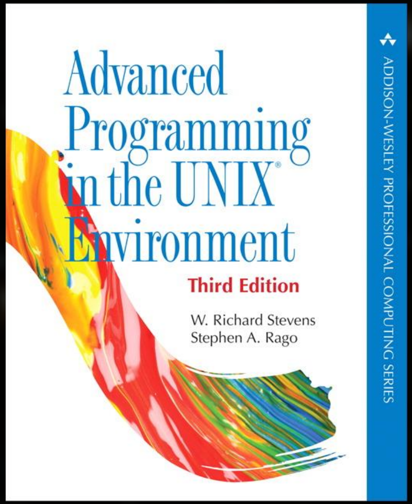
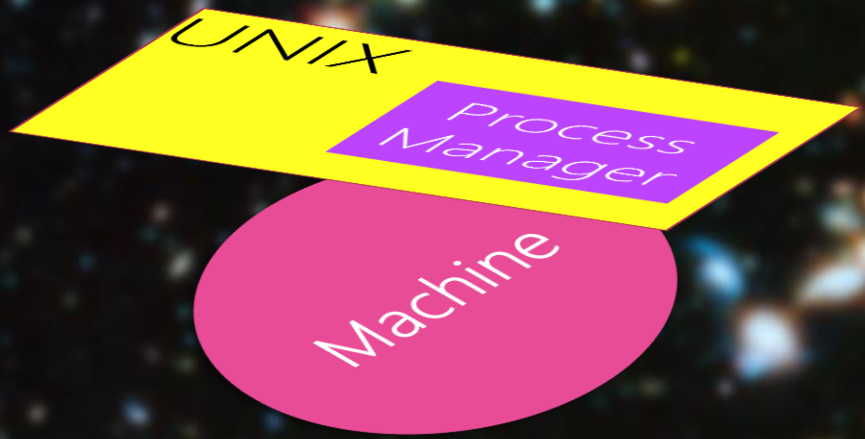
The Prestige (2006) - Christopher Nolan

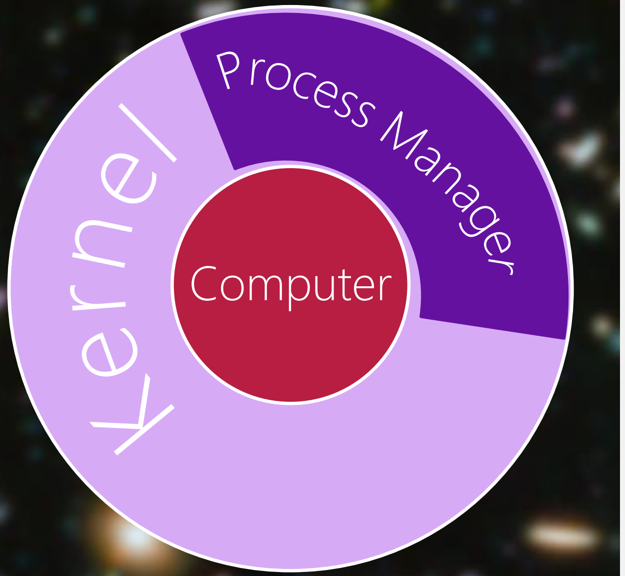

Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

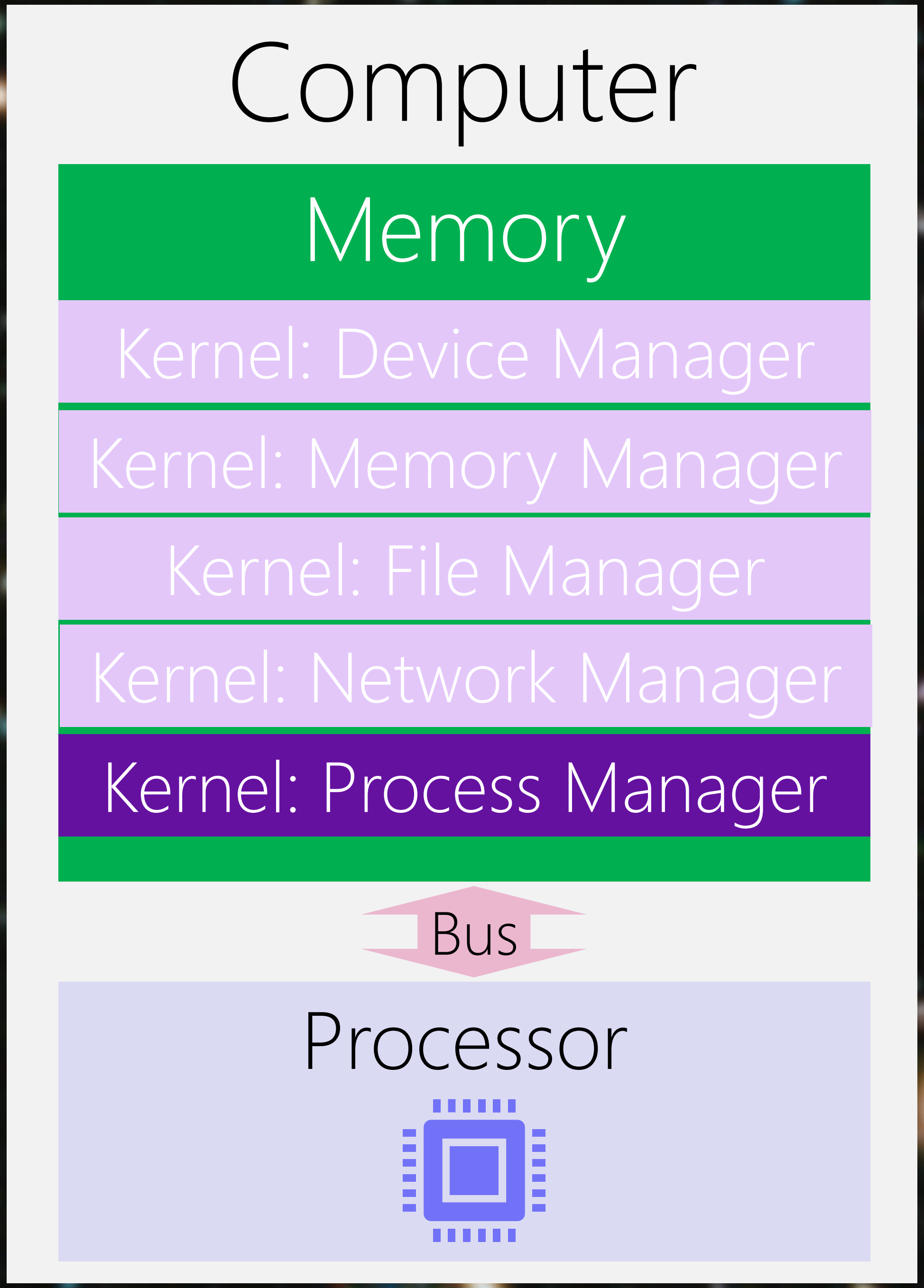
Chapter 07: Process Environment
Chapter 08: Process Control

Process Manager

aka. Process Control







Computer

Memory

Kernel: Device Manager

Kernel: Memory Manager

Kernel: File Manager

Kernel: Network Manager

Kernel: Process Manager

Bus

Processor

Program → Process → Run → Terminate

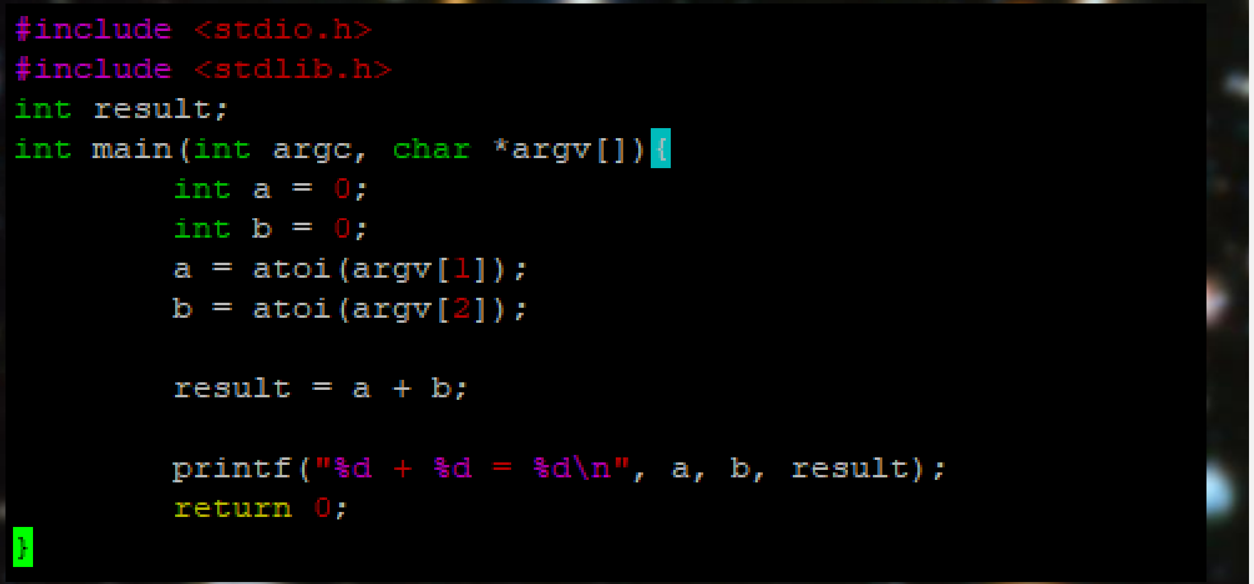
Program → Process → Run → Terminate

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

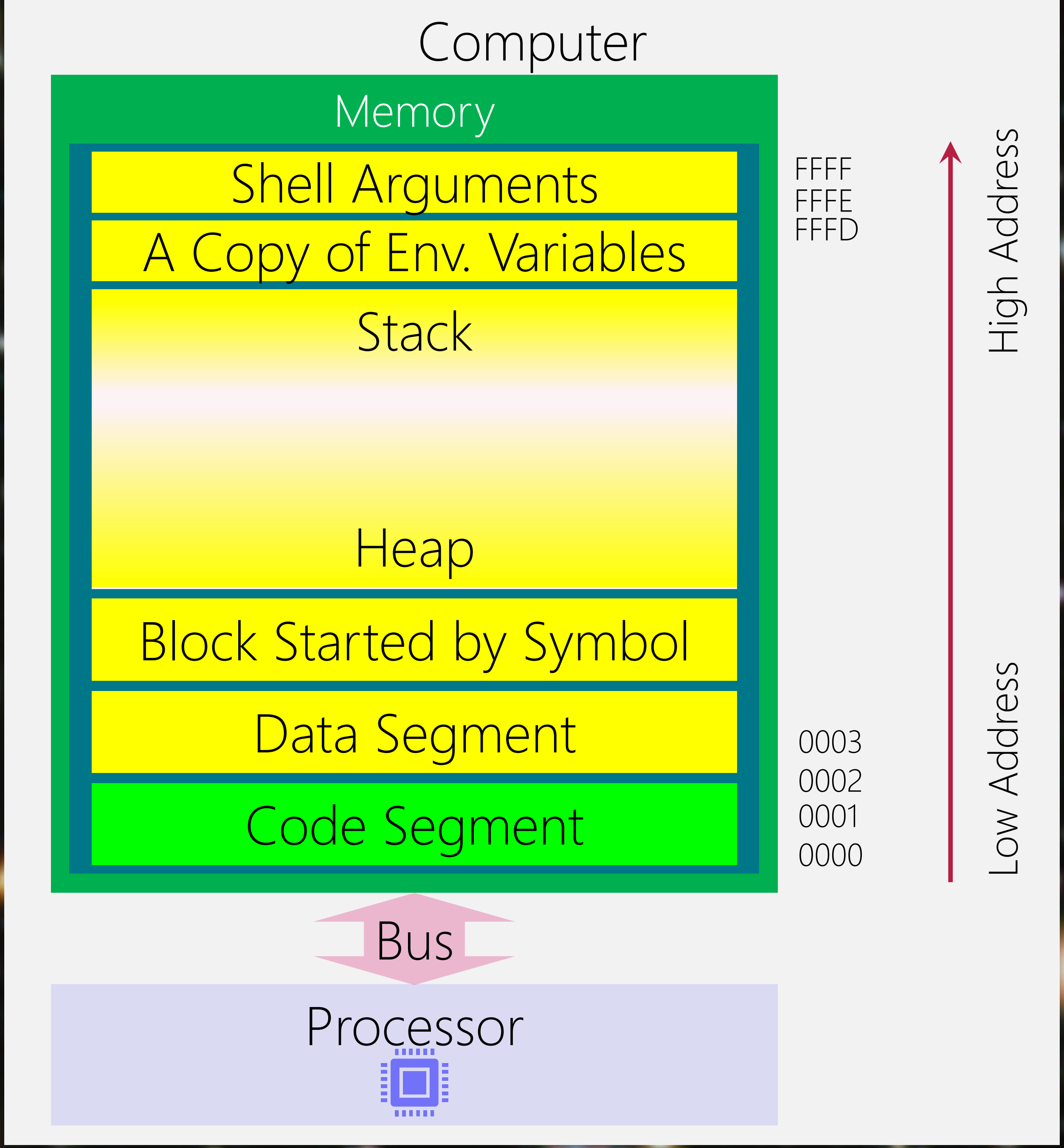
```
shell$ ./program arg1  arg2 arg3 ....
```

Into the Wild (2007) - Sean Penn

Program → Process → Run → Terminate



```
#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

FFFF
FFFE
FFFD

0003
0002
0001
0000

High Address

Low Address

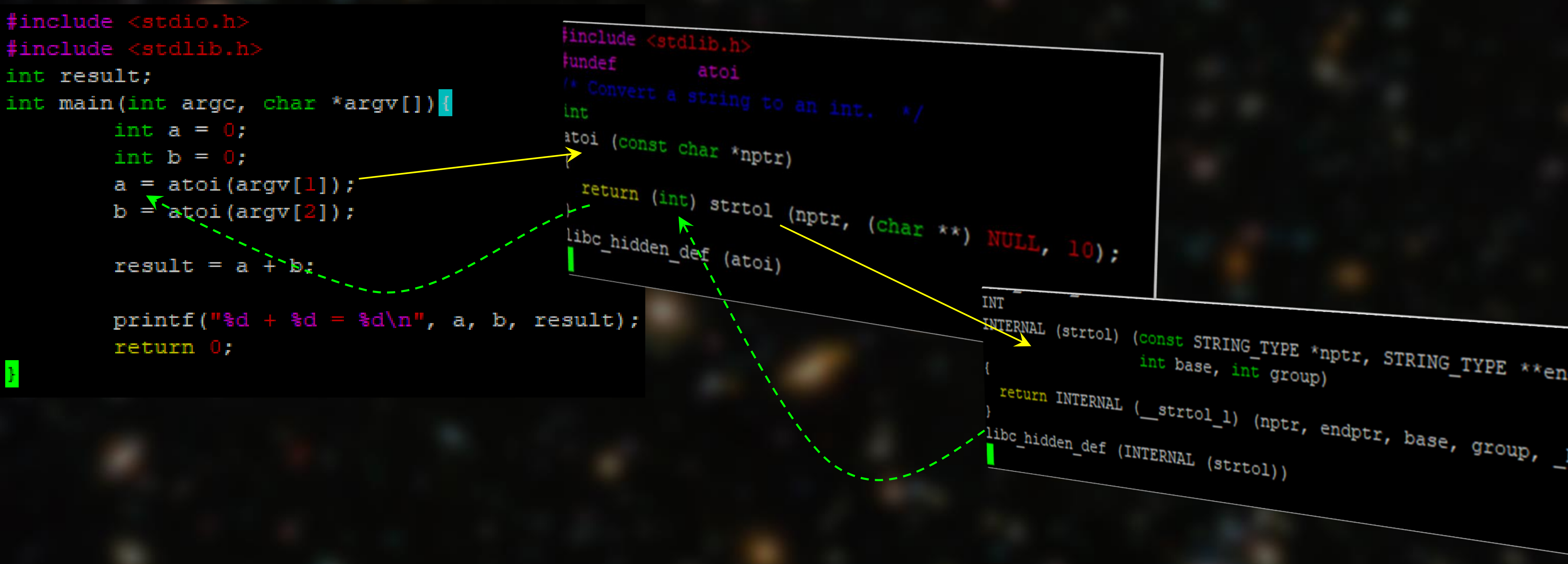
Bus

Processor

Into the Wild (2007) - Sean Penn

Stack

Functions Arguments, Local Variables, Return Address (runtime)



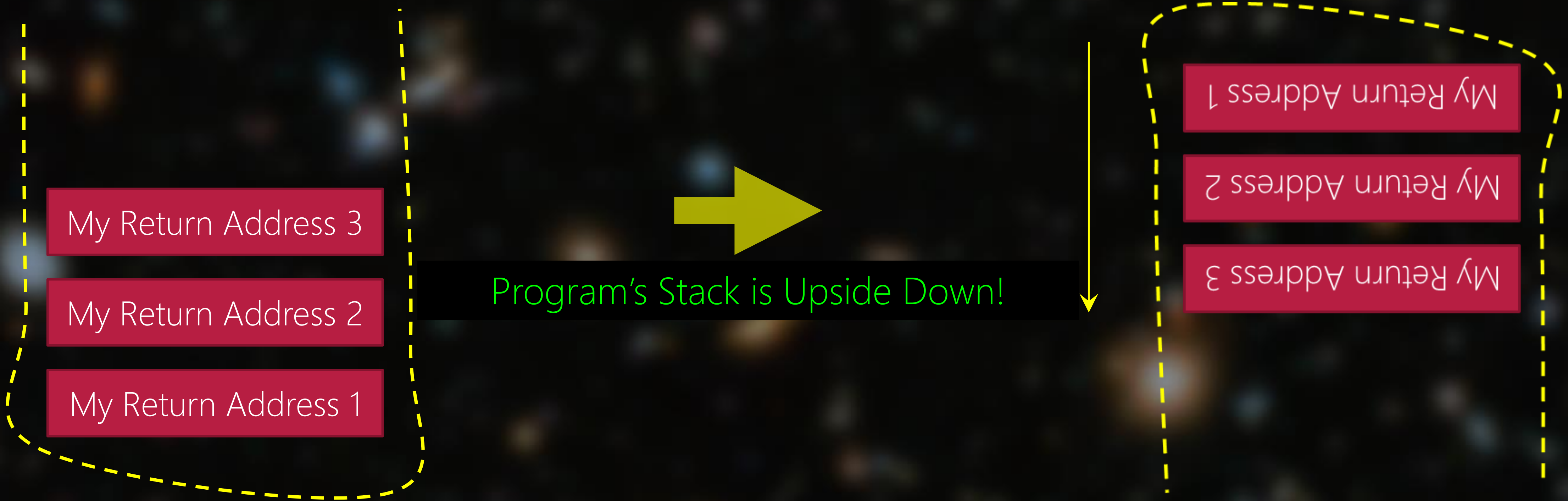
```
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int a = 0;
        int b = 0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);

        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

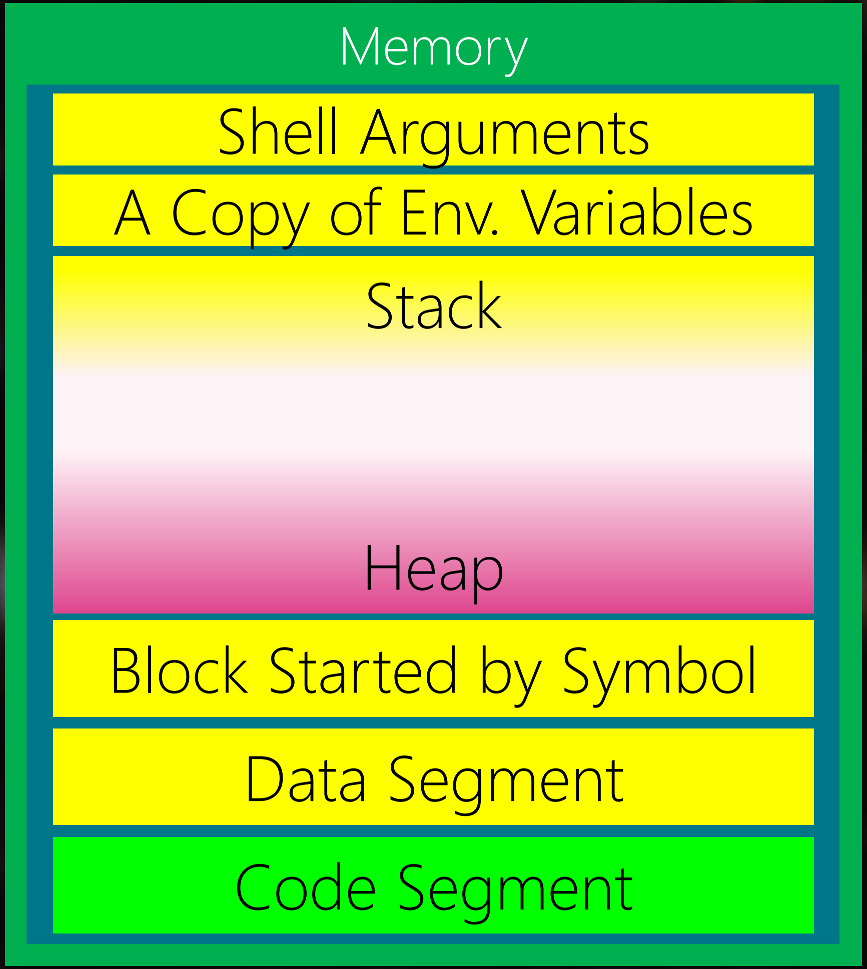
My Return Address 2

My Return Address 3

Into the Wild (2007) - Sean Penn

Heap

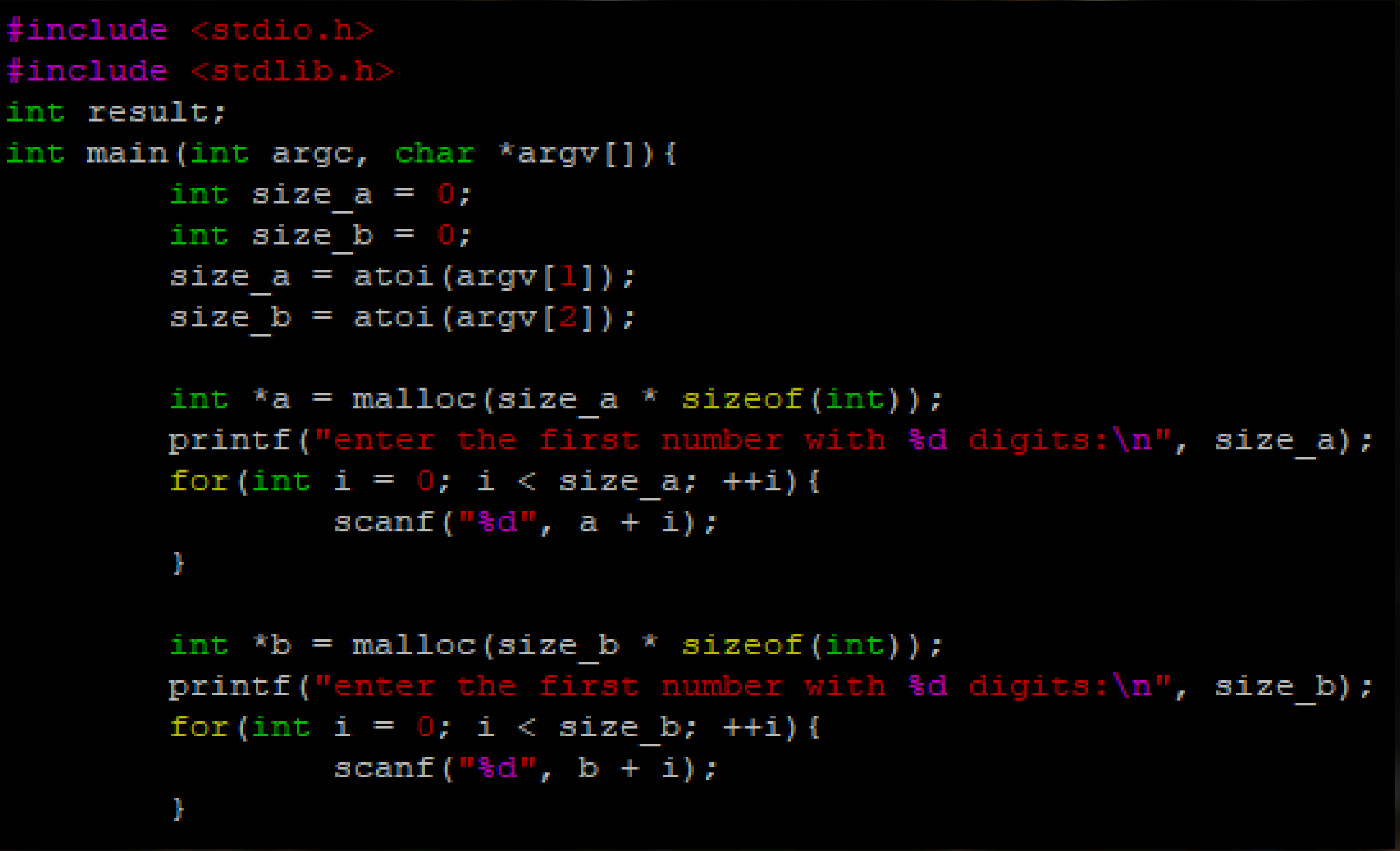
Dynamic memory allocation (runtime)



