

## Introduction

This activity explores the System Call interface in an operating system.

**Student Learning Objectives:** Students will be able to...

- Identify the other layers of the OS that the system call layer interfaces with.
- Identify the functionality in the services layer that the system call layer exposes.
- Understand the separation of interface from implementation in the system calls system.

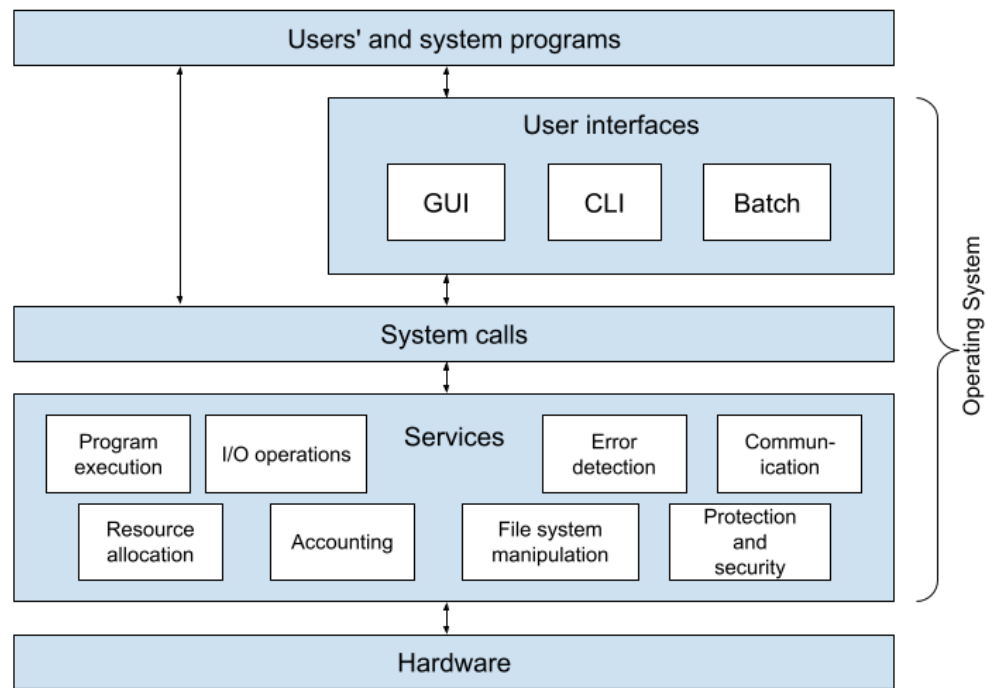
**Prerequisites:** Students must ...


- Have experience running commands and/or interfacing with an operating system.

<b>Date</b>	
<b>Team Members</b>	
<b>Member 1:</b>	
<b>Member 2:</b>	

## ***Model I. (10 minutes) Operating System Services***

The following is a picture of layers within software and hardware of a computer.



1. What are the 3 layers that comprise an operating system?
2.  With what layers does the system calls layer interface?
3. Suppose a user creates a new directory ("folder") in the file system using `mkdir CS112` in a terminal window (i.e., using the **C**ommand-**L**ine **I**nterface). Identify the layers of the system the call traverses, ending in the hardware layer.
4. Where would Siri or the Google Assistant ("OK Google") be represented in the model?

