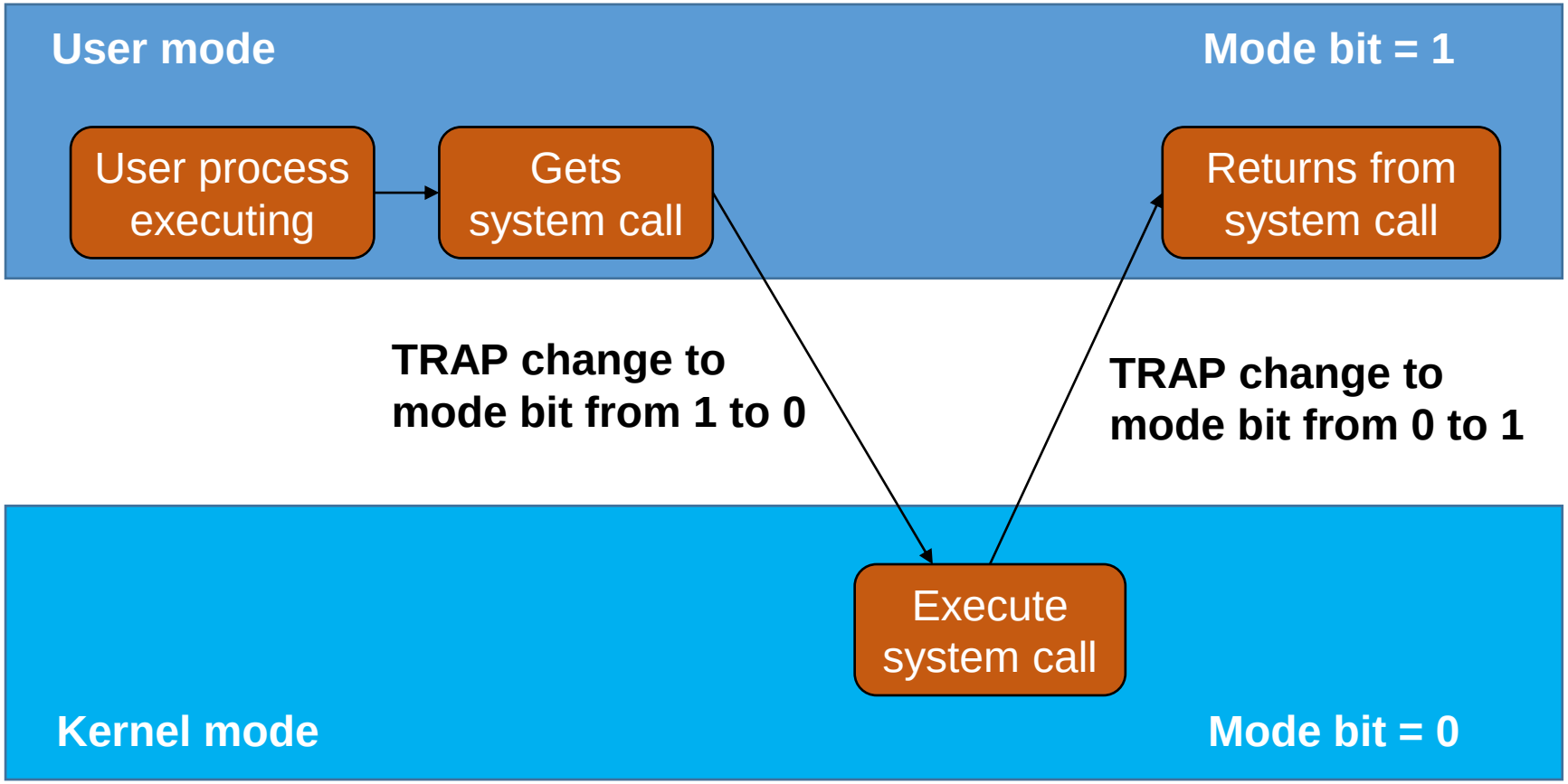
System Calls

What is a system call?