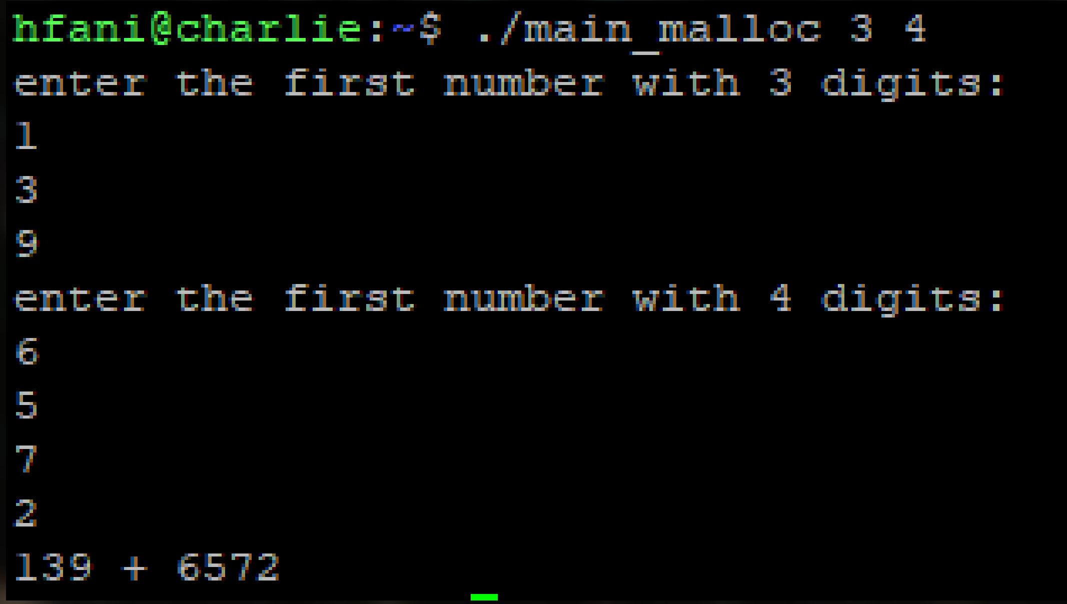
Memory Allocators by Library Routines

```
#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

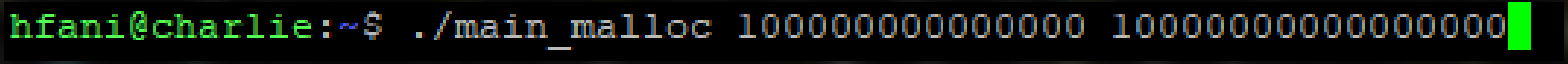


```c
#include <stdio.h>
#include <stdlib.h>
int result;
int main(int argc, char *argv[]){
        int size_a = 0;
        int size_b = 0;
        size_a = atoi(argv[1]);
        size_b = atoi(argv[2]);

        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);
        }
```

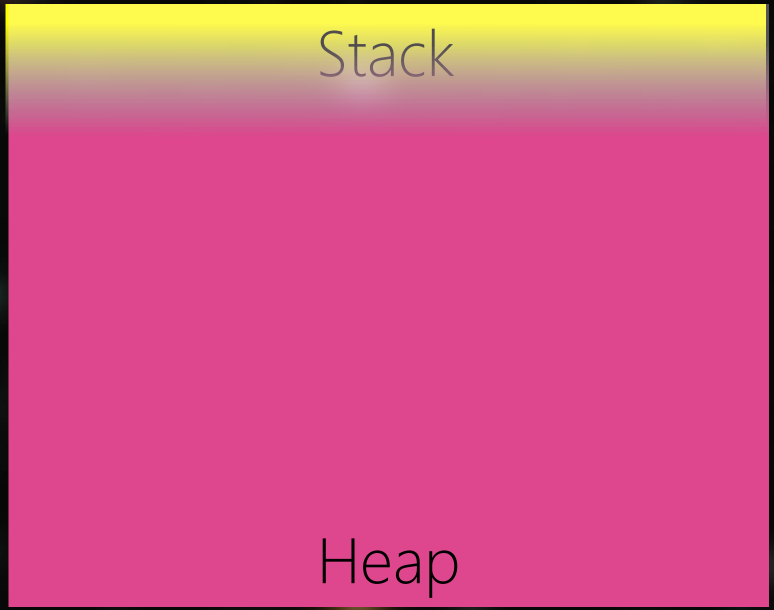


```
hfani@charlie:~$ ./main_malloc 3 4
enter the first number with 3 digits:
1
3
9
enter the first number with 4 digits:
6
5
7
2
139 + 6572
```

Size is dynamic during runtime
Value is dynamic during runtime



```
hfani@charlie:~$ ./main_malloc 100000000000000 10000000000000000
```



Stack

Heap

Heap

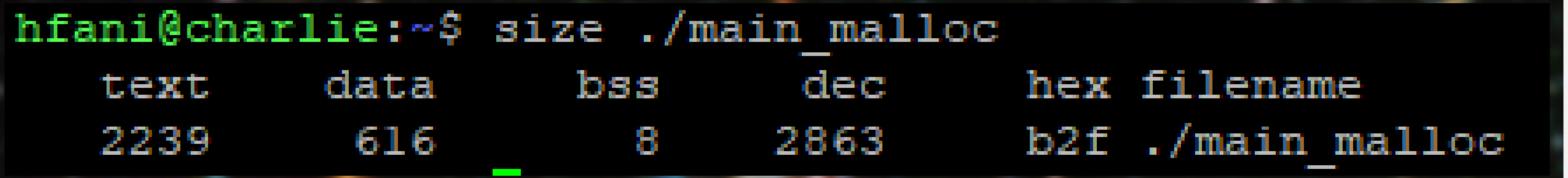
Dynamic memory allocation (runtime)

Memory Allocators by Library Routines

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

Memory Allocators by System Calls?

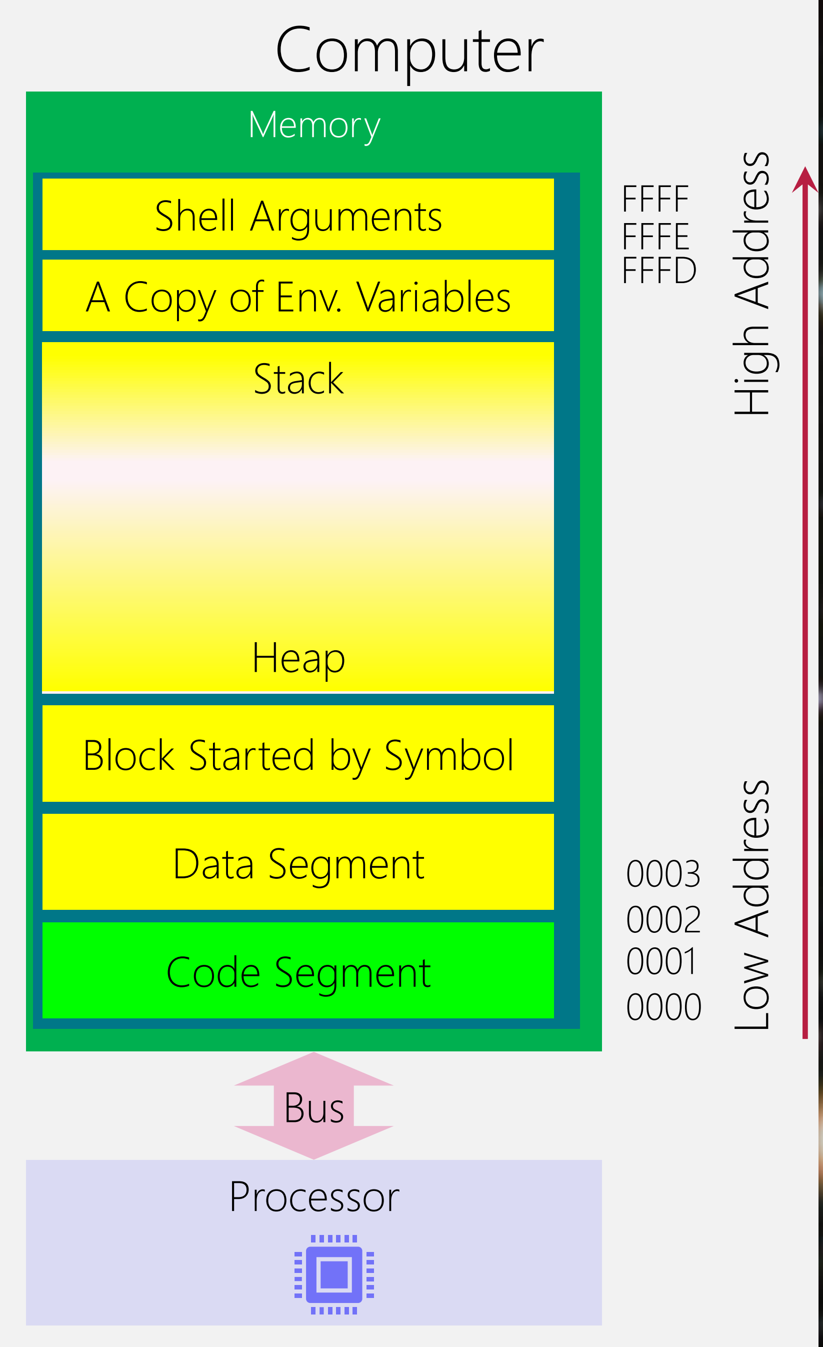
Shell’s `size` command



```
hfani@charlie:~$ size ./main_malloc
   text    data     bss     dec     hex filename
   2239     616       8    2863     b2f ./main_malloc
```

Why is not any info for:

- Stack?
- Heap?



Process Identifier (pid)

Non-negative

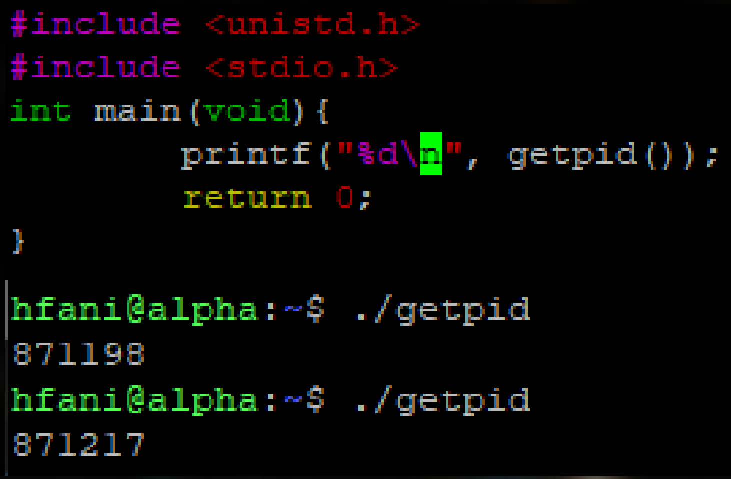
Unique among processes (live programs)

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

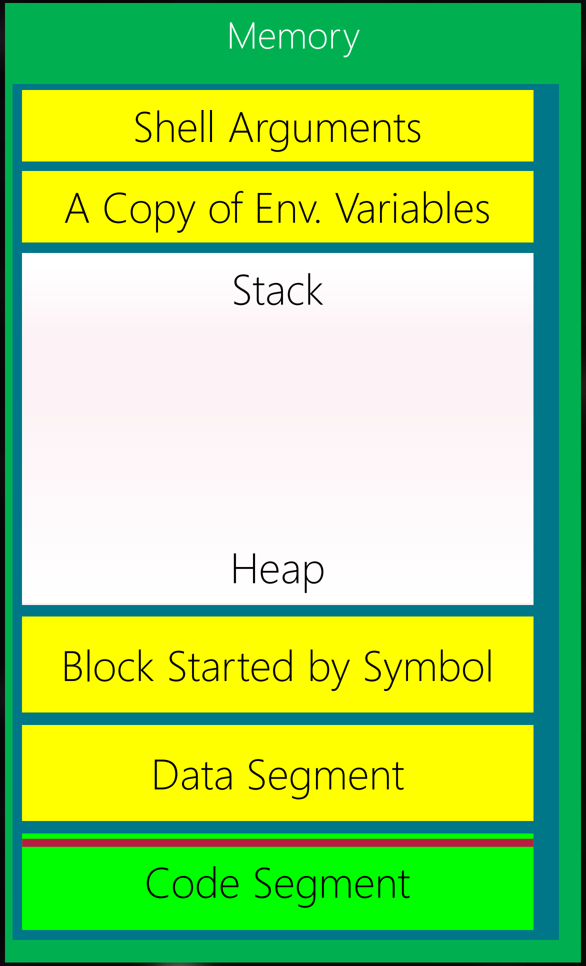


```
#include <unistd.h>
#include <stdio.h>
int main(void){
        printf("%d\n", getpid());
        return 0;
}
```

```
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

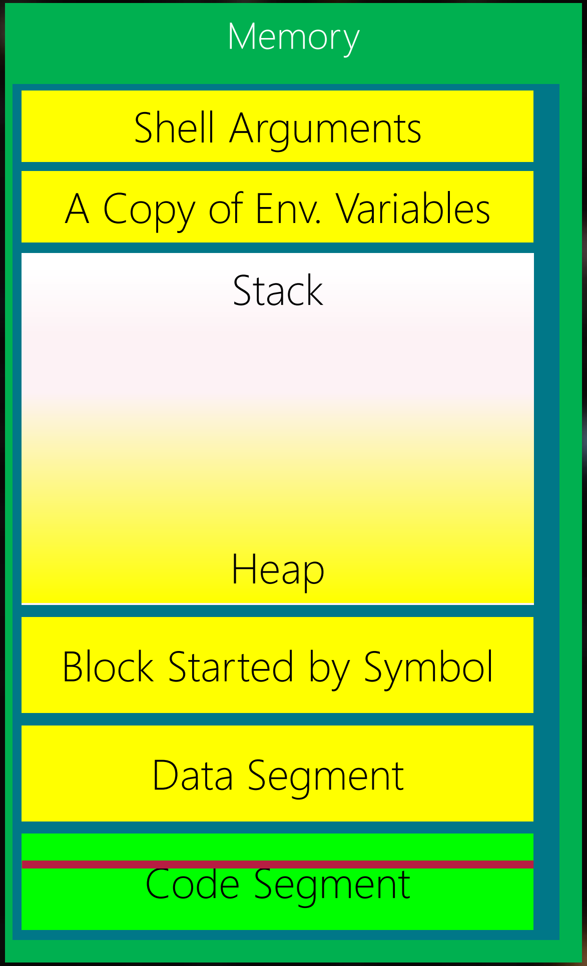
Heap

Block Started by Symbol

Data Segment

Code Segment

Program → Process → Run → Terminate



Memory

Shell Arguments

A Copy of Env. Variables

Stack

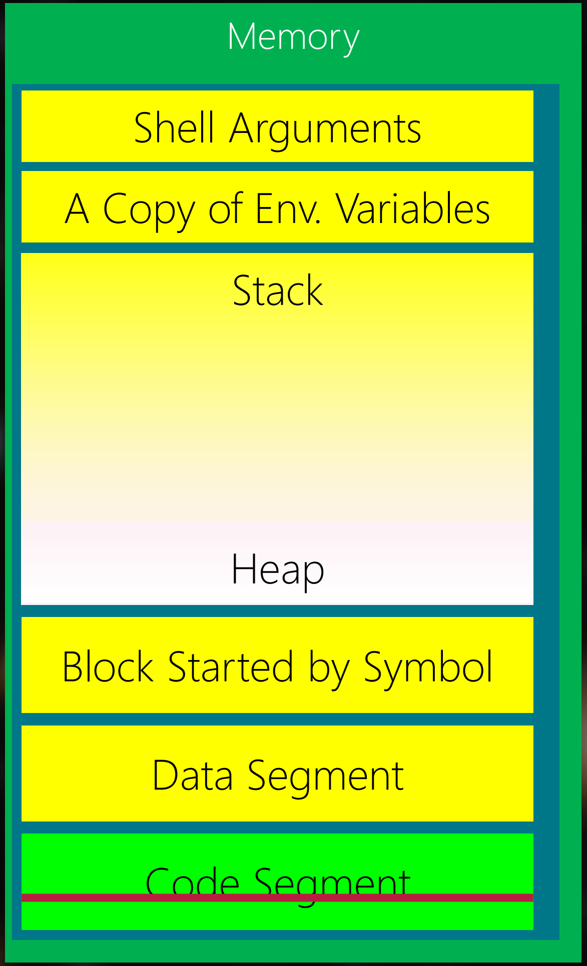
Heap

Block Started by Symbol

Data Segment

Code Segment

Program → Process → Run → Terminate



Memory

Shell Arguments

A Copy of Env. Variables

Stack

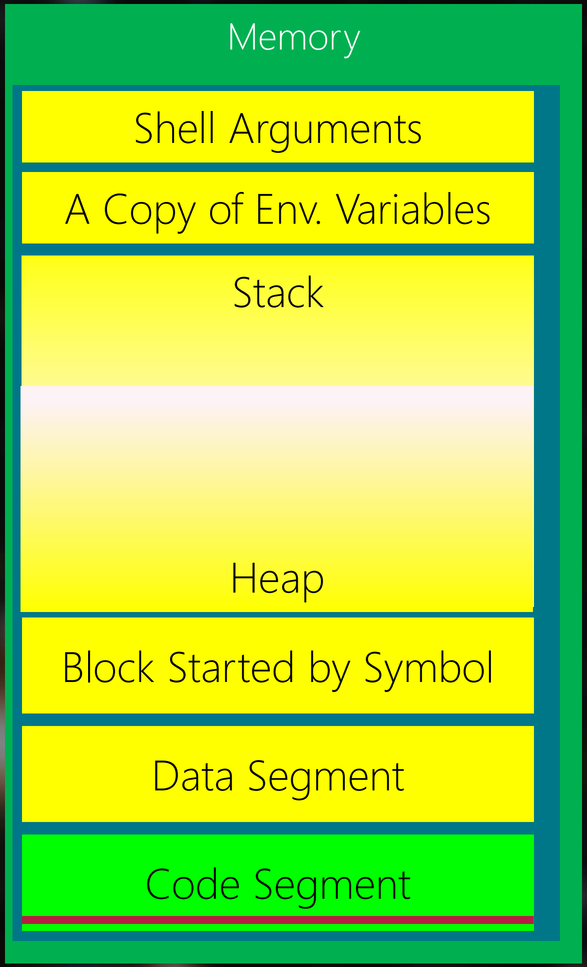
Heap

Block Started by Symbol

Data Segment

Code Segment

Program → Process → Run → Terminate



Memory

Shell Arguments

A Copy of Env. Variables

Stack

Heap

Block Started by Symbol

Data Segment

Code Segment

Into the Wild (2007) - Sean Penn

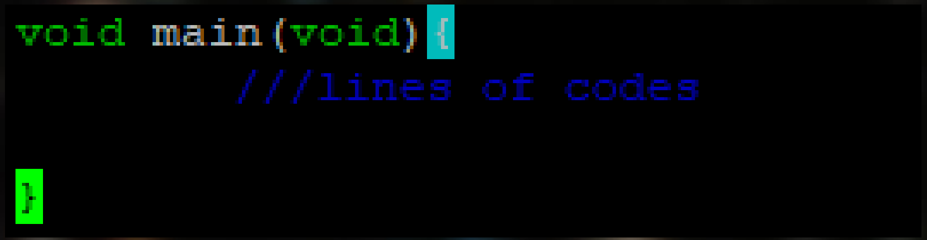
Program → Process → Run → Terminate

C has exit status (code)

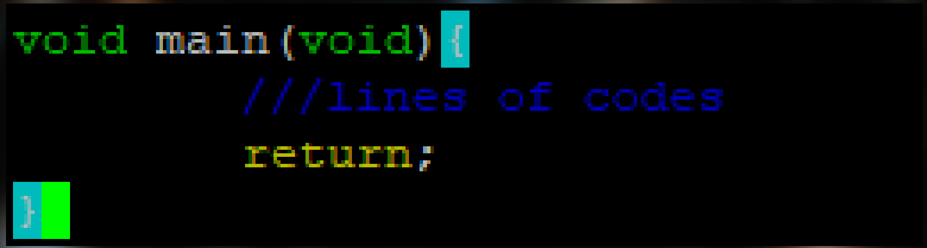
Normal vs. Abnormal Exits

C has exit status (code)

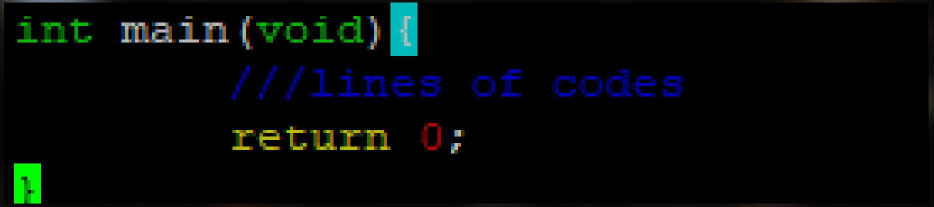
Normal



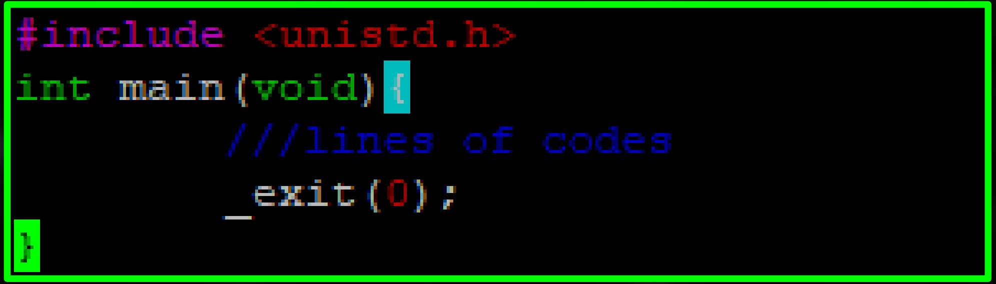
```
void main(void){
        ///lines of codes

}
```



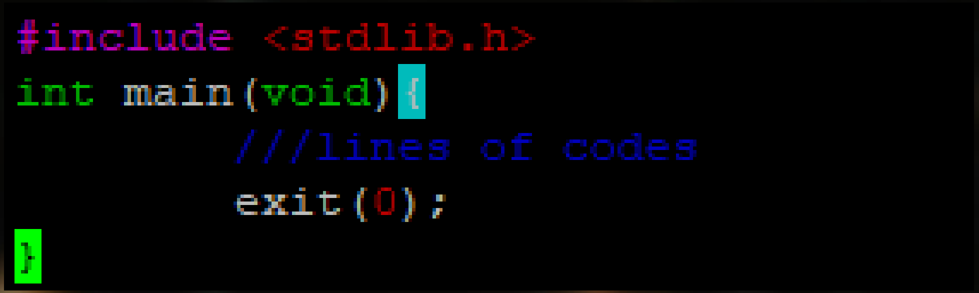
```
void main(void){
        ///lines of codes
        return;
}
```



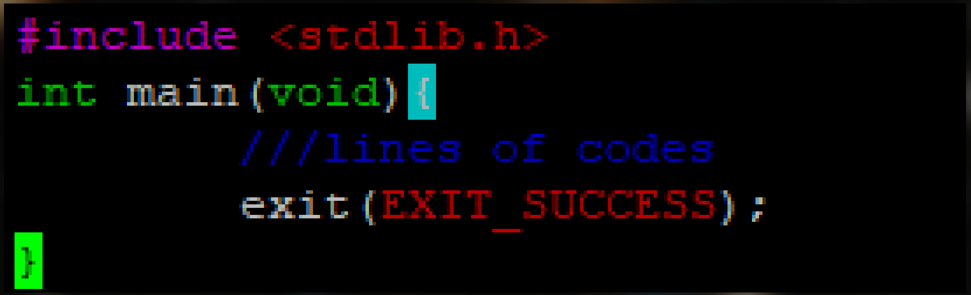
```
int main(void){
        ///lines of codes
        return 0;
}
```



```
#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.

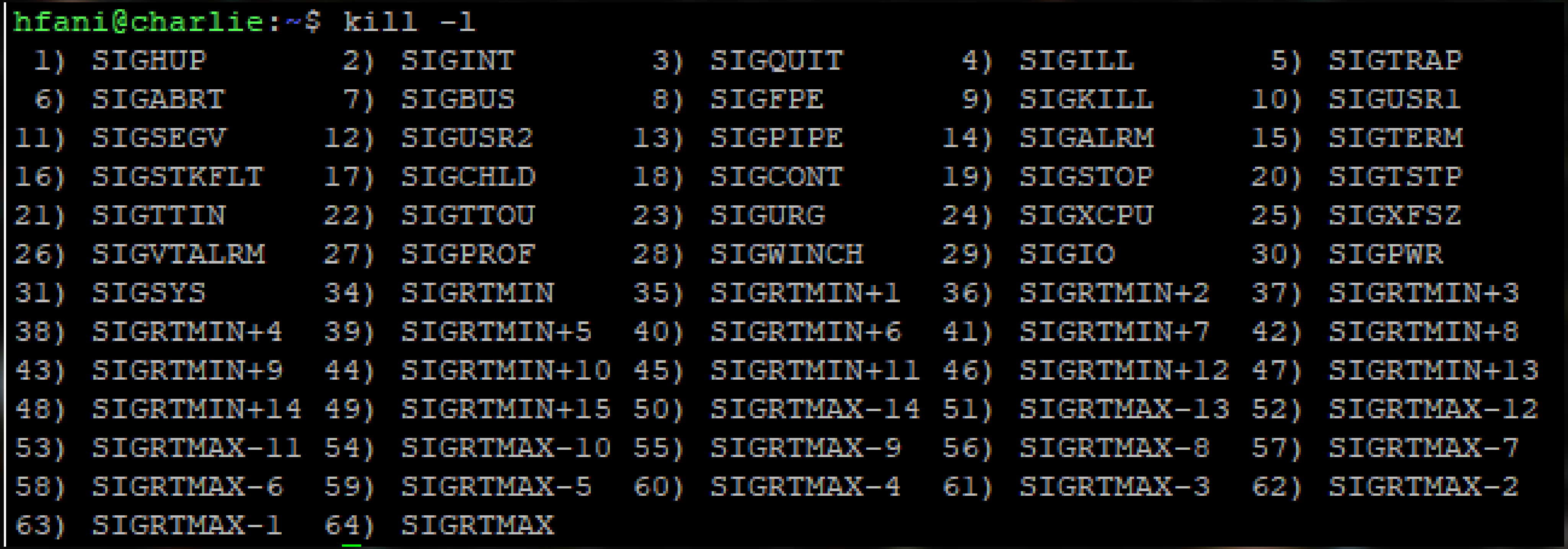
C has exit status (code)

Abnormal

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

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

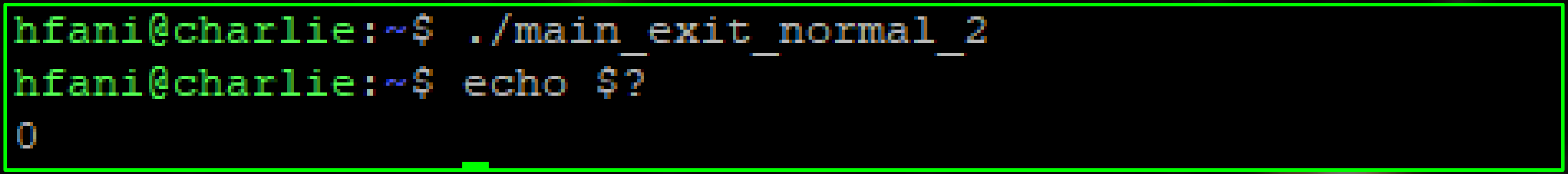
C has exit status (code)
Abnormal



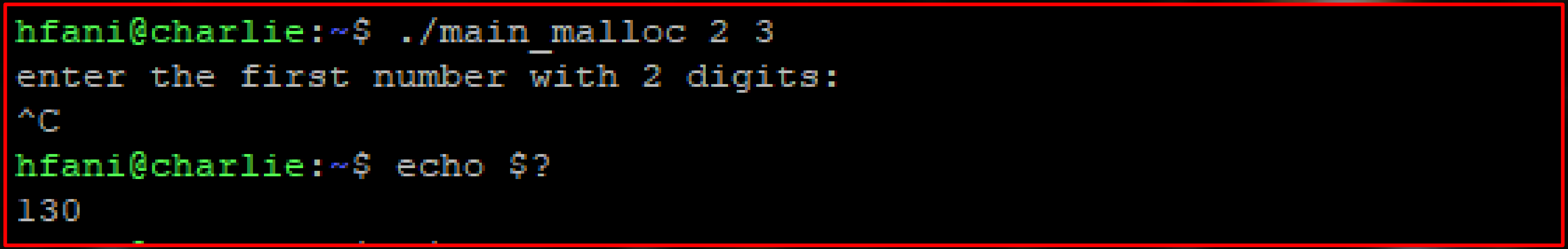
```
hfani@charlie:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Shell's Variable for Exit Status

`echo $?`



```
hfani@charlie:~$ ./main_exit_normal_2
hfani@charlie:~$ echo $?
0
```



```
hfani@charlie:~$ ./main_malloc 2 3
enter the first number with 2 digits:
^C
hfani@charlie:~$ echo $?
130
```

C does not have error/exception handling!

