

System Calls and Threads

1) Which of the following is FALSE ?

- A) Threads are cheaper to create than processes
- B) Threads are cheaper to context switch than processes
- C) A blocking user-level thread blocks the process
- D) System calls do not change to privilege mode of the processor

Solution D)

System calls do not change to privilege mode of the processor – False – a trap is generated into the kernel so we do change the privilege mode.

Threads are cheaper to create than processes – True

Threads are cheaper to context switch than processes – True – Threads don't have to save the address space.

A blocking user-level thread blocks the process – True

2) Consider the following code which invokes the fork system call.

```
char string1[] = "Hello";
char string2[] = "World";

int main( ) {
    if (fork()==0)
        printf("%s", string2);
    else
        printf("%s", string1);
    return 0;
}
```

Which of the following statement(s) is/are correct? **[MSQ]**

- A) The parent process prints "Hello".
- B) The child process prints "Hello".
- C) The parent process prints "World".
- D) The child process prints "World".

Solution: (A) (D)

Fork system call returns 0 to child and any integer > 0 to parent. Therefore, if body is executed by Child process and else body is executed by Parent. Therefore, The parent process prints

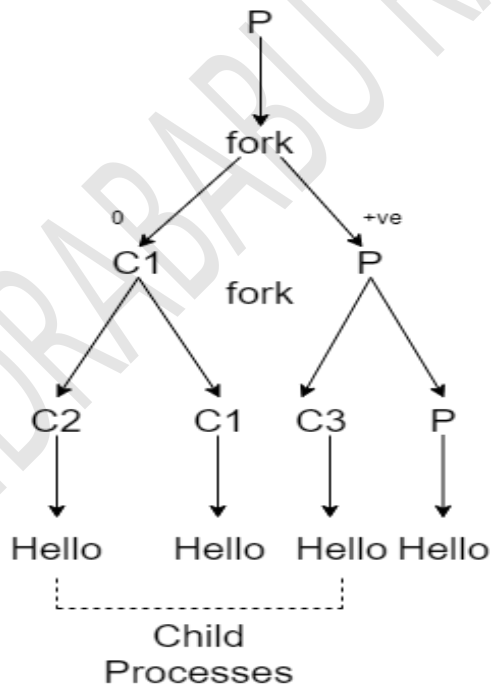
"Hello", and The child process prints "World"

3) The output of the following program is :

```
int main() {  
    fork();  
    fork();  
    printf("Hello");  
    return 0;  
}
```

- A) Hello
- B) Hello Hello
- C) Hello Hello Hello
- D) Hello Hello Hello Hello

Solution: (D)



4) How many child processes are created with following code : _____ [NAT]

```
int main() {  
    fork();  
    if(fork()==0)
```

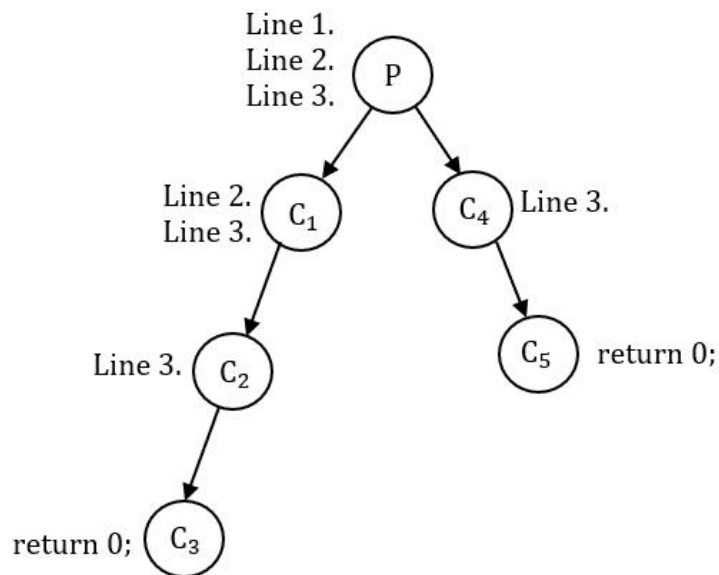
```

if(fork() != 0){
    printf("GATE EXAM 2022");
}
return 0;
}