- “method for a computer program to request a service from the kernel of the operating system on which it is running”
- “method of interacting with the operating system via programs”
- “a request from computer software to an operating system's kernel”

What is a system call? (contd.)



When do we need a system call in OS?

- Read or write a file
- Create or delete a file
- Sending and receiving data packets over network connections
- Access hardware devices, including printer and scanner
- Create and manage new processes
  - Process: any program that is executed

Read/write system call

- Read/write statement
  - Contains three parameters
    - File descriptor (fd)
    - Data
    - Count in characters

```
write(1,"hello\n",6);
```

File descriptor

- For each process, we have file descriptors tables with values 0, 1, and 2
- The tables are stored in /dev/tty
  - /dev/tty is a special file, representing the terminal for the current process
- **Read from stdin => read from fd 0**
- **Write to stdout => write to fd 1**
- **Write to stderr => write to fd 2**

Write system call – example 1

```
#include<unistd.h>
int main()
{
        write(1,"hello\n",6); //1 is the file descriptor, "hello\n" is the data, 6
                              is the count of characters in data
}
```

- Write, compile, and run this program
- Change the last number – decrease and increase

Write system call – example 2

```
#include<stdio.h>
#include<unistd.h>
int main()
{
        int count;
        count=write(1,"hello\n",6);
        printf("Total bytes written: %d\n", count);
}
```

- Printing total bytes written

Read system call – example 1

```
//read.c
#include<unistd.h>
int main()
{
        char buff[20];
        read(0,buff,10);//read 10 bytes from standard input
        device(keyboard), store in buffer (buff)
        write(1,buff,10);//print 10 bytes from the buffer on the screen
}
```

- Write, compile, and run the program
- Increase and decrease the buffer size
- Do you know how much a user is going to write?

Read system call – example 2

```
#include<unistd.h>
int main()
{
int nread;
char buff[20];
nread=read(0,buff,10);//read 10 bytes from standard input
device(keyboard), store in buffer (buff)
write(1,buff,nread);//print 10 bytes from the buffer on the screen
}
```

- Taking care of the write statement

Create a file

- create() function is used to create a new empty file in C

  int **create**(char ***filename*, mode_t *mode*);

- Parameter
  - filename: name of the file
  - mode: permission of the file

- Return value
  - first unused file descriptor (usually 3)

Open system call

```
int open (const char* Path, int flags);
```

- Path: Path to the file
  - Absolute and relative path both work
- Flags: how to open the file – read-only, write-only, etc.

Open system call – example 1

```
// C program to illustrate
// open system call
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

extern int errno;

int main()
{
    // if file does not have in directory, file foo.txt is created.
    int fd = open("foo.txt", O_RDONLY | O_CREAT);

    printf("fd = %d\n", fd);

    if (fd == -1) {
        printf("Error Number % d\n", errno); // print which type of error
have in a code
        perror("Program"); // print program detail "Success or failure"
    }
    return 0;
}
```

fork()

- Used to create a (child) process

The child process uses the same program counter, CPU registers, and open files used in the parent process. It takes no parameters and returns an integer value.

- **Negative Value**: The creation of a child process was unsuccessful.
- **Zero**: Returned to the newly created child process.
- **Positive value**: Returned to parent or caller. The value contains the process ID of the newly created child process.