Multiprocessing

aka multiprogramming

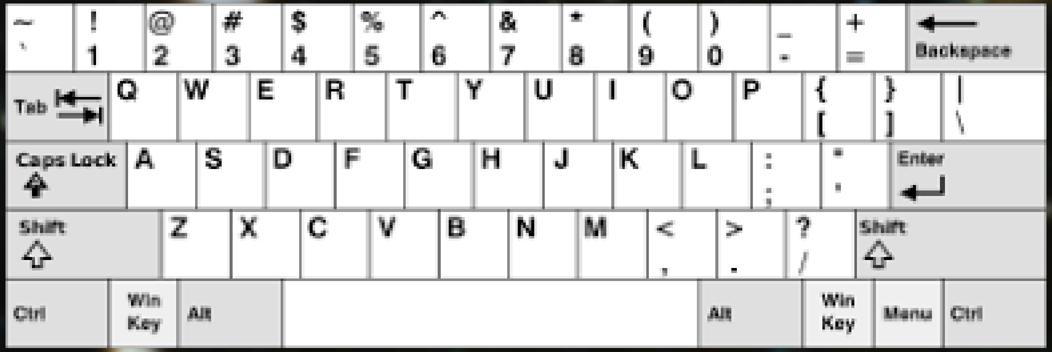
Single Processor ~~Multiprocessor~~

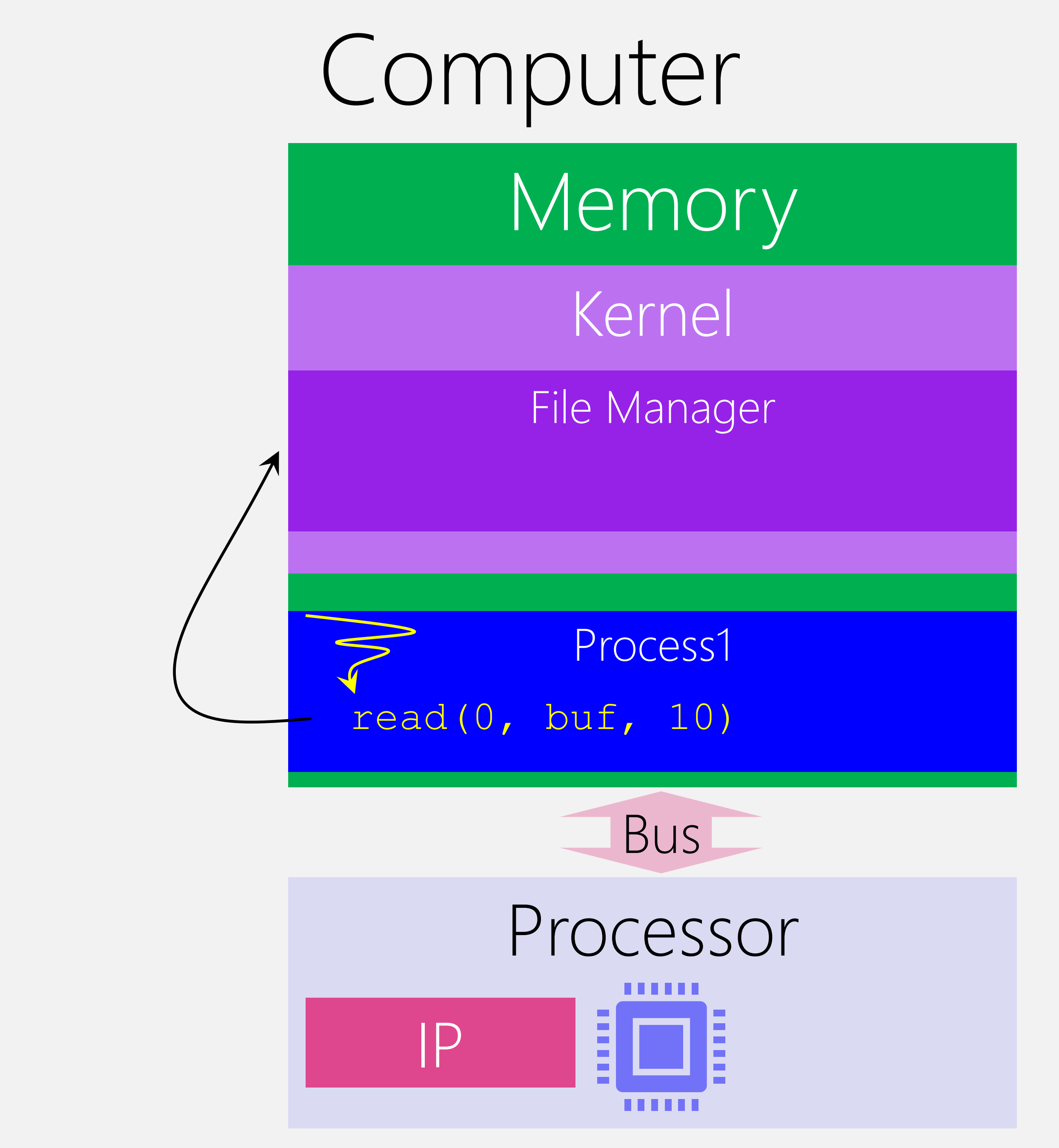
Week#2

Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?


~ ` ! 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

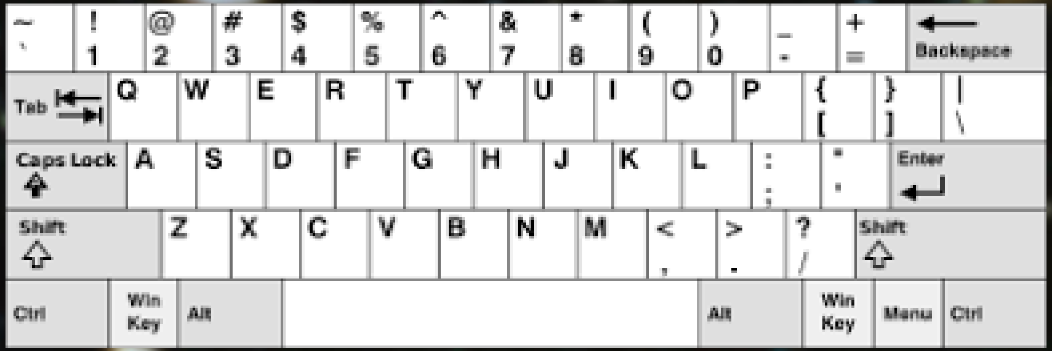
Processor

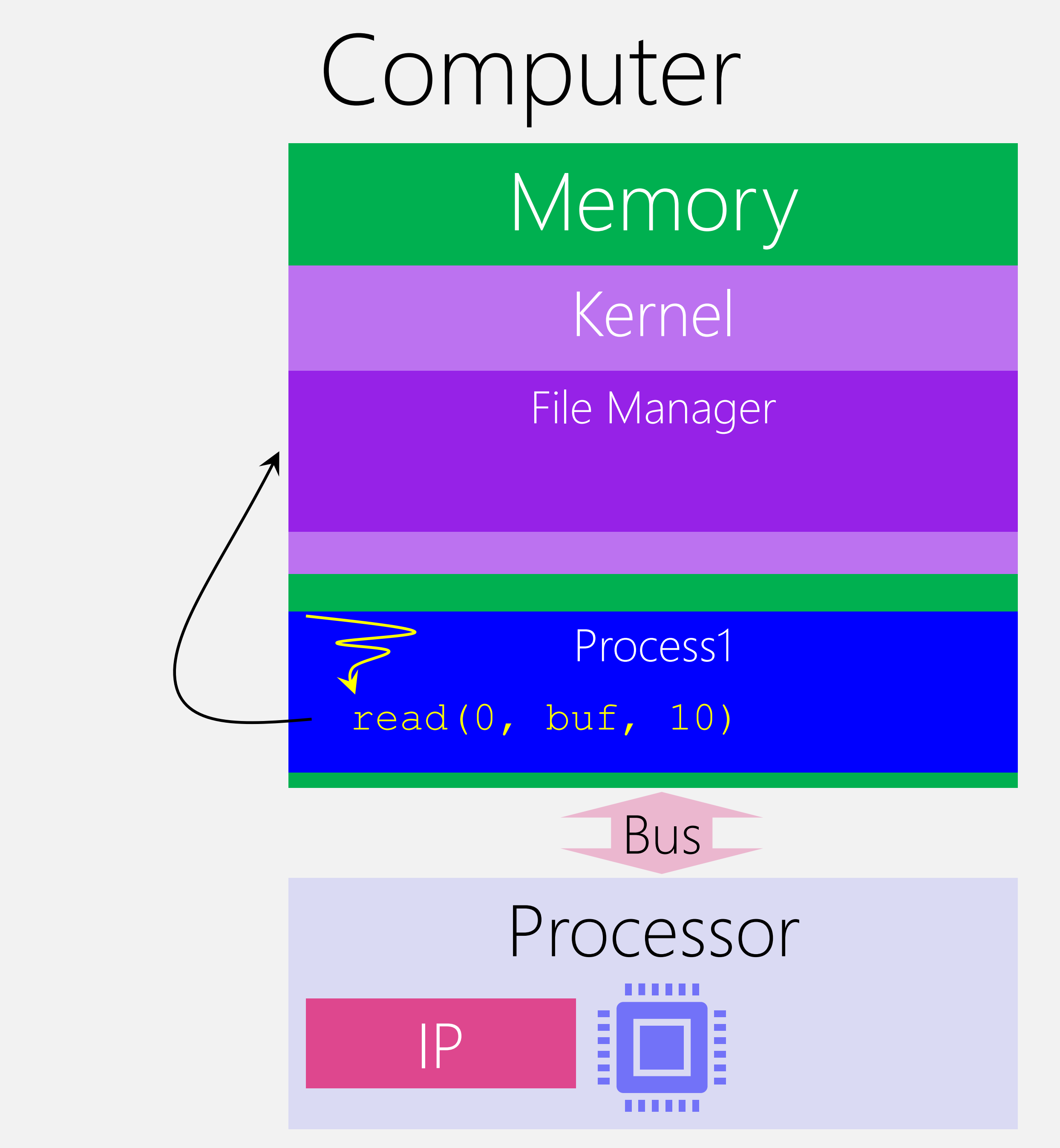
IP

Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?
What is the processor doing?


~ ` ! 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

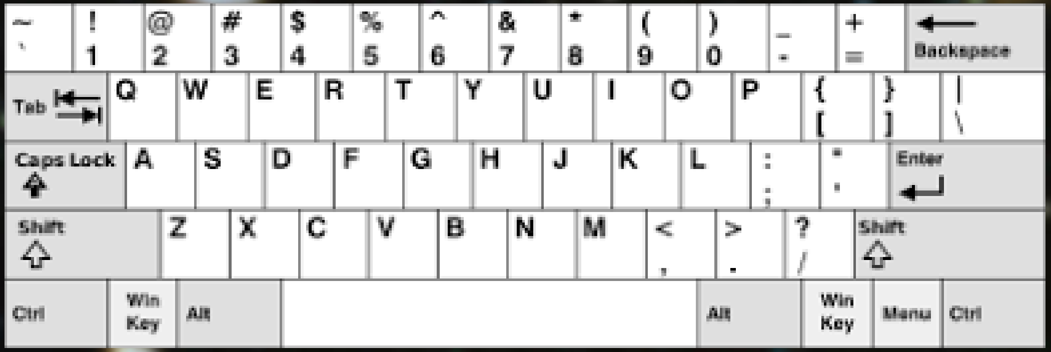


```
read(0, buf, 10)
```

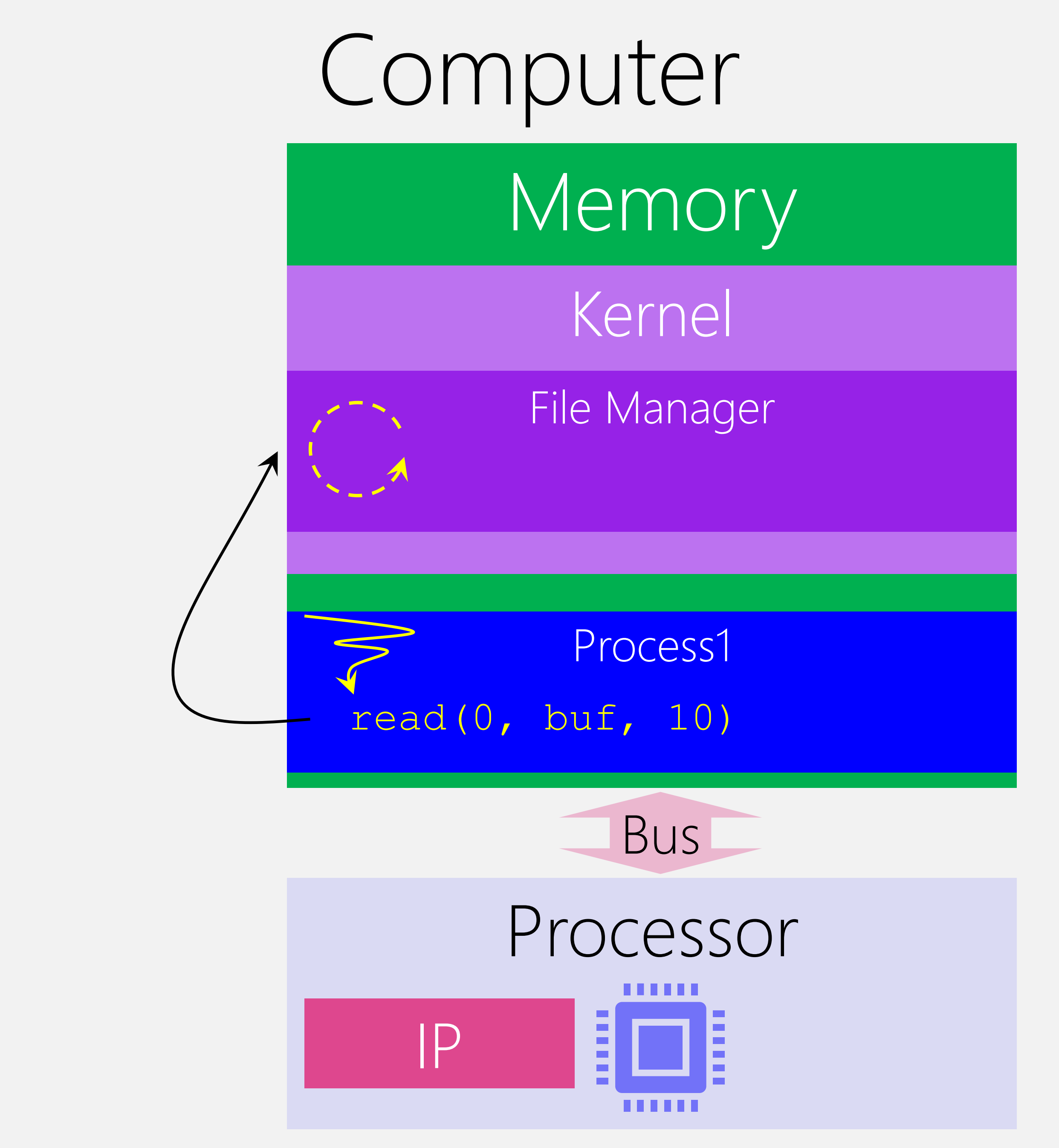
Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

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



~ ` ! 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

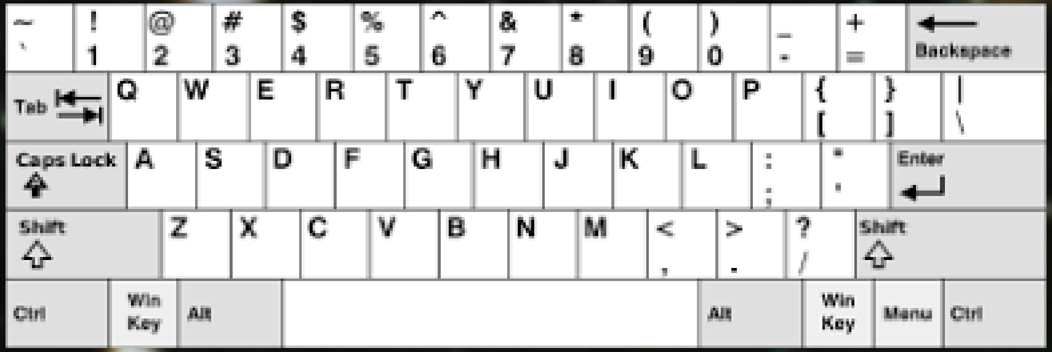
Week#2

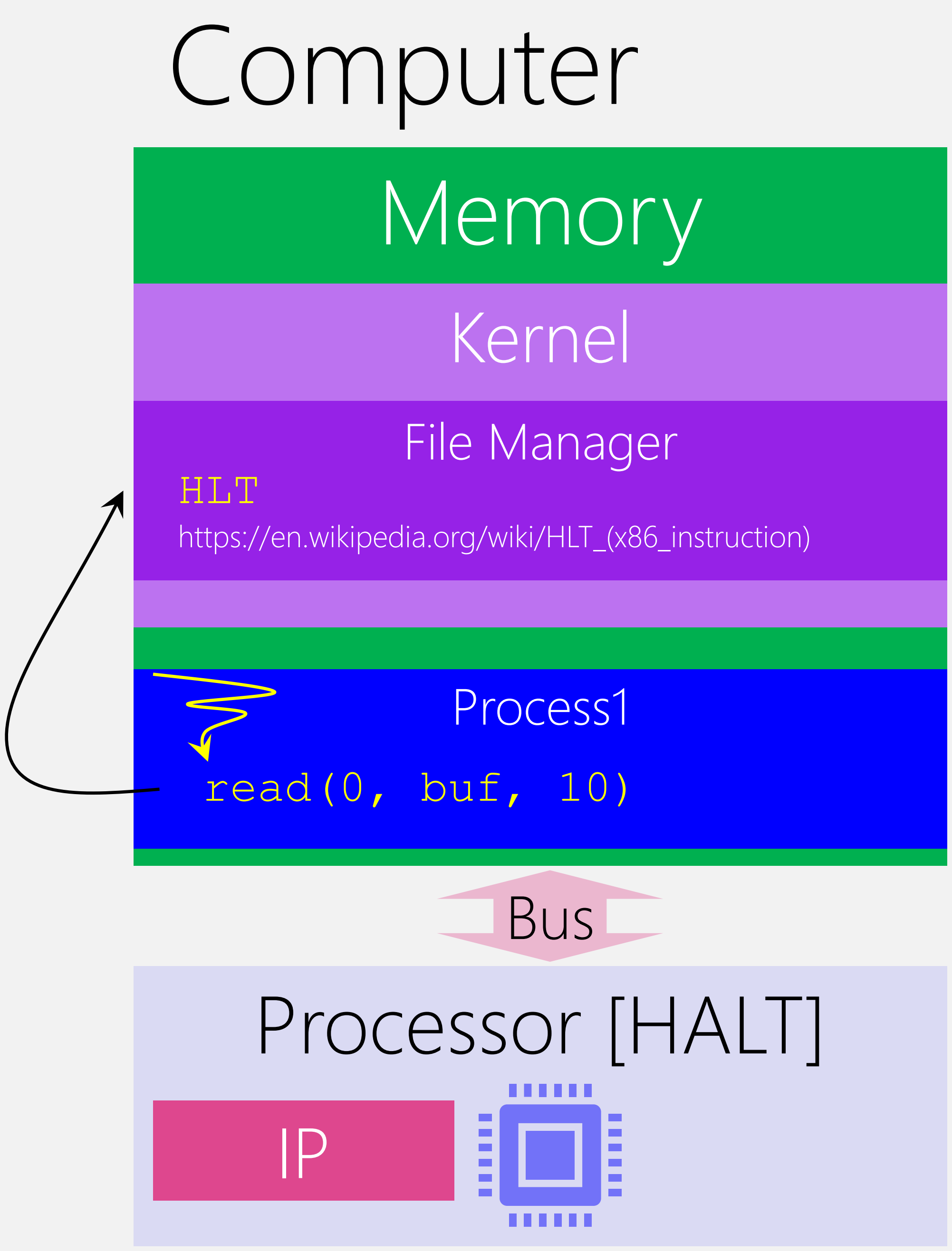
Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?

What is the processor doing?

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

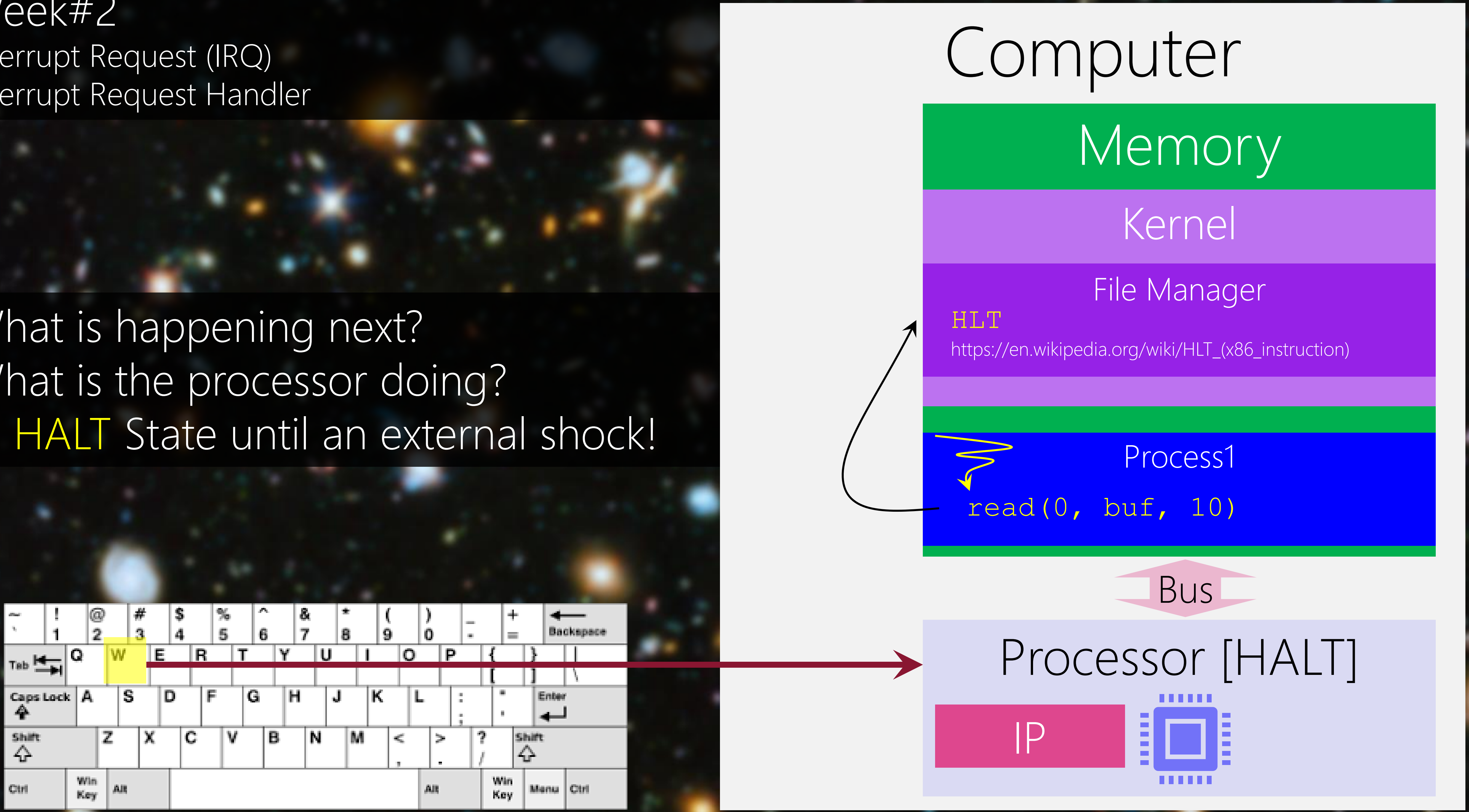
Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?

