store Process1 Return Address
IP=&Process2
Retrieve Process1 Return Address
IP=&Process1
```

Process1

```
read(0, buf, 10)
```

Bus

Processor

IP

Process Manager

aka. Process Control

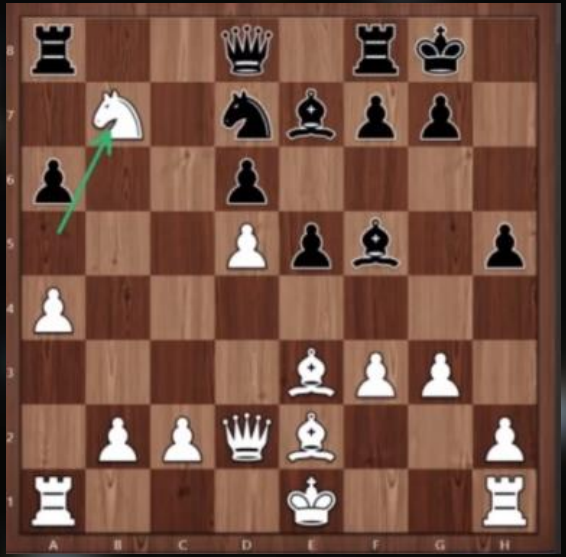
- Review: Program → Process → Run → Terminate
- Multiprocessing
    - HALT
    - Context Switch
    - Fork

It's not that simple, tho!

Further Reading → Process Context Switch → OS

Taking Processor and Give it to Another Process

Magnus Carlsen

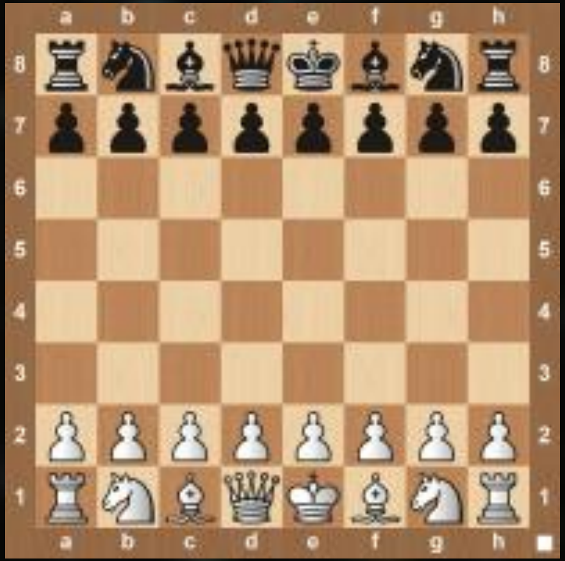


Hikaru Nakamura

Can we borrow your chessboard while you're thinking for the next move?

Magnus Carlsen

Sure! →

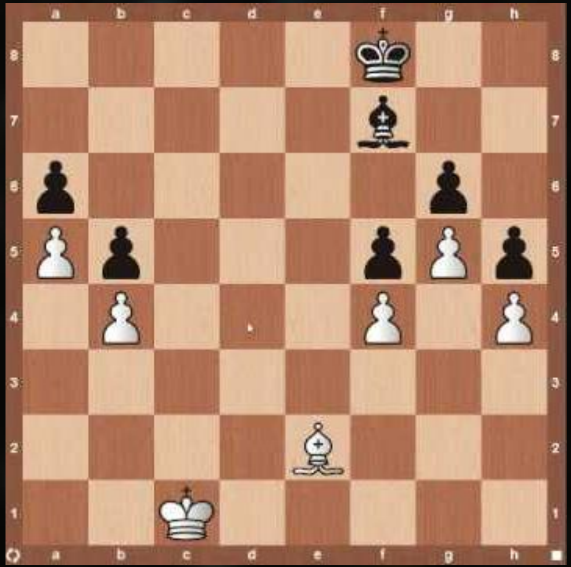


Hikaru Nakamura

Eris Li

Magnus Carlsen

Can we have it back?



Hikaru Nakamura

Eris Li

Magnus Carlsen

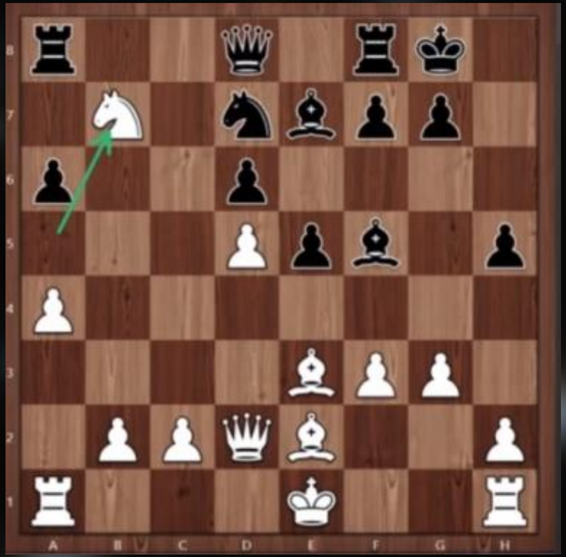
Where were we?!

← Sure!

Hikaru Nakamura

Eris Li

Magnus Carlsen



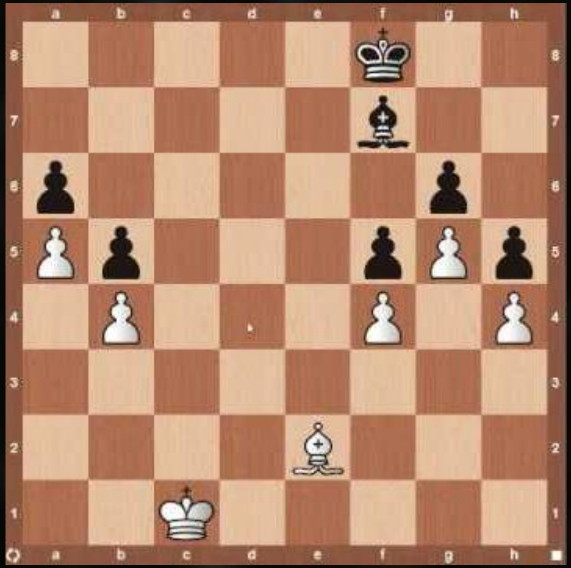
← Sure!

Hikaru Nakamura

Eris Li

Magnus Carlsen

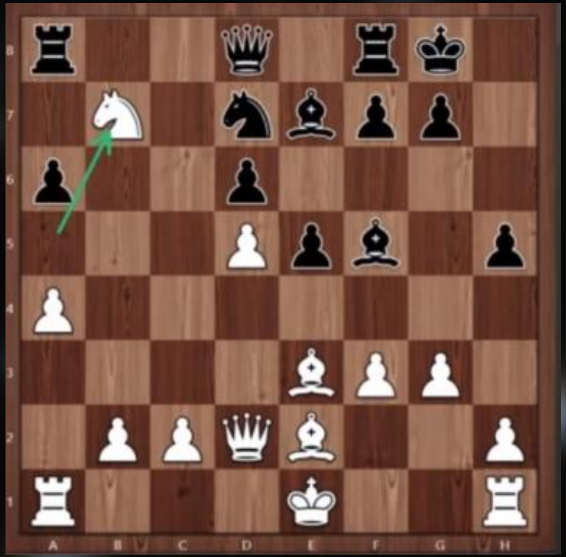
Sure! →



Hikaru Nakamura

Eris Li

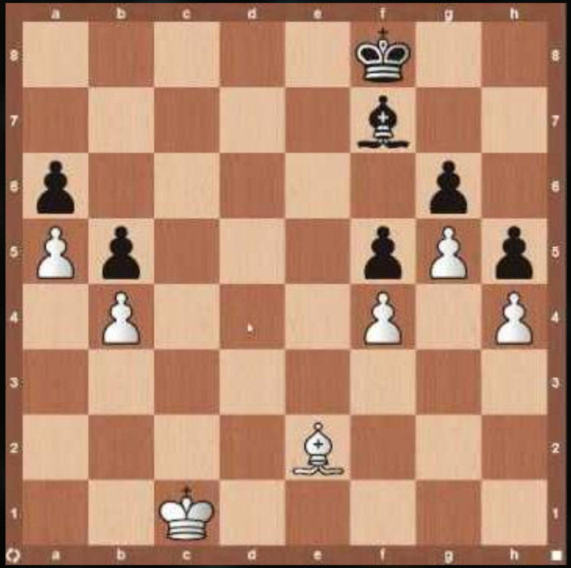
Magnus Carlsen



Sure! →

← Sure!

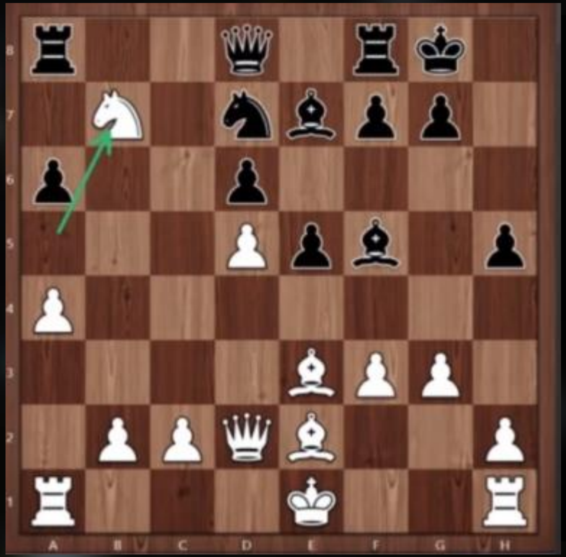
10 microseconds to 100 nano!



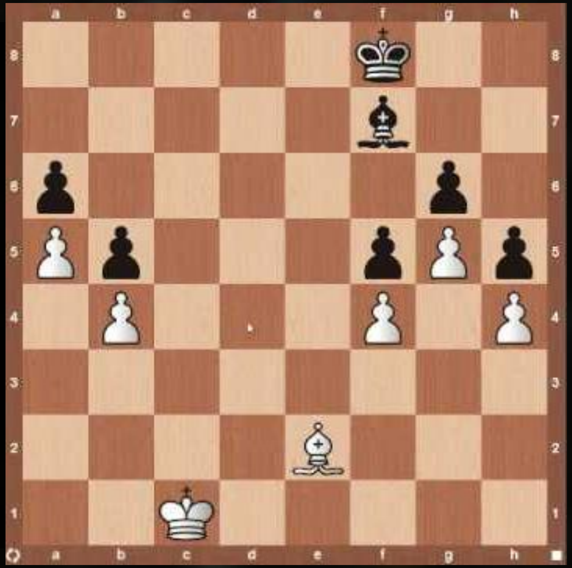
Hikaru Nakamura

Eris Li

Magnus Carlsen



Seems we have two chessboard
10 microseconds to 100 nano!



Hikaru Nakamura

Eris Li

"Normal people should see Naples before they die, but the great chess masters have to win the Wijk aan Zee tournament first of all"-Bent Larsen

a b c d e f g h
8 7 6 5 4 3 2 1

Sharing 1 Chessboard

OS

Processor Sharing Protocol (Scheduling)

Application-level programmer: *What is the processor sharing protocol? 1 move? 1 OPCODE? 1 Second?*

The Process Control (the Kernel): *Mind your own business! Whatever I like!!*

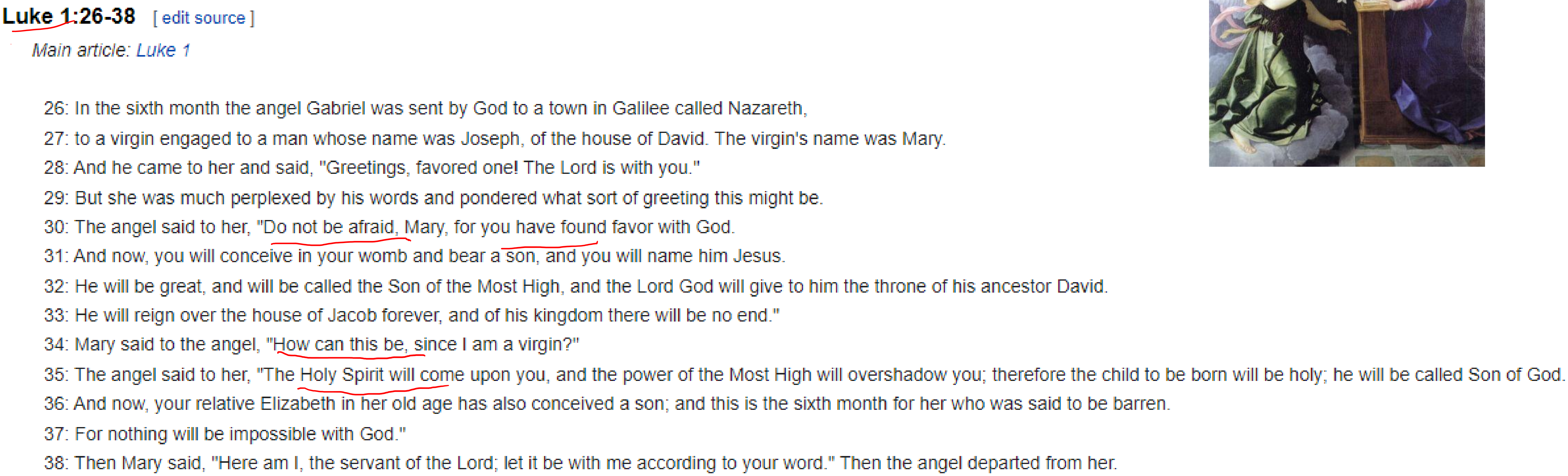
Process Manager

aka. Process Control

- Review: Program → Process → Run → Terminate
- Multiprocessing
  - HALT
  - Context Switch
  - Fork

The virgin birth of Jesus

Christian doctrine that Jesus was conceived by his mother, Mary, through the power of the Holy Spirit and without sexual intercourse.



Luke 1:26-38 [edit source]

Main article: Luke 1

26: In the sixth month the angel Gabriel was sent by God to a town in Galilee called Nazareth,

27: to a virgin engaged to a man whose name was Joseph, of the house of David. The virgin's name was Mary.

28: And he came to her and said, "Greetings, favored one! The Lord is with you."

29: But she was much perplexed by his words and pondered what sort of greeting this might be.

30: The angel said to her, "Do not be afraid, Mary, for you have found favor with God.

31: And now, you will conceive in your womb and bear a son, and you will name him Jesus.

32: He will be great, and will be called the Son of the Most High, and the Lord God will give to him the throne of his ancestor David.

33: He will reign over the house of Jacob forever, and of his kingdom there will be no end."

34: Mary said to the angel, "How can this be, since I am a virgin?"

35: The angel said to her, "The Holy Spirit will come upon you, and the power of the Most High will overshadow you; therefore the child to be born will be holy; he will be called Son of God.

36: And now, your relative Elizabeth in her old age has also conceived a son; and this is the sixth month for her who was said to be barren.

37: For nothing will be impossible with God."

38: Then Mary said, "Here am I, the servant of the Lord; let it be with me according to your word." Then the angel departed from her.

Creating a New Process

- Disclaimer: Use of real-world terms to explain the topics. Not necessary my viewpoints!
  - Parent, Mother, Child, Children, …
  - Create or give birth to a child, force the child to work, exit, …

Parent vs. Child Process

System Calls: `fork()` in `unistd.h`

Only an existing process can create a new process. Because somebody should do the system call!

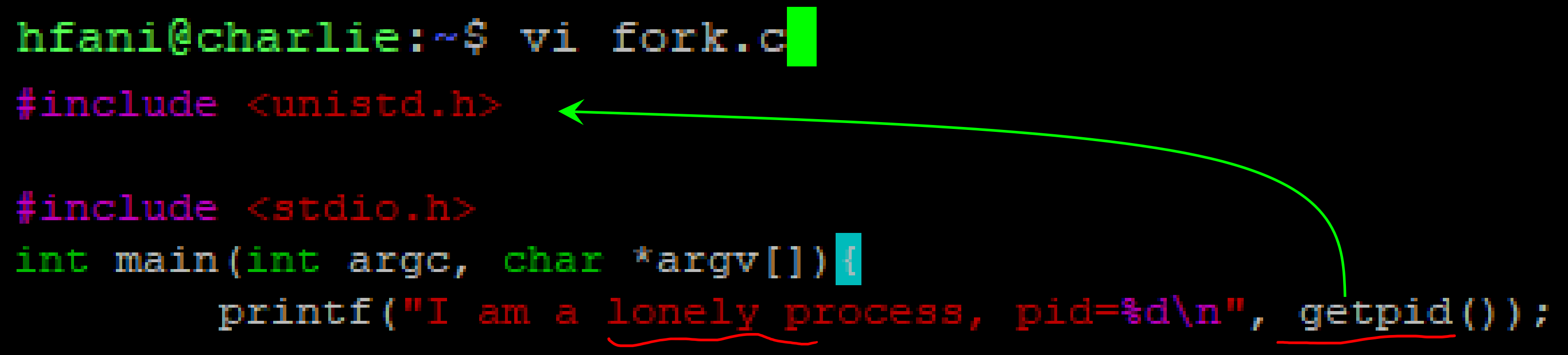
Creating a New Process

System Calls: `fork()` in `unistd.h`