Example 1.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

Example 2.c

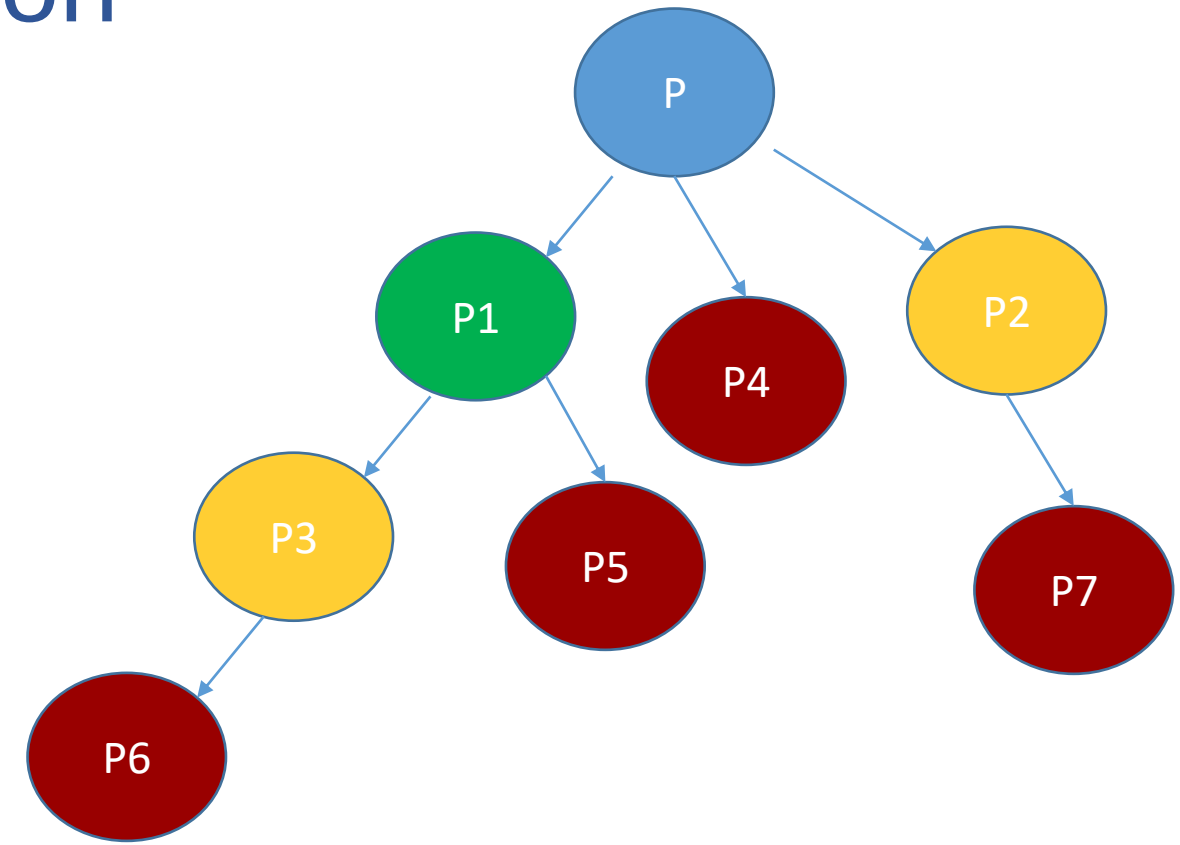
```c
// Calculate the number of times hello is printed.

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```

Example 2.c explanation

```c
// Calculate the number of t

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```



Question: How many child process?

```
for (i=0; i<n; i++)
{
        fork();
}
```

Example 3.c

```c
// Predict the Output of the following program.

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
        // child process because return value zero
        if (fork() == 0)
                printf("Hello from Child!\n");

        // parent process because return value non-zero.
        else
                printf("Hello from Parent!\n");
}
int main()
{
        forkexample();
        return 0;
}
```

```
nment/Simple_fork_programs$ ./a.out
Hello from Parent!
Hello from Child!
```

Example 4.c

```c
// Predict the Output of the following program

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void forkexample()
{
        int x = 1;

        if (fork() == 0)
                printf("Child has x = %d\n", ++x);
        else
                printf("Parent has x = %d\n", --x);
}
int main()
{
        forkexample();
        return 0;
}
```

```
Parent has x = 0
Child has x = 2
```