What is the processor doing?

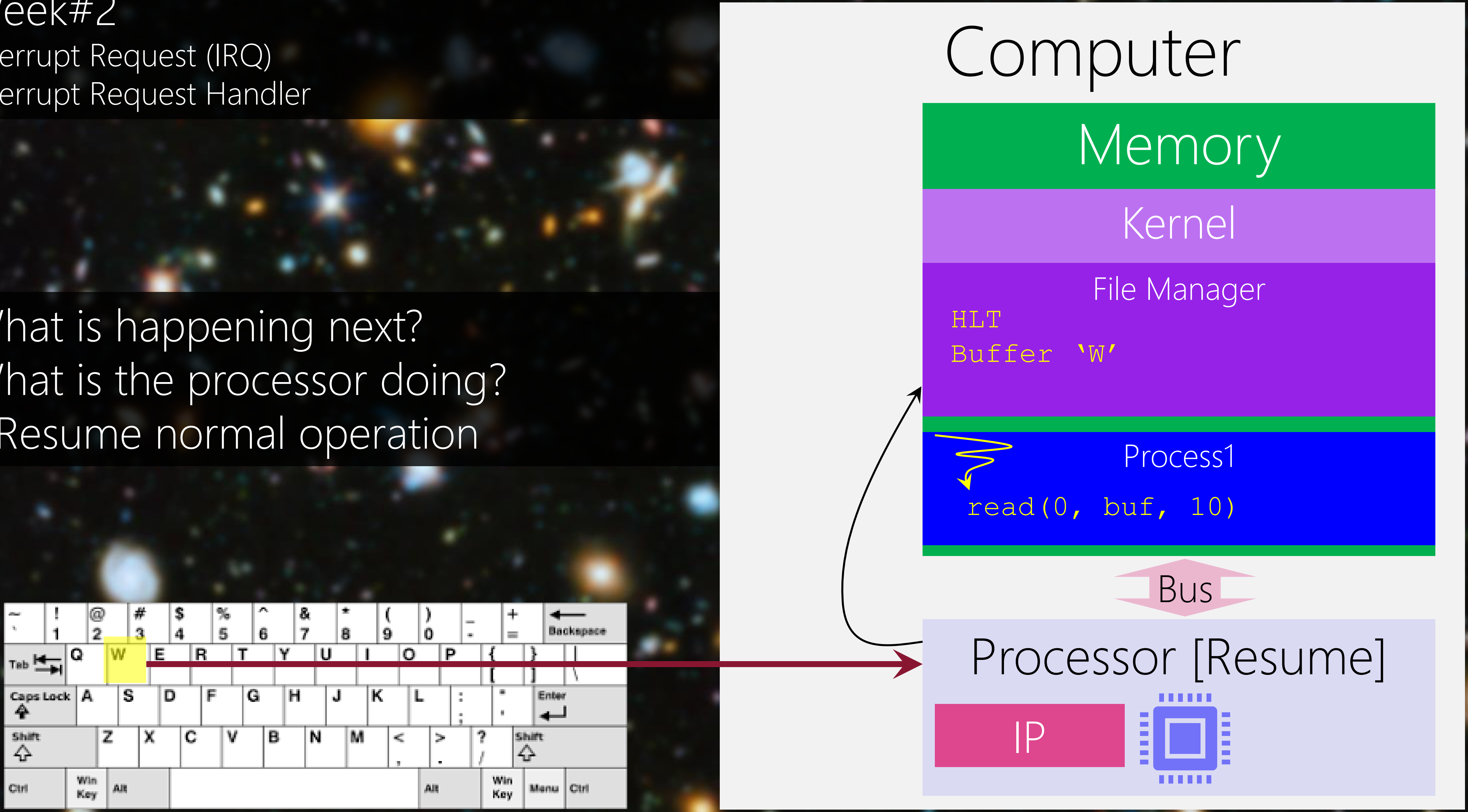
B) HALT State until an external shock!



Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

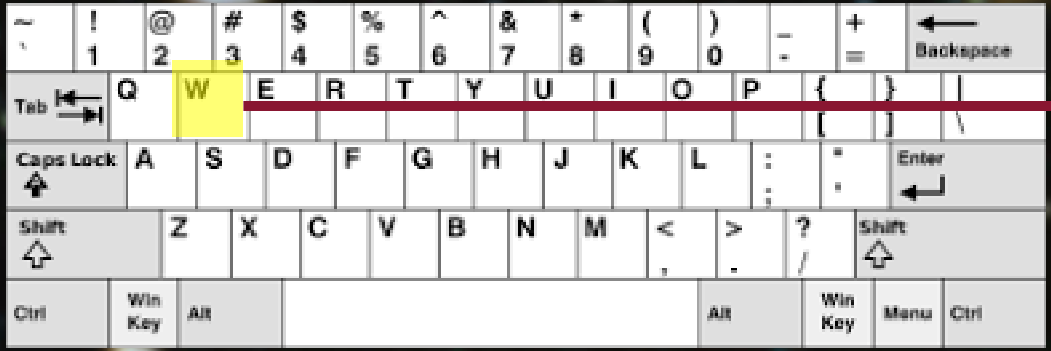
What is happening next?
What is the processor doing?
- Resume normal operation

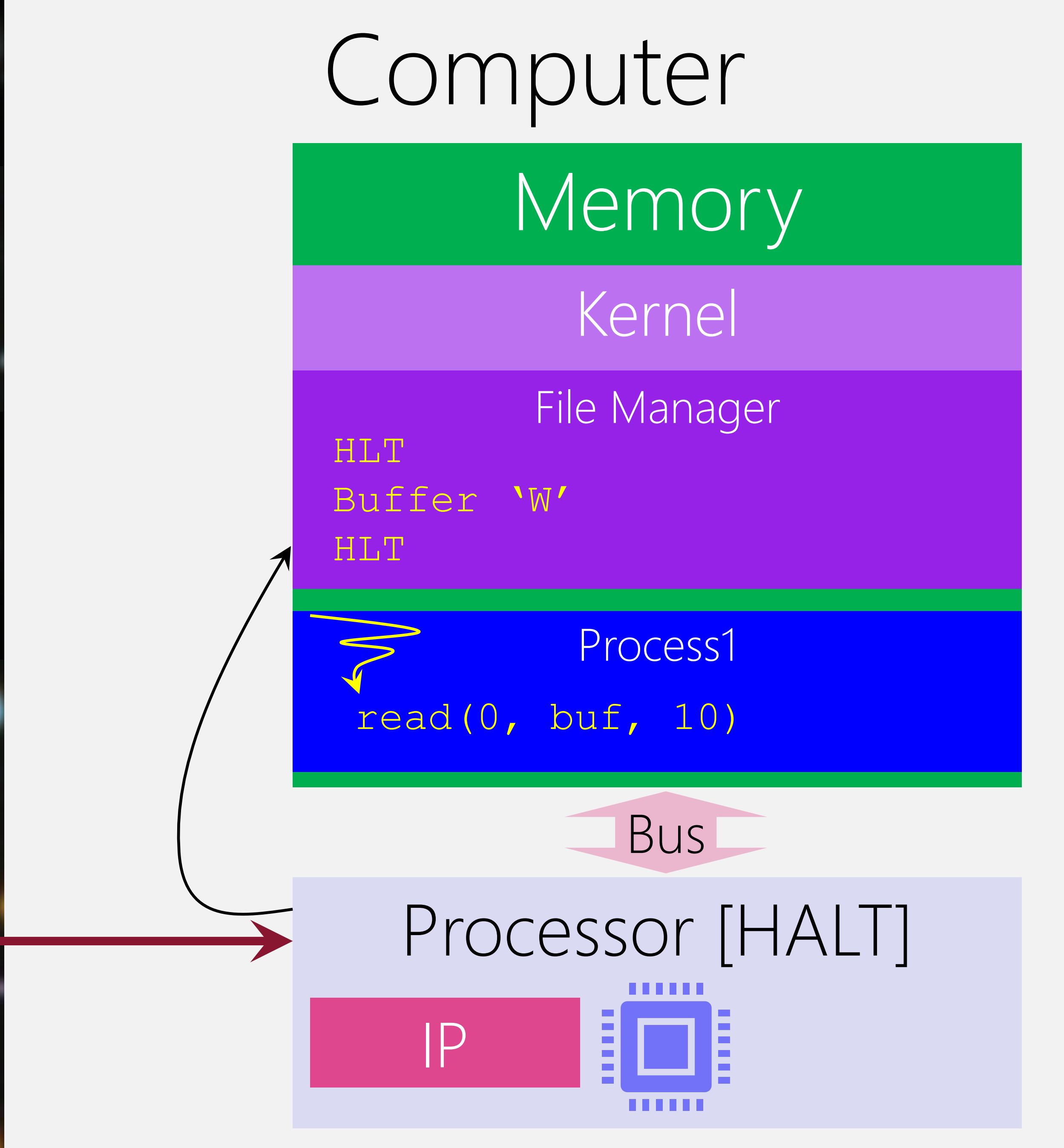


Week#2

Interrupt Request (IRQ)
Interrupt Request Handler

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





```
HLT
Buffer 'W'
HLT
```

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

Week#2

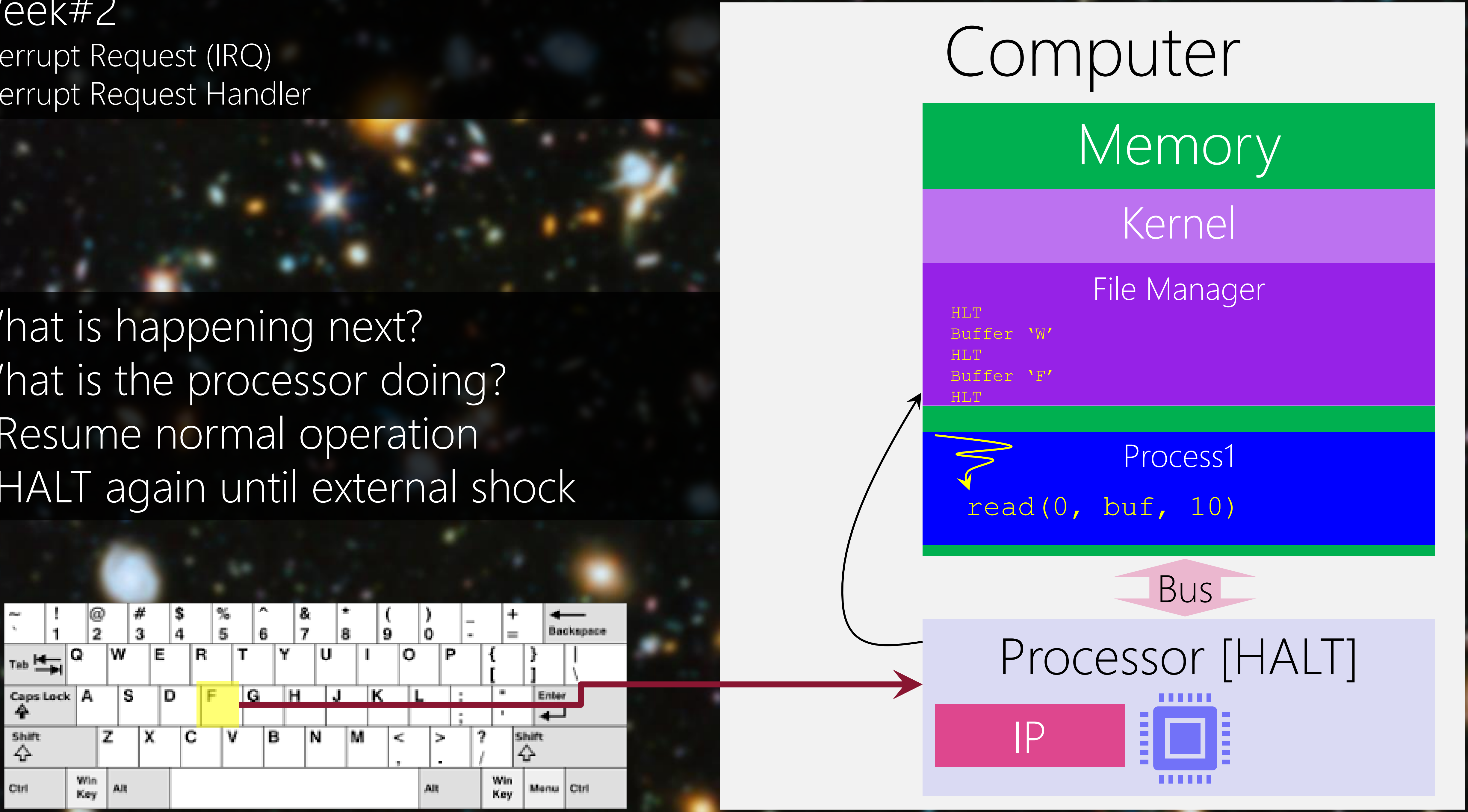
Interrupt Request (IRQ)

Interrupt Request Handler

What is happening next?

What is the processor doing?

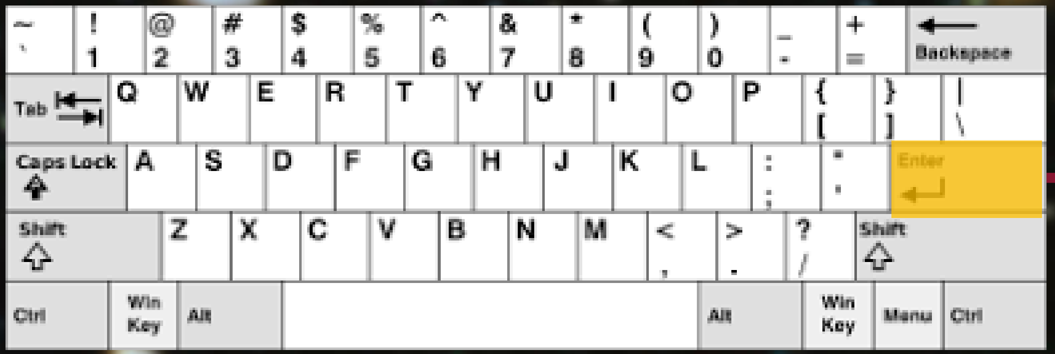
- Resume normal operation
- HALT again until external shock