```

Solution: 5

Line1: fork();
 Line2: if(fork()==0)
 Line3: if(fork() != 0){



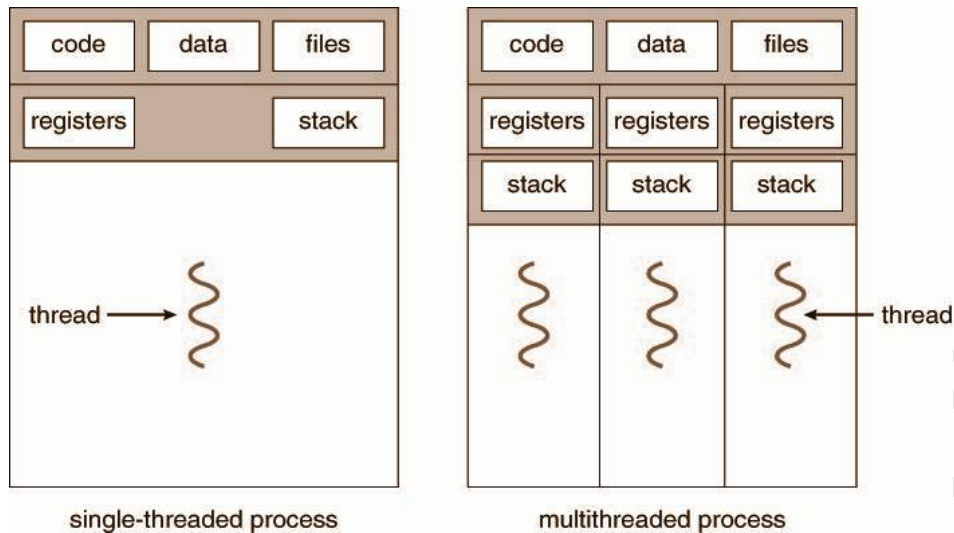
It should be noted that only one child can be created by C1. This is because line 3 is entered only if the line number 2 fork call returns 0, which creates C2. But C1 (as parent) can never enter line 3.

5) Which of the following are correct about Threads? **[MSQ]**

- A) A thread has no data segment.
- B) Each thread has its own stack.
- C) There can be more than one thread in a process.
- D) Each thread has its own heap.

Solution: (A) (B) (C)

A thread has no data segment or heap of its own. There can be more than one thread in a process. Each thread has its own stack.



6) Which of the following information about the process is identical for a parent and the newly created child processes, immediately after the fork system call?

- A) The process id (PID)
- B) The file descriptor (fd) opened by the parent.
- C) Both of these.
- D) None of these.

Solution (B)

Every process has its own unique PID in the system, and hence parent and child will have different PID. However, as the child gets an exact copy of the parent's file descriptor table, the fd will be the same.

7) Which of the following are correct about Threads? **[MSQ]**

- A) Threads in a process can execute different parts of the program code at the same time.
- B) Threads can also execute the same parts of the code at the same time, but with different execution state.
- C) Processes start out execution with a single main thread.
- D) A thread can also create new threads.

Solution: (A) (B) (C) (D)

Threads in a process can execute different parts of the program code at the same time. They can also execute the same parts of the code at the same time, but with different execution state:

- i) They have independent current instructions; that is, they have (or appear to have) independent program counters.
- ii) They are working with different data; that is, they are (or appear to be) working with independent registers.

Processes start out with a single main thread. The main thread can create new threads using a system call. The new threads can also use this system call to create more threads. Consequently, a thread not only belongs to a process; it also has a parent thread - the thread that created it.

8) Whenever there is a need to switch between the threads of the same process, the context of a thread will be saved and restored in the form of

- A) code, data, and stack
- B) code, CPU registers, and stack
- C) PC, CPU registers, and stack
- D) None

Solution (C)

There is no need to save or restore the code section, data section, or state of the resources. The code, data, and resource state is with the process only, but execution of a process is in the form of threads (PC, CPU registers and stack).

9) Consider the following code :

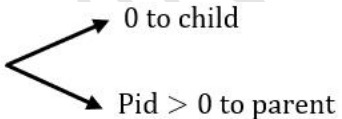
```
for(i=0; fork() ; i++)  
    if(i==n) break;
```

How many times the fork statement is called ?

