1.What is a system call ? Explain any 3 system calls with appropriate example program.

ANSWER 1: A system call is a way for programs to interact with the operating system. It provides an interface between a process and the operating system. A process can request services from the operating system by making a system call. The operating system then performs the requested service on behalf of the process.

Here are three examples of system calls:

1. **open()**: This system call is used to open a file. It takes two arguments: the name of the file to be opened and the mode in which it should be opened. Here’s an example program that uses open() to open a file:

#include <stdio.h>

#include <fcntl.h>

int main() {

int fd;

fd = open("file.txt", O\_RDONLY);

if (fd == -1) {

printf("Error opening file\n");

return 1;

}

printf("File opened successfully\n");

close(fd);

return 0;

}

1. **read()**: This system call is used to read data from a file. It takes three arguments: the file descriptor, a buffer to store the data, and the number of bytes to read. Here’s an example program that uses read() to read data from a file:

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[100];

ssize\_t nread;

fd = open("file.txt", O\_RDONLY);

if (fd == -1) {

printf("Error opening file\n");

return 1;

}

nread = read(fd, buffer, sizeof(buffer));

if (nread == -1) {

printf("Error reading file\n");

return 1;

}

printf("Read %ld bytes: %s\n", nread, buffer);

close(fd);

return 0;

}

1. **write()**: This system call is used to write data to a file. It takes three arguments: the file descriptor, a buffer containing the data to be written, and the number of bytes to write. Here’s an example program that uses write() to write data to a file:

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[] = "Hello, world!\n";

ssize\_t nwritten;

fd = open("file.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

printf("Error opening file\n");

return 1;

}

nwritten = write(fd, buffer, sizeof(buffer));

if (nwritten == -1) {

printf("Error writing file\n");

return 1;

}

printf("Wrote %ld bytes\n", nwritten);

close(fd);

return 0;

}

2.Write a simple program for Process creation with sample output.

ANSWER 2: A process is a program in execution. [A process creation in operating systems is achieved through the **fork()** system call1](https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_process_creation_termination.htm). [The newly created process is called the **child process**, and the process that initiated it is called the **parent process**1](https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_process_creation_termination.htm)[2](https://www.geeksforgeeks.org/process-creation-and-deletions-in-operating-systems/). [After the fork() system call, there are two processes - parent and child processes1](https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_process_creation_termination.htm)[2](https://www.geeksforgeeks.org/process-creation-and-deletions-in-operating-systems/). A process can create several new processes through creating process system calls during the process execution. Every new process creates another process, forming a tree-like structure[1](https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_process_creation_termination.htm).

Here is a simple program for process creation in C language:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

pid\_t pid, mypid, myppid; // declare variables for process ids

pid = getpid(); // get the current process id

printf("Before fork: Process id is %d\n", pid); // print the current process id

pid = fork(); // create a new child process

if (pid < 0) { // check if fork() failed

perror("fork() failure\n"); // print an error message

return 1; // exit with non-zero status

}

// Child process

if (pid == 0) {

printf("This is child process\n"); // print a message

mypid = getpid(); // get the child process id

myppid = getppid(); // get the parent process id

printf("Process id is %d and PPID is %d\n", mypid, myppid); // print the ids

}

else {

// Parent process

sleep(2); // wait for 2 seconds

printf("This is parent process\n"); // print a message

mypid = getpid(); // get the parent process id

myppid = getppid(); // get the parent of parent process id

printf("Process id is %d and PPID is %d\n", mypid, myppid); // print the ids

printf("Newly created process id or child pid is %d\n", pid); // print the child process id

}

return 0; // exit with zero status

}

Here is a sample output of the program:

Before fork: Process id is 1234

This is child process

Process id is 1235 and PPID is 1234

This is parent process

Process id is 1234 and PPID is 1000

Newly created process id or child pid is 1235

3.Differentiate between wait() and exit() system calls.