```
#include <unistd.h>
pid_t fork(void);
```

Returns: 0 in child, PID of child in parent, –1 on error

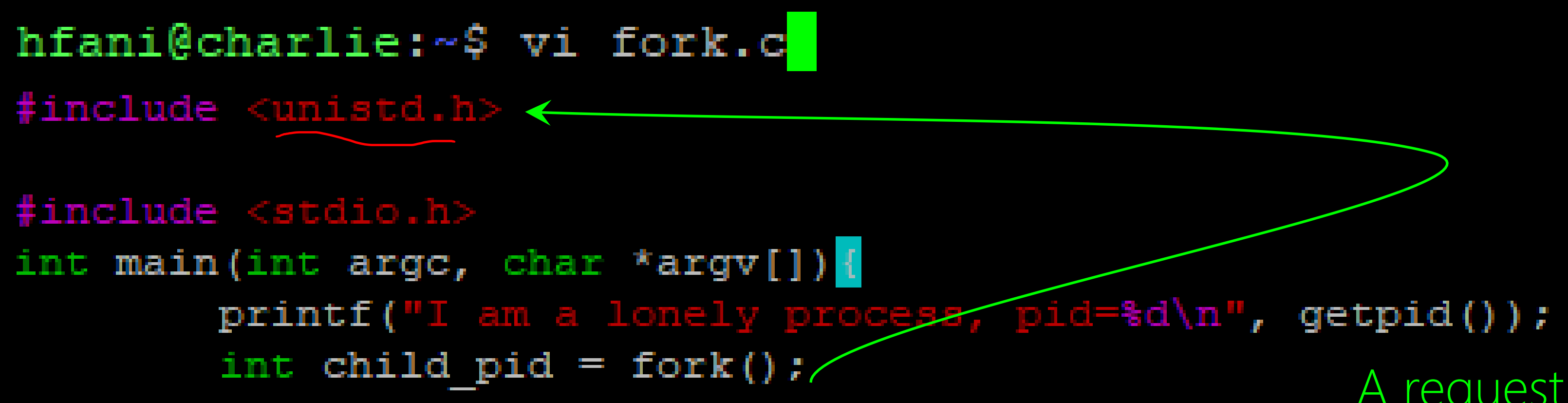
Me (my program) and the Kernel
A dialog



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>

#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
```

Compile Time Analysis

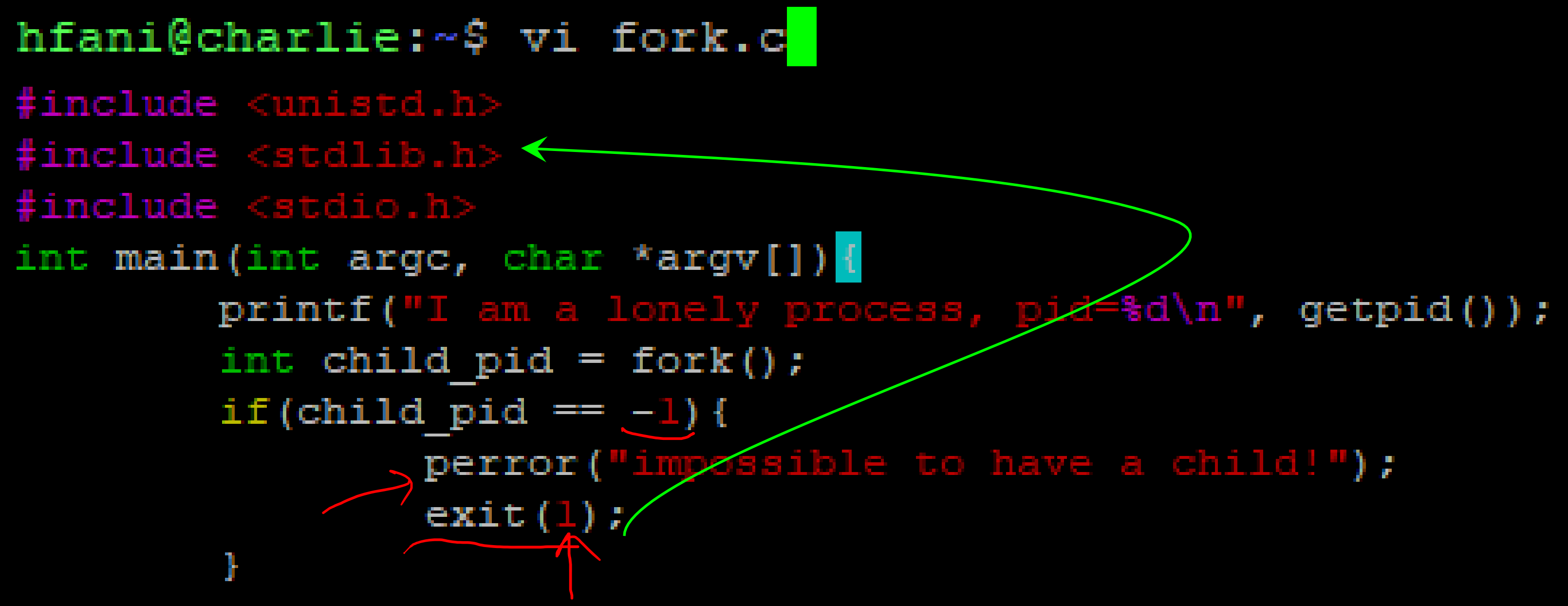
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>

#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
```

A request to ~~adopt~~ or give birth to a new child

Compile Time Analysis



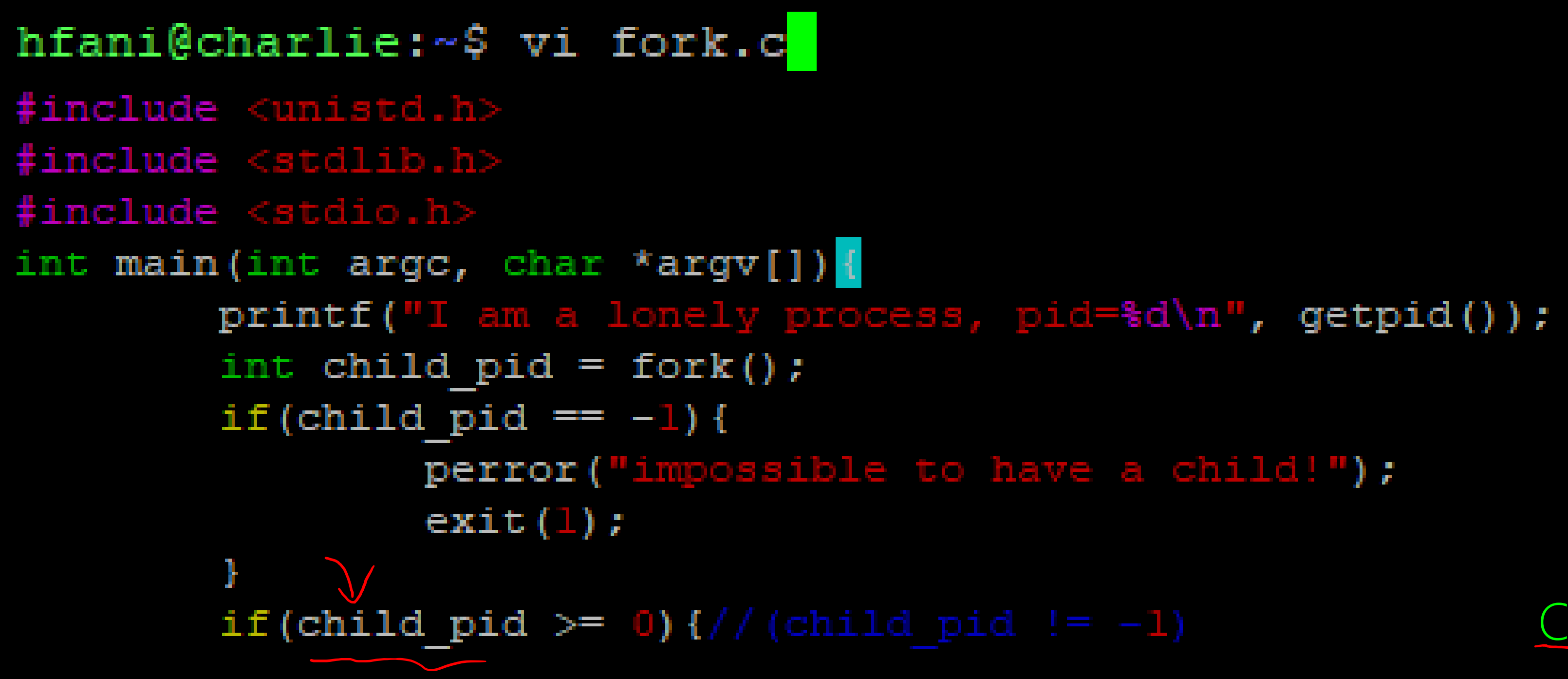
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
```

-1 on error in having a child

Exit the process with an error status

Nonzero!