- A) $2^{(n+1)}$
- B) 2^n
- C) n
- D) $n+1$

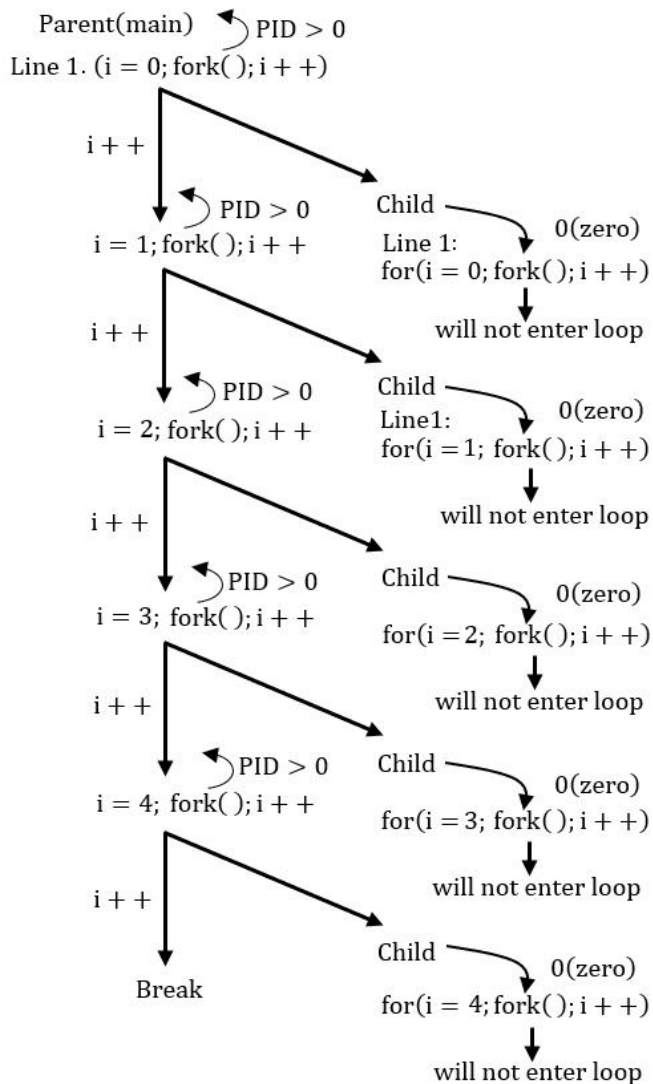
Solution (D)

Let us verify for $n=4$. Then we can generalize to n .

fork returns 

Line 1. for (i=0; fork(); i++)

Line 2. if (i==4) {break;}



Fork if called total 5 times.

10) Which of the followings is TRUE about threads?

- A. Different threads share the same heap.
- B. Different threads share the same stack.
- C. Different threads share the same Program Counter (PC).
- D. None of them.

Solution: (A)

Different threads share the same heap.

11) Which of the followings are true about the relationship between kernel and user level threads?

- (I) Both of them are faster to create than creating a process

(II) When a kernel level thread execute an I/O system call, the other kernel threads within the same process are blocked as well.

(III) When a user level thread execute an I/O system call, the other user threads within the same process are blocked as well.

A. I only.

B. I and II. C. I and III.

D. I, II and III.

Solution: (C)

By the definition and properties of user-level and kernel level threads, statement (I) and (III) holds.

12) Select the correct statement(s) from the following **[MSQ]**

A) Each thread has their own program counter, stack and set of registers.

B) Threads share common code, data, and certain structures such as open files.

C) Processes have a single thread of control.

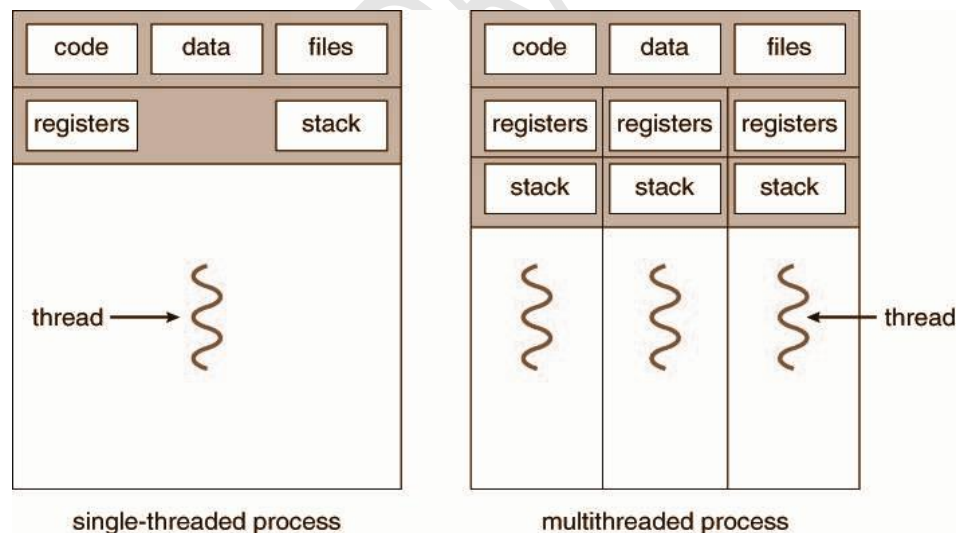
D) A thread is a basic unit of CPU utilization, also called a lightweight process.