ANSWER 3: The wait() and exit() system calls are related to the process termination in operating systems. Here are some differences between them:

| **wait()** | **exit()** |
| --- | --- |
| A call to wait() blocks the calling process until one of its child processes exits or a signal is received[1](https://www.geeksforgeeks.org/wait-system-call-c/)[2](https://programsolve.com/wait-system-call-in-c-program-with-examples/). | [A call to exit() terminates the current process and returns an exit status to its parent process3](https://vitux.com/fork-exec-wait-and-exit-system-call-explained-in-linux/)[4](https://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/pc/pc5.htm). |
| The wait() system call returns the process ID of the terminated child process, or -1 if there is no child process[1](https://www.geeksforgeeks.org/wait-system-call-c/). | [The exit() system call takes an integer argument as the exit status, which can be accessed by the parent process using WEXITSTATUS macro3](https://vitux.com/fork-exec-wait-and-exit-system-call-explained-in-linux/)[4](https://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/pc/pc5.htm). |
| The wait() system call also fills a pointer argument with the status information about the child process, which can be used to check the cause and mode of termination using WIF… macros[1](https://www.geeksforgeeks.org/wait-system-call-c/). | The exit() system call can be invoked explicitly by the process using exit() function, or implicitly by returning from the main function or receiving a terminating signal[3](https://vitux.com/fork-exec-wait-and-exit-system-call-explained-in-linux/)[4](https://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/pc/pc5.htm). |
| The wait() system call can be used to synchronize the parent and child processes, and to avoid zombie processes[1](https://www.geeksforgeeks.org/wait-system-call-c/)[2](https://programsolve.com/wait-system-call-in-c-program-with-examples/). | [The exit() system call can be used to indicate the success or failure of the process execution, and to release the resources held by the process3](https://vitux.com/fork-exec-wait-and-exit-system-call-explained-in-linux/)[4](https://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/pc/pc5.htm). |

4.What are the advantages of a multiprocessor system? Explain about time sharing system.

ANSWER 4: A multiprocessor system is a computer system that has two or more central processing units (CPUs) that share access to a common memory. A multiprocessor system can have several advantages over a single-processor system, such as:

* [**High throughput**: A multiprocessor system can execute more tasks in a given time than a single-processor system, as the workload can be distributed among the processors1](https://www.itrelease.com/2020/06/advantages-and-disadvantages-of-multiprocessor-systems/)[2](https://www.geeksforgeeks.org/introduction-of-multiprocessor-and-multicomputer/).
* [**Better scalability**: A multiprocessor system can be easily expanded to handle increased workloads by adding more processors to the system1](https://www.itrelease.com/2020/06/advantages-and-disadvantages-of-multiprocessor-systems/)[2](https://www.geeksforgeeks.org/introduction-of-multiprocessor-and-multicomputer/).
* **Increased reliability**: A multiprocessor system can continue to operate even if one processor fails, as the remaining processors can take over the tasks of the failed one[1](https://www.itrelease.com/2020/06/advantages-and-disadvantages-of-multiprocessor-systems/)[2](https://www.geeksforgeeks.org/introduction-of-multiprocessor-and-multicomputer/).
* **Reduced cost**: A multiprocessor system can be more cost-effective than building multiple single-processor systems to handle the same workload[1](https://www.itrelease.com/2020/06/advantages-and-disadvantages-of-multiprocessor-systems/)[2](https://www.geeksforgeeks.org/introduction-of-multiprocessor-and-multicomputer/).
* **Enhanced parallelism**: A multiprocessor system can perform different tasks simultaneously by using different processors, which can improve the performance of parallel applications[1](https://www.itrelease.com/2020/06/advantages-and-disadvantages-of-multiprocessor-systems/)[2](https://www.geeksforgeeks.org/introduction-of-multiprocessor-and-multicomputer/).

A time-sharing system is a computer system that allows multiple users to access and use the same computer resources at the same time. A time-sharing system uses a scheduling algorithm to allocate CPU time to each user according to some criteria, such as priority, fairness, or efficiency. A time-sharing system can have several advantages over a batch processing system, such as:

* **Improved resource utilization**: A time-sharing system can make better use of the CPU and other resources by switching between users rapidly and avoiding idle periods[3](https://www.britannica.com/technology/multiprocessing) .
* **Increased user interaction**: A time-sharing system can provide faster feedback and response to the users by allowing them to interact with the computer through terminals or graphical user interfaces[3](https://www.britannica.com/technology/multiprocessing) .
* [**Enhanced multitasking**: A time-sharing system can enable users to run multiple programs or tasks concurrently by dividing the CPU time among them3](https://www.britannica.com/technology/multiprocessing) .
* [**Reduced cost**: A time-sharing system can reduce the cost of hardware and software by sharing them among multiple users instead of requiring each user to have their own dedicated system3](https://www.britannica.com/technology/multiprocessing) .