Compile Time Analysis

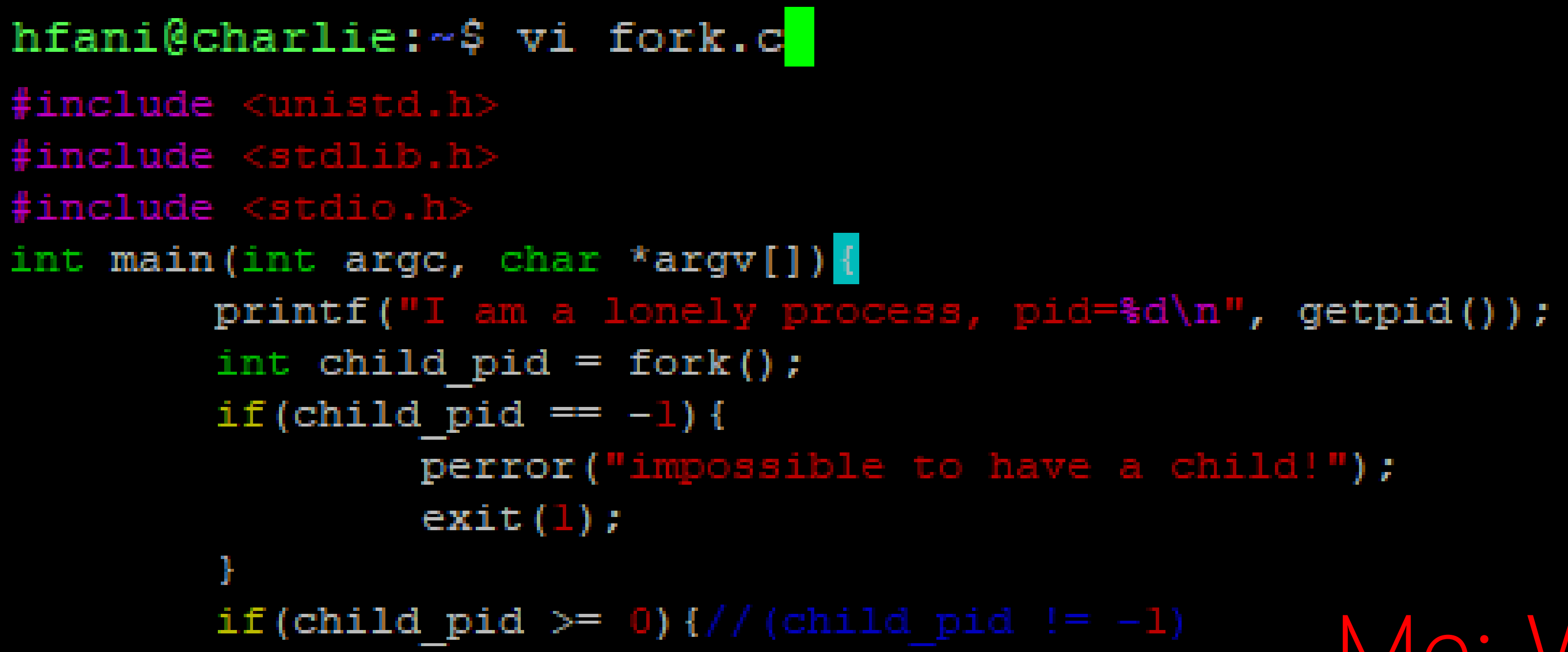


```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

Congratulation! You become a parent.

Here is the pid of your child.

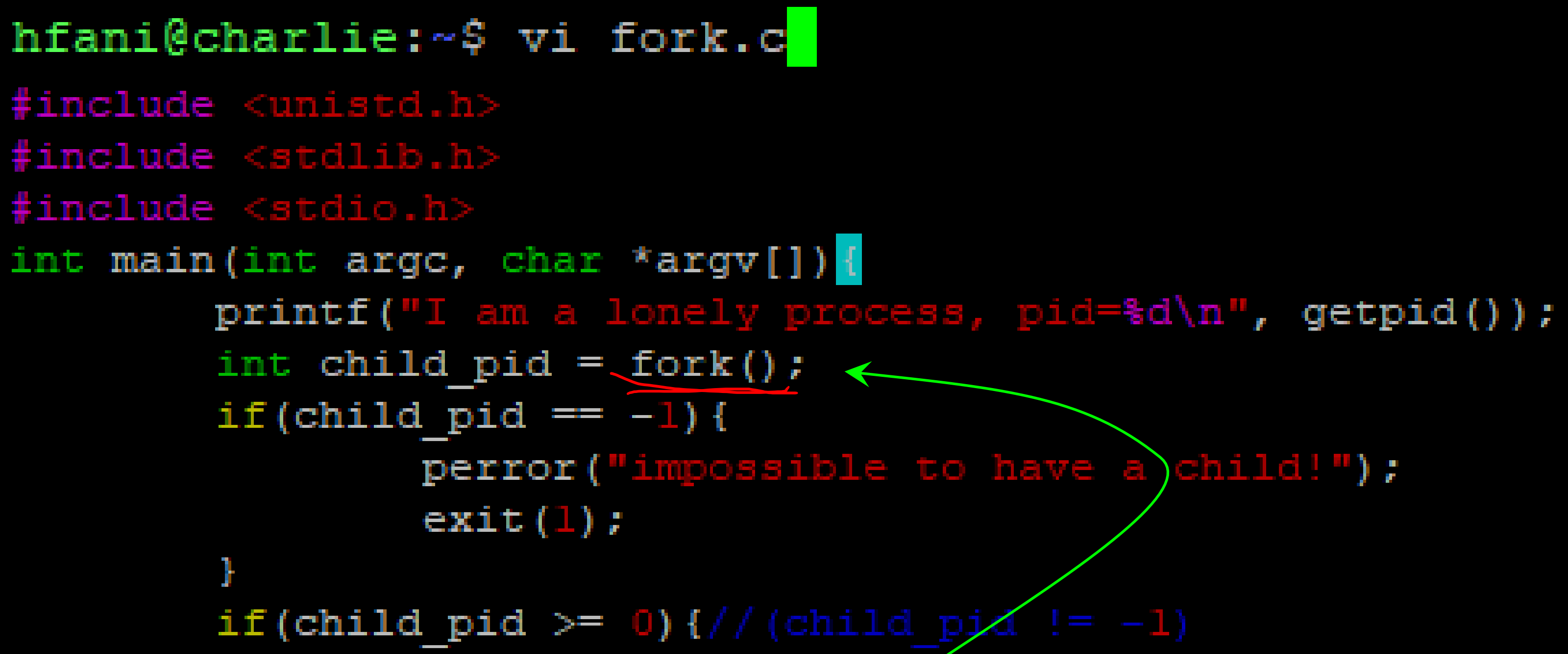
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

Me: Where is my child?!

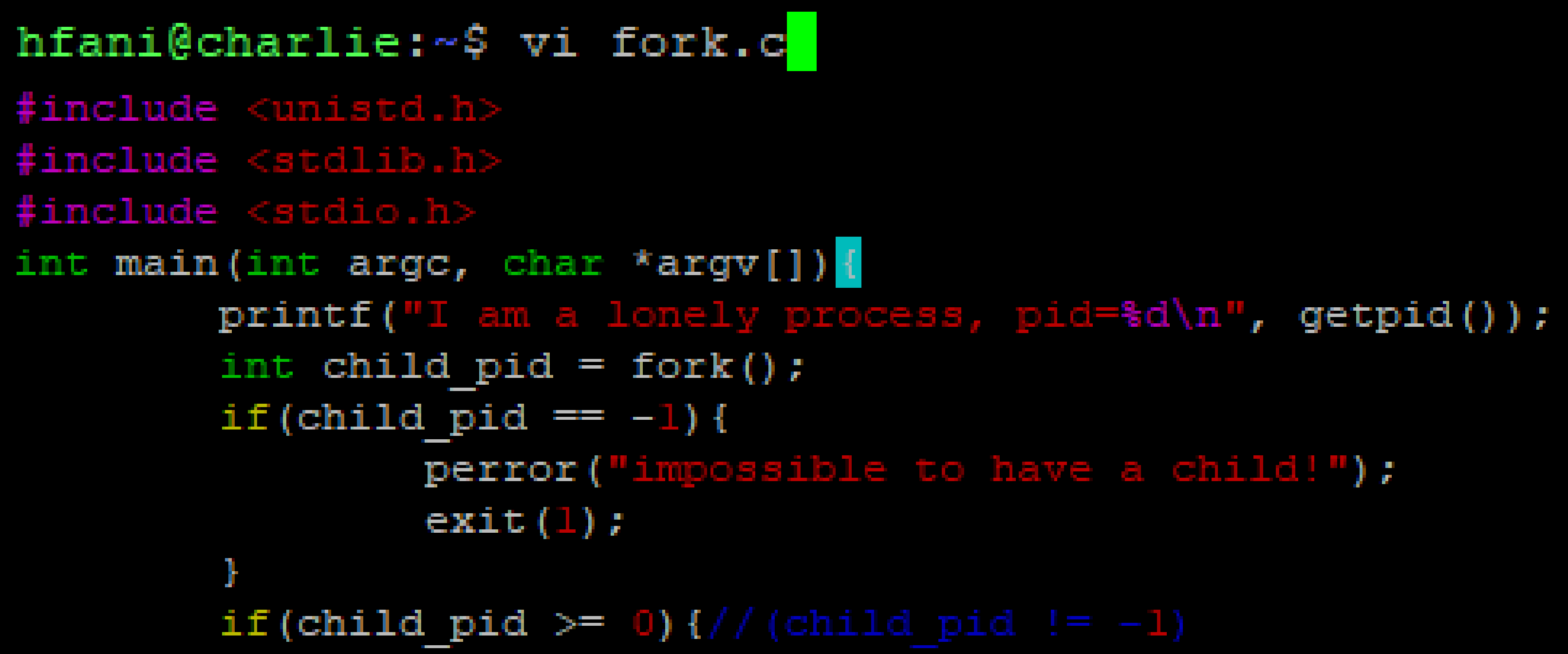
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

Kernel:
Your child was born here.
At runtime, we promise that your child is inside the memory somewhere.

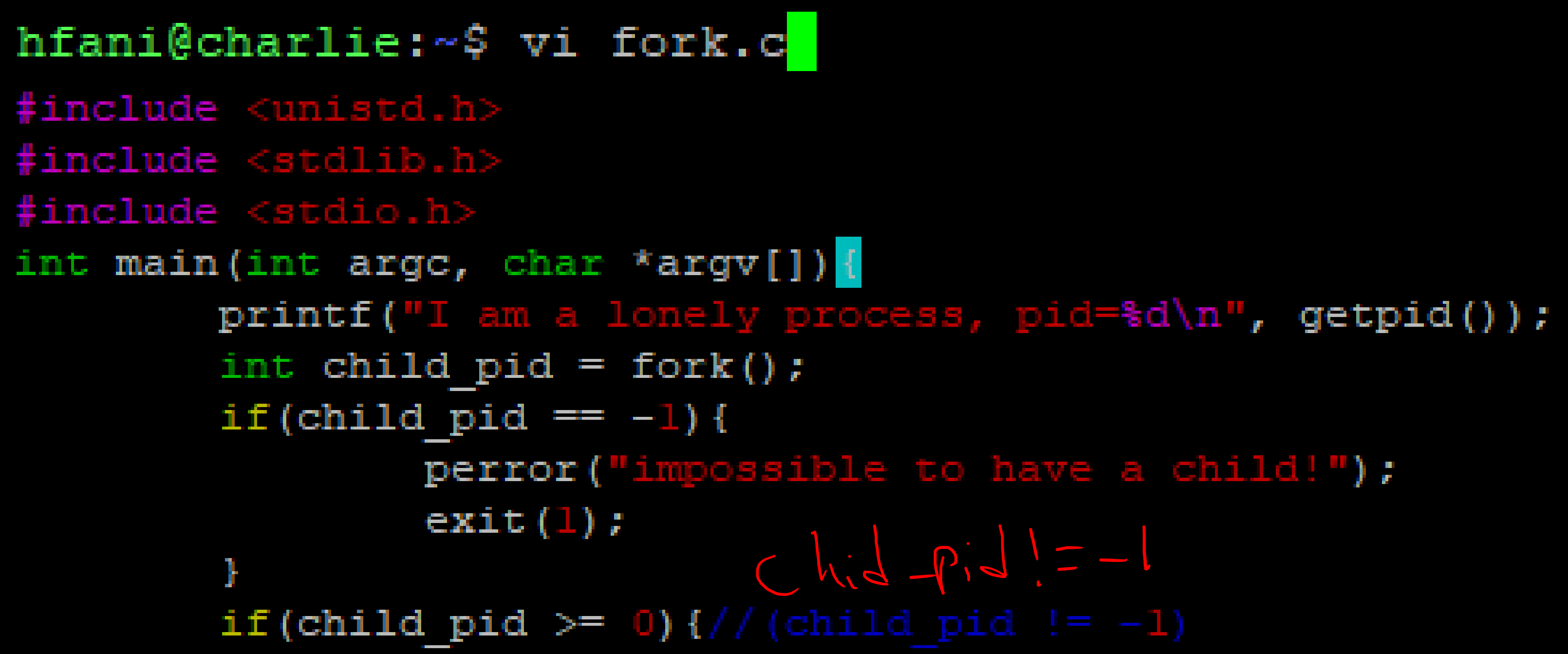
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

Me:
How does the child look like? Is the child girl or boy? What's the color of eye? Blue? ...

Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

chid_pid!=-1

Kernel:

What do you expect?! Your child is like you.

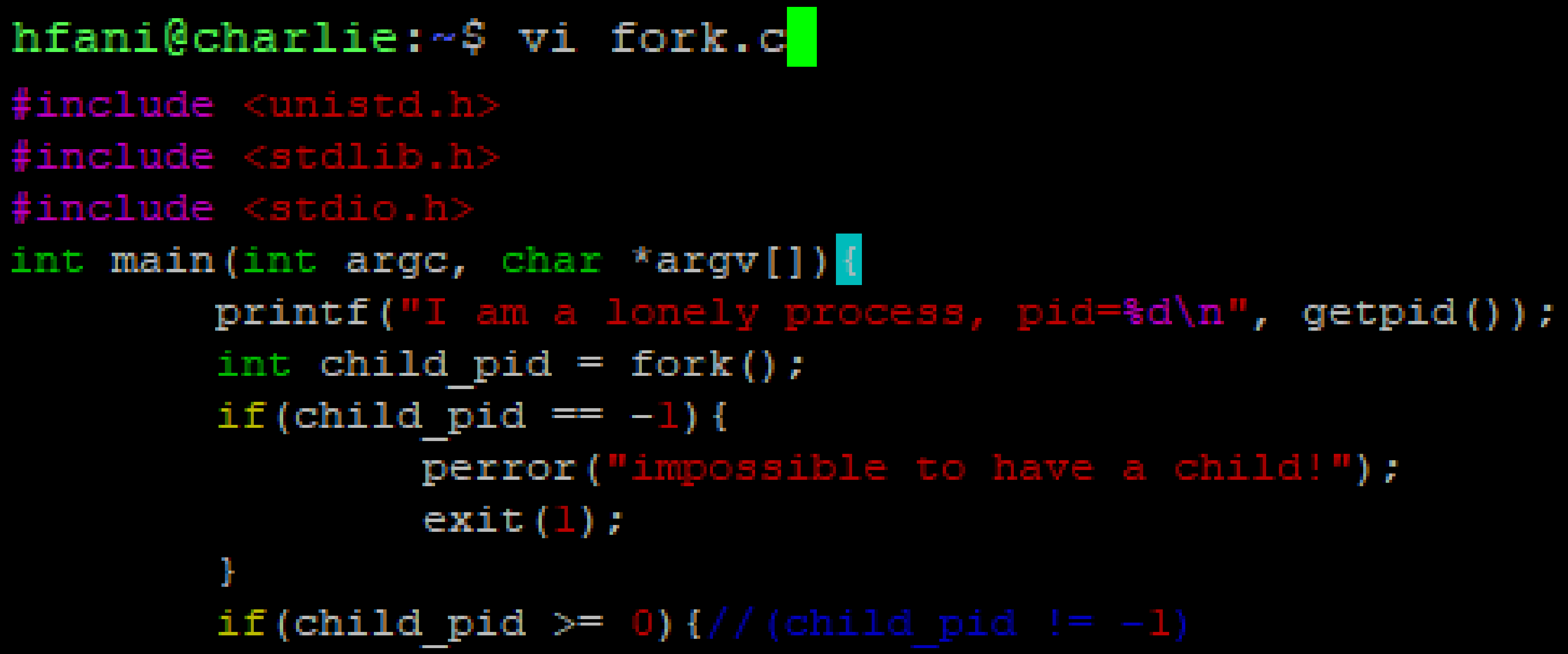
Oh, the child is exactly a copy of you (clone) indeed.

Same age, same gender, same color, …

As a matter of fact, it is very hard to distinguish yourself from your child.

Which one is my hat?, The Prestige (2006) - Christopher Nolan
https://www.youtube.com/watch?v=3utO5bNkdvg

Compile Time Analysis

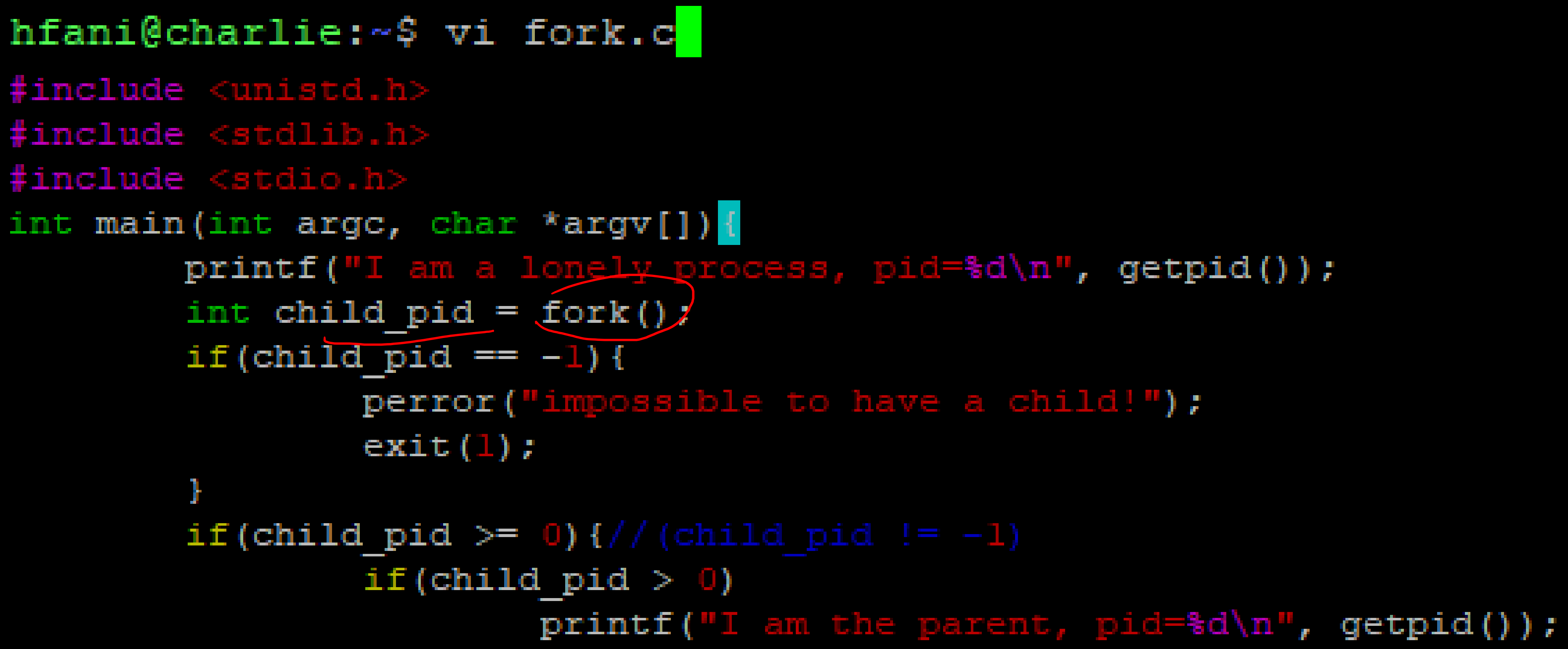


