**ASSIGNMENT 3**

**AIM:** Implement following programs to exhibit UNIX Process Control "Program where parent process sorts array elements in ascending order and child process sorts array elements in descending order. Show the demonstration of wait() and zombie process".

**THEORY:**

1. **What is a process?**

A process is a program in execution. It is defined as an entity which represents the basic unit of work to be implemented in the system. Attributes held by process include hardware state, memory, CPU etc. A process is an ‘active’ entity, as opposed to a program, which is considered to be a ‘passive’ entity. A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created).

1. **What is Process Control Block?**

While creating a process the operating system performs several operations. To identify these processes, it must identify each process, hence it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process’s execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc. All these information is required and must be saved when the process is switched from one state to another. When the process made transitions from one state to another, the operating system must update information in the process’s PCB.

1. **Explain the fork() system call.**

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: 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 process ID of newly created child process.

1. **Explain the exec() system call.**

The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program. It comes under the header file unistd.h.There are many members in the exec family like:

* execvp
* execv
* execlp and execl
* execvpe and execle

1. **Explain the wait() system call.**

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.  
Child process may terminate due to any of these:

* It calls exit();
* It returns (an int) from main
* It receives a signal (from the OS or another process) whose default action is to terminate.

If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.

If only one child process is terminated, then return a wait() returns process ID of the terminated child process.

If more than one child processes are terminated than wait() reap any arbitrarily child and return a process ID of that child process.  
When wait() returns they also define exit status (which tells our, a process why terminated) via pointer, If status are not NULL.

If any process has no child process then wait() returns immediately “-1”.

1. **What is a zombie process?**

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. Zombie processes usually occur for child processes, as the parent process still needs to read its child’s exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as reaping the zombie process.

The exit status of the zombie process zombie process can be read by the parent process using the wait() system call. After that, the zombie process is removed from the system. Then the process ID and the process table entry of the zombie process can be reused.

If the parent process is not running anymore, then the presence of a zombie process indicates an operating system bug. This may not be a serious problem if there are a few zombie processes but under heavier loads, this can create issues for the system such as running out of process table entries.

1. **What is an orphan process?**

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process. A process can be orphaned intentionally or unintentionally.

An intentionally orphaned process runs in the background without any manual support. This is usually done to start an indefinitely running service or to complete a long-running job without user attention.

An unintentionally orphaned process is created when its parent process crashes or terminates. Unintentional orphan processes can be avoided using the process group mechanism.