Example 5.c

```c
// Predict the output of the below program.

#include <stdio.h>
#include <unistd.h>
int main()
{
        fork();
        fork() && fork() || fork();
        fork();

        printf("forked\n");
        return 0;
}
```

forked printed 20 times

Example 5.c explanation - homework

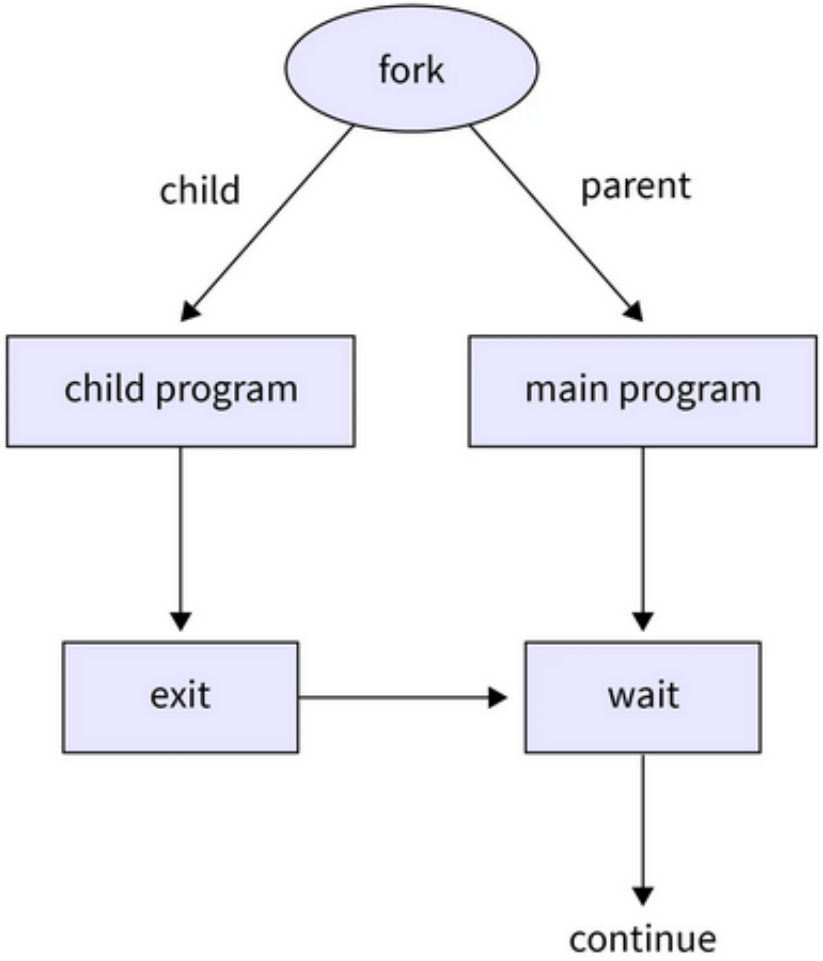
```c
// Predict the output of the below program.

#include <stdio.h>
#include <unistd.h>
int main()
{
        fork();
        fork() && fork() || fork();
        fork();

        printf("forked\n");
        return 0;
}
```

forked printed 20 times

fork() and wait(): What happens?



Example 6.c

```c
// C program to demonstrate working of wait()
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
        pid_t cpid;
        if (fork()== 0)
                exit(0);                /* terminate child */
        else
                cpid = wait(NULL); /* reaping parent */
        printf("Parent pid = %d\n", getpid());
        printf("Child pid = %d\n", cpid);

        return 0;
}
```

```
Parent pid = 1005
Child pid = 1006
```

```c
// C program to demonstrate working of wait()
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
        pid_t p;
        printf("before fork \n");
        p = fork();

        if(p == 0) //child
                {
                        printf("I am child with ID: %d \n", getpid());
                        printf("My parent's ID: %d \n", getppid());
                }
        else    // parent
                {
                        wait(NULL);
                        printf("My child's ID: %d\n", p);
                        printf("I am a parent having ID: %d\n", getpid());
                }

        printf("Common statement for Child and Parent\n");

        return 0;
}
```

Example 6.c

Acknowledgement

- The slides are prepared using information from several Online resources