### ***Model II. (10 minutes) Example Program using System Calls***

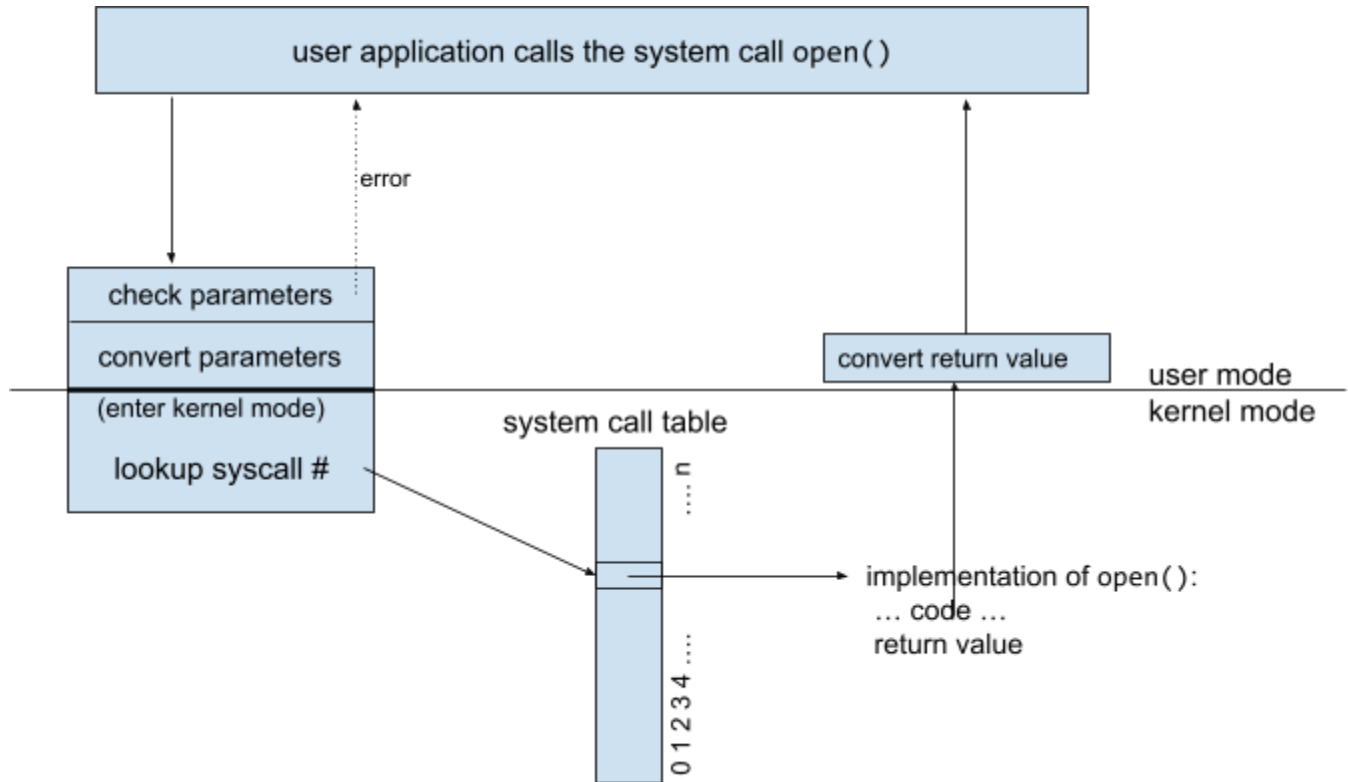
Suppose that you have a program that copies the contents of one "source file" to a new "destination file". The table below shows the steps needed to do this. The first two lines are finished for you.


Call Sequence	Line Includes System Call?	System call interfaces with...
Acquire input file name: 1) Write prompt to screen	Y	I/O operations (display)

2) Accept input from keyboard	Y	I/O operations (keyboard)
Acquire output file name: 1) Write prompt to screen		
2) Accept input from keyboard		
Open the input file		
If the file does not exist, abort the process.		
Create new output file		
If file exists, abort the process		
Repeat: 1) Read from input file		
2) Write to output file		
Until result of the read fails		
Close output file		
Write completion message to screen		
Terminate normally		

- Put a checkmark in the middle column for the lines that involve system calls.
- In the right column, indicate which boxes in the **services** box in Model I the system calls interface with. E.g., if you think the system call on a line interfaces with code in the “accounting” service, write “accounting” in the right column.