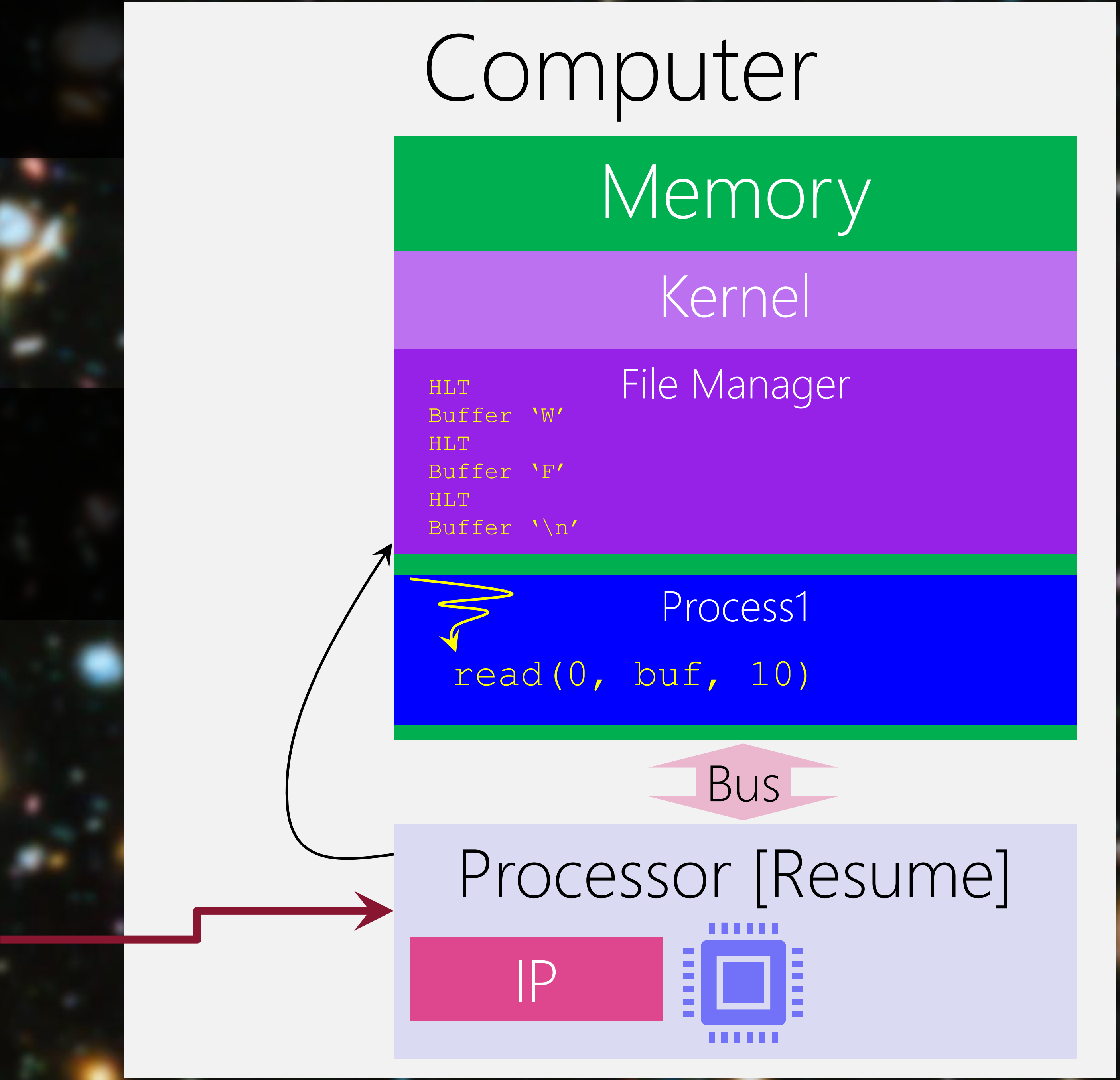


Week#2

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)
```

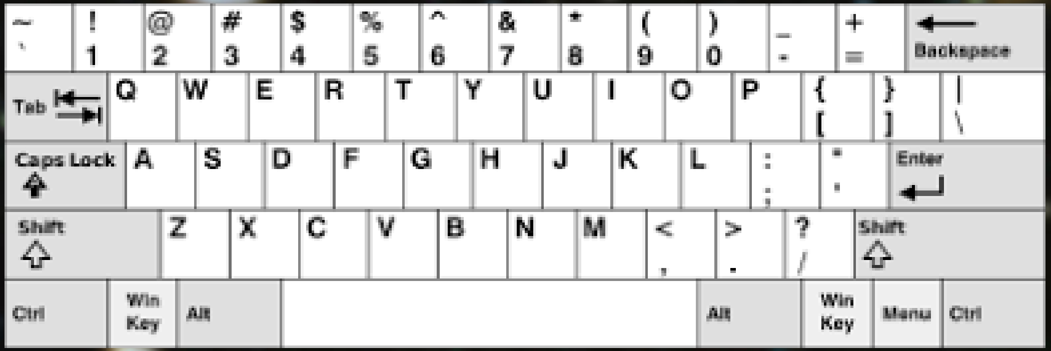
Week#2

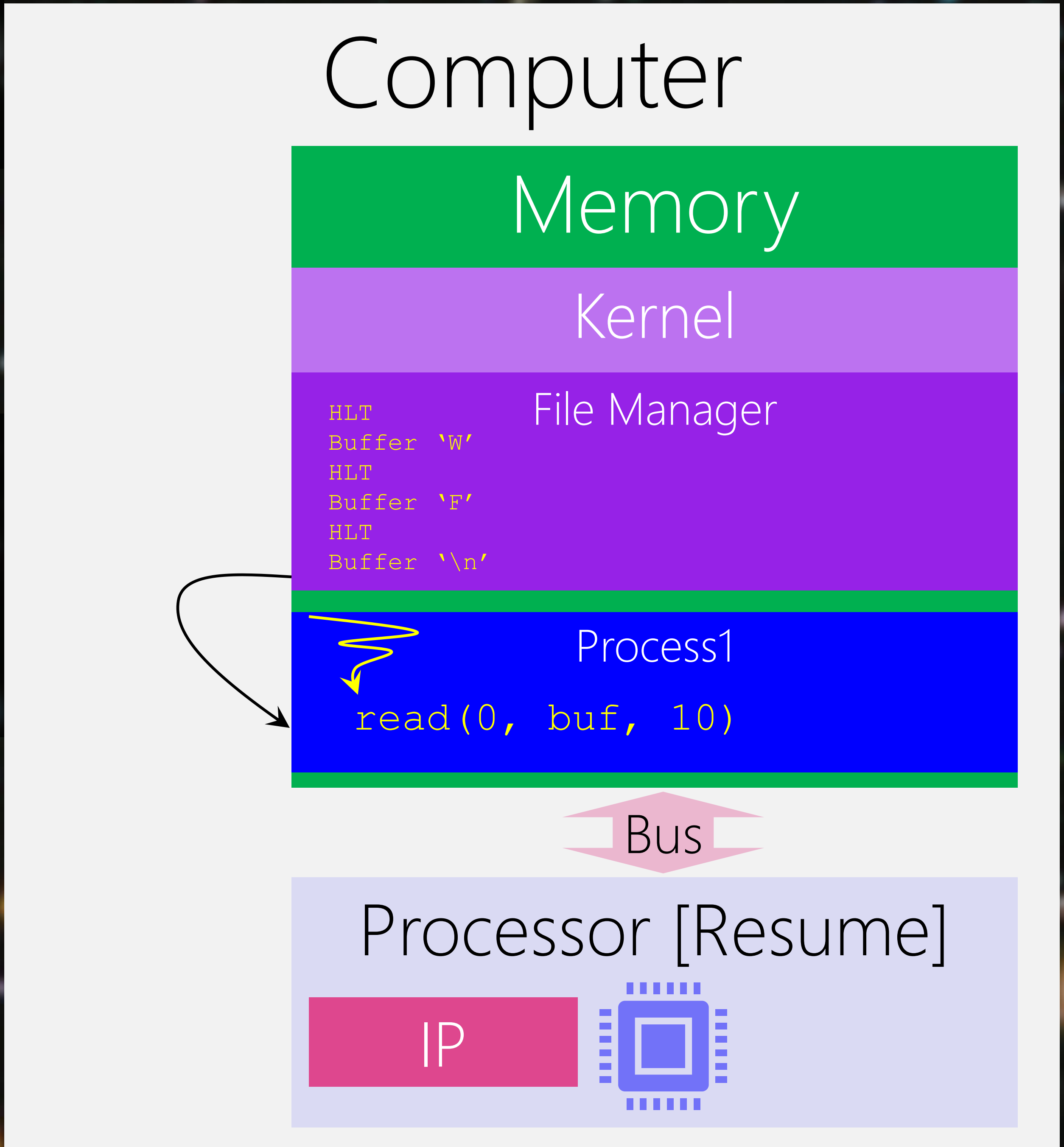
Interrupt Request (IRQ)
Interrupt Request Handler

What is happening next?

What is the processor doing?

- Resume normal operation
- Give control to the process

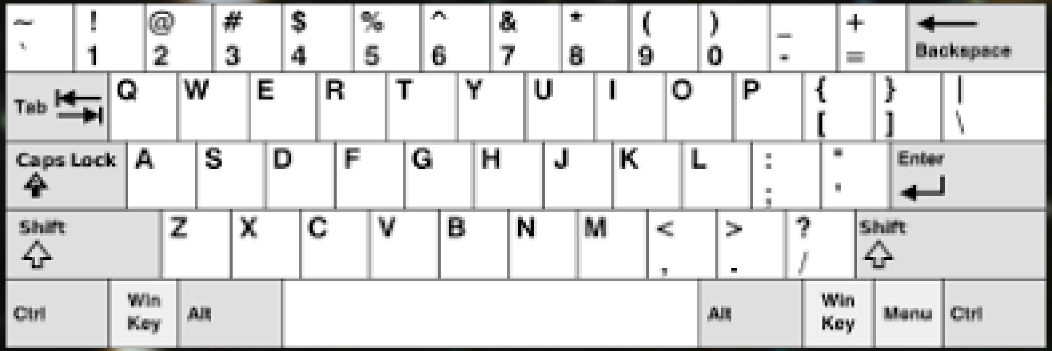


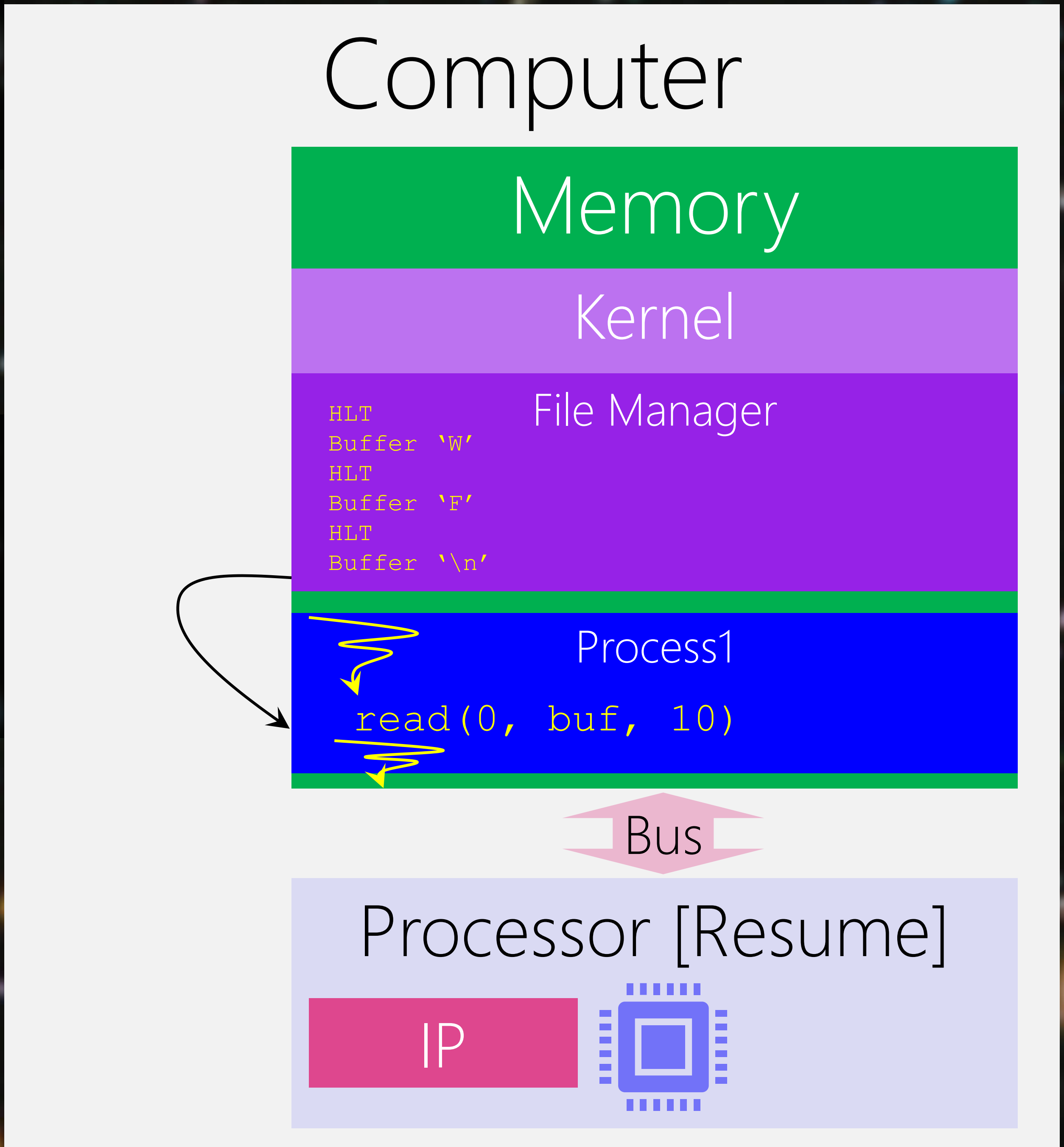


Week#2

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



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

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

Whether Busy Waiting or HALT

Waste of Processor

Single Processor ~~Multiprocessor~~

Whether Busy Waiting or HALT

Share it with another process

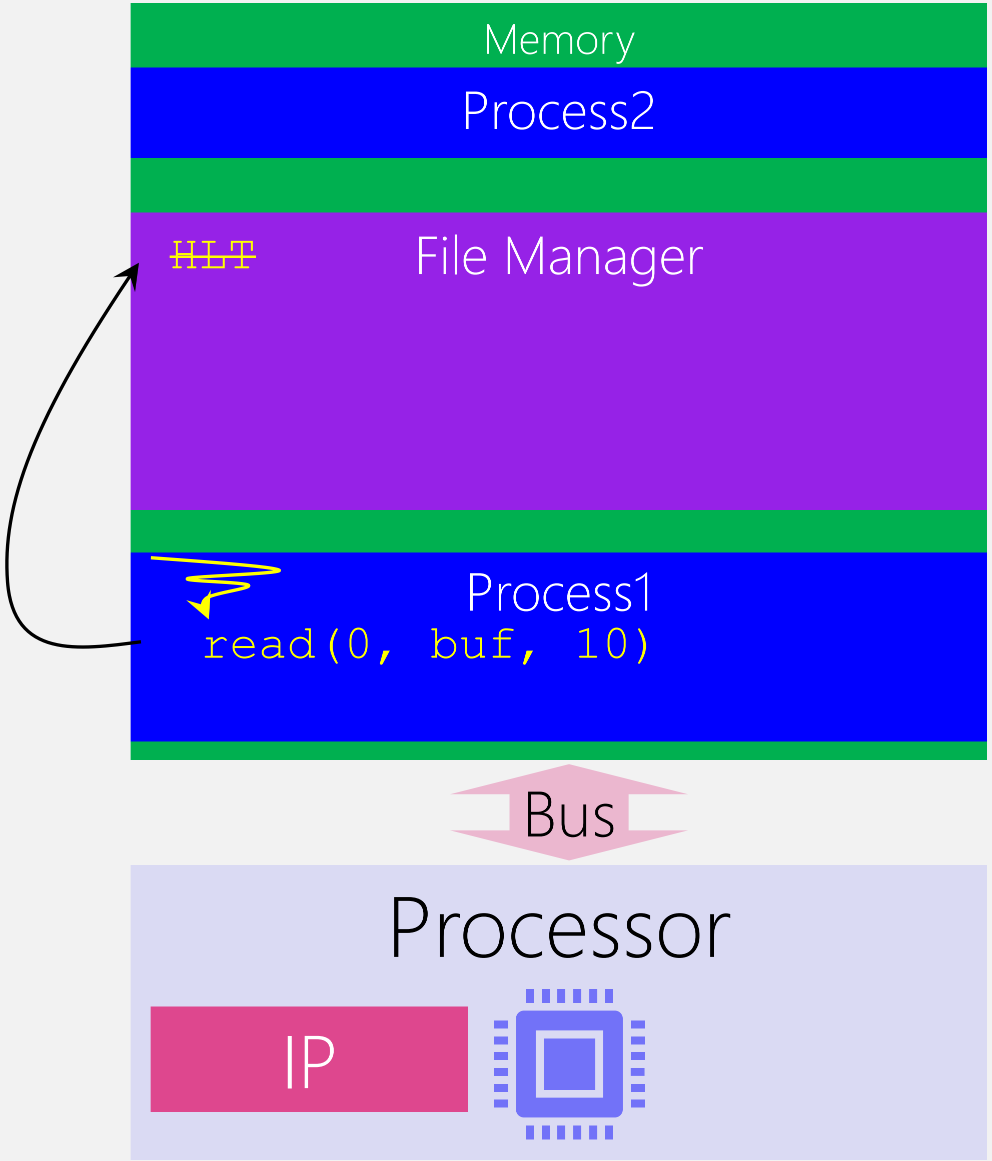
Single Processor ~~Multiprocessor~~

Whether Busy Waiting or HALT

Processor Sharing → Time Sharing/Slicing

Single Processor ~~Multiprocessor~~

Computer



Memory

Process2

~~HLT~~

File Manager

Process1

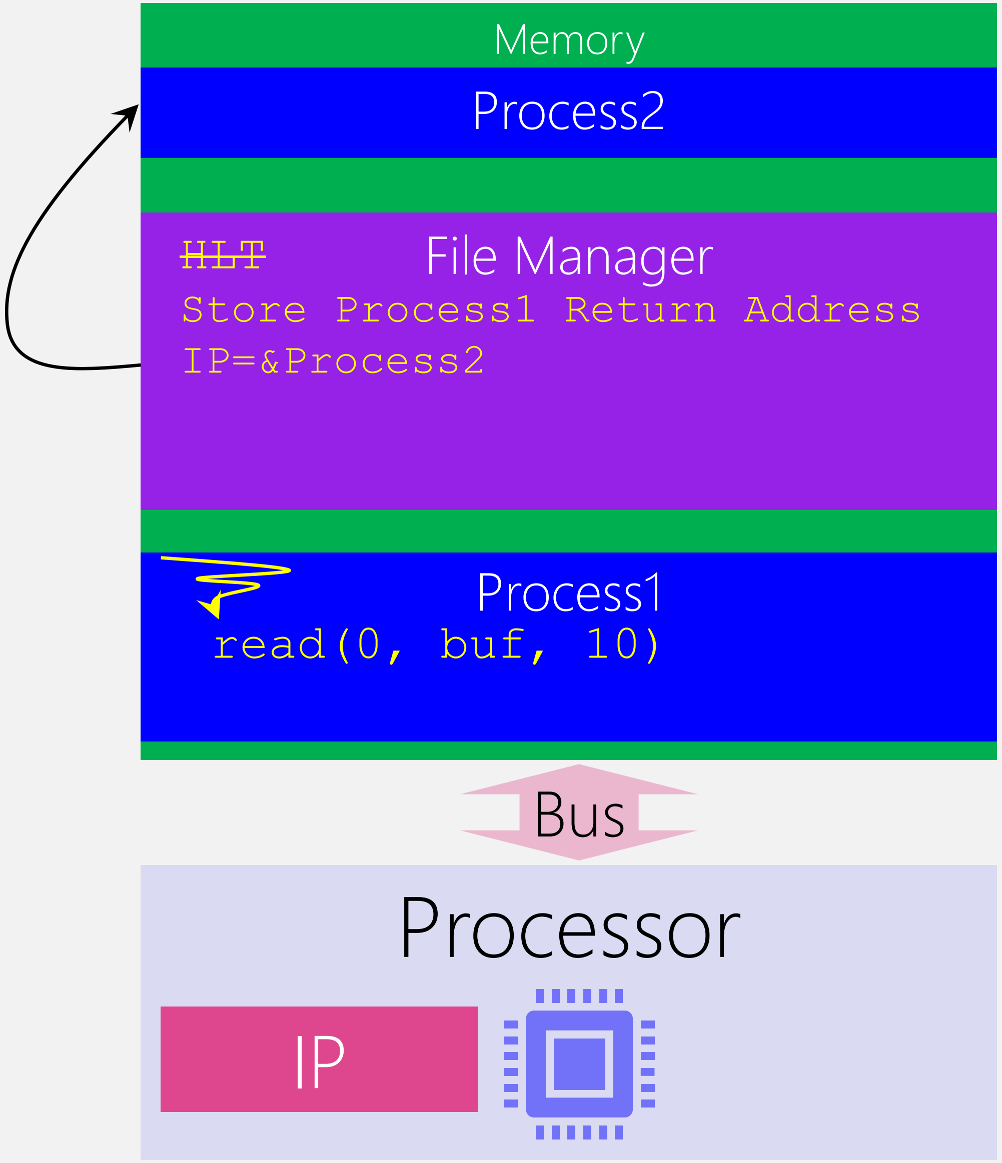
```
read(0, buf, 10)
```

Bus

Processor

IP

Computer



Memory

Process2

~~HLT~~

File Manager

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

Process1

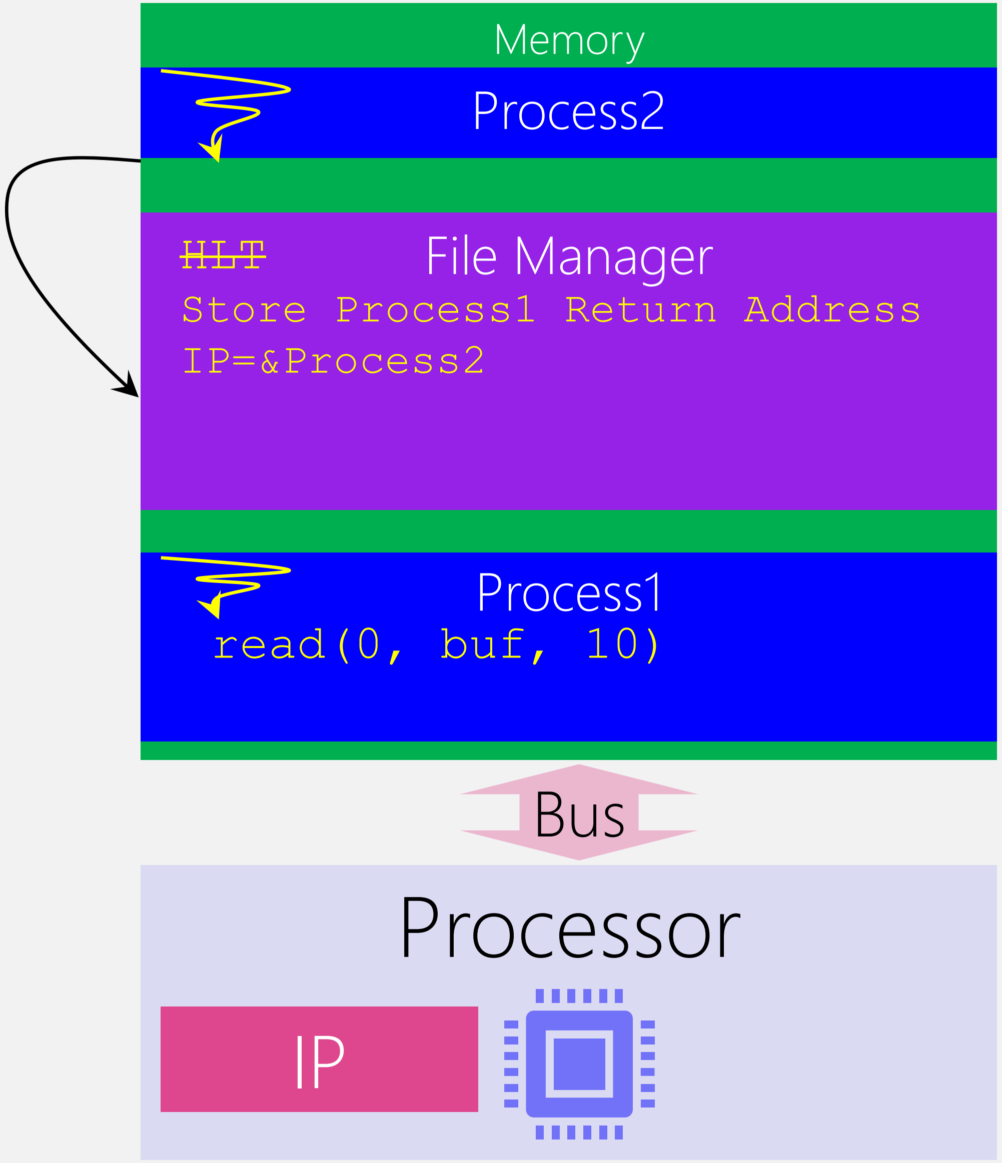
```
read(0, buf, 10)
```

Bus

Processor

IP

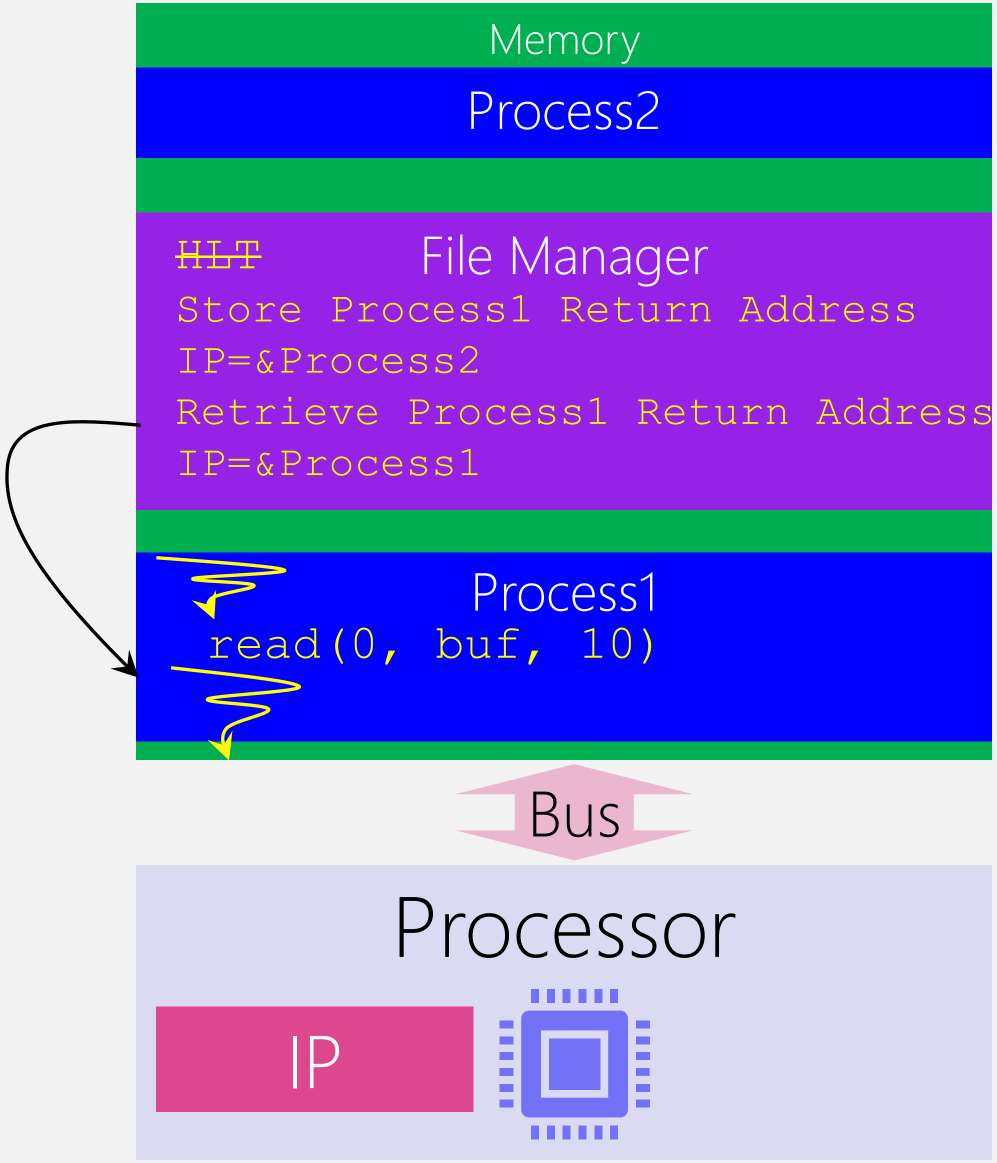
Computer



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

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

Computer



Memory

Process2

~~HLT~~

File Manager

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

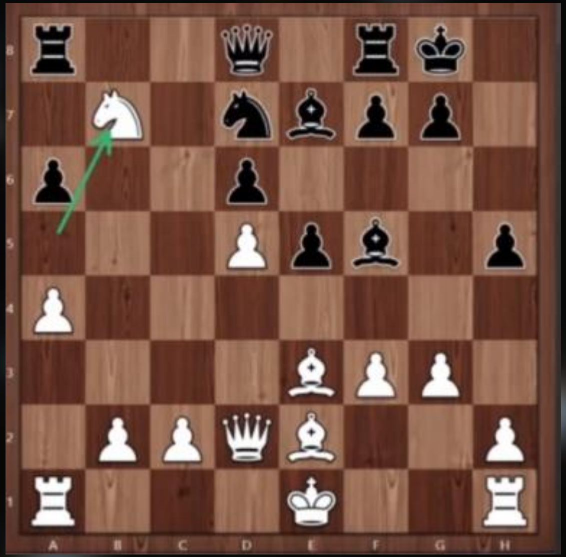
Process1

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

It’s not that simple, tho!

Further Reading → Process Context Switch

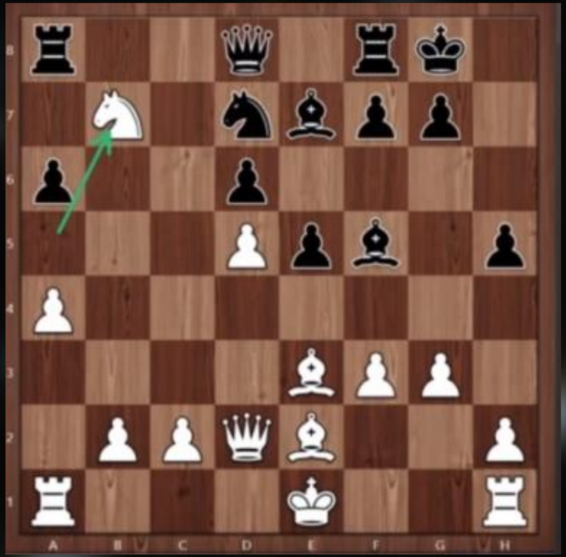
Magnus Carlsen



Hikaru Nakamura

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

Magnus Carlsen



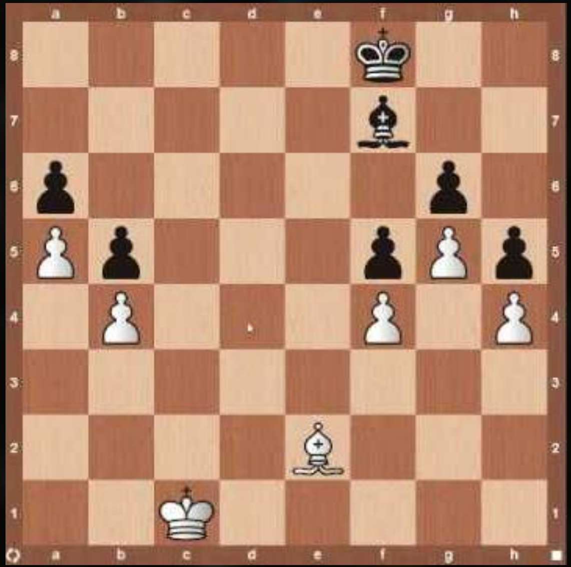
Hikaru Nakamura

Sure! →

Eris Li

Magnus Carlsen

Can we have it back?



Hikaru Nakamura

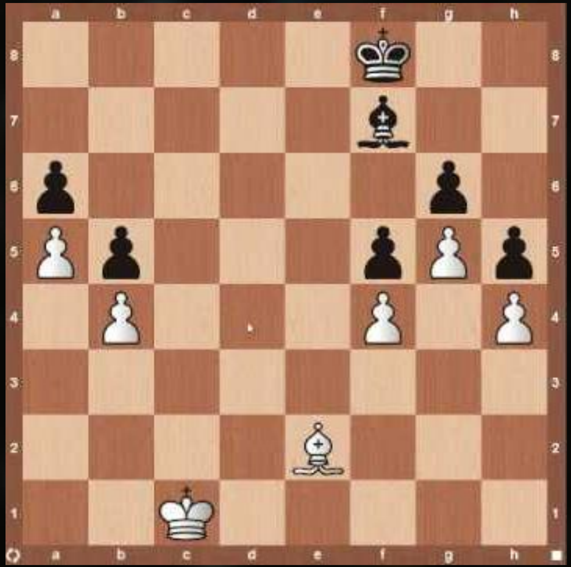
Eris Li

Magnus Carlsen

Where were we?!

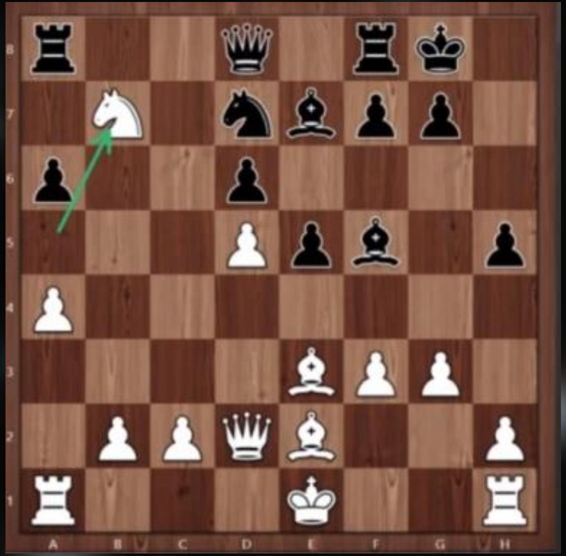
← Sure!

Hikaru Nakamura



Eris Li

Magnus Carlsen



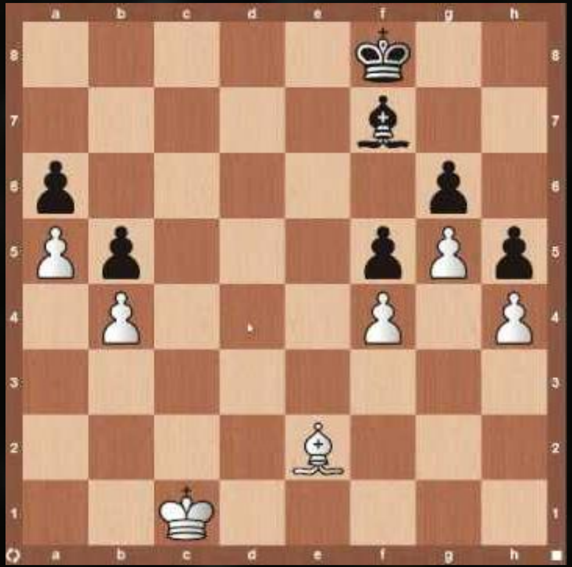
← Sure!

Hikaru Nakamura

Eris Li

Magnus Carlsen

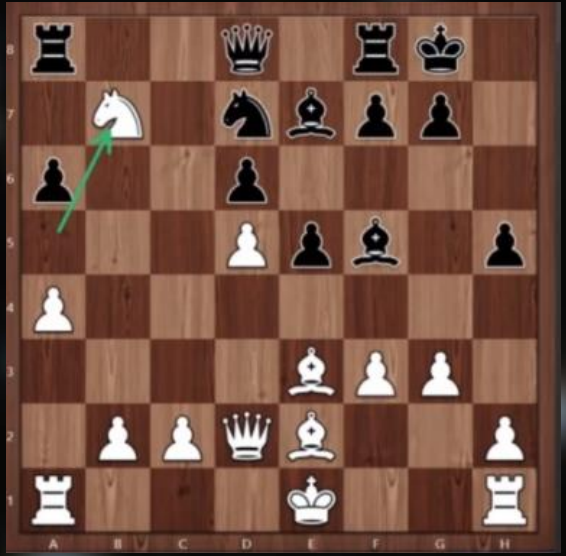
Sure! →



Hikaru Nakamura

Eris Li

Magnus Carlsen

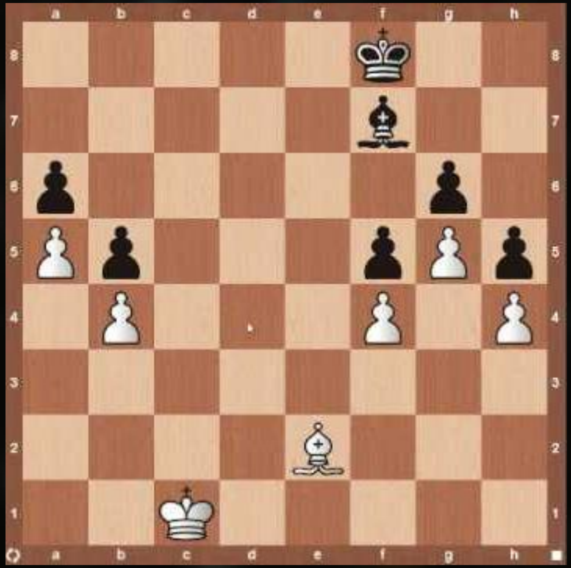


Hikaru Nakamura

Sure! →

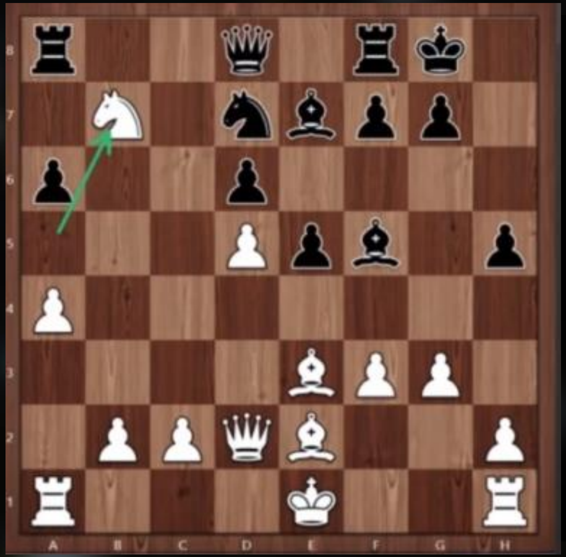
← Sure!

100 nano to 10 microseconds!



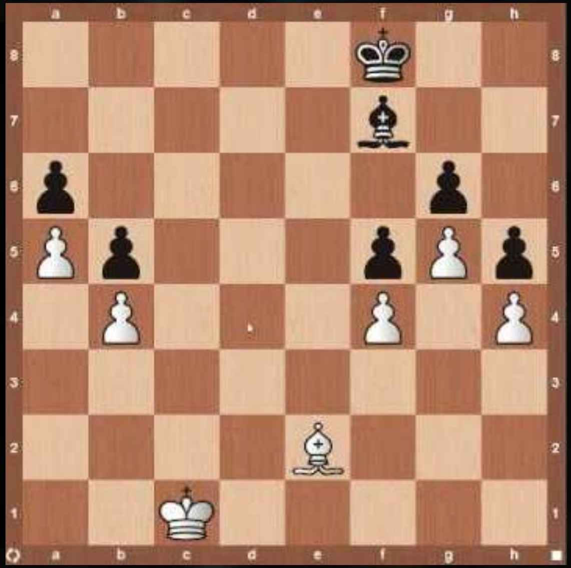
Eris Li

Magnus Carlsen



Seems we have two chessboard

100 nano to 10 microseconds



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

Sharing 1 Chessboard

Creating a New Process

Creating a New Process

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

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

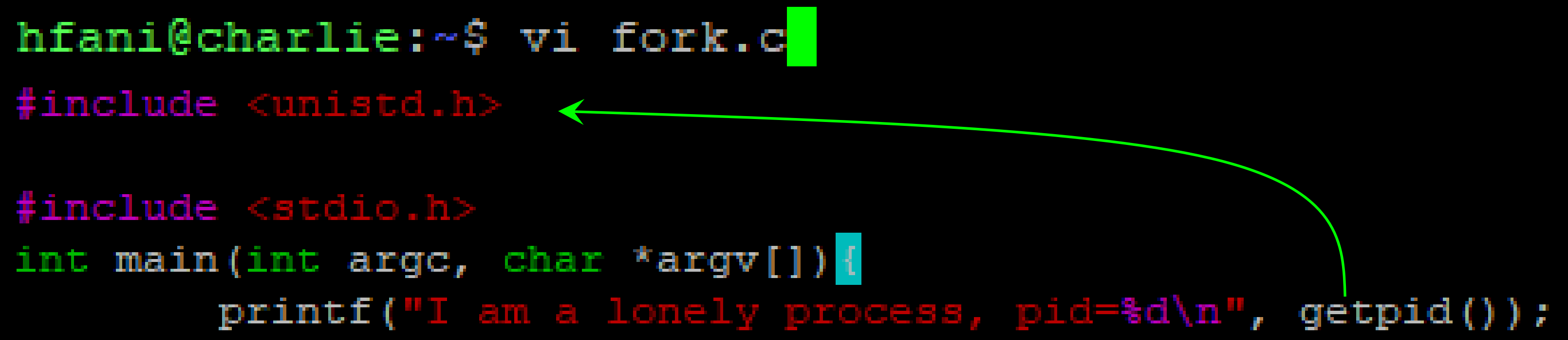
Returns: 0 in child, PID of child in parent, –1 on error

Parent vs. Child Process

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

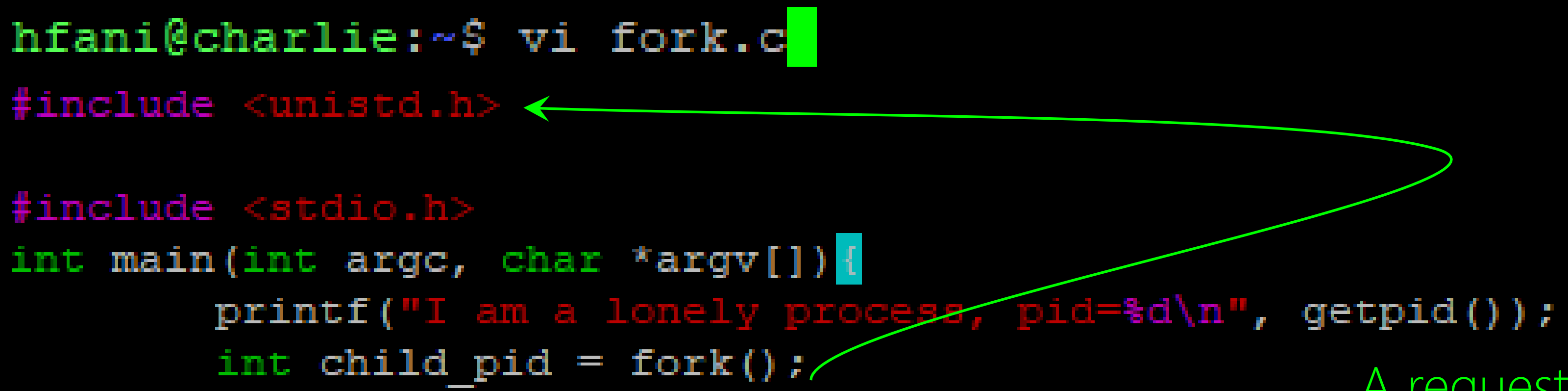
Only an existing process can create a new process.
Because somebody should do the system call!

Compile Time Analysis



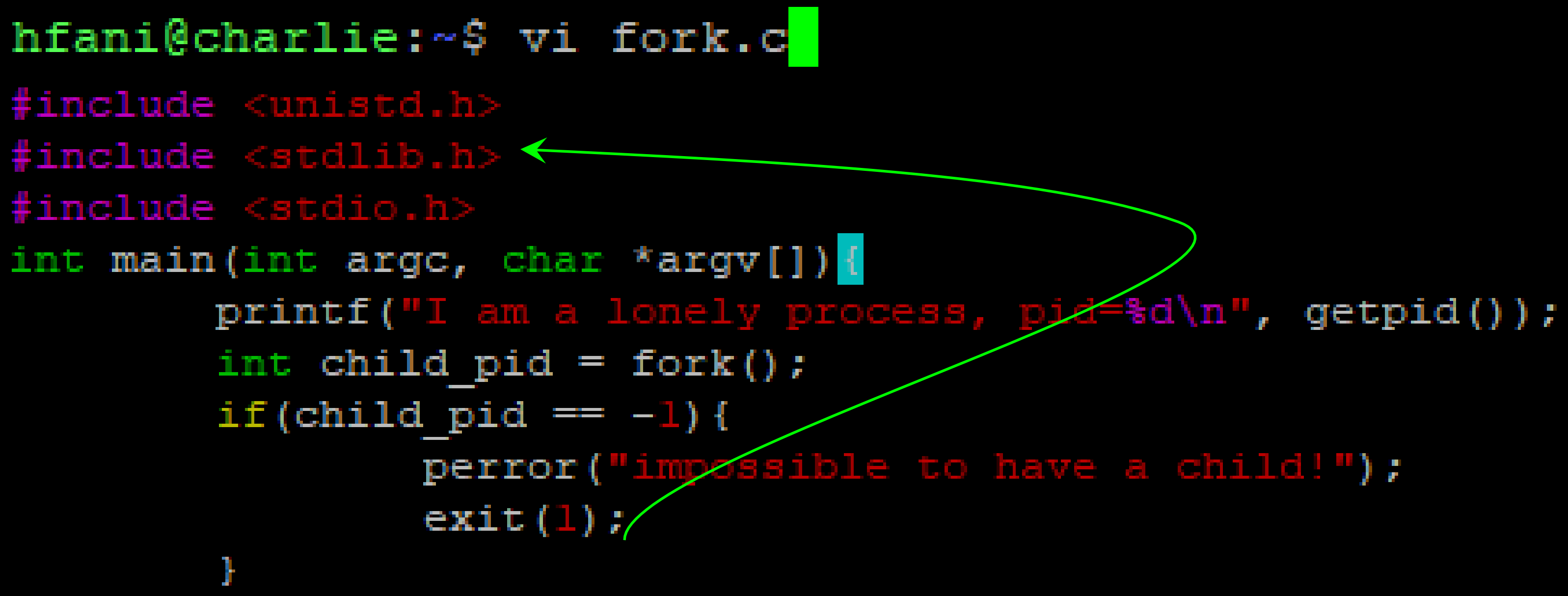
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>

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

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



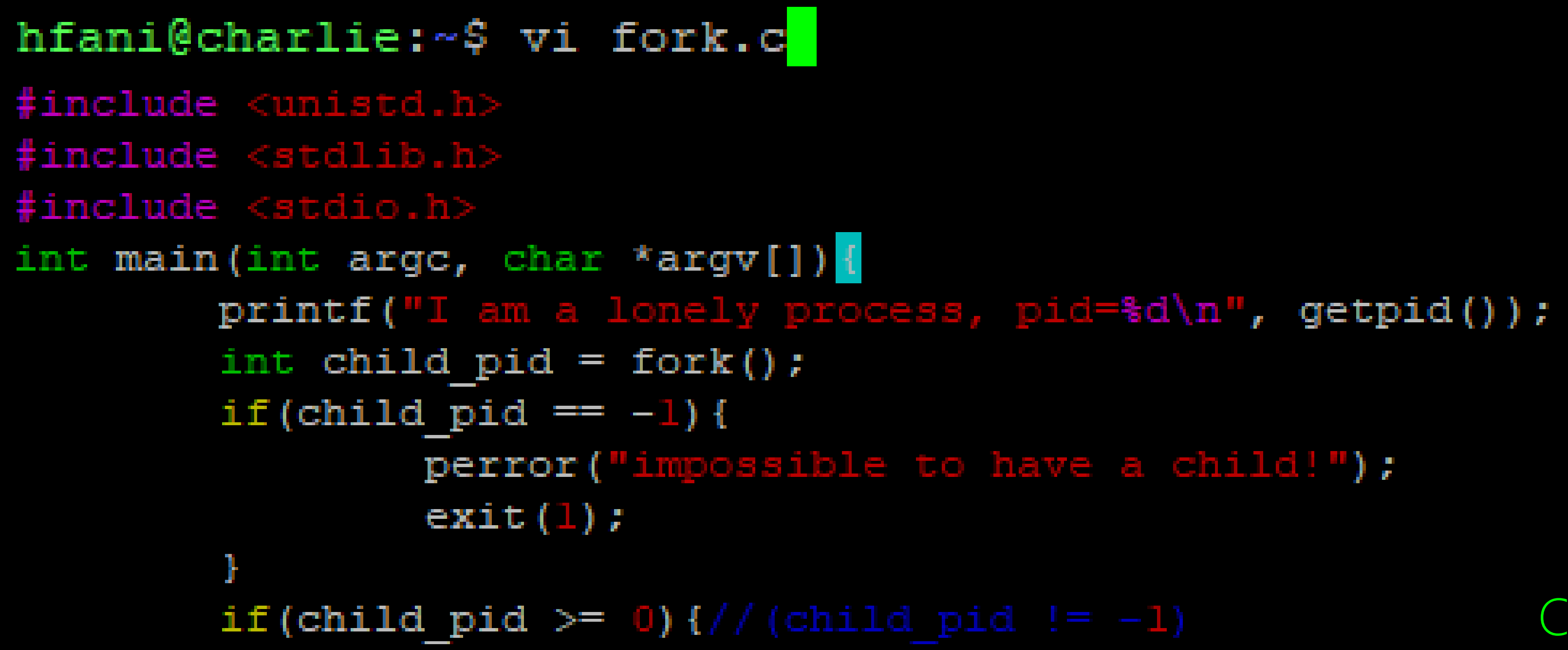
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");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

Compile Time Analysis

-1 on error in having a child

Exit the process with an error status Nonzero!

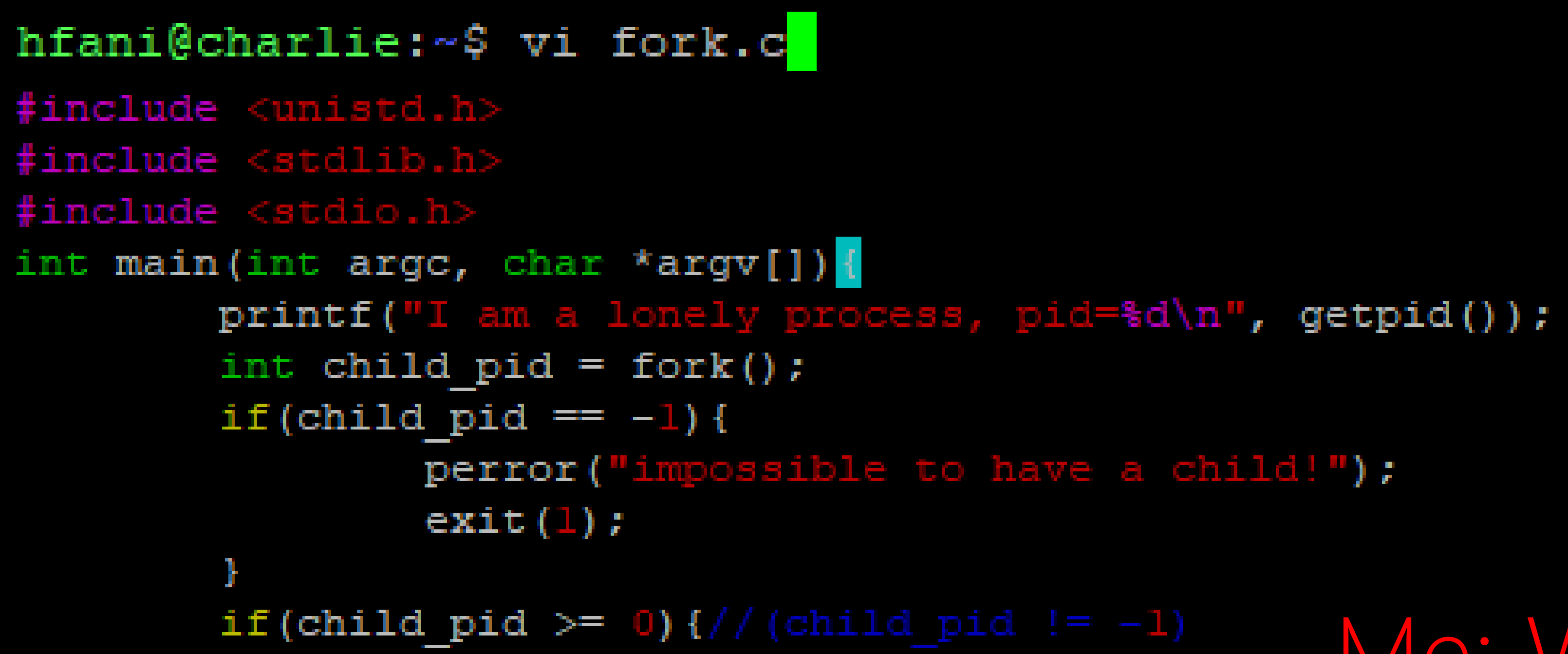
Compile Time Analysis