Solution : (A) (B) (C) (D)

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.)

Traditional (heavyweight) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.

As shown in Figure, multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.



13) What is the purpose of the fork() system call in Unix-like operating systems?

A) To terminate a process

B) To create a new process

- C) To read data from a file
- D) To write data to a file

Solution: B) To create a new process

Explanation: The fork() system call is used to create a new process by duplicating the existing process.

14) Which system call is used to terminate a process in Unix-like operating systems?

- A) fork()
- B) exec()
- C) getpid()
- D) exit()

Solution: D) exit()

Explanation: The exit() system call is used to terminate a process in Unix-like operating systems. It is typically called by a process when it has finished its execution.

15) What system call is used to open a file in Unix-like operating systems?

- A) open()
- B) close()
- C) read()
- D) write()

Solution: A) open()

Explanation: The open() system call is used to open a file in Unix-like operating systems. It returns a file descriptor that can be used for subsequent I/O operations on the file.

16) Which of the following system calls is used to read data from a file in Unix-like operating systems?

- A) read()
- B) write()
- C) getpid()
- D) fork()

Solution: A) read()

Explanation: The read() system call is used to read data from a file in Unix-like operating systems. It takes a file descriptor, a buffer to read the data into, and the number of bytes to read as arguments.

17) What is the purpose of the getpid() system call in Unix-like operating systems?

- A) To create a new process
- B) To retrieve the process ID of the calling process
- C) To open a file
- D) To terminate a process

Solution: B)

To retrieve the process ID of the calling process

Explanation: The getpid() system call is used to retrieve the process ID (PID) of the calling process. The PID is a unique identifier assigned to each process by the operating system.

18) Which system call is used to allocate a contiguous block of memory in Unix-like operating systems?

- A) malloc()
- B) free()
- C) mmap()
- D) brk()

Solution: A) malloc()

Explanation: The malloc() system call is used to allocate a contiguous block of memory dynamically in Unix-like operating systems.

19) What is the purpose of the free() system call in memory management?

- A) To allocate memory
- B) To deallocate memory
- C) To map files into memory
- D) To change the size of the data segment

Solution: B) To deallocate memory

Explanation: The free() system call is used to deallocate memory that was previously allocated using malloc() or a similar memory allocation function.

20) Which system call is used to change the size of the data segment in Unix-like operating systems?

- A) sbrk()
- B) mmap()
- C) munmap()
- D) brk()

Solution: D) brk()

Explanation: The brk() system call is used to set the end of the data segment to a specified value, effectively changing the size of the data segment.

21) Which of the following is a benefit of using threads in a program?

- A) Increased process isolation
- B) Reduced concurrency
- C) Improved responsiveness
- D) Simplified resource management

Solution: C) Improved responsiveness

Explanation: Threads allow concurrent execution within a process, which can lead to improved responsiveness, as tasks can be executed concurrently without blocking the main thread.

22) Identify the correct statement?

- A) User-level threads are managed by the kernel, while kernel-level threads are managed by user-space libraries.
- B) User-level threads have their own execution context, while kernel-level threads share the same execution context.
- C) User-level threads are more efficient than kernel-level threads.
- D) Kernel-level threads have higher priority than user-level threads.

Solution: B) User-level threads have their own execution context, while kernel-level threads share the same execution context.

Explanation: User-level threads are managed entirely by user-space libraries and have their own execution context, while kernel-level threads are managed by the operating system kernel and share the same execution context.

23) What is the primary advantage of User-Level Threads (ULT) over Kernel-Level Threads (KLT)?

- A) ULTs provide better concurrency
- B) ULTs are easier to implement
- C) ULTs have lower context-switching overhead
- D) ULTs have higher priority than KLTs

Solution: C) ULTs have lower context-switching overhead

Explanation: User-Level Threads (ULTs) have lower context-switching overhead compared to Kernel-Level Threads (KLTs) because thread management is handled entirely in user space without kernel involvement.

24) Which of the following is true regarding Kernel-Level Threads (KLT)?

- A) Each KLT has its own program counter
- B) KLTs are managed entirely by user-space libraries
- C) KLTs provide better application isolation
- D) KLTs are created and managed by the operating system kernel

Solution: D) KLTs are created and managed by the operating system kernel

Explanation: Kernel-Level Threads (KLTs) are created and managed by the operating system kernel, and each KLT has its own kernel-level data structure representing the thread.