```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
```

Me:

There should be a way that tells me is me and the child is the child.

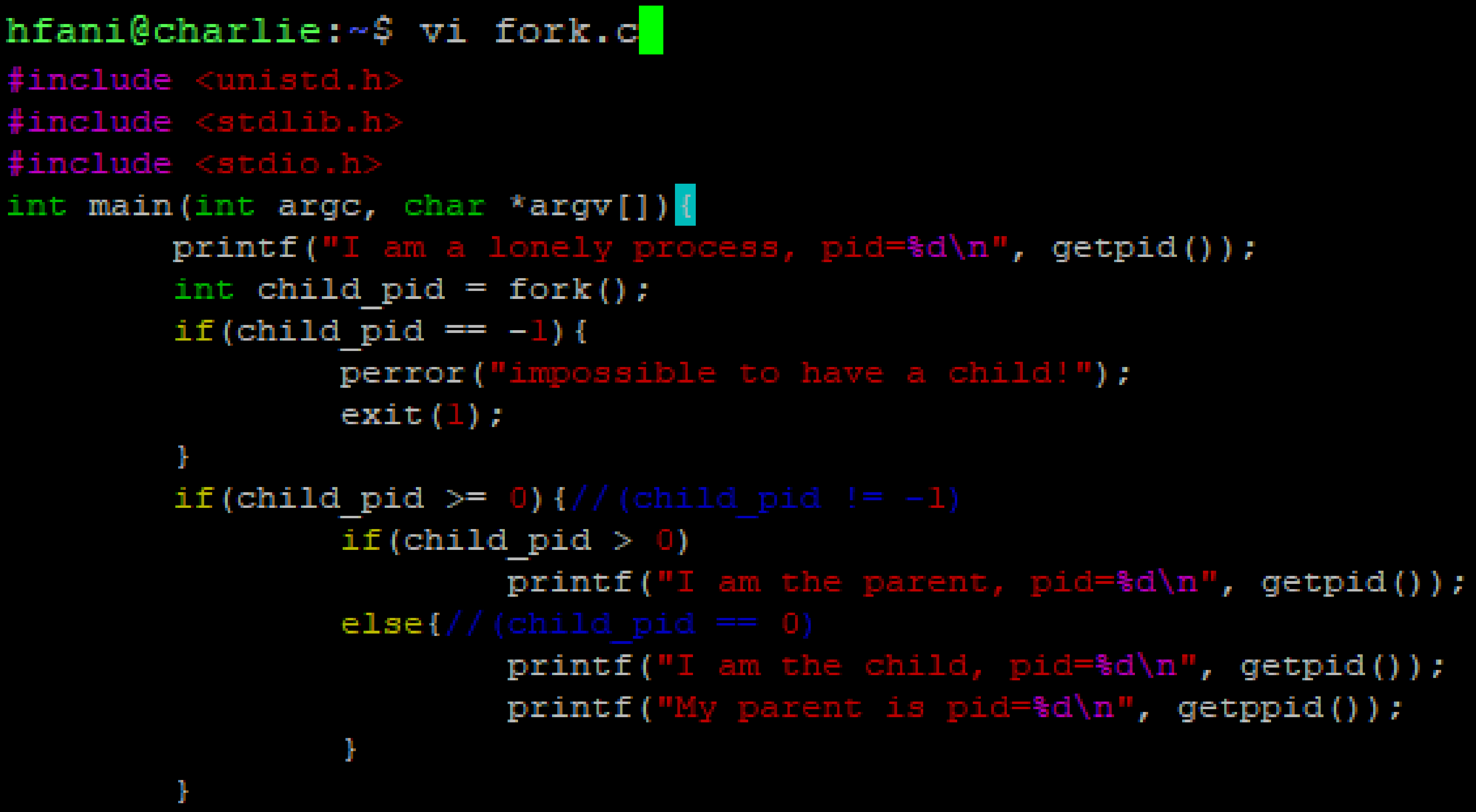
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
```

Kernel:
If the child_pid is a non-zero positive number, it means you're are the parent.
Because we only give children's pid to their parents.

Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
```

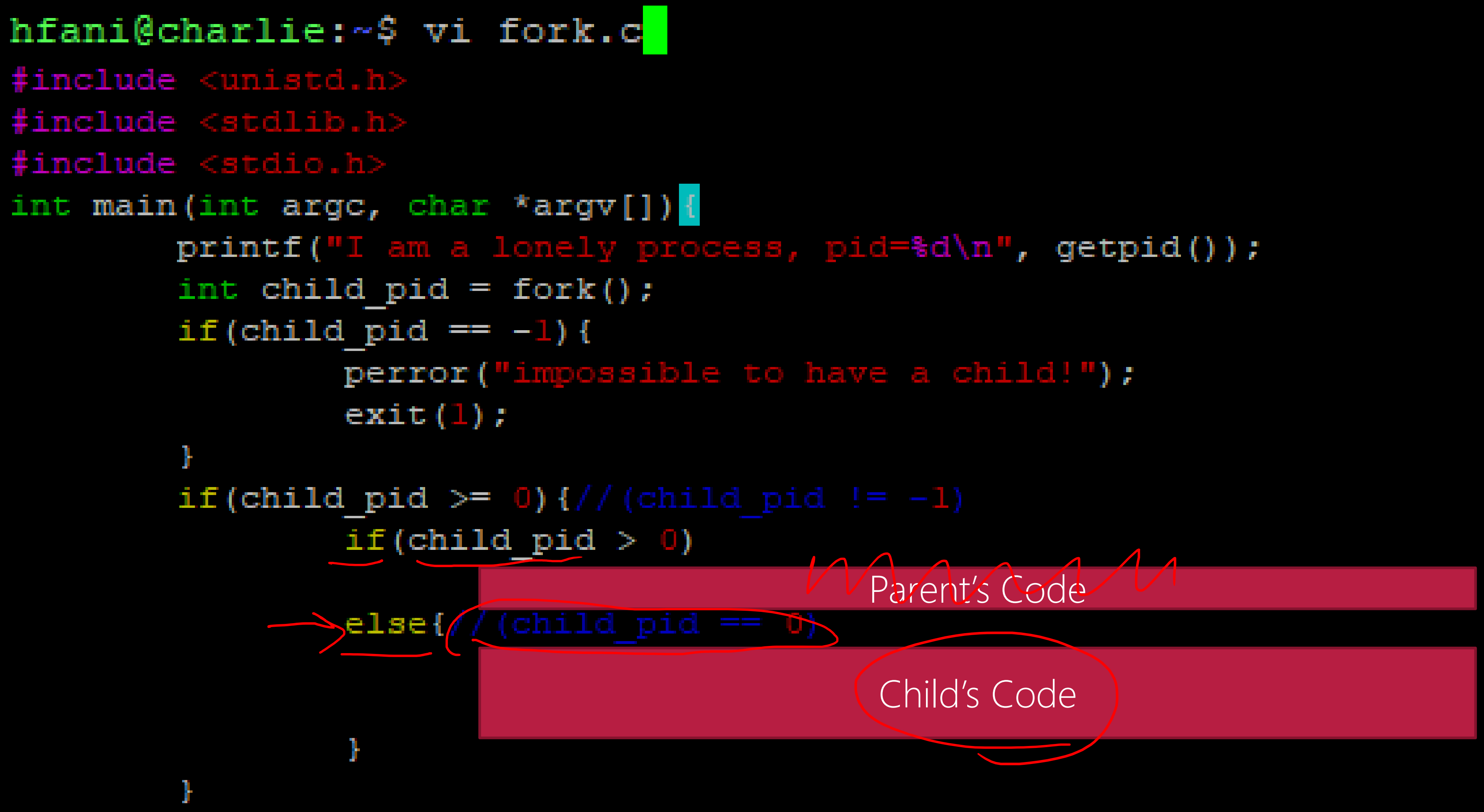
Kernel:

If the child_pid is 0, it means you're are the child.

If you want to know your pid, use `getpid()` system call.

If you want to know your parent pid, use `getppid()` system call.

Compile Time Analysis



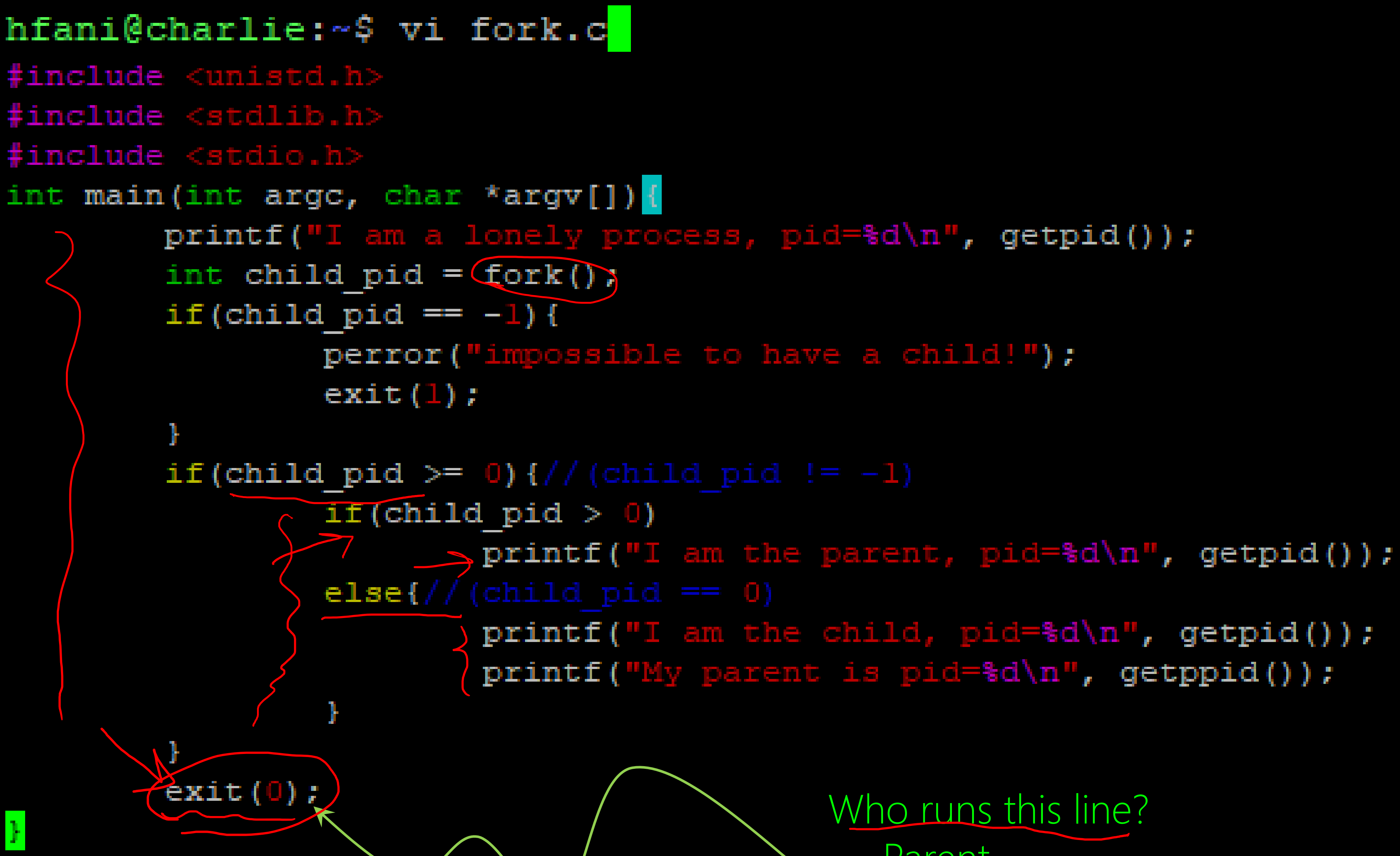
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)

                else{//(child_pid == 0)


                }
        }
```

Parent's Code

Child's Code

Kernel:
If the child_pid is 0, it means you're are the child.
If you want to know your pid, use `getpid()` system call.
If you want to know your parent pid, use `getppid()` system call.