```
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.

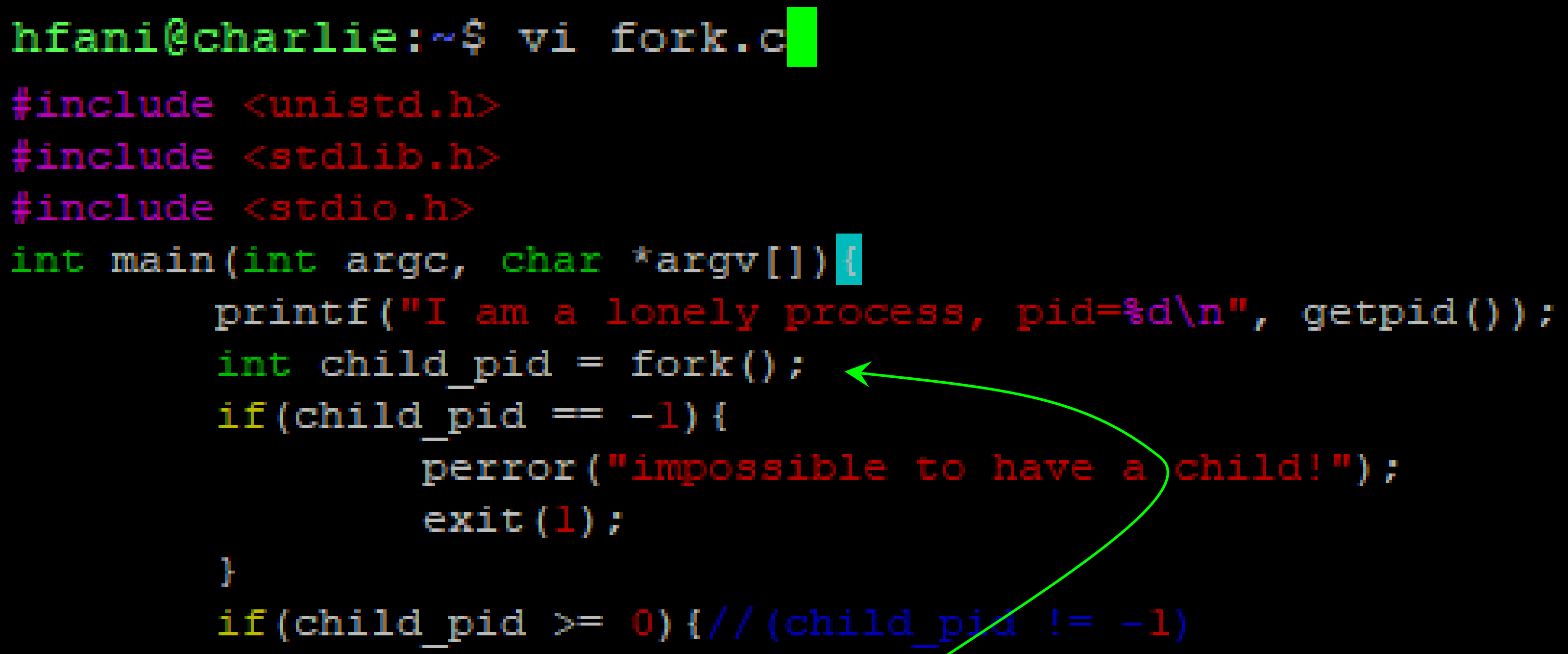
Compile Time Analysis



```
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?!

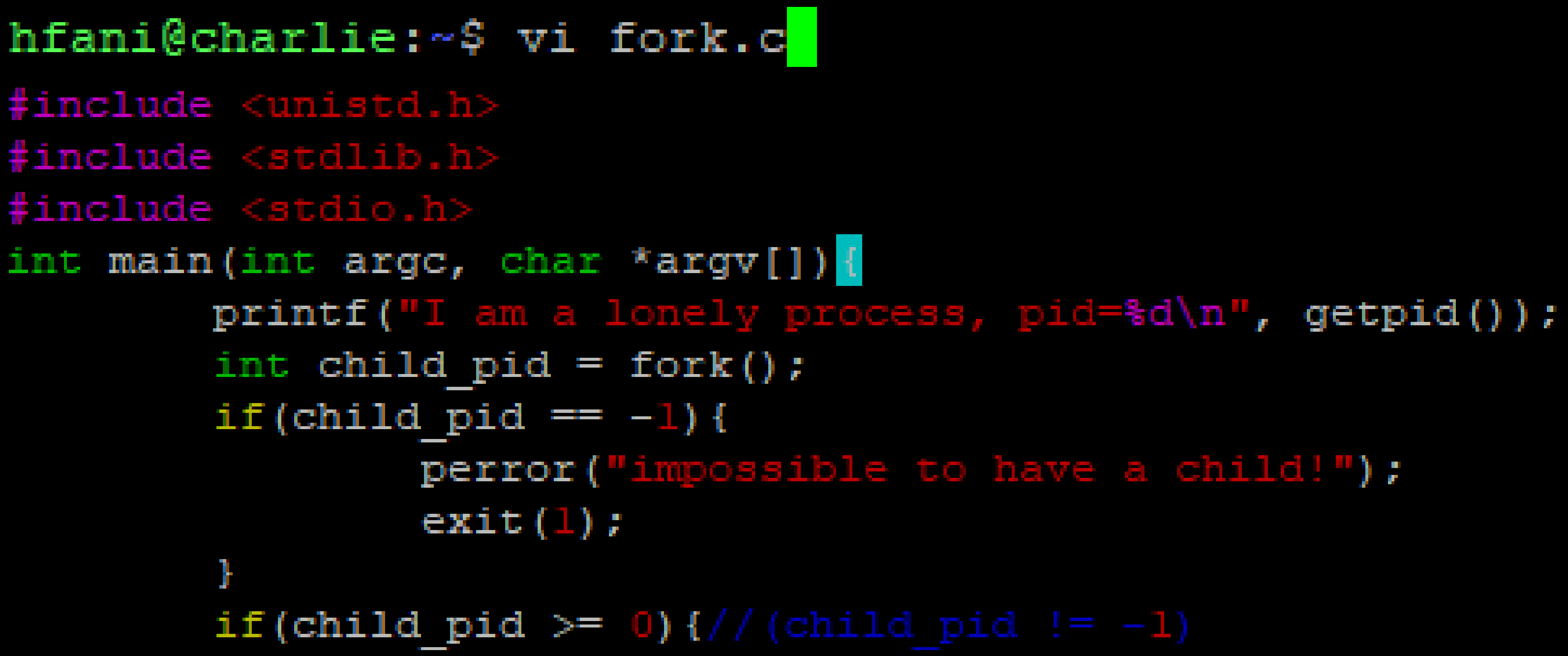
Compile Time Analysis