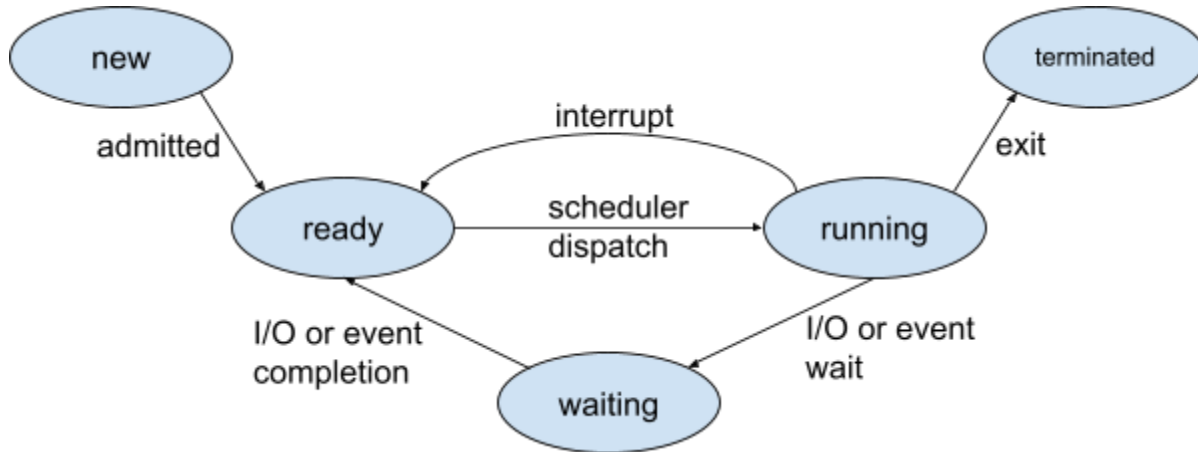
## **Model III. (5 minutes) System Call Implementation**




7. What system call is the user application making?
8. What are the two modes the code executes in?
9. What work is done in user mode?
10.  Describe the system call table. How is it indexed? What values does it hold?

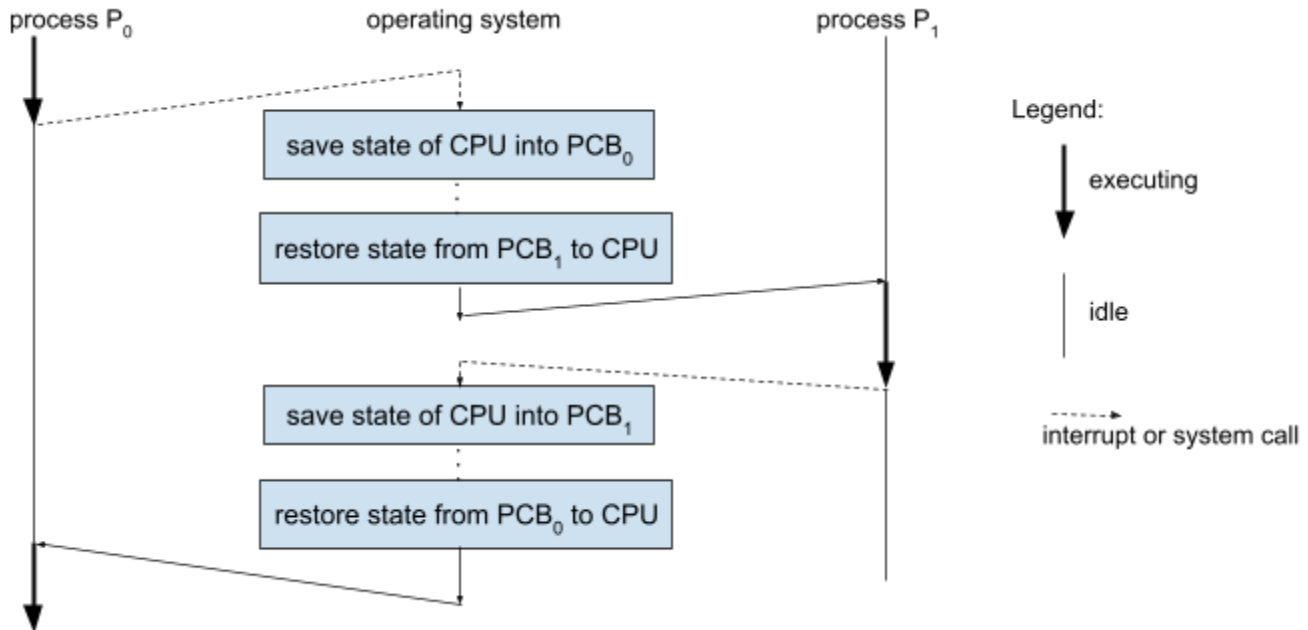
## ***Model IV. Process States (10 minutes)***