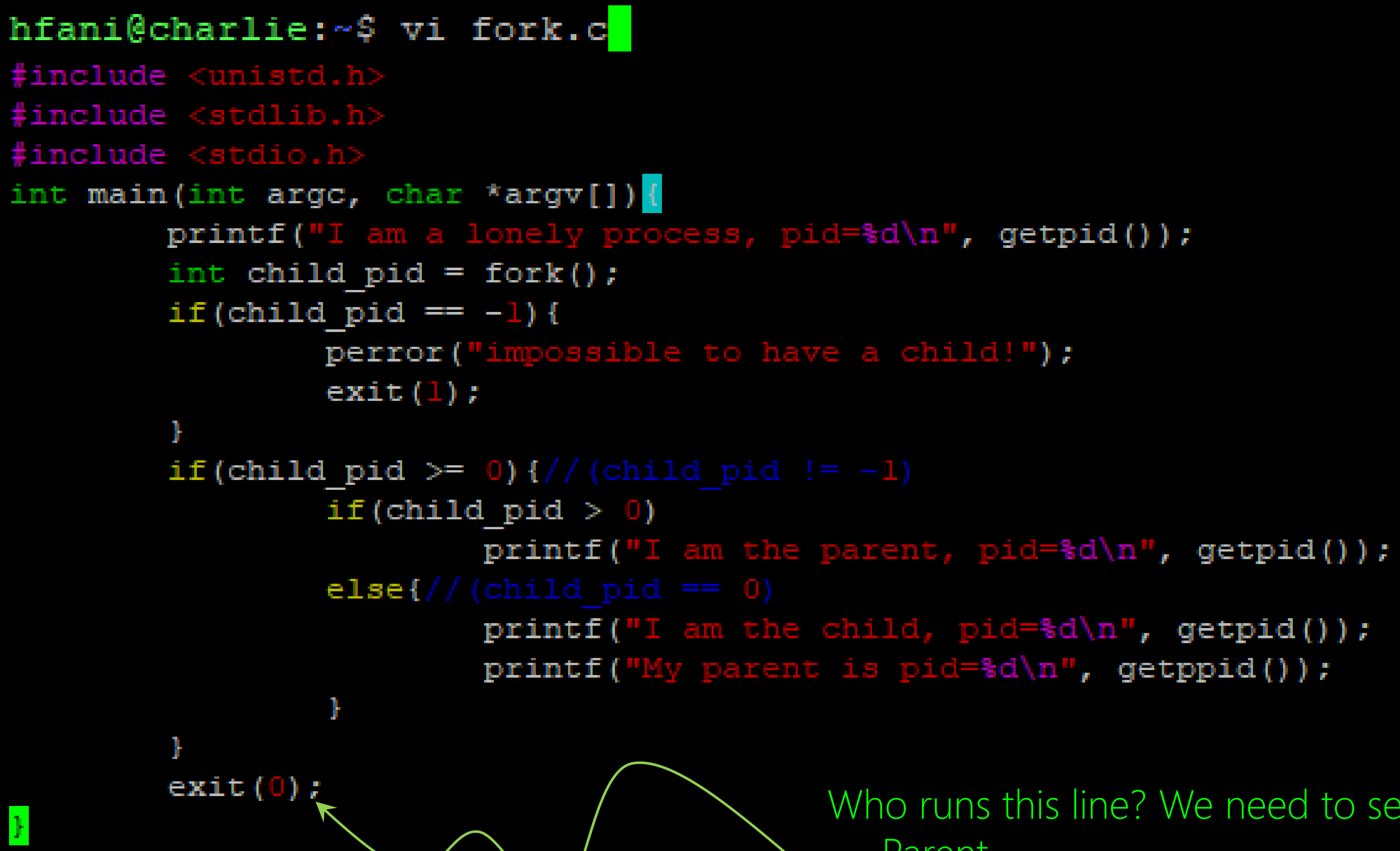
Compile Time Analysis



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

Who runs this line?

- Parent
- Child
- [✓ Both]
- None



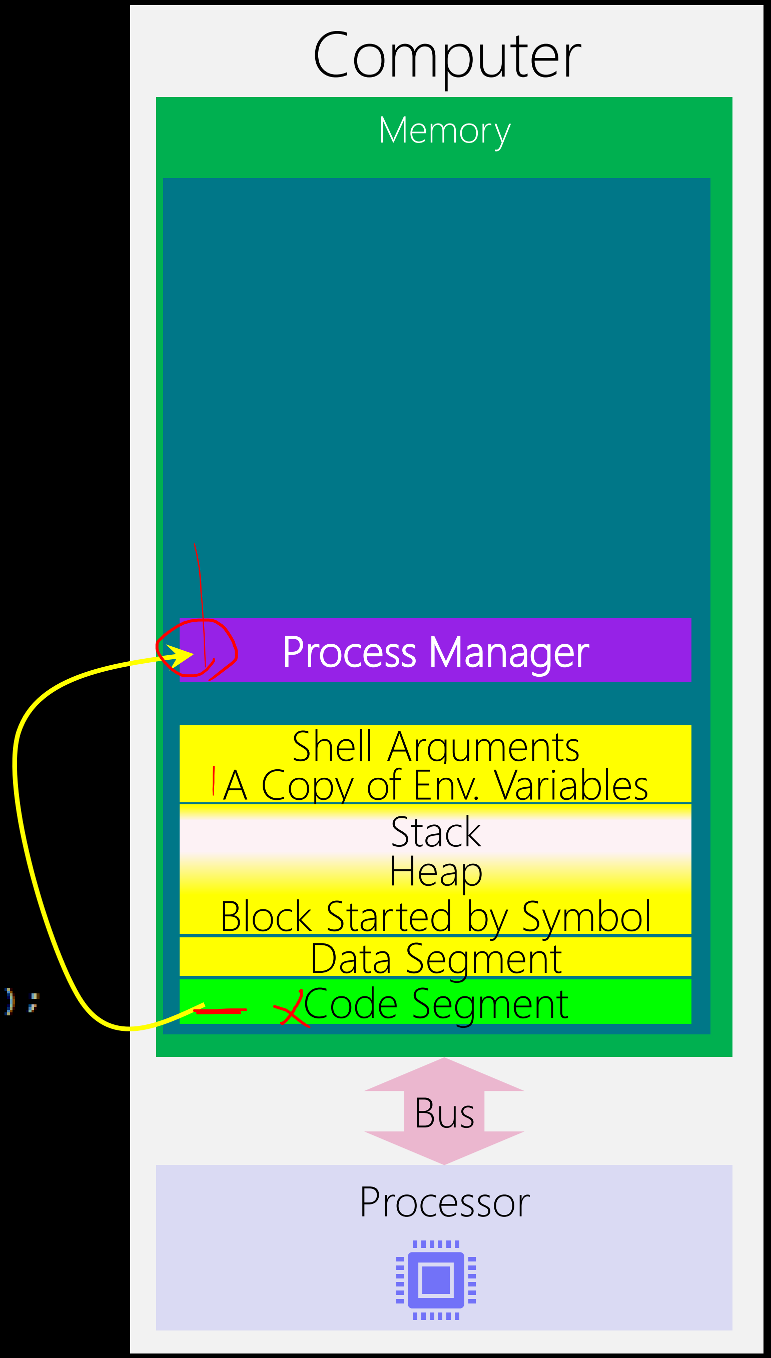
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

Compile Time Analysis

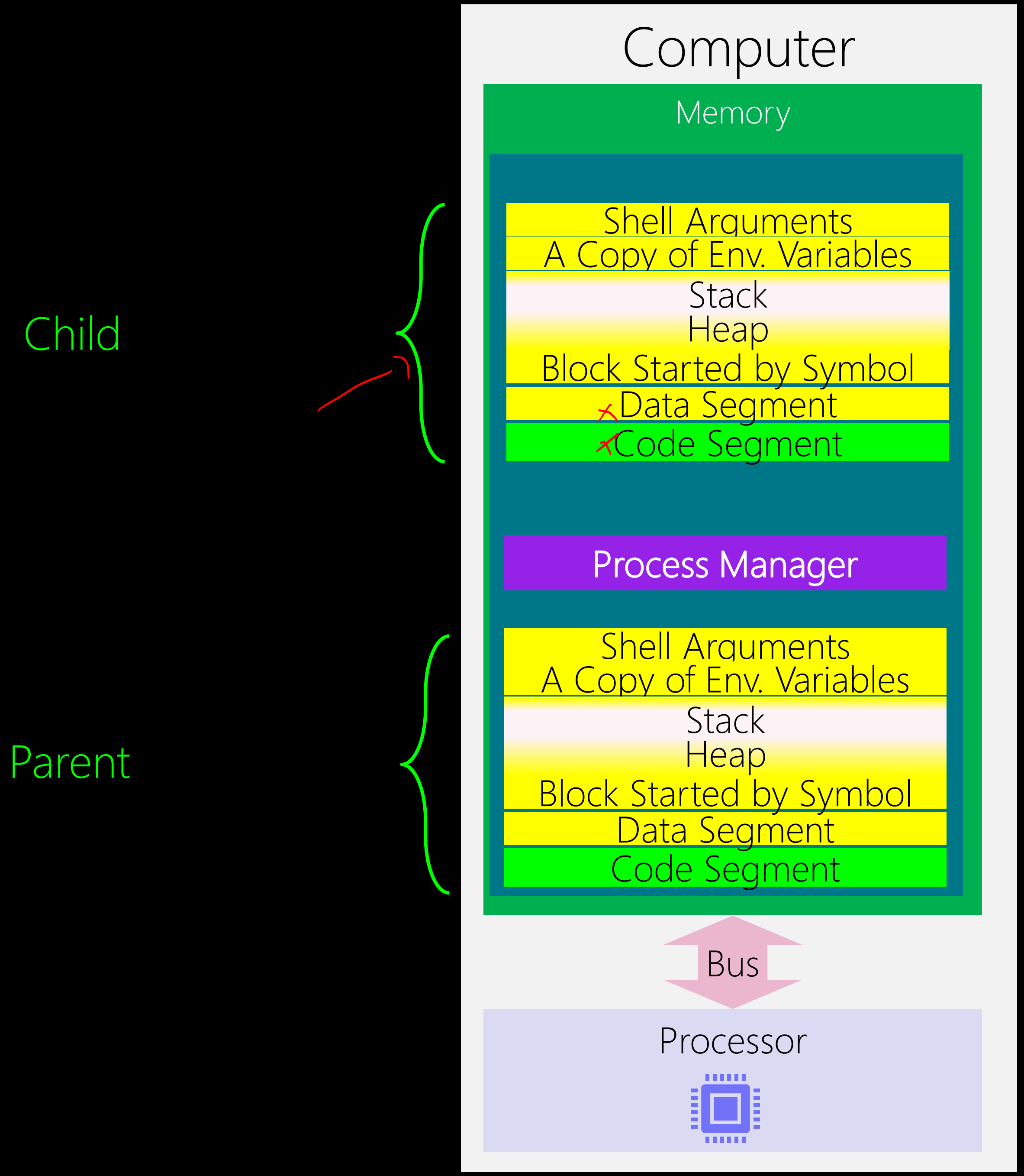
Who runs this line? We need to see what's going on in runtime.