```
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

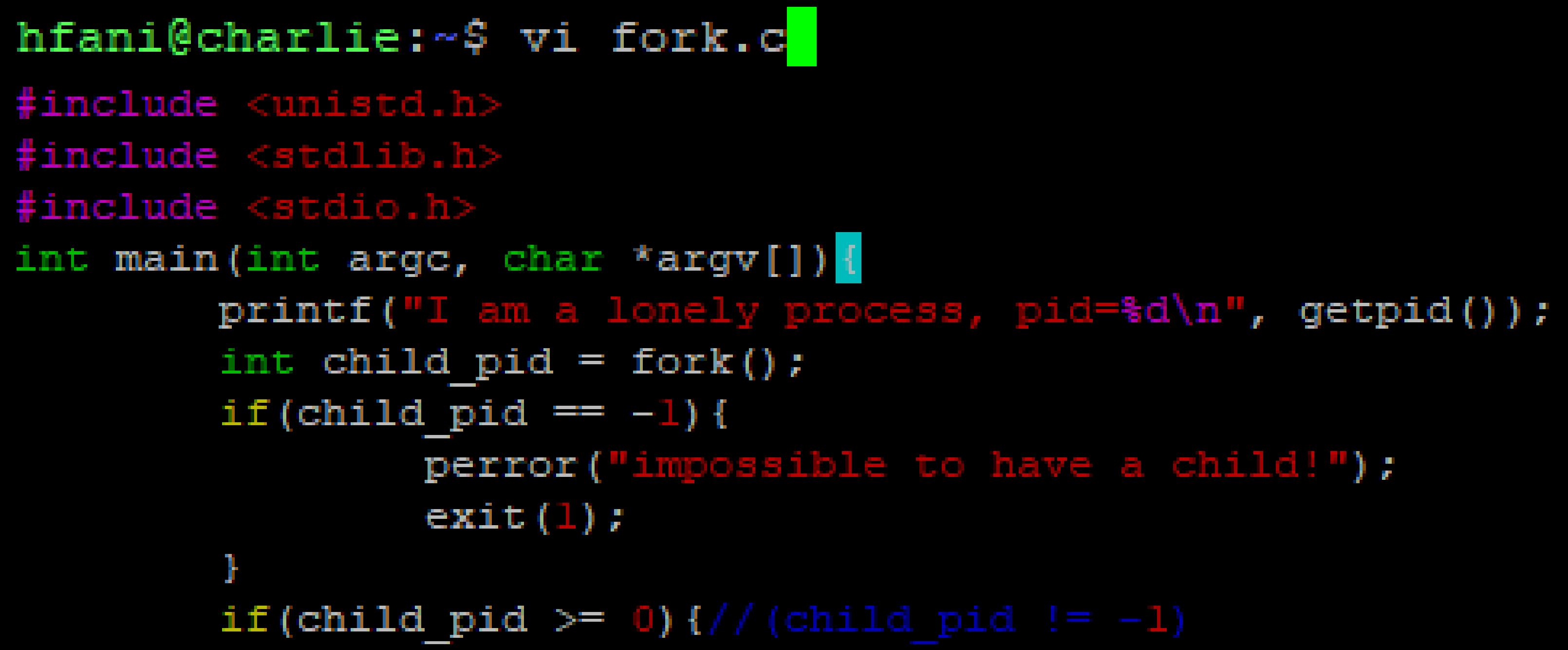


```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");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

Kernel:

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

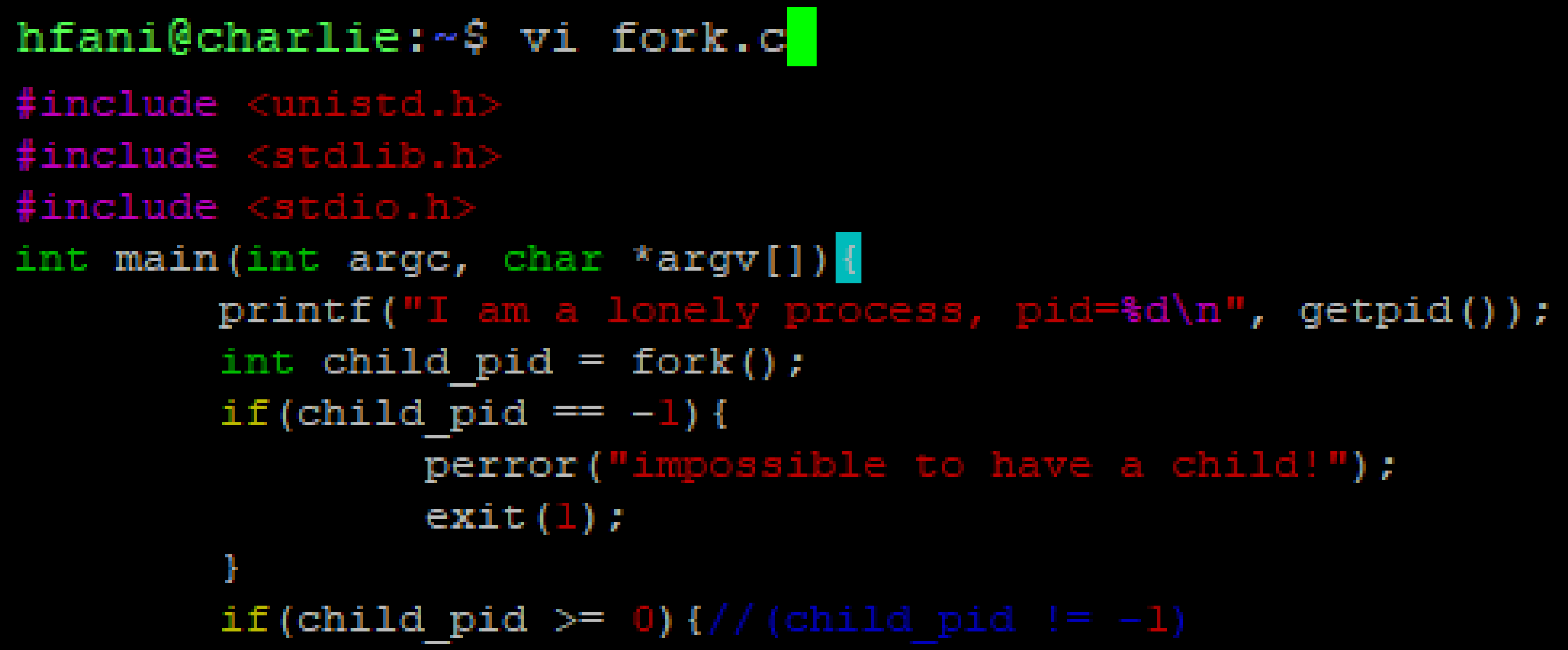
Oh, the child is exactly a copy of you (clone)

Same age, same gender, same color, …

Indeed, it is very hard to distinguish yourself from your child.

Which one is my hat?, The Prestige (2006) - Christopher Nolan

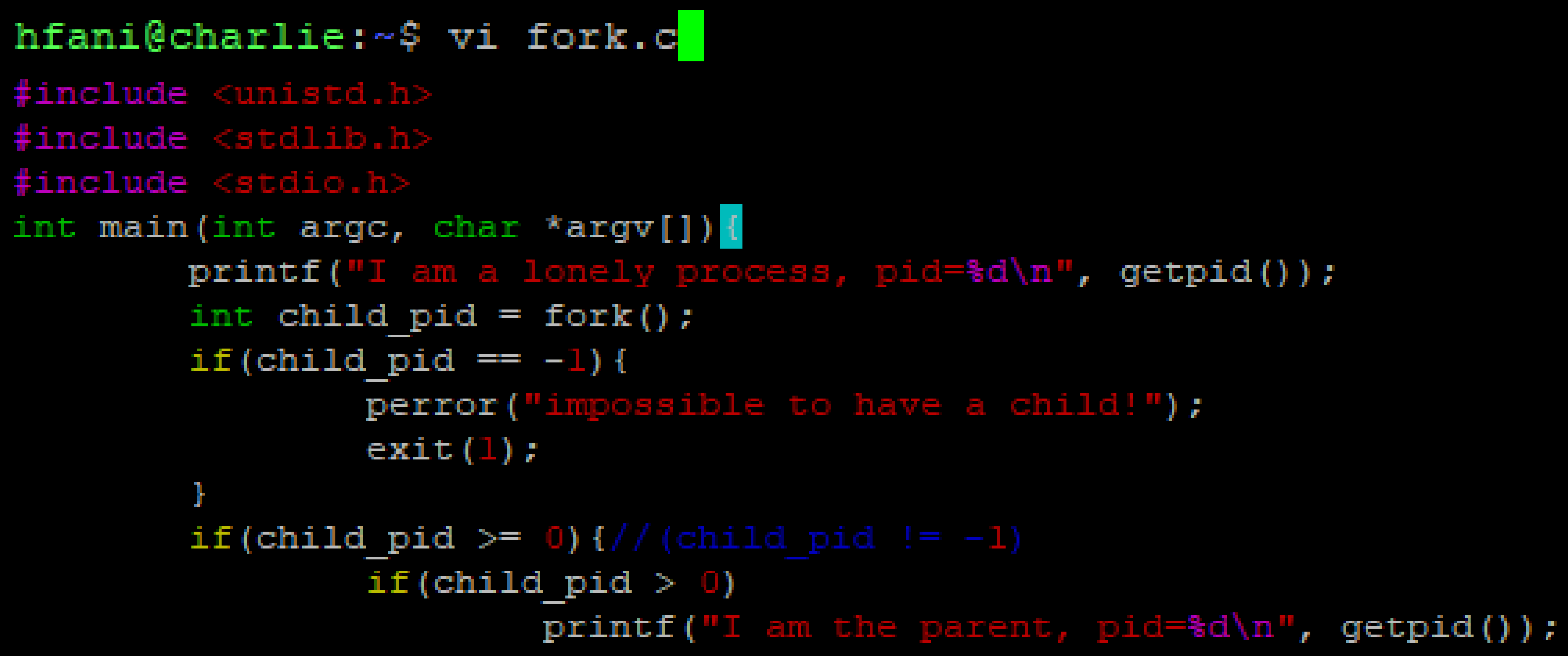
Compile Time Analysis



```
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



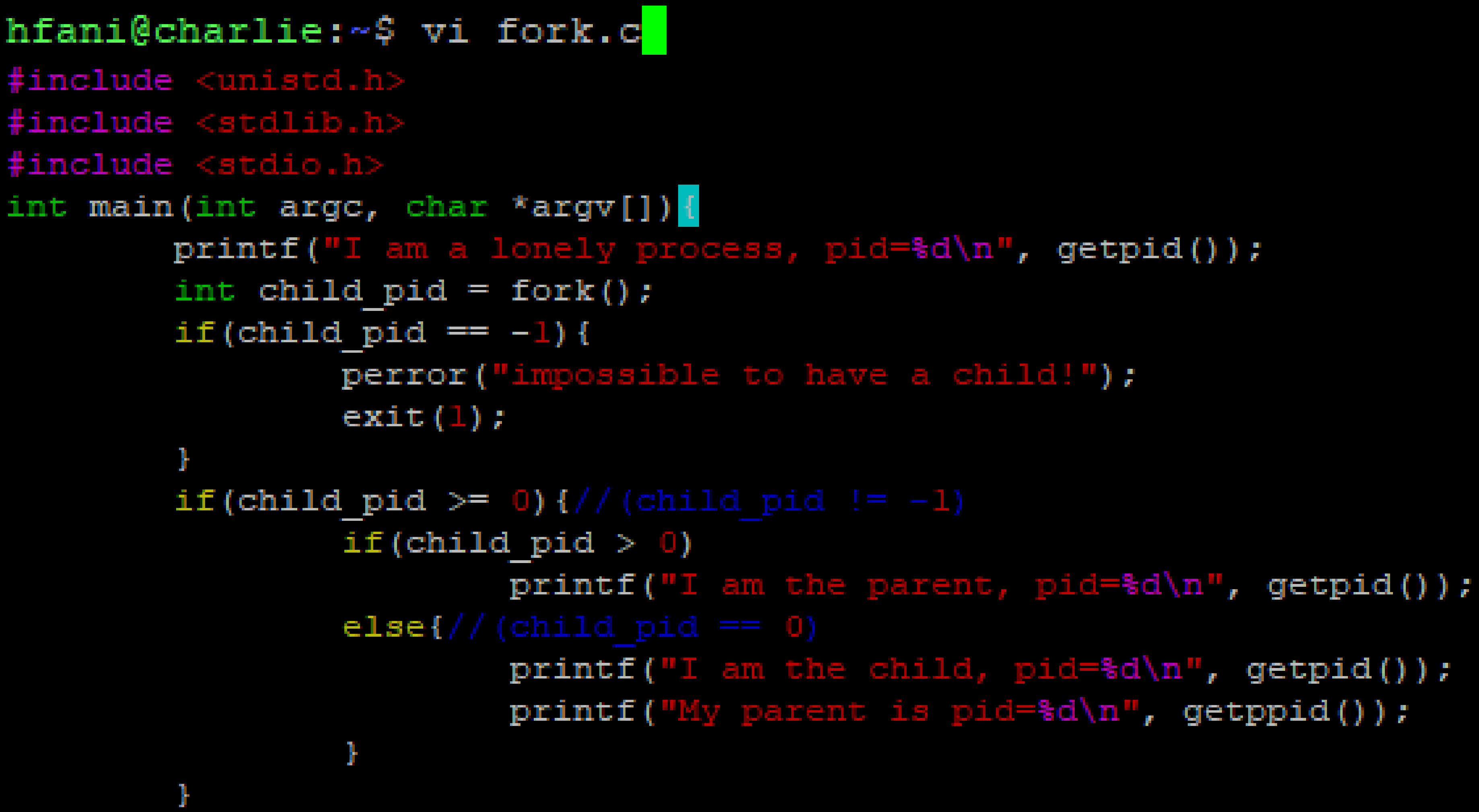
```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");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

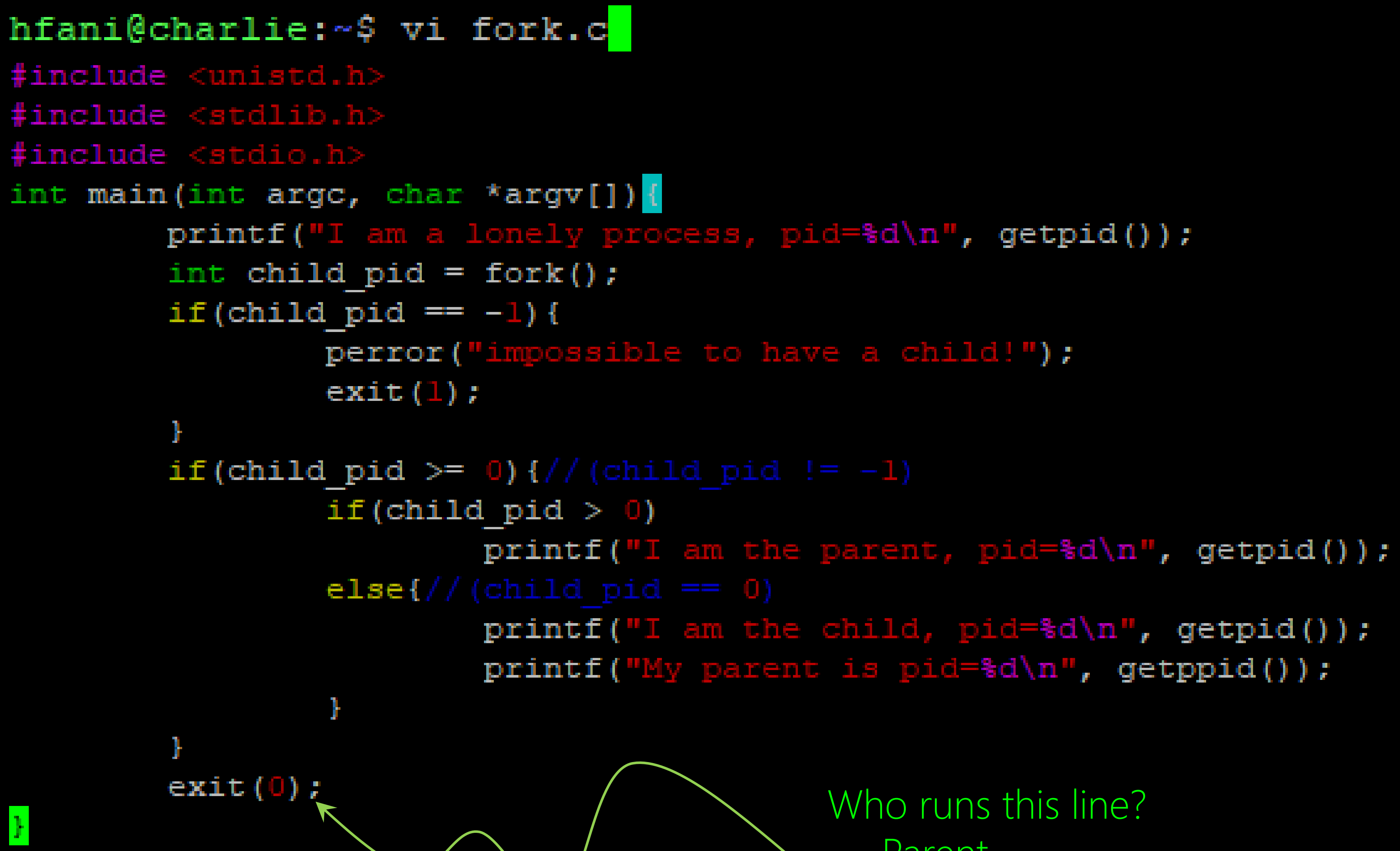
Compile Time Analysis



```
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());
                }
        }
```

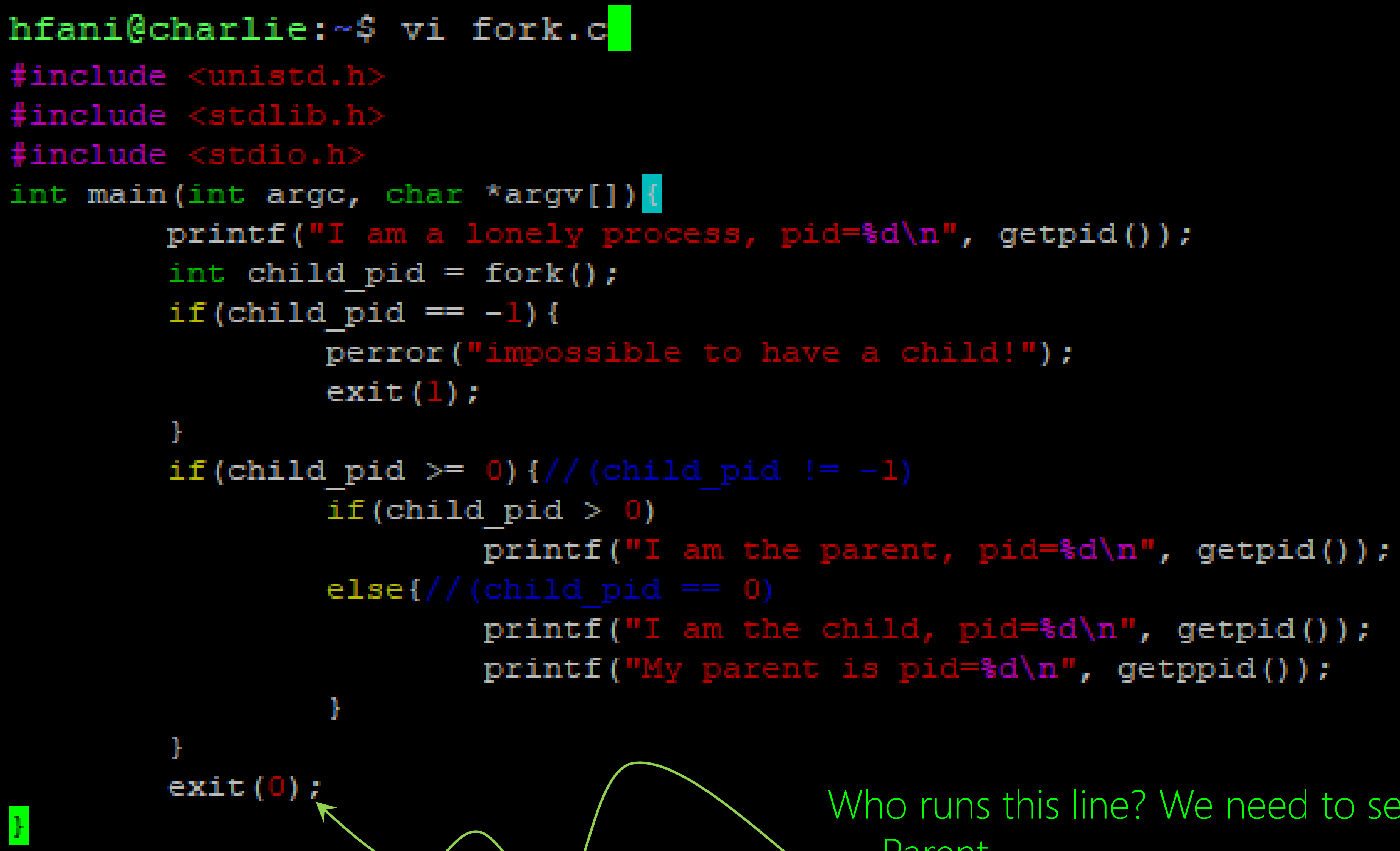
Kernel:
If the child_pid is 0, it means you're are the child.
If you want to know your pid, use `getpid()` system call.
If you want to know your parent pid, use `getppid()` system call.

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)
                        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

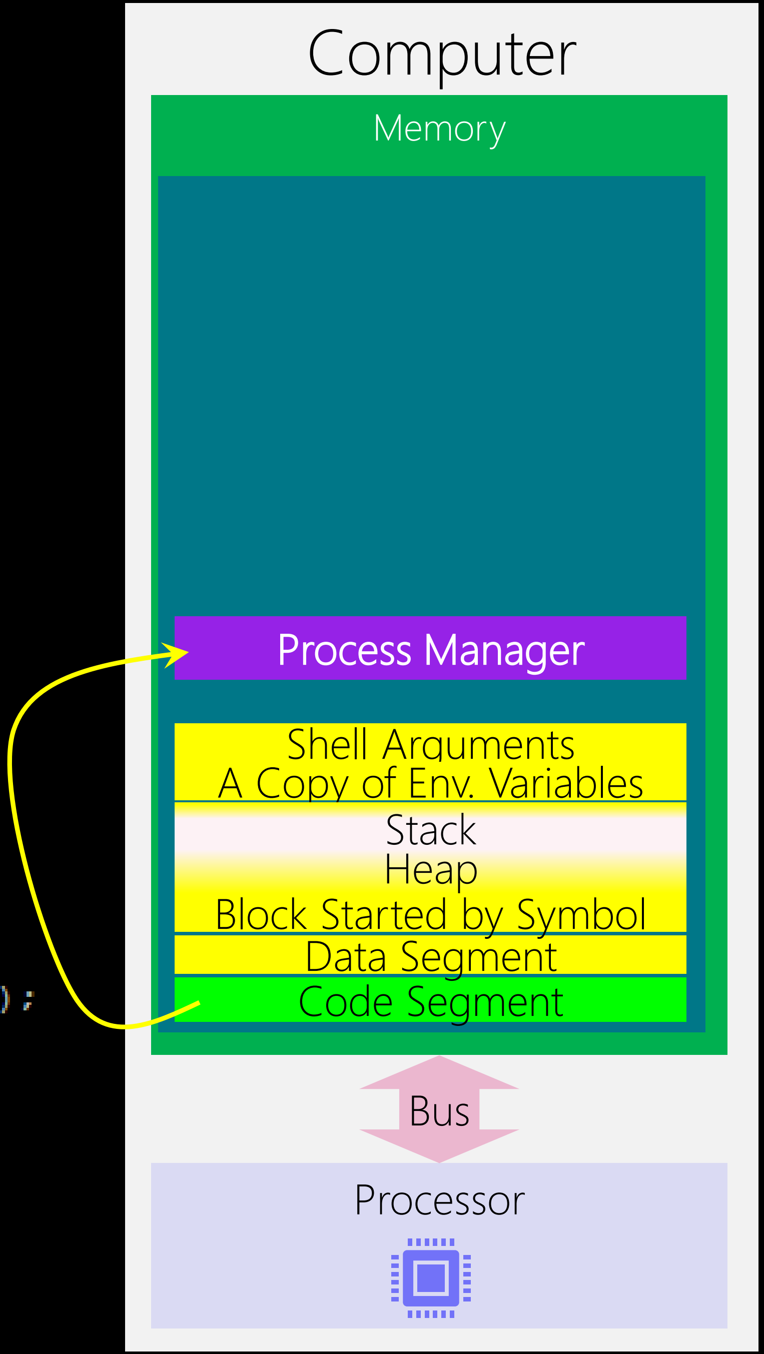


```
hfani@charlie:~$ vi fork.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");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