Each process starts in the **new** state and always belongs to a state. The following is a picture of the states a process can be in and how a process transitions between states.




11. What causes a process to move from **running** directly to **ready**?
12. After a process is admitted to the system, what state is it in?
13. What action causes a process to move from the **ready** state to the **running** state?
14. What two actions need to take place for a process to move from **running** to **ready** through the **waiting** state?
15. A student writes a program that does  $10000 + 10000000000$  but does not print anything out. The student runs the program. Predict the states the process goes through.
16.  The student modifies the program to do the computation and then **print out** the result. The student runs the program. Predict what states the process goes through.
17. Speculate: Can multiple processes be in the **running** state at the same time? Can multiple processes be in the **ready** state at the same time?

## ***Model V: Context Switches between Processes (5 minutes)***

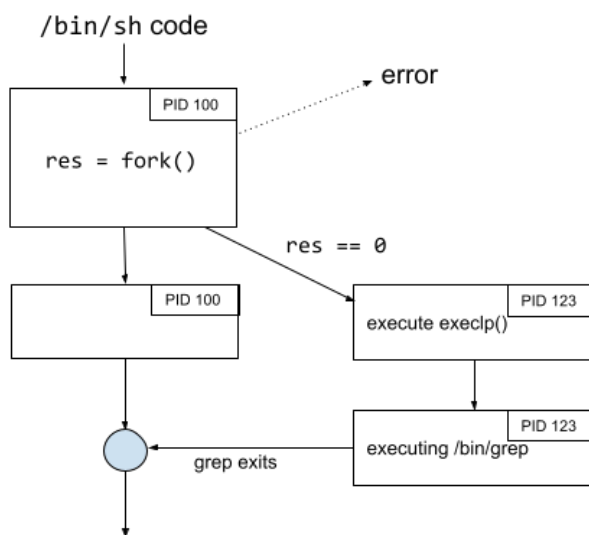


A PCB is a **Process Control Block** -- a data structure created by the operating system to hold information about each process. The illustration shows two **context switches**, from  $P_0$  to  $P_1$  and from  $P_1$  back to  $P_0$ , with the OS reading and writing values into  $PCB_0$  and  $PCB_1$ .

18. How many processes are represented in the model?
19. What event stops a process ( $P_0$  or  $P_1$ ) from executing?
20. What is executing the code that saves and reloads the state for processes?
21.  Suppose  $P_{old}$  is interrupted, the operating system takes some actions, and  $P_{new}$  starts executing. Describe what the operating system does to the two PCBs.
22. Suppose  $P_0$  stops executing because of an interrupt. What state is  $P_0$  in while it is idle (refer to what you learned in Model I)?
23. Speculate: When the OS saves the state of the CPU into  $PCB_0$ , what information might it be saving?