- Parent
- Child
- Both
- None

```
int child_pid = fork();
```



Exact copy at `fork()`



Child

Computer

Memory

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

Data Segment

Code Segment

Process Manager

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

Data Segment

Code Segment

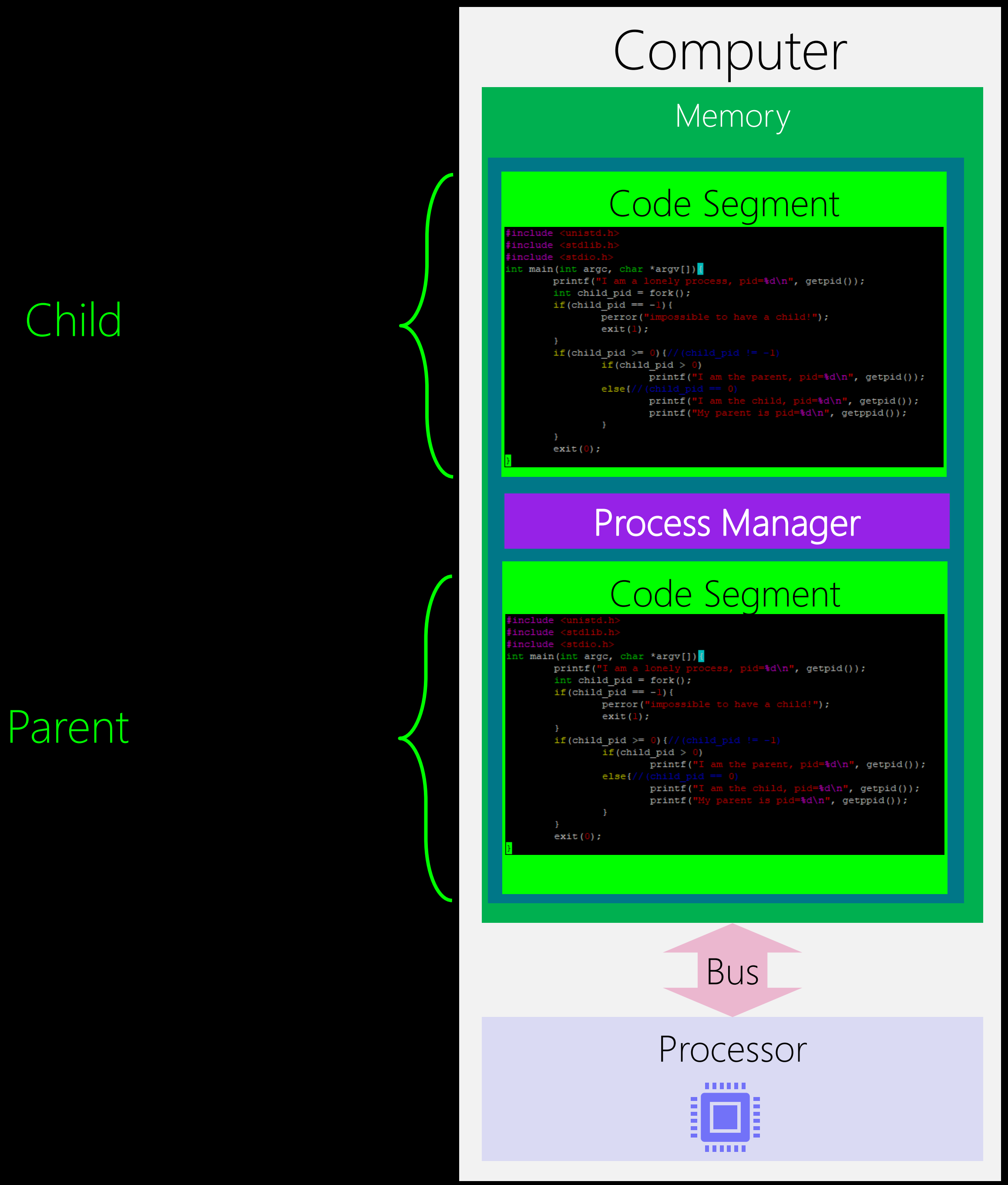
Parent

Bus

Processor

Any change by the child is in the child copy

Any change by the parent is in the parent copy



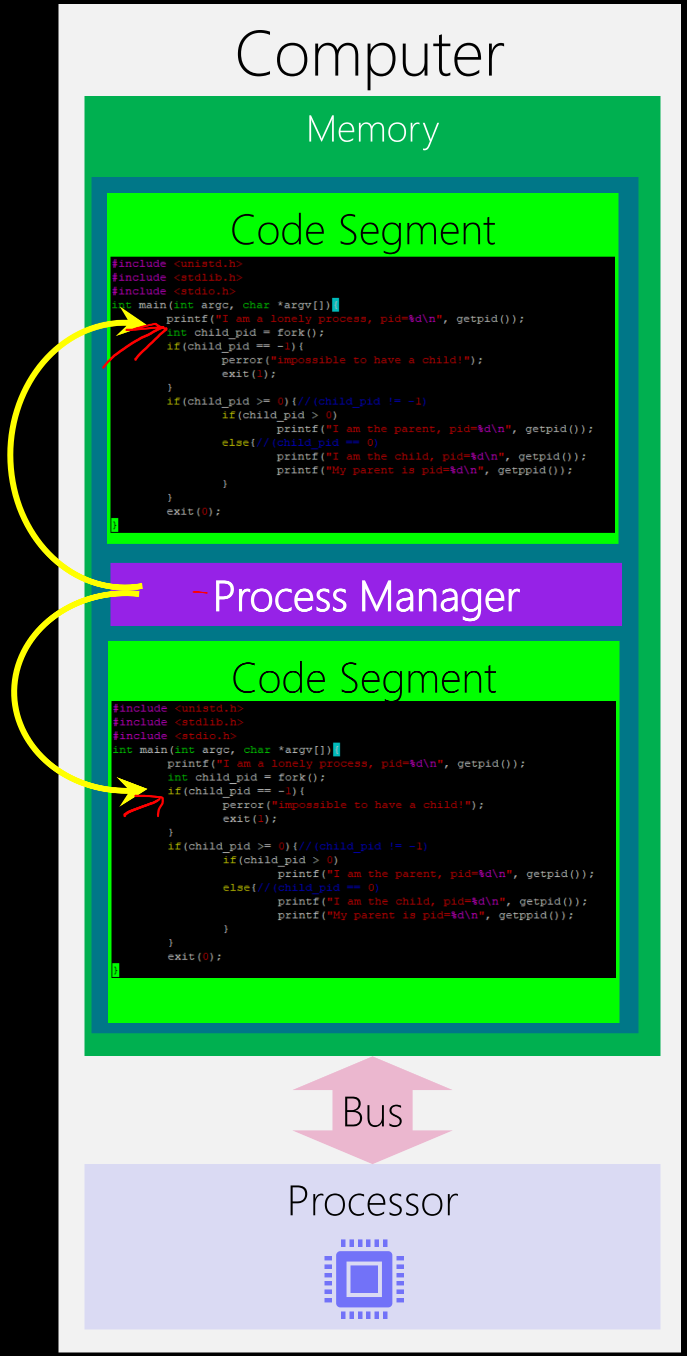
```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

If we zoom in to the code segment, which line is the current line in child and parent?

Child

Parent



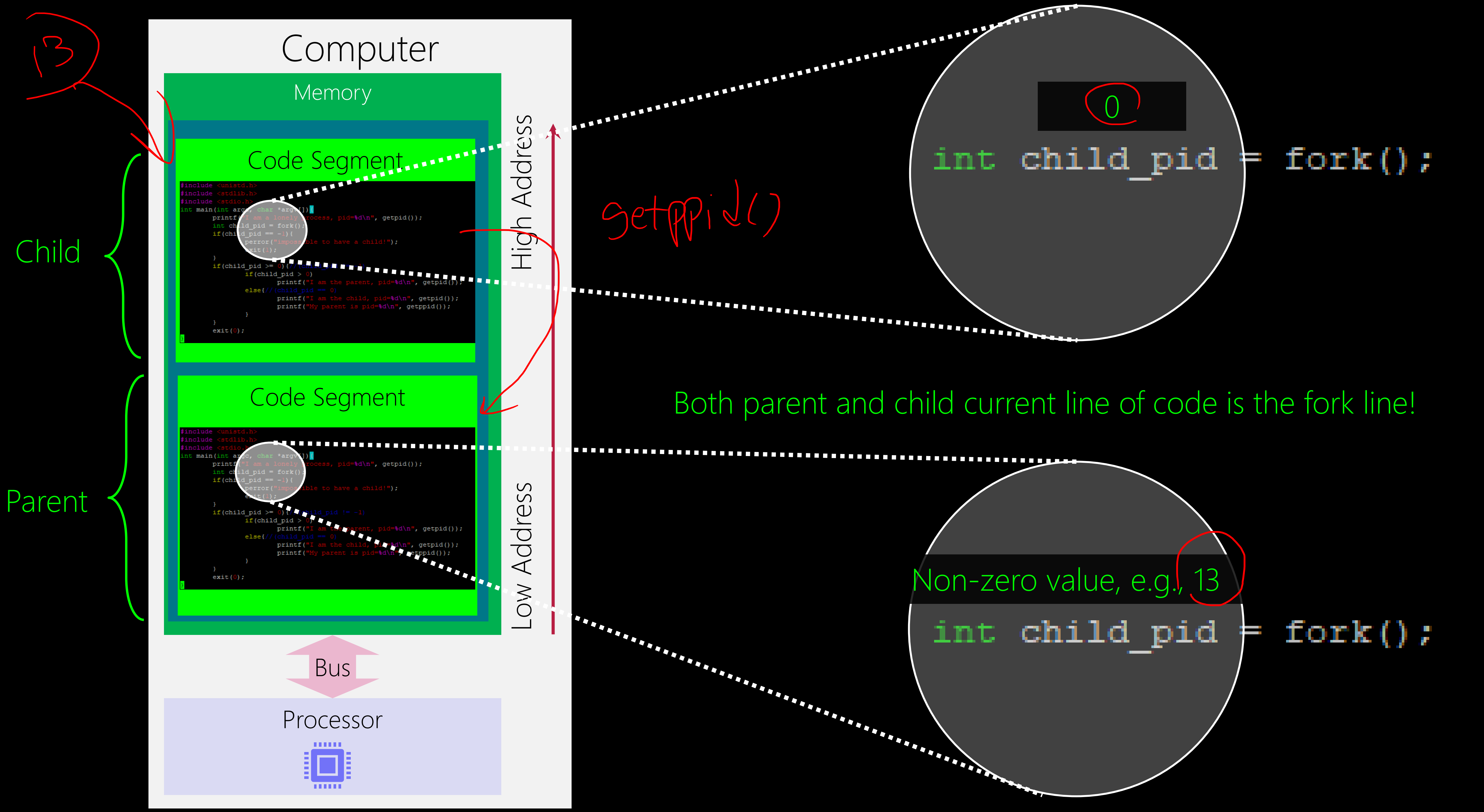
```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

0 at `fork()` for child

This system call is amazing as it returns two values to two different processes!

child_pid at `fork()` for parent



```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

```
int child_pid = fork();
```

Both parent and child current line of code is the fork line!

```
int child_pid = fork();
```

Parent (pid=4)

13

```
int child_pid = fork();
```

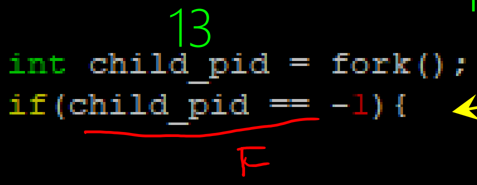
Child (pid=13)

0

```
int child_pid = fork();
```

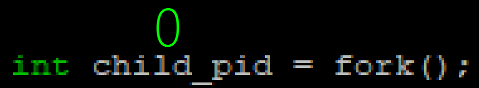
Parent (pid=4)

Processor

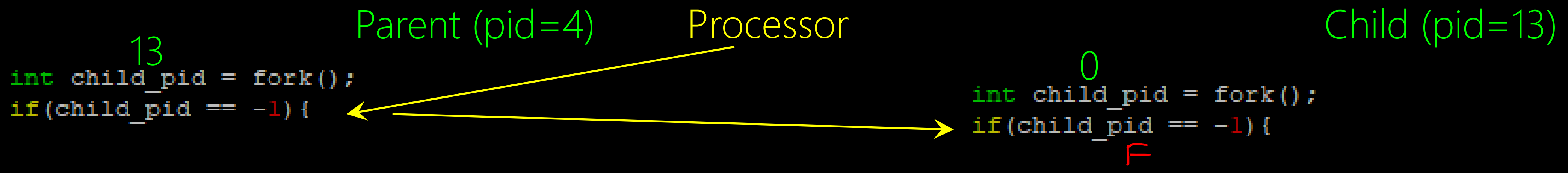


```
int child_pid = fork();
if(child_pid == -1){
```

Child (pid=13)



```
int child_pid = fork();
```

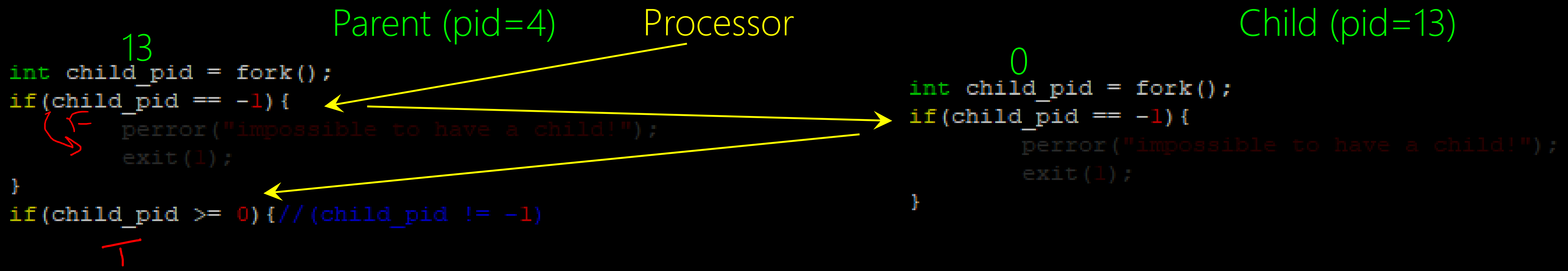


Parent (pid=4)

```
int child_pid = fork();
if(child_pid == -1){
```

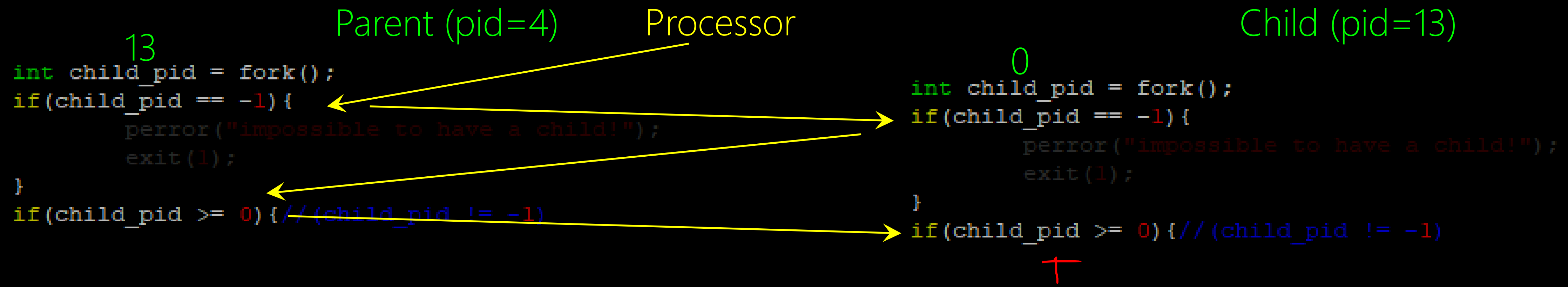
Child (pid=13)

```
int child_pid = fork();
if(child_pid == -1){
```



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
```



Parent (pid=4)

Processor

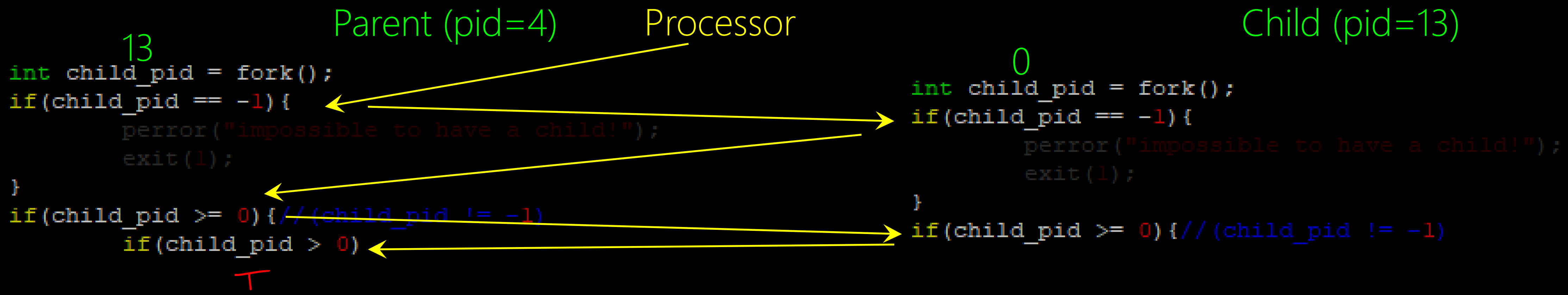
Child (pid=13)

13

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
```

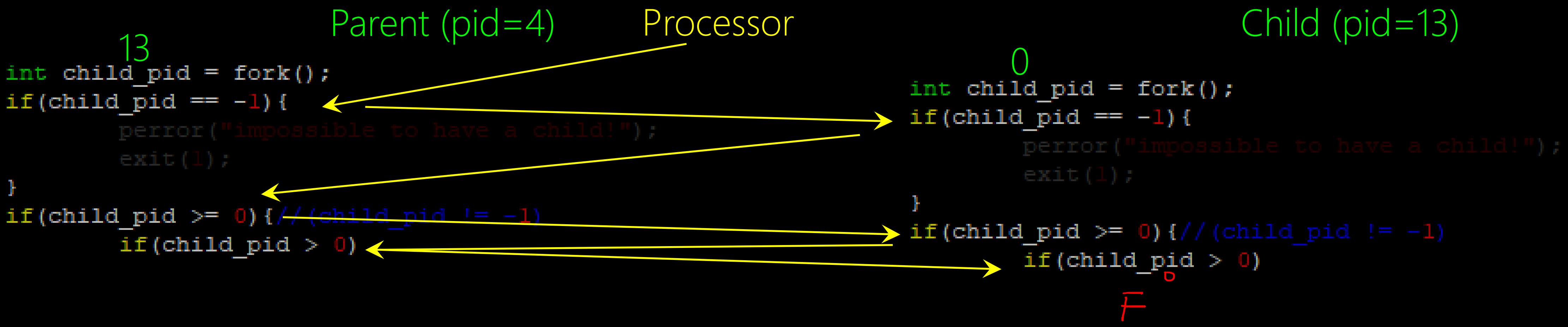
0

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
```