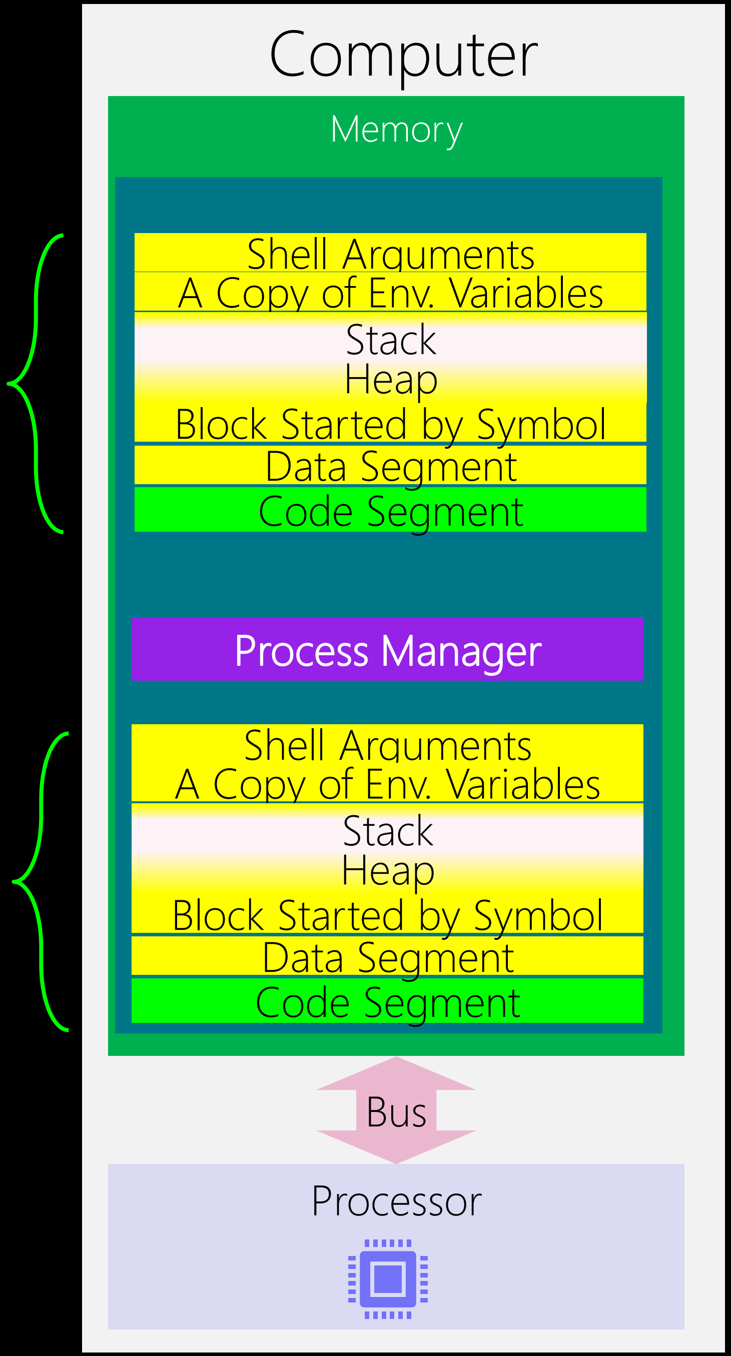
- Parent
- Child
- Both
- None



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

Exact copy at `fork()`

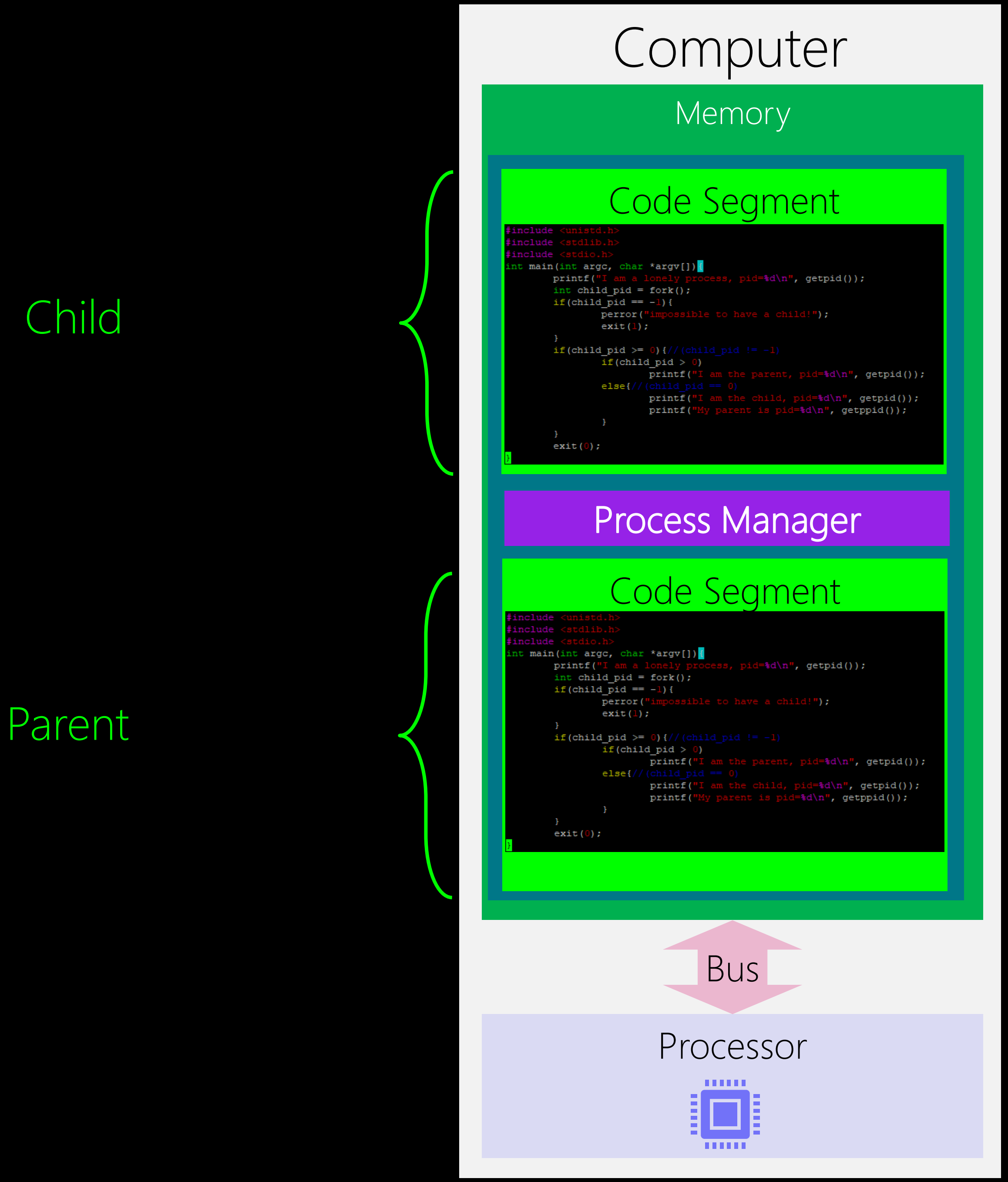
Child



Any change by the child is in the child copy

Parent

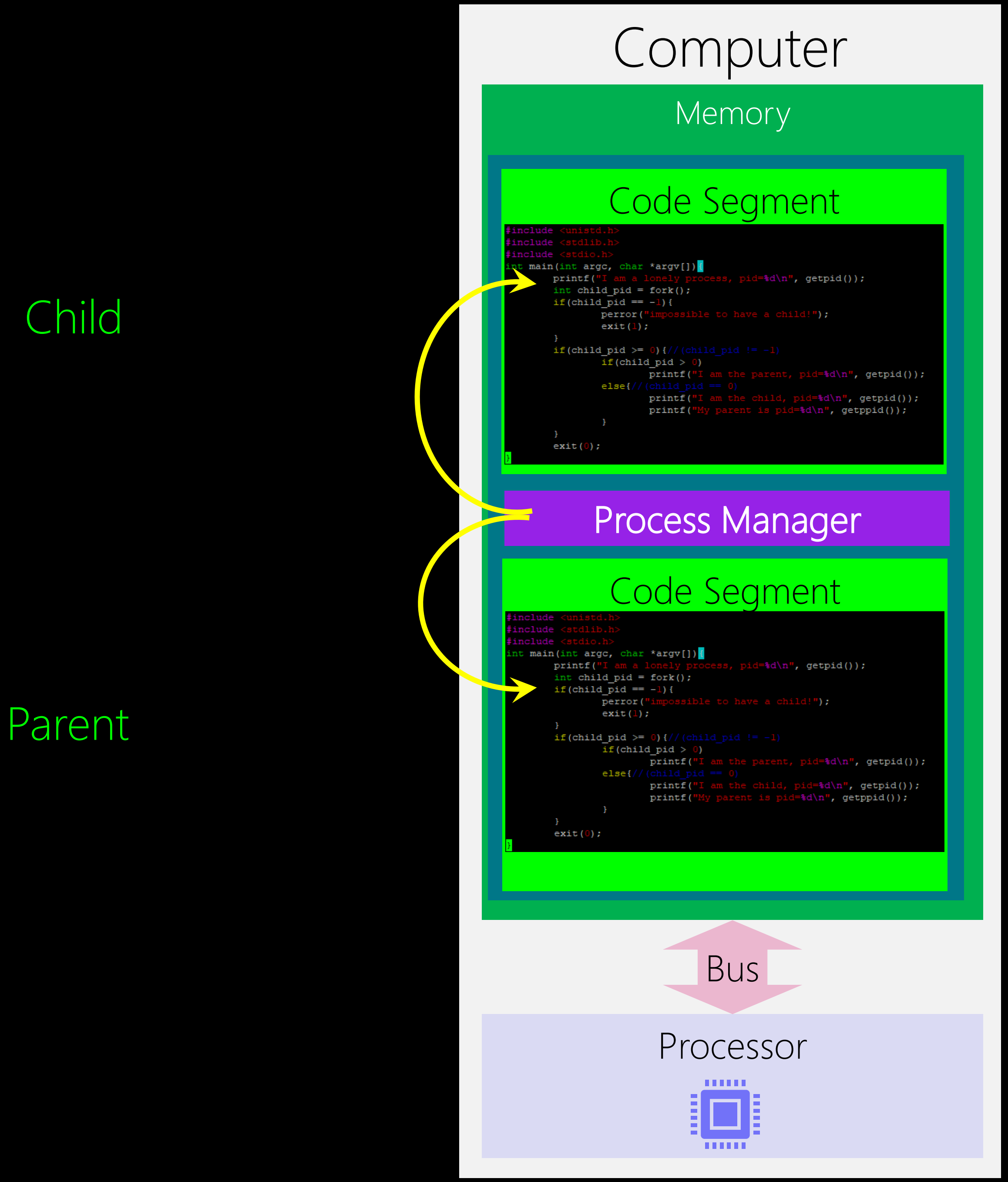
Any change by the parent is in the parent copy



```
#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);
}
```

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



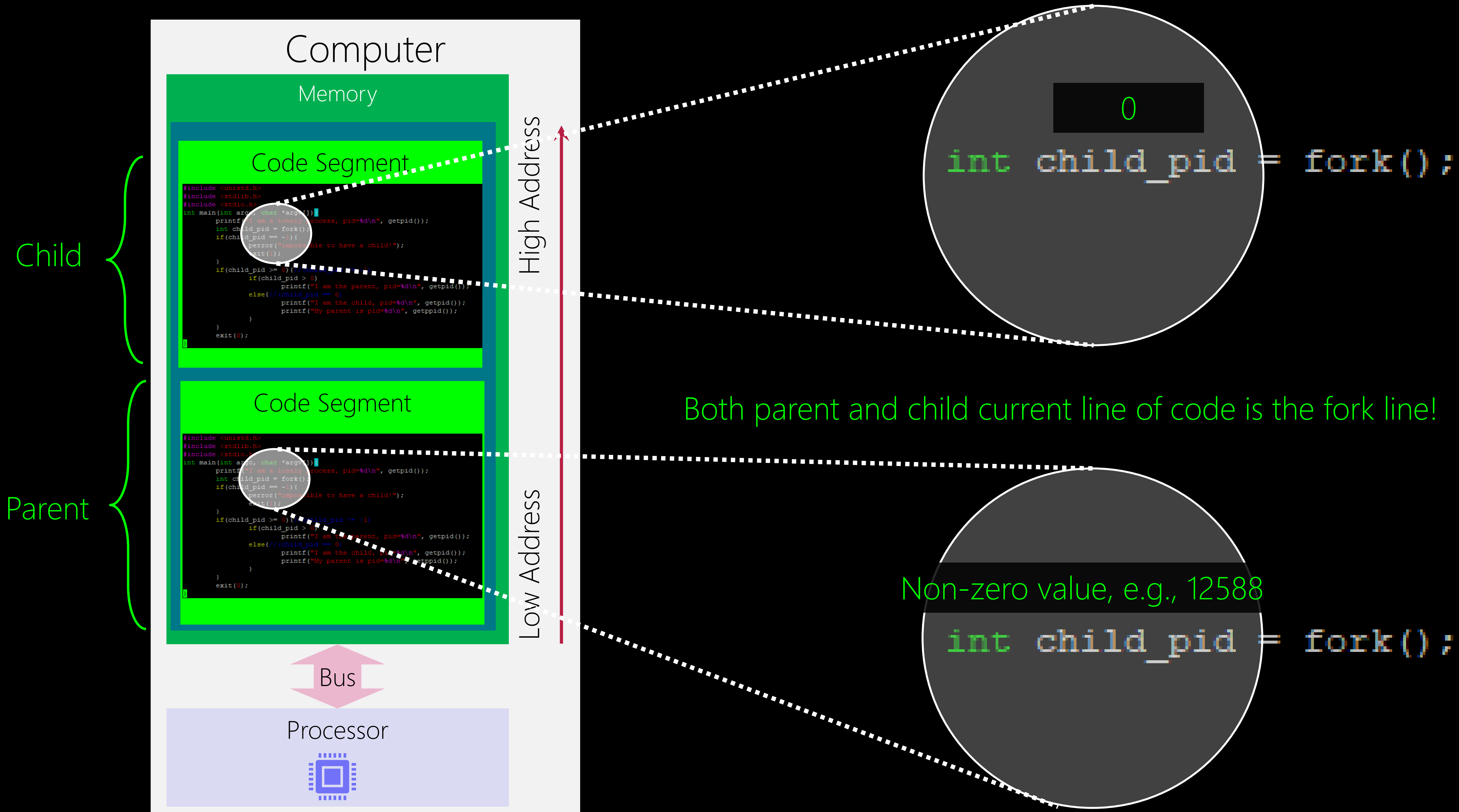
```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        printf("I am a lonely process, pid=%d\n", getpid());
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!");
                exit(1);
 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

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

Parent

13

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

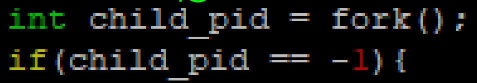
Child

0

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

Parent

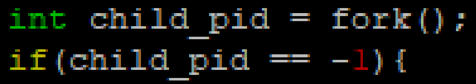
13



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

Child

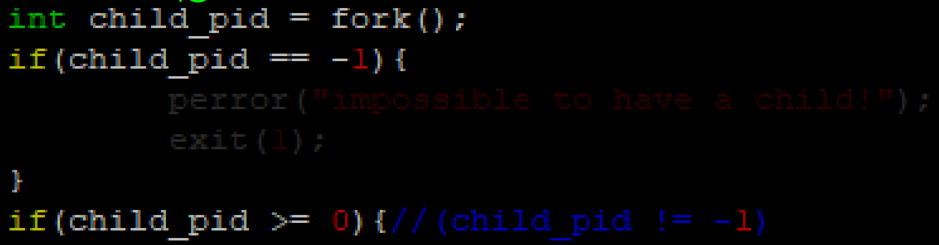
0



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

Parent

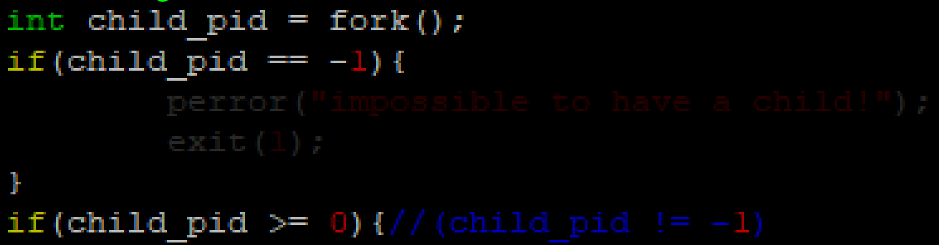
13



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

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)
```

Parent

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)
```

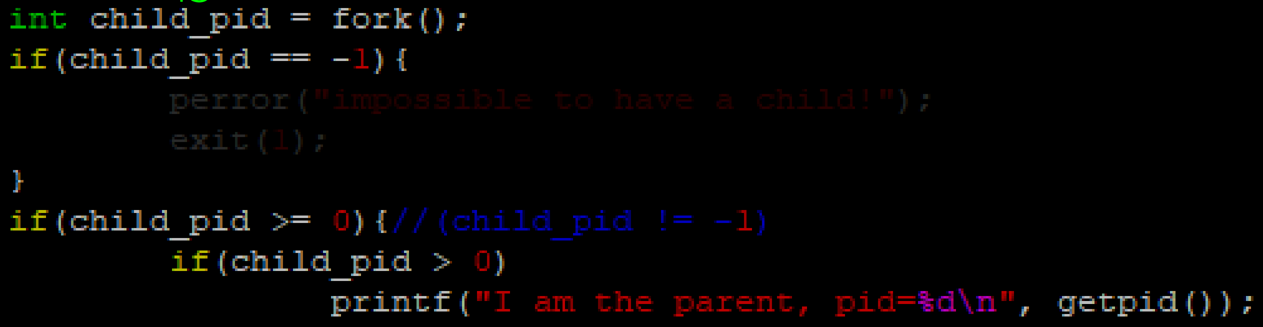
Child

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)
```

Parent

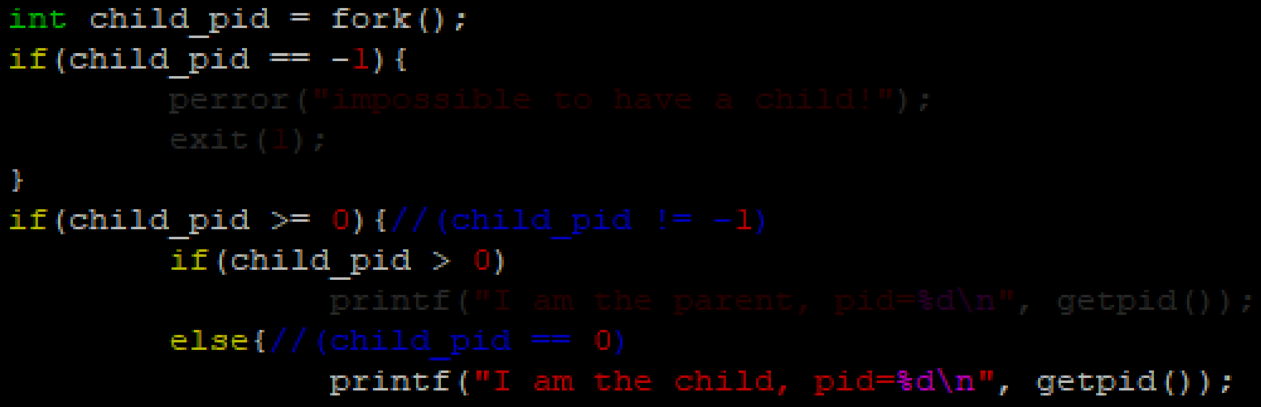
13



```
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());
```

Child

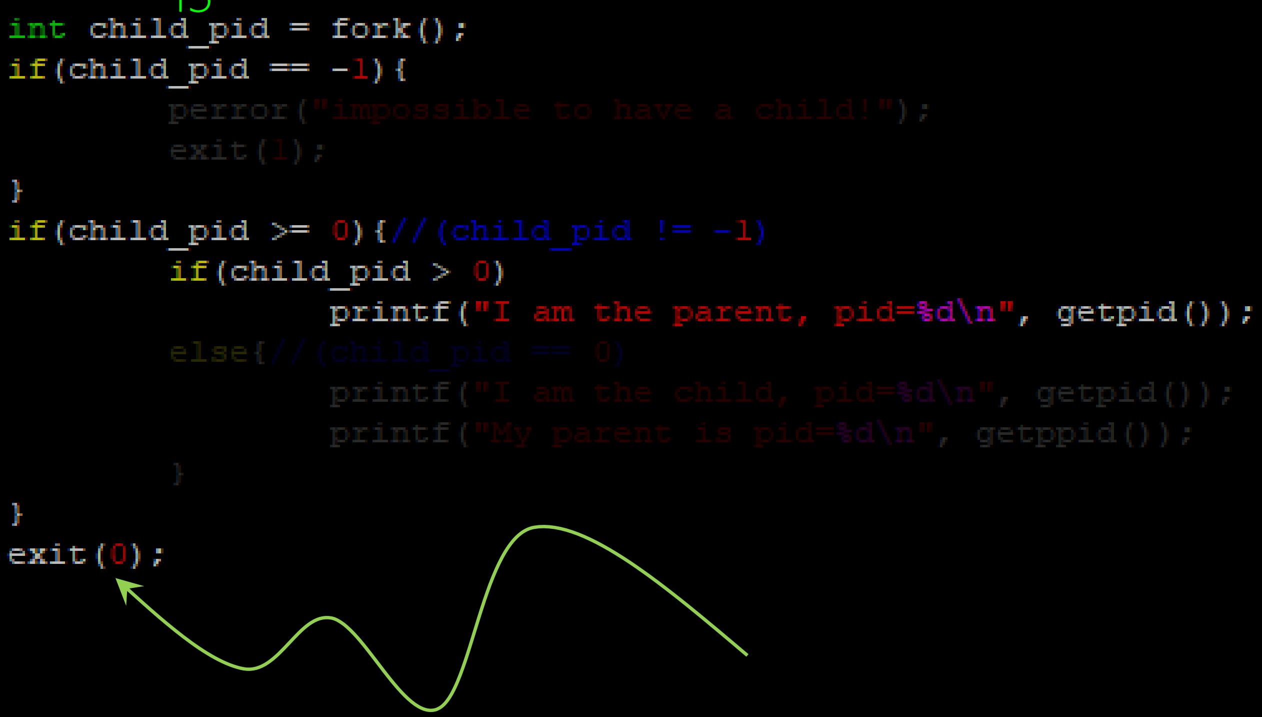
0



```
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());
```

Parent

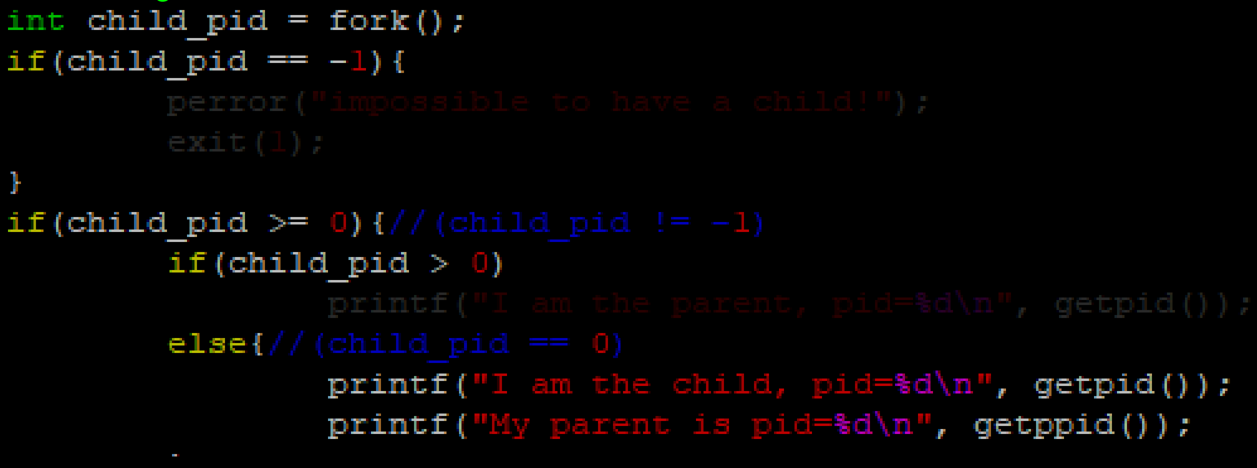
13



```
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

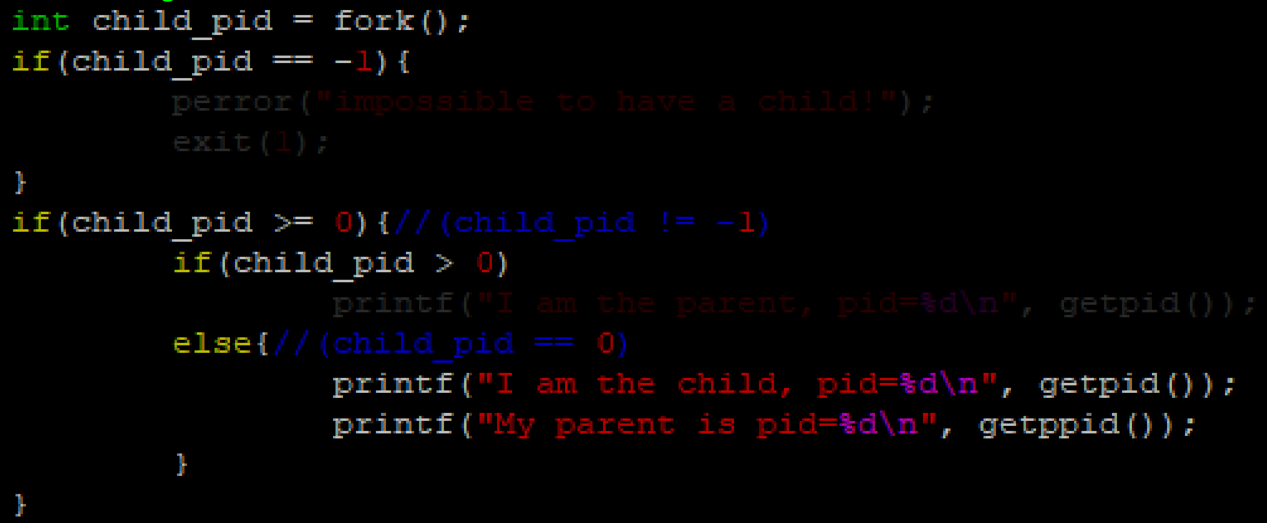
0



```
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

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());
        }
}
```

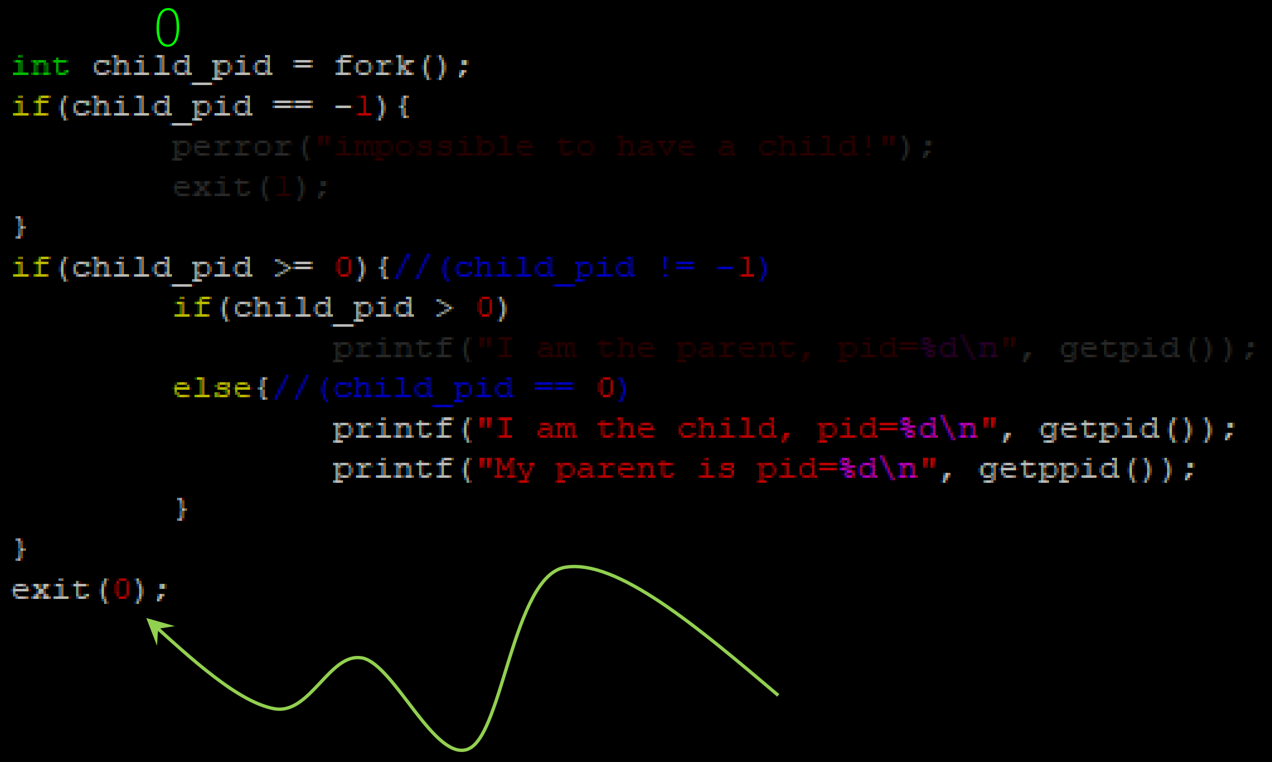
Orphan

No Parent → Kernel Adopts the Child

Child' PPID → ?

What's Kernel PID?

Child



```
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