**Model VI: Creating Processes on Linux/Unix (10 minutes)**

To create a new process in Linux, Unix, and MacOS, two system calls must be made -- `fork()` and `exec*()` (there are many versions of `exec*()` -- `execve()`, `execlp()`, `execle()`, etc.). The `fork()` system call creates a new process in memory with new text, stack, data, and heap sections. In the example below, a student is running the `sh` command shell (`/bin/sh`), types `grep hello afile.txt`, which causes the shell to execute the given code to run `grep`.




```

res = fork() // create a child process

if res < 0:
    print "Fork failed."
    return -1
// parent and child both execute from here
else if res == 0: // child process
    execlp("/bin/grep", "grep", NULL)
    // never get here
else: // parent process
    wait(NULL)
    print "Child complete"
  
```

24. What PID does the parent process running `/bin/sh` have? What PID does the child process get?
25. The code determines that the child process is running when `res` has what value?
26. What value does `res` have in the parent process?
27. Label the unlabeled arrow coming straight down out of the `"res = fork()"` box with a label like `res ? 0`, replacing `?` with a comparison operator.
28. Correctly label the boxes pointed to by `"res == 0"` and the answer to the previous question with `"child"` and `"parent"`
29. Fill in the empty box with the name of the function called by the parent.
30. Circle the parts of the **code** where there are 2 processes (parent and child) executing that code.

31.  A function call typically returns one value. In a complete sentence (or 2), explain how a single call to `fork()` can return (at once) two different values -- `res == 0` and `res > 0`.
32. If the parent process does not run `wait(NULL)`, how does the system behavior change?
33. Draw a diagram demonstrating the creation of 2 new child processes by the first parent process (i.e., the shell). The first child runs `emacs` and the second child runs `code`.
34. When the child process runs the executable `/bin/grep` (by calling `exec1p()`), what happens to the text area of the child process (which contained the code for `/bin/sh`)?



**Model VII: Increment with Wrap-around (10 minutes)**

```
#define BUFFER_SIZE 4
typedef struct {
    . . .
} Item;
Item buffer[BUFFER_SIZE];
int index = 0; /* next location to read from */
```

35. How many Items can be stored in the buffer array?

36. The first item in the buffer array is at index 0. What is the index of the last item in the array?

37. Suppose code accesses each item in the array in order, and after accessing the last item, it needs to "wrap around" to begin reading from the beginning again. Complete the code below to increment the index and wrap it around to 0 when the index value is off the end of the array.


```
index = index + 1;

if (                ) {
    index = 0;
}
```

Recall that the % operator returns **the remainder when dividing**. Consider the following code:

```
index = (index + 1) % BUFFER_SIZE;
```

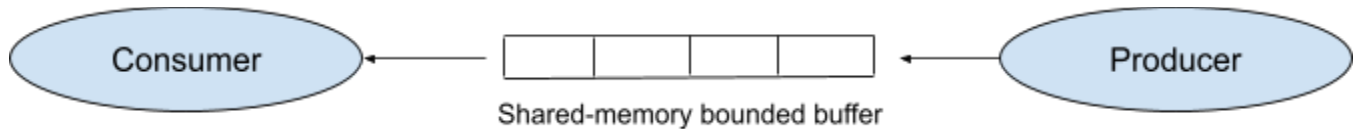
38. If the value of index was 1 initially, what value does index end up as? If the value of index was 3 initially, what value does index end up as?

39.  Suppose index has the value 0. What output is produced when the following code is executed?

```
loop 10 times:
    index = (index + 1) % BUFFER_SIZE;
    print index
```

### ***Model VIII: Producer-Consumer Problem using Shared-Memory Bounded Buffer (20 minutes)***

Two processes request a block of shared memory from the operating system, and use that memory to communicate fixed-size Items. One process produces Items while the other consumes (and processes) them in order. Consider the code below which implements this communication mechanism. The Producer process cannot produce more Items if the bounded buffer is full, and the Consumer process cannot consume Items if the bounded buffer is empty.



```

#define BUFFER_SIZE 4
typedef struct {
    . . .
} Item;

/* SHARED VARIABLES */
Item buffer[BUFFER_SIZE];
int in = 0; /* an index: the next location to write to */
int out = 0; /* an index: the next location to read from */

/* PRODUCER CODE */
Item next_produced;
while (true) {
    /* produce an Item into next_produced */
    next_produced = code_that_produces_data();
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing while buffer is full */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}

-----

/* CONSUMER CODE */
Item next_consumed;
while (true) {
    while (in == out) // while buffer is empty