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
```



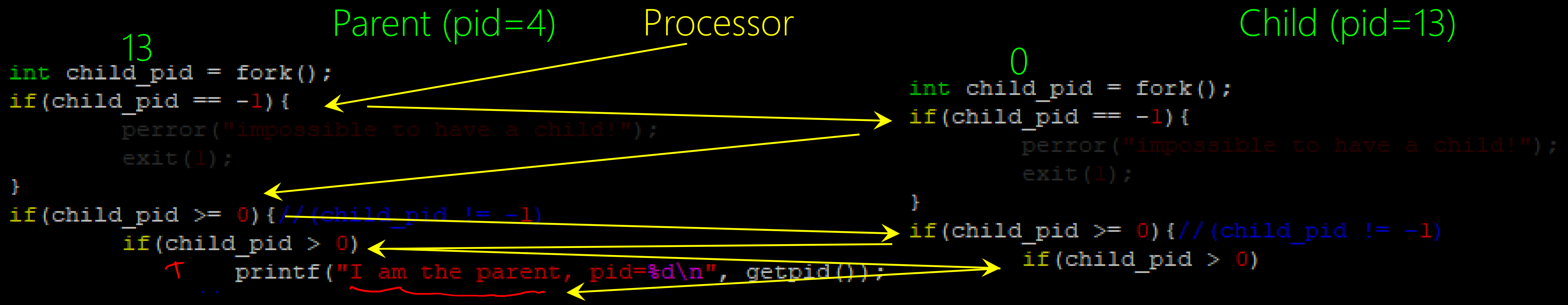
Parent (pid=4)

Processor

Child (pid=13)

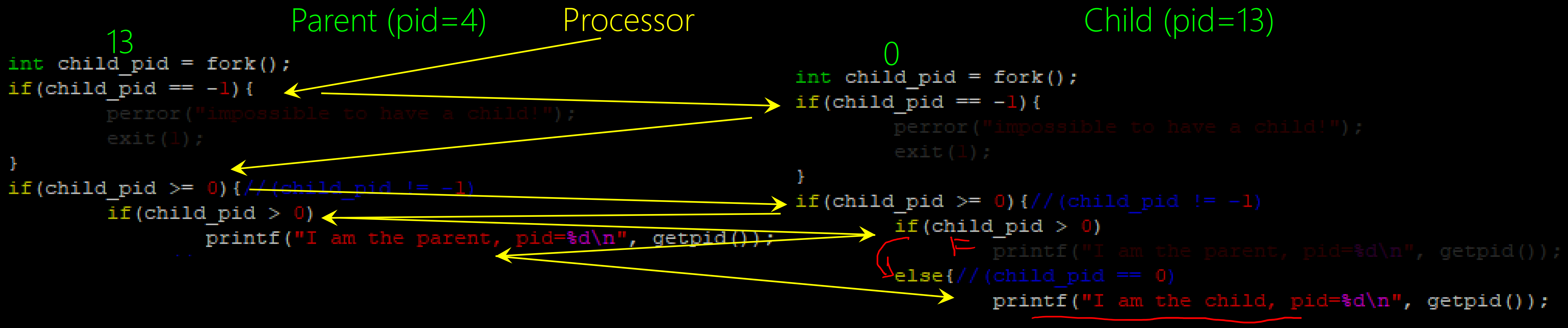
```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
```



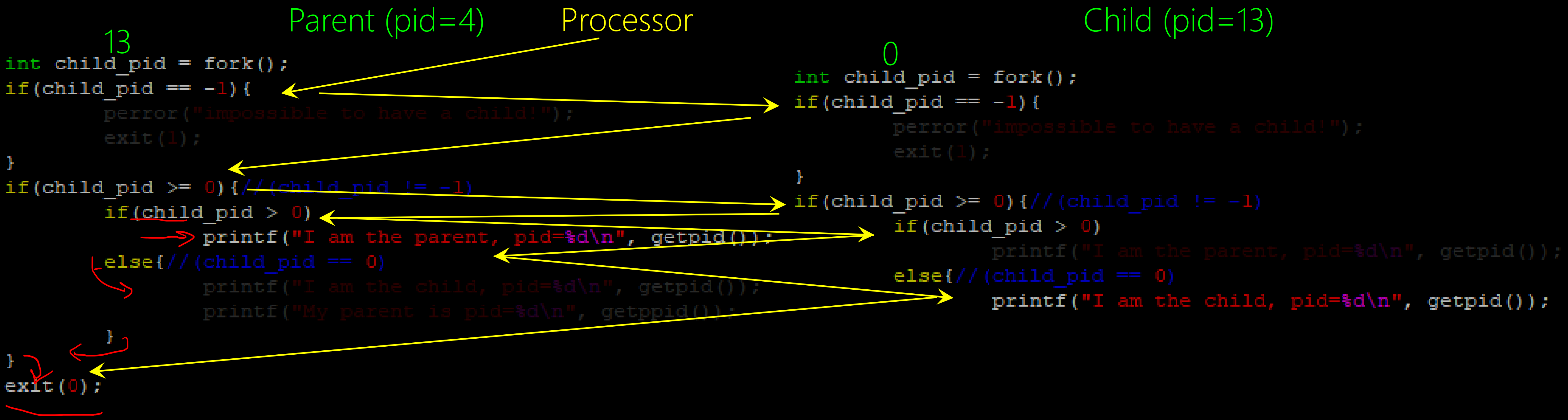
```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
```



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
```

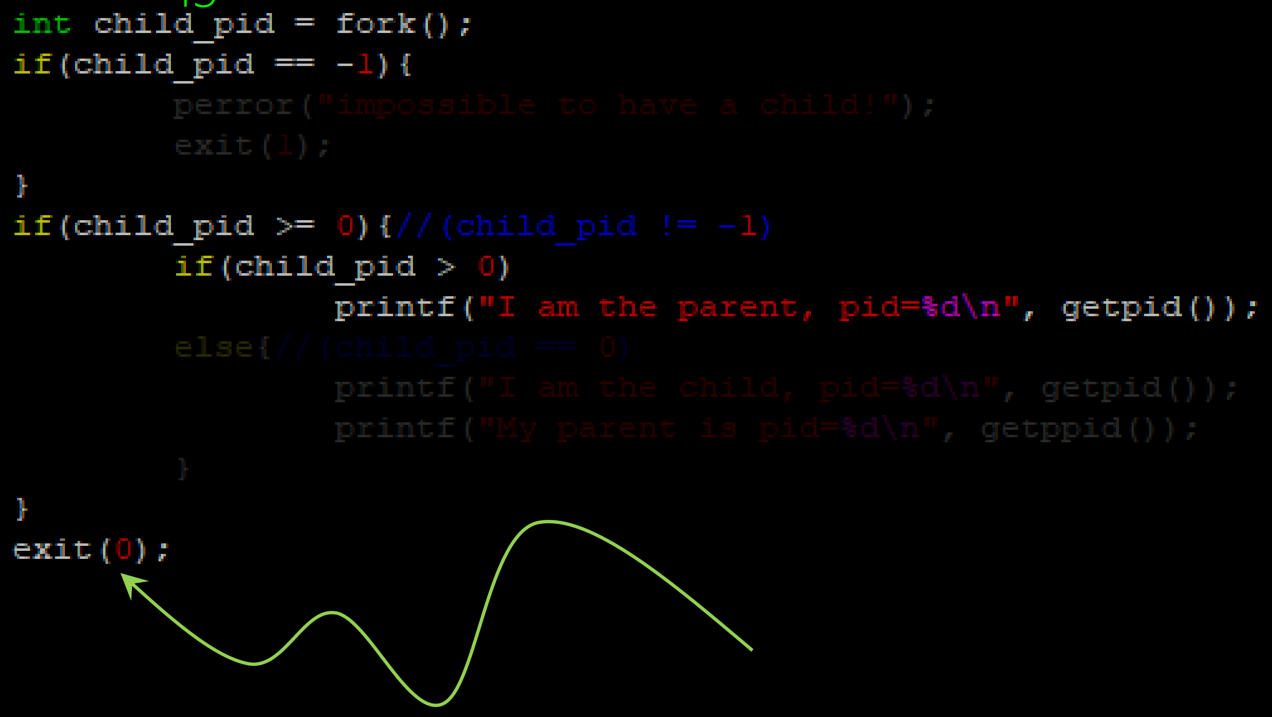


```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```

```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
```

Parent (pid=4)

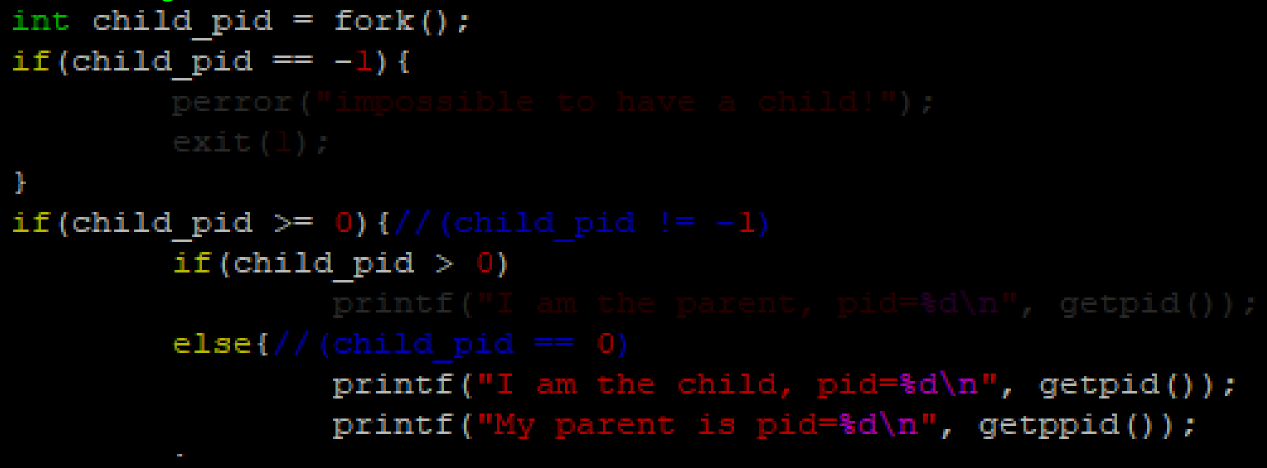
13



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```

Child (pid=13)

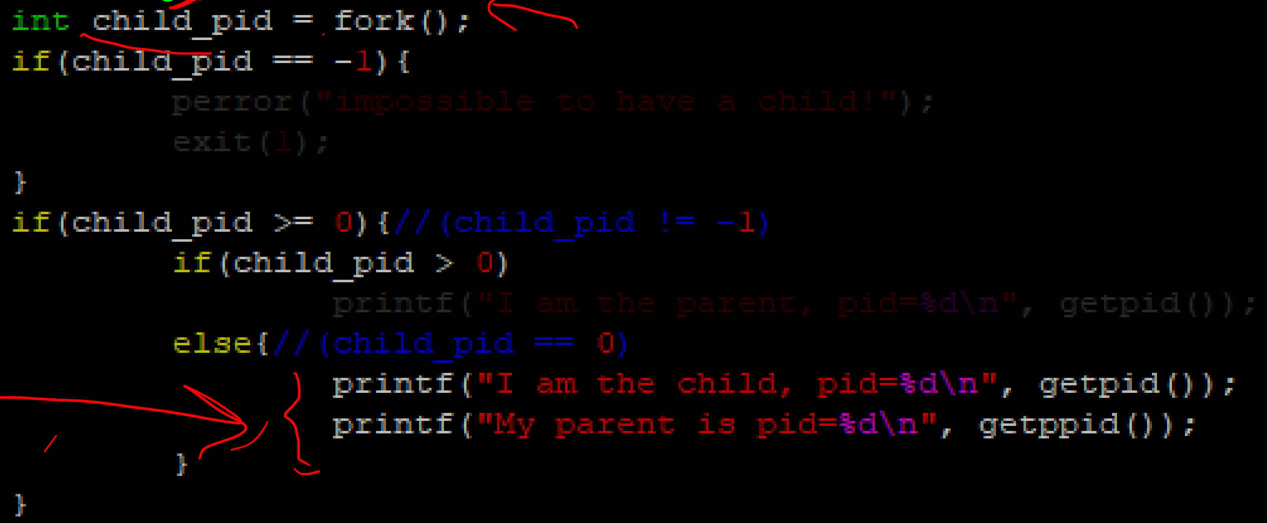
0



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
```

Child (pid=13)