```

```

        ;    /* do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    /* code to consume the Item stored in next_consumed */
    code_to_consume_data(next_consumed);
}

```

40. Initially there are 0 items in the buffer. What are the initial values for in and out?
41. What is the test the consumer uses in its while statement to check if the buffer is empty?
42. Explain why the consumer loops quickly while `in == out` ? Why can it not proceed?
43. Suppose the consumer is looping quickly in its while `in == out` code. What would have to change to make this looping stop -- i.e. to make `in == out` be false?
44. What process changes the value of in?
45. in is defined as the next location where the producer will write. Does the producer code move the in index before or after writing the Item into the buffer?
46. out is defined as the next location where the consumer will read. Does the consumer code move the out index before or after reading the Item from the buffer?
47. Assume each Item is a single character, the buffer is empty, and in and out are 0. Fill out the following table, while filling in values into the buffer below the table. Note that in is only changed by the producer and out is only changed by the consumer.


Action	Value of in after production	Value of out after consumption
Producer produces A.	1	0
Consumer consumes the next Item.		


Producer produces B.		
Producer produces C.		
Producer produces D.		

value:	A			
index:	0	1	2	3

48. Using the buffer above, `in` should have the value 0, `out` should have the value 1, and the buffer should hold the values A, B, C, and D. If the producer were to produce a new value E into slot 0, overwriting the value A, would that be OK? Why or why not? (Has A already been consumed?)


49. Assume the producer did not produce E into slot 0. How much space is left in the buffer for new values?

50.  If the producer were to produce the value E into slot 0 and change `in` to 1 (and `out` is still 1), then all slots in the buffer would have values ready to be consumed. However, look back at the answer to question 7. What problem do we have?

51.  Explain why the code in the producer adds 1 to `in` before checking against the value of `out`. I.e., why is this the test to see if the buffer is full?:

```
while (((in + 1) % BUFFER_SIZE) == out)
    ; /* do nothing while buffer is full */
```

52. For a buffer of size 4, how many items can be produced into the table before the table is full?

53.  For a buffer of size `n`, how many items can be produced into the table before the table is full?

54. Suppose the producer needs to send strings of varying lengths to the consumer. How could you use the solution above which assumes fixed-sized items?

55. When the buffer is not full, do you think the code in the producer is efficient? Why or why not?

## ***Model IX: Producer-Consumer Problem using Message Passing System Calls (5 minutes)***

Suppose that our operating system provides a message passing mechanism through these **system calls**:


- `q = create_message_q(limit);`
- `send(q, message);`
- `message = receive(q);`

The message passing mechanism implements a queue that has a maximum fixed size (like in the previous Model). When that limit is reached, a `send()` call will block until the message can be added to the queue, which will only be after a message has been removed from the queue -- that is, a consumer process has called `receive()`. Similarly, the `receive()` call will block only if there are no messages available to be received, i.e., the message queue is empty. Note that the internal implementation of the message queue does not waste any space as was the case with the shared-memory implementation.

The code of the producer is:

```
Item next_produced;  
  
while (true) {  
    next_produced = code_that_produces_data();  
    send(q, next_produced);  
}
```

56. At which line of code might the code block?

57.  Write the (4-line) code for the consumer process.

58. How does this implementation compare with the bounded-buffer implementation in terms of the amount of overhead involved **for each message that is communicated** between the two.

© Victor Norman at Calvin College, 2022 [vtn2@calvin.edu](mailto:vtn2@calvin.edu)

Version 2.1

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.