0



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
```

X = fork()

getppid()

Orphan

~~No Parent → Grandparent adopts the Child~~

~~Child' PPID → Grandparent → ... → Shell → Kernel~~

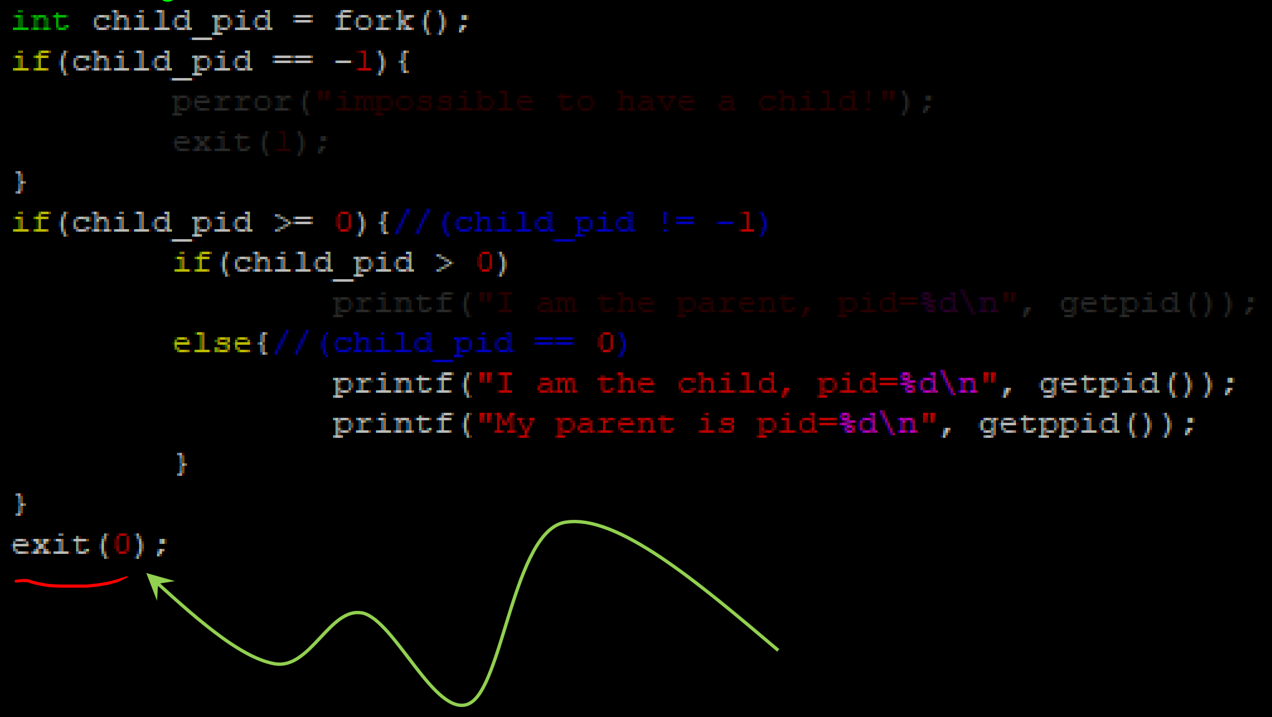
1

0

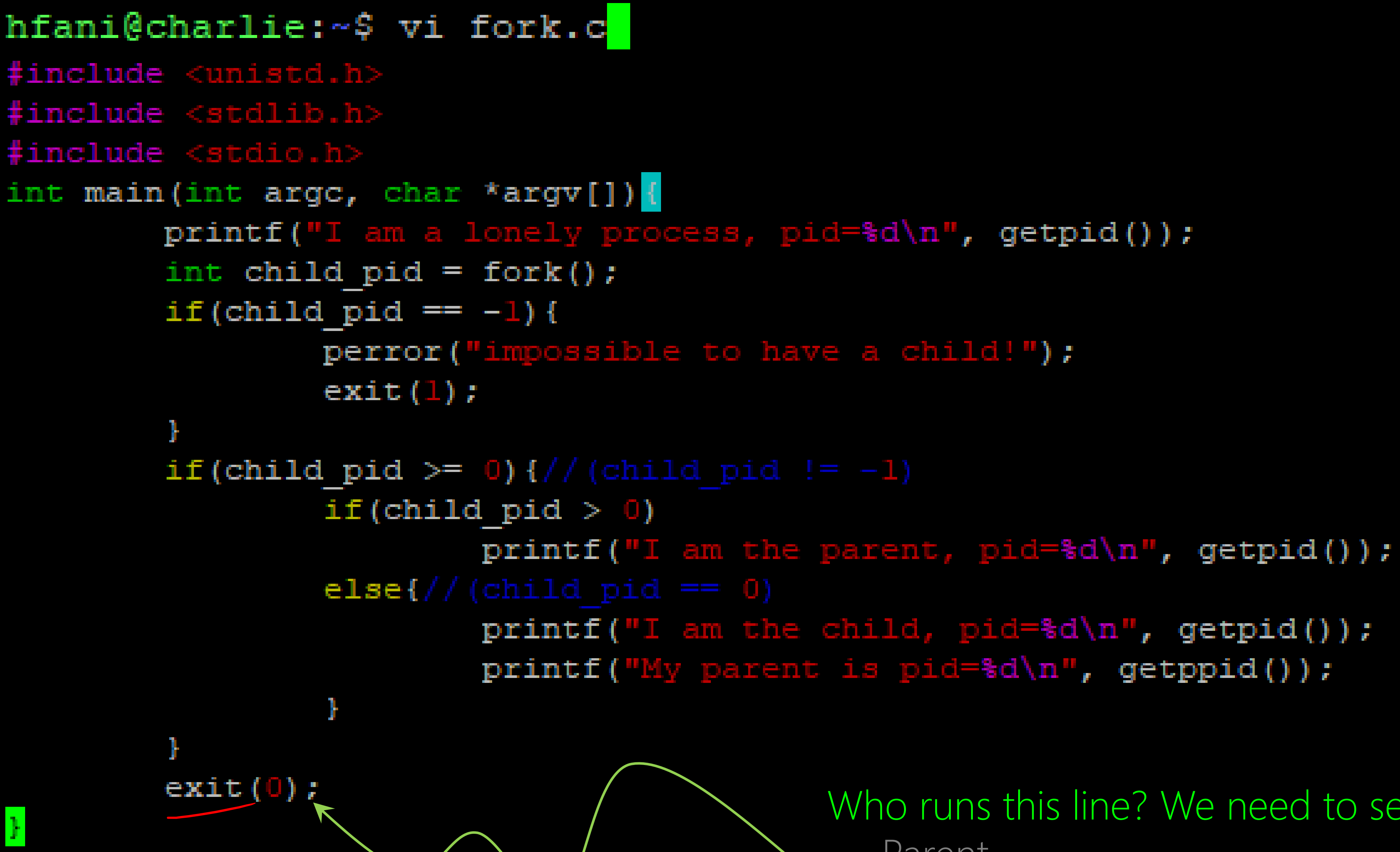
Kernel becomes the parent of any orphan process!

Child

0



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```



```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d\n", getpid());
                        printf("My parent is pid=%d\n", getppid());
                }
        }
        exit(0);
}
```

Compile Time Analysis

Who runs this line? We need to see what's going on in runtime.

- Parent
- Child
- [✓ Both]
- None

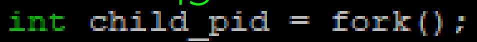
Put the child first, please!

Parent (pid=4)

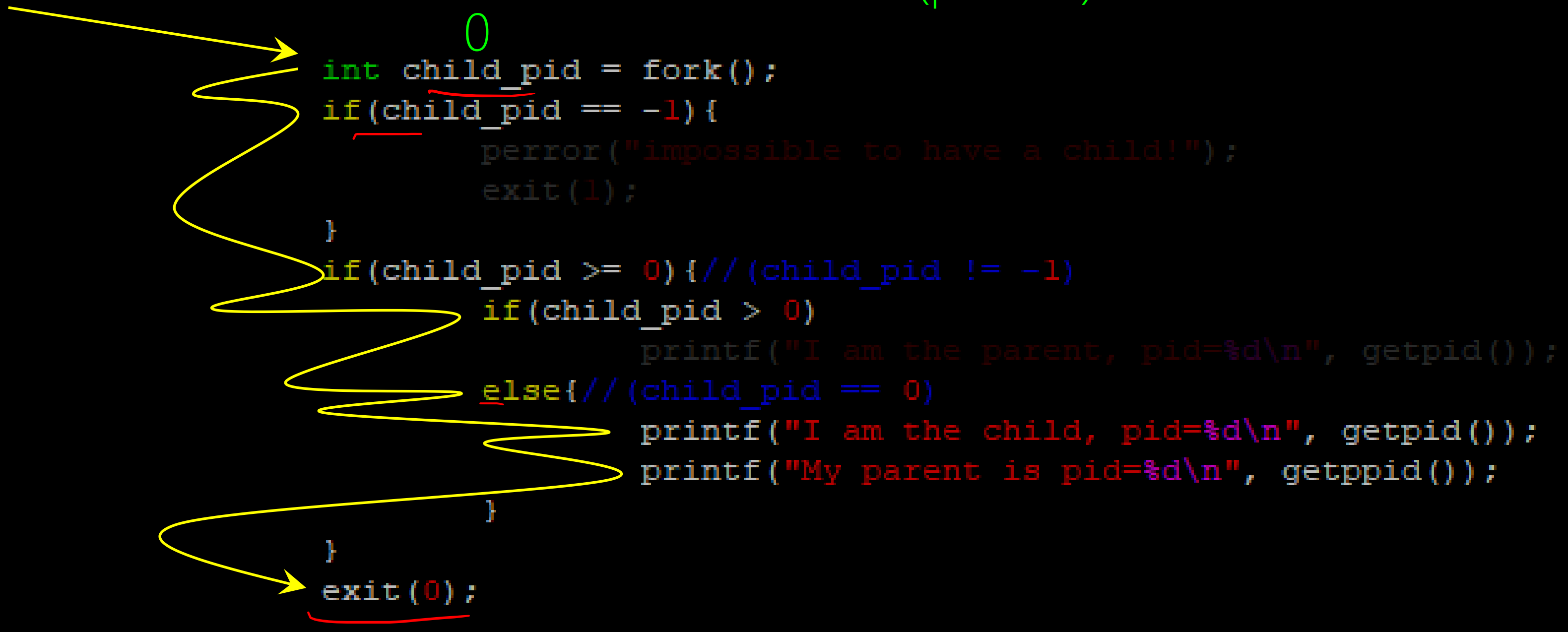
Processor

Child (pid=13)

13



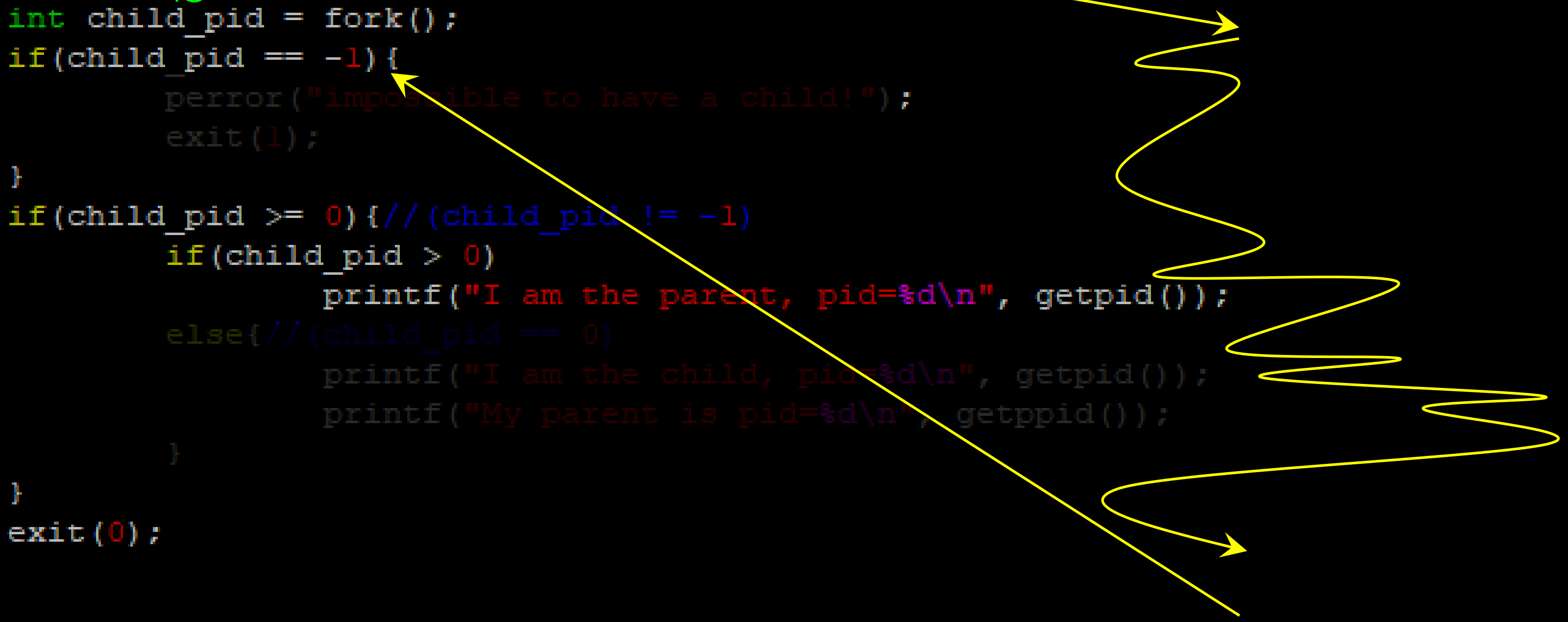
```
int child_pid = fork();
```



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```

Parent (pid=4) Processor

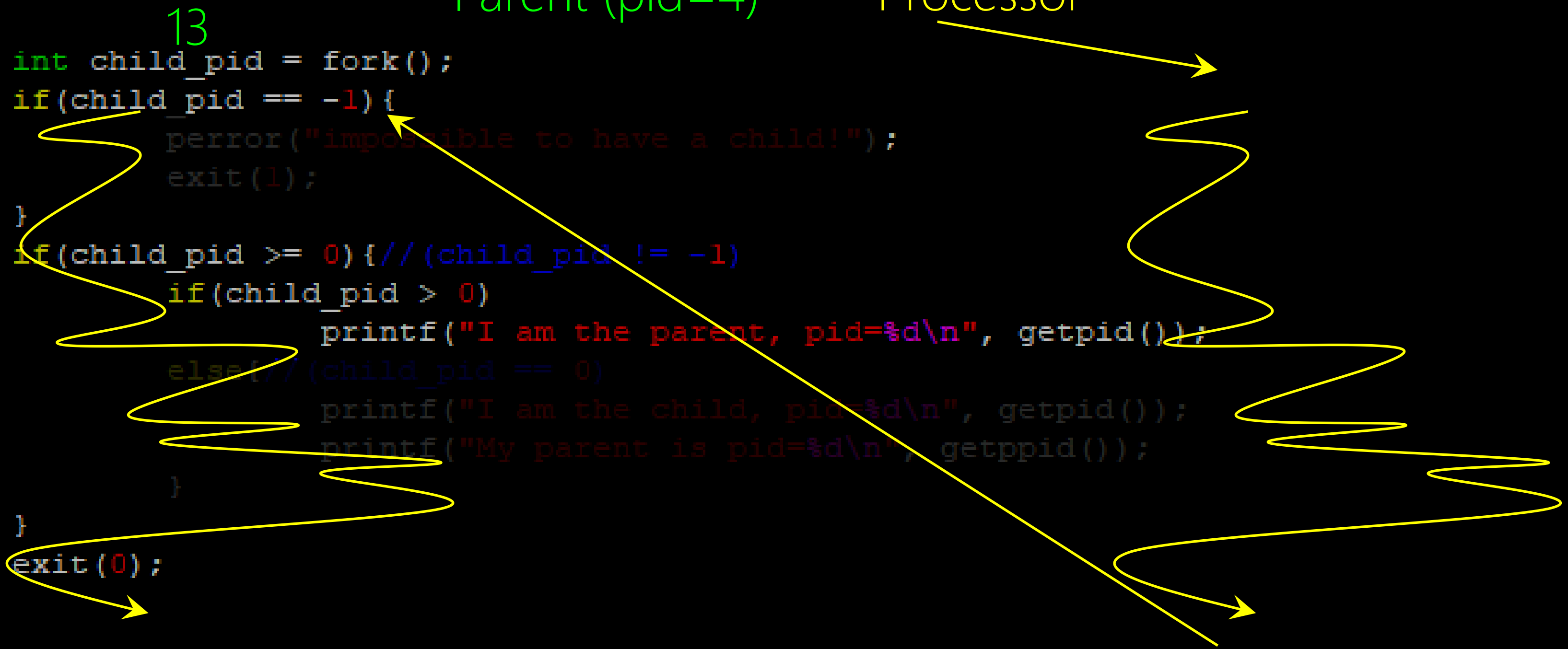
13



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```

Parent (pid=4)

Processor



```
int child_pid = fork();
if(child_pid == -1){
        perror("impossible to have a child!");
        exit(1);
}
if(child_pid >= 0){//(child_pid != -1)
        if(child_pid > 0)
                printf("I am the parent, pid=%d\n", getpid());
        else{//(child_pid == 0)
                printf("I am the child, pid=%d\n", getpid());
                printf("My parent is pid=%d\n", getppid());
        }
}
exit(0);
```

Laser Room, Resident Evil (2002) - Paul W. S. Anderson